International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

H.264

(06/2011)

SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS

Infrastructure of audiovisual services – Coding of moving video

Advanced video coding for generic audiovisual services

Recommendation ITU-T H.264

ITU-T

International Telecommunication Union

ITU-T H-SERIES RECOMMENDATIONS

AUDIOVISUAL AND MULTIMEDIA SYSTEMS

CHARACTERISTICS OF VISUAL TELEPHONE SYSTEMS	H.100–H.199
INFRASTRUCTURE OF AUDIOVISUAL SERVICES	
General	H.200–H.219
Transmission multiplexing and synchronization	H.220–H.229
Systems aspects	H.230–H.239
Communication procedures	H.240–H.259
Coding of moving video	**H.260–H.279**
Related systems aspects	H.280–H.299
Systems and terminal equipment for audiovisual services	H.300–H.349
Directory services architecture for audiovisual and multimedia services	H.350–H.359
Quality of service architecture for audiovisual and multimedia services	H.360–H.369
Supplementary services for multimedia	H.450–H.499
MOBILITY AND COLLABORATION PROCEDURES	
Overview of Mobility and Collaboration, definitions, protocols and procedures	H.500–H.509
Mobility for H-Series multimedia systems and services	H.510–H.519
Mobile multimedia collaboration applications and services	H.520–H.529
Security for mobile multimedia systems and services	H.530–H.539
Security for mobile multimedia collaboration applications and services	H.540–H.549
Mobility interworking procedures	H.550–H.559
Mobile multimedia collaboration inter-working procedures	H.560–H.569
BROADBAND, TRIPLE-PLAY AND ADVANCED MULTIMEDIA SERVICES	
Broadband multimedia services over VDSL	H.610–H.619
Advanced multimedia services and applications	H.620–H.629
IPTV MULTIMEDIA SERVICES AND APPLICATIONS FOR IPTV	
General aspects	H.700–H.719
IPTV terminal devices	H.720–H.729
IPTV middleware	H.730–H.739
IPTV application event handling	H.740–H.749
IPTV metadata	H.750–H.759
IPTV multimedia application frameworks	H.760–H.769
IPTV service discovery up to consumption	H.770–H.779

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T H.264

Advanced video coding for generic audiovisual services

Summary

Recommendation ITU-T H.264 | International Standard ISO/IEC 14496-10 represents an evolution of the existing video coding standards (ITU-T H.261, ITU-T H.262, and ITU-T H.263) and it 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.

The revision approved 2005-03 contained modifications of the video coding standard to add four new profiles, referred to as the High, High 10, High 4:2:2, and High 4:4:4 profiles, to improve video quality capability and to extend the range of applications addressed by the standard (for example, by including support for a greater range of picture sample precision and higher-resolution chroma formats). Additionally, a definition of new types of supplemental data was specified to further broaden the applicability of the video coding standard. Finally, a number of corrections to errors in the published text were included.

Corrigendum 1 to Rec. ITU-T H.264 corrected and updated various minor aspects to bring the ITU-T version of the text up to date relative to the April 2005 output status approved as a new edition of the corresponding jointly-developed and technically-aligned text ISO/IEC 14496-10. It additionally fixed a number of minor errors and needs for clarification and defined three previously-reserved sample aspect ratio indicators.

Amendment 1 "Support of additional colour spaces and removal of the High 4:4:4 Profile" contained alterations to Rec. ITU-T H.264 | ISO/IEC 14496-10 Advanced Video Coding to specify the support of additional colour spaces and to remove the definition of the High 4:4:4 Profile.

NOTE – Rec. ITU-T H.264 is a twin text with ISO/IEC 14496-10 and this amendment was published in two different documents in the ISO/IEC series:

– The removal of the High 4:4:4 profile was found in ISO/IEC 14496-10:2005/Cor.2.

– The specification for support of additional colour spaces was found in ISO/IEC 14496-10:2005/Amd.1.

Amendment 2 "New profiles for professional applications" contained extensions to Rec. ITU-T H.264 | ISO/IEC 14496-10 Advanced Video Coding to specify the support of five additional profiles intended primarily for professional applications (the High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, CAVLC 4:4:4 Intra, and High 4:4:4 Predictive profiles) and two new types of supplemental enhancement information (SEI) messages (the post-filter hint SEI message and the tone mapping information SEI message).

Amendment 3 "Scalable video coding" contained extensions to Rec. ITU-T H.264 | ISO/IEC 14496-10 Advanced Video Coding to specify a scalable video coding extension in three profiles (the Scalable Baseline, Scalable High, and Scalable High Intra profiles).

The ITU-T H.264 edition published in 2005-11 included the text approved 2005-03 and its Corrigendum 1 approved 2005-09. ITU-T H.264 (2005) Amd.2 (2007) was available only as pre-published text since it was superseded by ITU-T H.264 Amd.3 (2007-11) before its publication; further, ITU-T H.264 Amd.3 was not published separately. This third edition integrated into the ITU-T H.264 edition published in 2005-11 all changes approved in Amendments 1 (2006-06), 2 (2007-04) and 3 (2007-11).

Corrigendum 1 (2009) provides a significant number of minor corrections, clarifications, consistency improvements and formatting improvements drafted in response to accumulated errata reports collected since publication of the 2nd edition (dated 2005-03, which included a Cor.1 approved 2005-09).

The ITU-T H.264 edition published in 2009-05 contained enhancement extensions to support multiview video coding (MVC), specification of a "Constrained Baseline Profile", and some miscellaneous corrections and clarifications.

The ITU-T H.264 edition published in 2010-03 contained the specification of a new profile (the Stereo High profile) for two-view video coding with support of interlaced coding tools, the specification a new SEI message (the frame packing arrangement SEI message), and some miscellaneous corrections and clarifications.

This revision to Rec. ITU-T H.264 contains the specification of a new level (Level 5.2) supporting higher processing rates in terms of maximum macroblocks per second, a new profile (the Progressive High profile) to enable implementation of decoders supporting only the frame coding tools of the previously specified High profile, and includes miscellaneous corrections and clarifications.

History

Edition	Recommendation	Approval	Study Group
1.0	ITU-T H.264	2003-05-30	16
1.1	ITU-T H.264 (2003) Cor. 1	2004-05-07	16
2.0	ITU-T H.264	2005-03-01	16
2.1	ITU-T H.264 (2005) Cor. 1	2005-09-13	16
2.2	ITU-T H.264 (2005) Amd. 1	2006-06-13	16
2.3	ITU-T H.264 (2005) Amd. 2	2007-04-06	16
3.0	ITU-T H.264	2007-11-22	16
3.1	ITU-T H.264 (2007) Cor. 1	2009-01-13	16
4.0	ITU-T H.264	2009-03-16	16
5.0	ITU-T H.264	2010-03-09	16
6.0	ITU-T H.264	2011-06-29	16

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

Foreword

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardising telecommunications on a world-wide basis. The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups that, in turn, produce Recommendations on these topics. The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1. In some areas of information technology that fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialised system for world-wide standardisation. National Bodies that are members of ISO and IEC participate in the development of International Standards through technical committees established by the respective organisation to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organisations, 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. 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.

This Recommendation | International Standard was prepared jointly by ITU-T SG 16 Q.6, also known as VCEG (Video Coding Experts Group), and by ISO/IEC JTC 1/SC 29/WG 11, also known as MPEG (Moving Picture Experts Group). VCEG was formed in 1997 to maintain prior ITU-T video coding standards and develop new video coding standard(s) appropriate for a wide range of conversational and non-conversational services. MPEG was formed in 1988 to establish standards for coding of moving pictures and associated audio for various applications such as digital storage media, distribution, and communication.

In this Recommendation | International Standard Annexes A through E, G and H contain normative requirements and are an integral part of this Recommendation | International Standard.

Recommendation ITU-T H.264

Advanced video coding for generic audiovisual services

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 Publication and versions of this Specification

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

This Specification has been jointly developed by ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group. It is published as technically-aligned twin text in both organizations ITU-T and ISO/IEC.

Rec. ITU-T H.264 | ISO/IEC 14496-10 version 1 refers to the first approved version of this Recommendation | International Standard. Version 1 was approved by ITU-T on 30 May 2003. The first published version in ISO/IEC corresponded to version 1.

Rec. ITU-T H.264 | ISO/IEC 14496-10 version 2 refers to the integrated text containing the corrections specified in the first technical corrigendum. The first fully-published version in the ITU-T was version 2 as approved by ITU-T on 7 May 2004, due to the development of the corrigendum during the publication process. Version 2 was also published in integrated form by ISO/IEC.

Rec. ITU-T H.264 | ISO/IEC 14496-10 version 3 refers to the integrated text containing both the first technical corrigendum (2004) and the first amendment, which is referred to as the "Fidelity range extensions". Version 3 was approved by ITU-T on 1 March 2005.

Rec. ITU-T H.264 | ISO/IEC 14496-10 version 4 refers to the integrated text containing the first technical corrigendum (2004), the first amendment (the "Fidelity range extensions"), and an additional technical corrigendum (2005). Version 4 was approved by ITU-T on 13 September 2005. In both ITU-T and ISO/IEC, the next complete published version after version 2 was version 4.

Rec. ITU-T H.264 | ISO/IEC 14496-10 version 5 refers to the integrated version 4 text with its specification of the High 4:4:4 profile removed.

Rec. ITU-T H.264 | ISO/IEC 14496-10 version 6 refers to the integrated version 5 text after its amendment to support additional colour space indicators. In the ITU-T, the changes for versions 5 and 6 were approved on 13 June 2006 and were published as a single amendment.

Rec. ITU-T H.264 | ISO/IEC 14496-10 version 7 refers to the integrated version 6 text after its amendment to define five new profiles intended primarily for professional applications (the High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, CAVLC 4:4:4 Intra, and High 4:4:4 Predictive profiles) and two new types of supplemental enhancement information (SEI) messages (the post-filter hint SEI message and the tone mapping information SEI message). Version 7 was approved by ITU-T on 6 April 2007.

Rec. ITU-T H.264 | ISO/IEC 14496-10 version 8 refers to the integrated version 7 text after its amendment to specify scalable video coding in three profiles (Scalable Baseline, Scalable High, and Scalable High Intra profiles). Version 8 was approved by ITU-T on 22 November 2007.

Rec. ITU-T H.264 | ISO/IEC 14496-10 version 9 refers to the integrated version 8 text after applying the corrections specified in a third technical corrigendum. Version 9 was approved by ITU-T on 13 January 2009.

Rec. ITU-T H.264 | ISO/IEC 14496-10 version 10 refers to the integrated version 9 text after its amendment to specify a profile for multiview video coding (the Multiview High profile) and to define additional SEI messages.

Rec. ITU-T H.264 | ISO/IEC 14496-10 version 11 refers to the integrated version 10 text after its amendment to define a new profile (the Constrained Baseline profile) intended primarily to enable implementation of decoders supporting only the common subset of capabilities supported in various previously-specified profiles. In the ITU-T, the changes for versions 10 and 11 were approved on 16 March 2009.

Rec. ITU-T H.264 | ISO/IEC 14496-10 version 12 refers to the integrated version 11 text after its amendment to define a new profile (the Stereo High profile) for two-view video coding with support of interlaced coding tools and to specify an additional SEI message specified as the frame packing arrangement SEI message. The changes for versions 11 and 12 were processed as a single amendment in the ISO/IEC approval process.

Rec. ITU-T H.264 | ISO/IEC 14496-10 version 13 refers to the integrated version 12 text with various minor corrections and clarifications as specified in a fourth technical corrigendum. In the ITU-T, the changes for versions 12 and 13 were approved on 9 March 2010.

Rec. ITU-T H.264 | ISO/IEC 14496-10 version 14 refers to the integrated version 13 text after its amendment to define a new level (Level 5.2) supporting higher processing rates in terms of maximum macroblocks per second and a new profile (the Progressive High profile) to enable implementation of decoders supporting only the frame coding tools of the previously-specified High profile.

Rec. ITU-T H.264 | ISO/IEC 14496-10 version 15 (the current Specification) refers to the integrated version 14 text with miscellaneous corrections and clarifications as specified in a fifth technical corrigendum. In the ITU-T, the changes for versions 14 and 15 were approved on 29 June 2011.

0.5 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.

0.6 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. With the exception of the transform bypass mode of operation for lossless coding in the High 4:4:4 Intra, CAVLC 4:4:4 Intra, and High 4:4:4 Predictive profiles, and the I_PCM mode of operation in all profiles, the algorithm is typically 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 coding or arithmetic coding.

Scalable video coding is specified in Annex G allowing the construction of bitstreams that contain sub-bitstreams that conform to this Specification. For temporal bitstream scalability, i.e., the presence of a sub-bitstream with a smaller temporal sampling rate than the bitstream, complete access units are removed from the bitstream when deriving the sub-bitstream. In this case, high-level syntax and inter prediction reference pictures in the bitstream are constructed accordingly. For spatial and quality bitstream scalability, i.e., the presence of a sub-bitstream with lower spatial resolution or quality than the bitstream, NAL units are removed from the bitstream when deriving the sub-bitstream. In this case, inter-layer prediction, i.e., the prediction of the higher spatial resolution or quality signal by data of the lower spatial resolution or quality signal, is typically used for efficient coding. Otherwise, the coding algorithm as described in the previous paragraph is used.

Multiview video coding is specified in Annex H allowing the construction of bitstreams that represent multiple views. Similar to scalable video coding, bitstreams that represent multiple views may also contain sub-bitstreams that conform to this Specification. For temporal bitstream scalability, i.e., the presence of a sub-bitstream with a smaller temporal sampling rate than the bitstream, complete access units are removed from the bitstream when deriving the sub-bitstream. In this case, high-level syntax and inter prediction reference pictures in the bitstream are constructed accordingly. For view bitstream scalability, i.e. the presence of a sub-bitstream with fewer views than the bitstream, NAL units are removed from the bitstream when deriving the sub-bitstream. In this case, inter-view prediction, i.e., the prediction of one view signal by data of another view signal, is typically used for efficient coding. Otherwise, the coding algorithm as described in the previous paragraph is used.

0.6.1 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.

0.6.2 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.

0.6.3 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.

0.6.4 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.

0.7 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, G, and H also form an integral part of this Recommendation | International Standard.

Annex A specifies thirteen profiles (Baseline, Constrained Baseline, Main, Extended, High, Progressive High, High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, and CAVLC 4:4:4 Intra), 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. Annex E specifies syntax and semantics of the video usability information parameters of the sequence parameter set.

Annex G specifies scalable video coding (SVC). The reader is referred to Annex G for the entire decoding process for SVC, which is specified there with references being made to clauses 2-9 and Annexes A-E. Subclause G.10 specifies three profiles for SVC (Scalable Baseline, Scalable High, and Scalable High Intra).

Annex H specifies multiview video coding (MVC). The reader is referred to Annex H for the entire decoding process for MVC, which is specified there with references being made to clauses 2-9 and Annexes A-E. Subclause H.10 specifies two profiles for MVC (Multiview High and Stereo High).

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

1 Scope

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

2 Normative references

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

– Recommendation ITU-T 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 11664-1:2007, *Colorimetry — Part 1: 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* that are consecutive in *decoding order* and contain exactly one *primary coded picture*. In addition to the *primary coded picture*, an access unit may also contain one or more *redundant coded pictures*, one *auxiliary coded picture*, 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 **alpha blending**: A process not specified by this Recommendation | International Standard, in which an *auxiliary coded picture* is used in combination with a *primary coded picture* and with other data not specified by this Recommendation | International Standard in the *display process.* In an alpha blending process, the samples of an *auxiliary coded picture* are interpreted as indications of the degree of opacity (or, equivalently, the degrees of transparency) associated with the corresponding *luma* samples of the *primary coded picture.*

3.6 **arbitrary slice order (ASO)**: A *decoding order* of *slices* in which the *macroblock address* of the first *macroblock* of some *slice* of a *slice group* may be less than the *macroblock address* of the first *macroblock* of some other preceding *slice* of the same *slice group* or, in the case of a *picture* that is coded using three separate colour planes, some other preceding *slice* of the same *slice group* within the same colour plane, or in which the *slices* of a *slice group* of a picture may be interleaved with the *slices* of one or more other *slice groups* of the *picture* or, in the case of a *picture* that is coded using three separate colour planes, with the *slices* of one or more other *slice groups* within the same colour plane.

3.7 **auxiliary coded picture**: A *picture* that supplements the *primary coded picture* that may be used in combination with other data not specified by this Recommendation | International Standard in the *display process.* An auxiliary coded picture has the same syntactic and semantic restrictions as a monochrome *redundant coded picture.* An auxiliary coded picture must contain the same number of *macroblocks* as the *primary coded picture*. Auxiliary coded pictures have no normative effect on the *decoding process*. See also *primary coded picture* and *redundant coded picture.*

3.8 **B slice**: A *slice* that may be decoded using *intra prediction* or *inter prediction* using at most two *motion vectors* and *reference indices* to *predict* the sample values of each *block*.

3.9 **bin**: One bit of a *bin string*.

3.10 **binarization**: A set of *bin strings* for all possible values of a *syntax element*.

3.11 **binarization process**: A unique mapping process of all possible values of a *syntax element* onto a set of *bin strings*.

3.12 **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*.

3.13 **bi-predictive slice**: See *B slice***.**

3.14 **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*.

3.15 **block**: An MxN (M-column by N-row) array of samples, or an MxN array of *transform coefficients*.

3.16 **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*.

3.17 **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.*

3.18 **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*.

3.19 **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.

3.20 **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.

3.21 **byte stream**: An encapsulation of a *NAL unit stream* containing *start code prefixes* and *NAL units* as specified in Annex B.

3.22 **can**: A term used to refer to behaviour that is allowed, but not necessarily required.

3.23 **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.

3.24 **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.

3.25 **coded field**: A *coded representation* of a *field*.

3.26 **coded frame**: A *coded representation* of a *frame*.

3.27 **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.

3.28 **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.

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

3.30 **coded slice data partition NAL unit**: A *NAL unit* containing a *slice data partition*.

3.31 **coded slice NAL unit**: A *NAL unit* containing a *slice* that is not a *slice* of an *auxiliary coded picture*.

3.32 **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*.

3.33 **component**: An array or single sample from one of the three arrays (*luma* and two *chroma*) that make up a *field* or *frame* in 4:2:0, 4:2:2, or 4:4:4 colour format or the array or a single sample of the array that make up a *field* or *frame* in monochrome format.

3.34 **complementary field pair**: A collective term for a *complementary reference field pair* or a *complementary non-reference field pair*.

3.35 **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 and share the same value of the frame_num *syntax element*, where the first *field* is not already a paired *field.*

3.36 **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.

3.37 **context variable**: A variable specified for the *adaptive binary arithmetic decoding process* of a *bin* by an equation containing recently decoded *bins*.

3.38 **DC transform coefficient**: A *transform coefficient* for which the *frequency index* is zero in all dimensions.

3.39 **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*.

3.40 **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.

3.41 **decoder**: An embodiment of a *decoding process*.

3.42 **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*.

3.43 **decoding order**: The order in which *syntax elements* are processed by the *decoding process*.

3.44 **decoding process**: The process specified in this Recommendation | International Standard that reads a *bitstream* and derives *decoded pictures* from it.

3.45 **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.

3.46 **display process**: A process not specified in this Recommendation | International Standard having, as its input, the cropped decoded *pictures* that are the output of the *decoding process.*

3.47 **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*.

3.48 **encoder**: An embodiment of an *encoding process*.

3.49 **encoding process**: A process, not specified in this Recommendation | International Standard, that produces a *bitstream* conforming to this Recommendation | International Standard.

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

3.51 **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.

3.52 **field macroblock pair**: A *macroblock pair* decoded as two *field macroblocks.*

3.53 **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.*

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

3.55 **frame**: A *frame* contains an array of *luma* samples in monochrome format or an array of *luma* samples and two corresponding arrays of *chroma* samples in 4:2:0, 4:2:2, and 4:4:4 colour format. A *frame* consists of two *fields*, a *top field* and a *bottom field*.

3.56 **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.

3.57 **frame macroblock pair**: A *macroblock pair* decoded as two *frame macroblocks.*

3.58 **frequency index**: A one-dimensional or two-dimensional index associated with a *transform coefficient* prior to an *inverse transform* part of the *decoding process.*

3.59 **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.

3.60 **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*.

3.61 **I slice**: A *slice* that is not an *SI slice* that is decoded using *intra prediction* only.

3.62 **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.

3.63 **instantaneous decoding refresh (IDR) access unit**: An *access unit* in which the *primary coded picture* is an *IDR picture*.

3.64 **instantaneous decoding refresh (IDR) picture**: A *coded picture* for which the variable IdrPicFlag is equal to 1. An IDR picture causes the *decoding process* to mark all *reference pictures* as "unused for reference" immediately after the decoding of the IDR picture. All *coded pictures* that follow an IDR picture in *decoding order* can be decoded without *inter prediction* from any *picture* that precedes the IDR picture in *decoding order*. The first *picture* of each *coded video sequence* in *decoding order* is an IDR picture.

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

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

3.67 **interpretation sample value**: A possibly-altered value corresponding to a decoded sample value of an *auxiliary coded picture* that may be generated for use in the *display process*. Interpretation sample values are not used in the *decoding process* and have no normative effect on the *decoding process*.

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

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

3.70 **intra slice**: See *I slice*.

3.71 **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*.

3.72 **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.

3.73 **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, a level is the value of a *transform coefficient* prior to *scaling* (see the definition of *transform coefficient level*).

3.74 **list:** A one-dimensional array of *syntax elements* or variables.

3.75 **list 0 (list 1) motion vector**: A *motion vector* associated with a *reference index* pointing into *reference picture list 0* (*list 1*).

3.76 **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*).

3.77 **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.

3.78 **macroblock**: A 16x16 *block* of *luma* samples and two corresponding *blocks* of *chroma* samples of a *picture* that has three sample arrays, or a 16x16 *block* of samples of a monochrome *picture* or a *picture* that is coded using three separate colour planes. The division of a *slice* or a *macroblock pair* into macroblocks is a *partitioning*.

3.79 **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.*

3.80 **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.

3.81 **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*.

3.82 **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*.

3.83 **macroblock partition**: A *block* of *luma* samples and two corresponding *blocks* of *chroma* samples resulting from a *partitioning* of a *macroblock* for *inter prediction* for a *picture* that has three sample arrays or a *block* of *luma* samples resulting from a *partitioning* of a *macroblock* for *inter prediction* for a monochrome *picture* or a *picture* that is coded using three separate colour planes.

3.84 **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.

3.85 **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.

3.86 **matrix**: A two-dimensional array of *syntax elements* or variables.

3.87 **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.

3.88 **memory management control operation**: Seven operations that control *reference picture marking*.

3.89 **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*.

3.90 **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.

3.91 **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*.

3.92 **NAL unit stream**: A sequence of *NAL units*.

3.93 **non-paired field**: A collective term for a *non-paired reference field* or a *non-paired non-reference field*.

3.94 **non-paired non-reference field**: A decoded *non-reference field* that is not part of a *complementary non-reference field pair.*

3.95 **non-paired reference field**: A decoded *reference field* that is not part of a *complementary reference field pair.*

3.96 **non-reference field**: A *field* coded with nal_ref_idc equal to 0.

3.97 **non-reference frame**: A *frame* coded with nal_ref_idc equal to 0.

3.98 **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*.

3.99 **note**: A term used to prefix *informative* remarks. This term is used exclusively in an *informative* context.

3.100 **opposite parity**: The *opposite parity* of *top* is *bottom*, and vice versa.

3.101 **output order**: The order in which the *decoded pictures* are output from the *decoded picture buffer*.

3.102 **P slice**: A *slice* that is not an *SP slice* that may be decoded using *intra prediction* or *inter prediction* using at most one *motion vector* and *reference index* to *predict* the sample values of each *block*.

3.103 **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*.

3.104 **parity**: The parity of a *field* can be *top* or *bottom*.

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

3.106 **picture**: A collective term for a *field* or a *frame*.

3.107 **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.*

3.108 **picture order count**: A variable that is associated with each *coded field* and each *field* of a *coded frame* and has a value that is non-decreasing with increasing *field* position in *output order* relative to the first output *field* of the previous *IDR picture* in *decoding order* or relative to the first output *field* of the previous *picture*, in *decoding order*, that contains a *memory management control operation* that marks all *reference pictures* as "unused for reference".

3.109 **prediction**: An embodiment of the *prediction process*.

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

3.111 **predictive slice**: See *P slice*.

3.112 **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.

3.113 **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.*

3.114 **profile**: A specified subset of the syntax of this Recommendation | International Standard.

3.115 **quantisation parameter**: A variable used by the *decoding process* for *scaling* of *transform coefficient levels*.

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

3.117 **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.

3.118 **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.

3.119 **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.*

3.120 **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*.

3.121 **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*.

3.122 **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*.

3.123 **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*.

3.124 **reference index**: An index into a *reference picture list*.

3.125 **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*.

3.126 **reference picture list**: A list of *reference pictures* that is used for *inter prediction* of a *P, B,* or *SP slice.* For the *decoding process* of a *P* or *SP slice,* there is one reference picture list. For the *decoding process* of a *B slice*, there are two reference picture lists.

3.127 **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*.

3.128 **reference picture list 1**: A *reference picture list* used for *inter prediction* of a *B slice*. Reference picture list 1 is one of two *reference picture lists* used for *inter prediction* for a *B slice*, with the other being *reference picture list 0*.

3.129 **reference picture marking**: Specifies, in the bitstream, how the *decoded pictures* are marked for *inter prediction*.

3.130 **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.

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

3.132 **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*.

3.133 **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).

3.134 **scaling**: The process of multiplying *transform coefficient levels* by a factor, resulting in *transform coefficients*.

3.135 **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*.

3.136 **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.

3.137 **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.

3.138 **SI slice**: A *slice* that is coded using *intra prediction* only 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*.

3.139 **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*.

3.140 **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 *macroblock addresses* are derived from the first *macroblock address* in a slice (as represented in the *slice header*) and the *macroblock to slice group map*, and, when a *picture* is coded using three separate colour planes, a colour plane identifier.

3.141 **slice data partition**: A non-empty subset of the *syntax elements* of the slice data *syntax structure* for a *slice*. The *syntax elements* of a slice data partition are associated with the same *category*.

3.142 **slice data partitioning**: A method of *partitioning* selected *syntax elements* into *syntax structures* based on a *category* associated with each *syntax element*.

3.143 **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*.

3.144 **slice group map units**: The units of the *map unit to slice group map.*

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

3.146 **source**: Term used to describe the video material or some of its attributes before encoding.

3.147 **SP slice**: A *slice* that may be coded using *intra prediction* or *inter prediction* with quantisation of the *prediction* samples 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*.

3.148 **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*.

3.149 **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.

3.150 **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* for a *picture* that has three sample arrays or an 8x8 *luma block* of which one corner is located at a corner of the *macroblock* for a monochrome *picture* or a *picture* that is coded using three separate colour planes.

3.151 **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* for a *picture* that has three sample arrays or a *block* of *luma* samples resulting from a *partitioning* of a *sub-macroblock* for *inter prediction* for a monochrome *picture* or a *picture* that is coded using three separate colour planes.

3.152 **switching I slice**: See *SI slice*.

3.153 **switching P slice**: See *SP slice*.

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

3.155 **syntax structure**: Zero or more *syntax elements* present together in the *bitstream* in a specified order.

3.156 **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*.

3.157 **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.*

3.158 **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*.

3.159 **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.

3.160 **universal unique identifier (UUID)**: An identifier that is unique with respect to the space of all universal unique identifiers.

3.161 **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.

3.162 **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.

3.163 **VCL NAL unit**: A collective term for *coded slice NAL units* and *coded slice data partition NAL units*.

3.164 **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

For the purposes of this Recommendation | International Standard, the following abbreviations apply:

CABAC Context-based Adaptive Binary Arithmetic Coding

CAVLC Context-based Adaptive Variable Length Coding

CBR Constant Bit Rate

CPB Coded Picture Buffer

DPB Decoded Picture Buffer

DUT Decoder under test

FIFO First-In, First-Out

HRD	Hypothetical Reference Decoder
HSS	Hypothetical Stream Scheduler
IDR	Instantaneous Decoding Refresh
LSB	Least Significant Bit
MB	Macroblock
MBAFF	Macroblock-Adaptive Frame-Field Coding
MSB	Most Significant Bit
MVC	Multiview Video Coding
NAL	Network Abstraction Layer
RBSP	Raw Byte Sequence Payload
SEI	Supplemental Enhancement Information
SODB	String Of Data Bits
SVC	Scalable Video Coding
UUID	Universal Unique Identifier
VBR	Variable Bit Rate
VCL	Video Coding Layer
VLC	Variable Length Coding
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, including matrix 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.
÷	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 >= 0 and y > 0.

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.

When a relational operator is applied to a syntax element or variable that has been assigned the value "na" (not applicable), the value "na" is treated as a distinct value for the syntax element or variable. The value "na" is considered not to be equal to any other value.

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.
^	Bit-wise "exclusive 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 >> 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 have a value equal to the MSB of x prior to the shift operation.
x << 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:

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

Ceil(x) the smallest integer greater than or equal to x. (5-2)

$$\text{Clip1}_Y(\ x\ ) = \text{Clip3}(\ 0,\ (\ 1 << \text{BitDepth}_Y\ ) - 1,\ x\ ) \tag{5-3}$$

$$\text{Clip1}_C(\ x\ ) = \text{Clip3}(\ 0,\ (\ 1 << \text{BitDepth}_C\ ) - 1,\ x\ ) \tag{5-4}$$

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

Floor(x) the greatest integer less than or equal to x. (5-6)

$$\text{InverseRasterScan}(\ a, b, c, d, e\ ) = \begin{cases} (a\%(d/b))*b & ;\quad e == 0 \\ (a/(d/b))*c & ;\quad e == 1 \end{cases} \tag{5-7}$$

Log2(x) returns the base-2 logarithm of x. (5-8)

Log10(x) returns the base-10 logarithm of x. (5-9)

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

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

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

$$\text{Round}(\ x\ ) = \text{Sign}(\ x\ ) * \text{Floor}(\ \text{Abs}(\ x\ ) + 0.5\ ) \tag{5-13}$$

$$\text{Sign}(\ x\ ) = \begin{cases} 1 & ;\quad x >= 0 \\ -1 & ;\quad x < 0 \end{cases} \tag{5-14}$$

$$\text{Sqrt}(\ x\ ) = \sqrt{x} \tag{5-15}$$

5.8 Order of operation precedence

When order of precedence in an expression is not indicated explicitly by use of parentheses, the following rules apply:

– operations of a higher precedence are evaluated before any operation of a lower precedence,

– operations of the same precedence are evaluated sequentially from left to right.

Table 5-1 specifies the precedence of operations from highest to lowest; a higher position in the table indicates a higher precedence.

NOTE – For those operators that are also used in the C programming language, the order of precedence used in this Specification is the same as used in the C programming language.

Table 5-1 – Operation precedence from highest (at top of table) to lowest (at bottom of table)

operations (with operands x, y, and z)
"x++", "x− −"
"!x", "−x" (as a unary prefix operator)
x^y
"x * y", "x / y", "x ÷ y", " $\frac{x}{y}$ ", "x % y"
"x + y", "x − y" (as a two-argument operator), " $\sum_{i=x}^{y} f(i)$ "
"x << y", "x >> y"
"x < y", "x <= y", "x > y", "x >= y"
"x = = y", "x != y"
"x & y"
"x | y"
"x && y"
"x | | y"
"x ? y : z"
"x = y", "x += y", "x −= y"

5.9 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 without 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 that specify properties of the current position in the bitstream are referred to as syntax functions. These functions are specified in subclause 7.2 and 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. Syntax functions are described by their names, which are constructed as syntax element names and end 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).

Functions that are not syntax functions (including mathematical functions specified in subclause 5.7) are described by their names, which start with an upper case letter, contain a mixture of lower and upper case letters without any underscore character, and end with left and right parentheses including zero or more variable names (for definition) or values (for usage) separated by commas (if more than one variable).

Subscripts or square parentheses are used for the indexing of arrays. In reference to a visual depiction of a matrix, the first subscript is used as a row (vertical) index and the second subscript is used as a column (horizontal) index. The indexing order is reversed when using square parentheses rather than subscripts for indexing. Thus, an element of a matrix s at horizontal position x and vertical position y may be denoted either as s[x, y] or 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 (counted from the most to the least significant bit) 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 (counted from the most to the least significant bit) 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 value different from zero.

5.10 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

5.11 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.

When invoking a process, the assignment of variables is specified as follows:

– If the variables at the invoking and the process specification do not have the same name, the variables are explicitly assigned to lower case input or output variables of the process specification.

– Otherwise (the variables at the invoking and the process 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 one or more sample arrays:

– Luma (Y) only (monochrome), with or without an auxiliary array.

– Luma and two Chroma (YCbCr or YCgCo), with or without an auxiliary array.

– Green, Blue and Red (GBR, also known as RGB), with or without an auxiliary array.

– Arrays representing other unspecified monochrome or tri-stimulus colour samplings (for example, YZX, also known as XYZ), with or without an auxiliary array.

For convenience of notation and terminology in this Specification, the variables and terms associated with these arrays are referred to as luma (or L or Y) and chroma, where the two chroma arrays are referred to as Cb and Cr; regardless of the actual colour representation method in use. The actual colour representation method in use can be indicated in syntax that is specified in Annex E. The (monochrome) auxiliary arrays, which may or may not be present as auxiliary pictures in a coded video sequence, are optional for decoding and can be used for such purposes as alpha blending.

The variables SubWidthC, and SubHeightC are specified in Table 6-1, depending on the chroma format sampling structure, which is specified through chroma_format_idc and separate_colour_plane_flag. An entry marked as "-" in Table 6-1 denotes an undefined value for SubWidthC or SubHeightC. Other values of chroma_format_idc, SubWidthC, and SubHeightC may be specified in the future by ITU-T | ISO/IEC.

Table 6-1 – SubWidthC, and SubHeightC values derived from chroma_format_idc and separate_colour_plane_flag

chroma_format_idc	separate_colour_plane_flag	Chroma Format	SubWidthC	SubHeightC
0	0	monochrome	-	-
1	0	4:2:0	2	2
2	0	4:2:2	2	1
3	0	4:4:4	1	1
3	1	4:4:4	-	-

In monochrome sampling there is only one sample array, which is nominally considered the 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, depending on the value of separate_colour_plane_flag, the following applies:

– If separate_colour_plane_flag is equal to 0, each of the two chroma arrays has the same height and width as the luma array.
– Otherwise (separate_colour_plane_flag is equal to 1), the three colour planes are separately processed as monochrome sampled pictures.

The width and height of the luma sample arrays are each an integer multiple of 16. In coded video sequences using 4:2:0 chroma sampling, the width and height of chroma sample arrays are each an integer multiple of 8. In coded video sequences using 4:2:2 sampling, the width of the chroma sample arrays is an integer multiple of 8 and the height is an integer multiple of 16. The height of a luma array that is coded as two separate fields or in macroblock-adaptive frame-field coding (see below) is an integer multiple of 32. In coded video sequences using 4:2:0 chroma sampling, the height of each chroma array that is coded as two separate fields or in macroblock-adaptive frame-field coding (see below) 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 syntax for the luma and (when present) chroma arrays are ordered such when data for all three colour components is present, the data for the luma array is first, followed by any data for the Cb array, followed by any data for the Cr array, unless otherwise specified.

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 number of bits necessary for the representation of each of the samples in the luma and chroma arrays in a video sequence is in the range of 8 to 14, and the number of bits used in the luma array may differ from the number of bits used in the chroma arrays.

When the value of chroma_format_idc is equal to 1, 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).

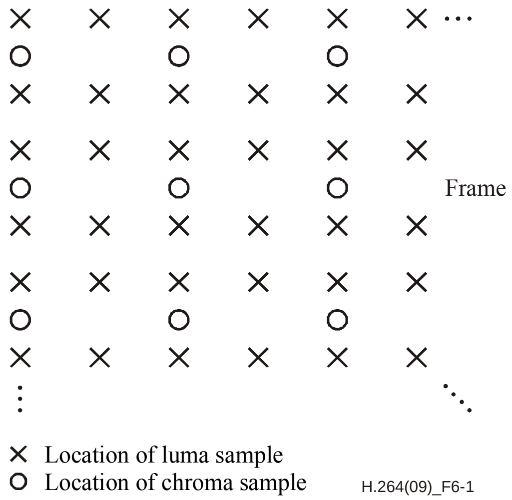


Figure 6-1 – Nominal vertical and horizontal locations of 4:2:0 luma and chroma samples in a frame

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. A top field consists of only the top field rows of a decoded frame. When the top field or bottom field of a decoded frame is used as a reference field (see below) only the even rows (for a top field) or the odd rows (for a bottom field) of the decoded frame are used.

When the value of chroma_format_idc is equal to 1, 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.

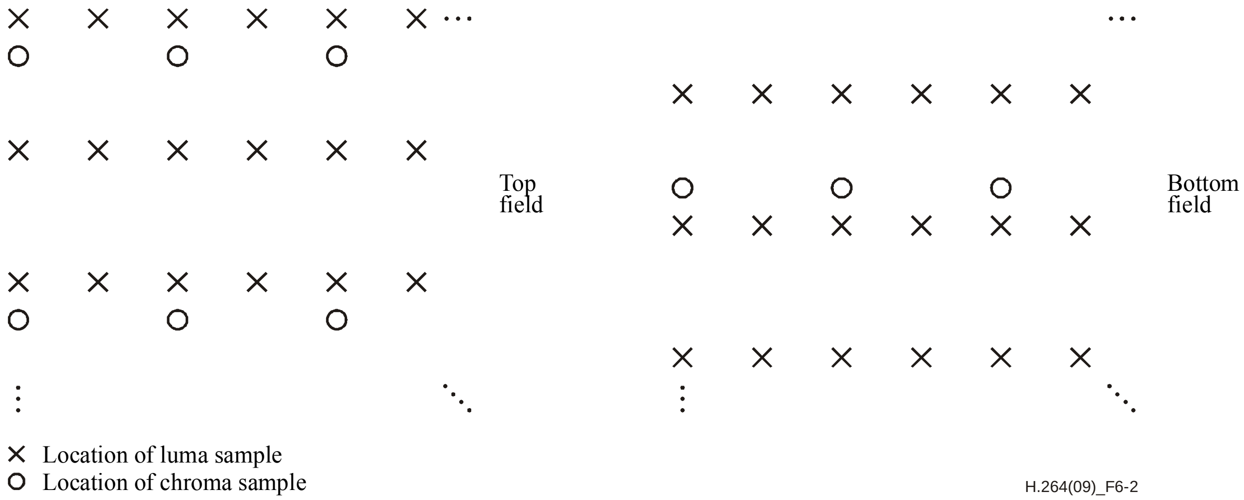


Figure 6-2 – Nominal vertical and horizontal sampling locations of 4:2:0 samples in top and bottom fields

When the value of chroma_format_idc is equal to 2, the chroma samples are co-sited with the corresponding luma samples and the nominal locations in a frame and in fields are as shown in Figures 6-3 and 6-4, respectively.

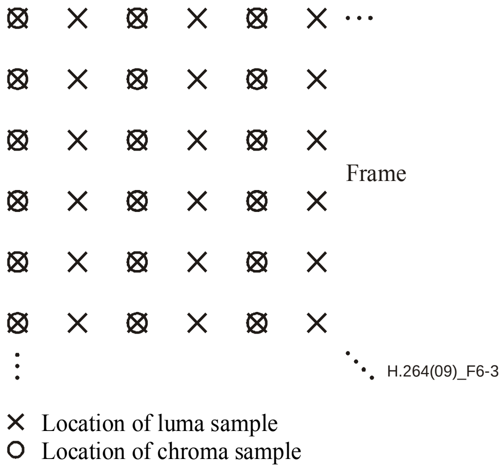


Figure 6-3 – Nominal vertical and horizontal locations of 4:2:2 luma and chroma samples in a frame

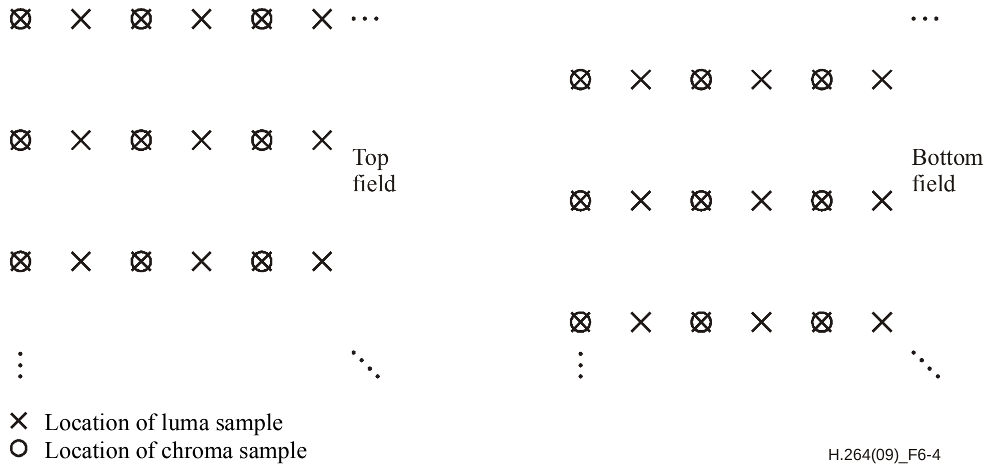


Figure 6-4 – Nominal vertical and horizontal sampling locations of 4:2:2 samples top and bottom fields

When the value of chroma_format_idc is equal to 3, all array samples are co-sited for all cases of frames and fields and the nominal locations in a frame and in fields are as shown in Figures 6-5 and 6-6, respectively.

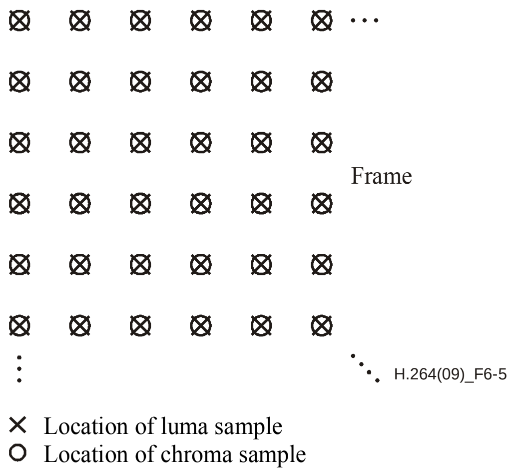


Figure 6-5 – Nominal vertical and horizontal locations of 4:4:4 luma and chroma samples in a frame

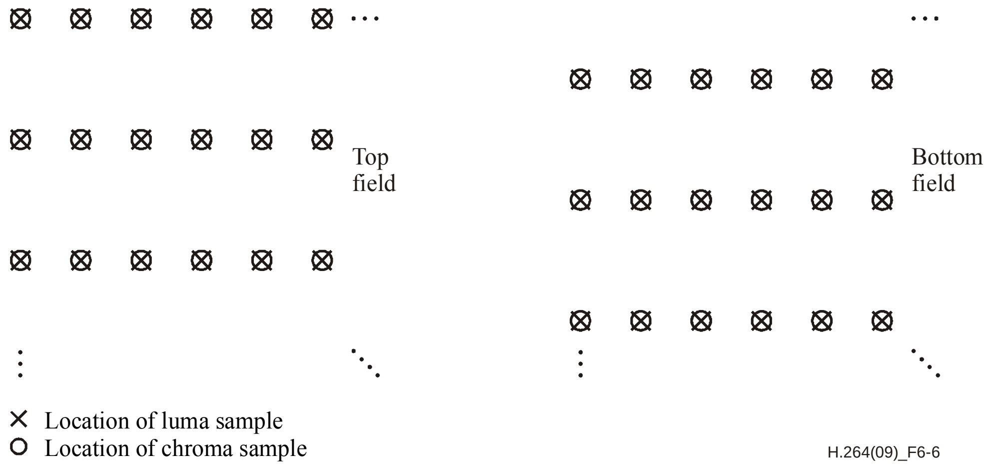


Figure 6-6 – Nominal vertical and horizontal sampling locations of 4:4:4 samples top and bottom fields

The samples are processed in units of macroblocks. The luma array for each macroblock is 16 samples in both width and height. The variables MbWidthC and MbHeightC, which specify the width and height, respectively, of the chroma arrays for each macroblock, are derived as follows:

– If chroma_format_idc is equal to 0 (monochrome) or separate_colour_plane_flag is equal to 1, MbWidthC and MbHeightC are both equal to 0.

– Otherwise, MbWidthC and MbHeightC are derived as

$$\text{MbWidthC} = 16 \,/\, \text{SubWidthC} \quad (6\text{-}1)$$
$$\text{MbHeightC} = 16 \,/\, \text{SubHeightC} \quad (6\text{-}2)$$

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 array and, when the chroma sampling format is not equal to 4:0:0 and separate_colour_plane_flag is equal to 0, two corresponding chroma sample arrays. When separate_colour_plane_flag is equal to 1, each macroblock is comprised of one 16x16 luma or chroma sample array. 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 6-7.

When a picture is coded using three separate colour planes (separate_colour_plane_flag is equal to 1), a slice contains only macroblocks of one colour component being identified by the corresponding value of colour_plane_id, and each colour component array of a picture consists of slices having the same colour_plane_id value. Coded slices with different values of colour_plane_id within an access unit can be interleaved with each other under the constraint that for each value of colour_plane_id, the coded slice NAL units with that value colour_plane_id shall be in the order of increasing macroblock address for the first macroblock of each coded slice NAL unit.

NOTE – When separate_colour_plane_flag is equal to 0, each macroblock of a picture is contained in exactly one slice. When separate_colour_plane_flag is equal to 1, each macroblock of a colour component is contained in exactly one slice (i.e., information for each macroblock of a picture is present in exactly three slices and these three slices have different values of colour_plane_id).

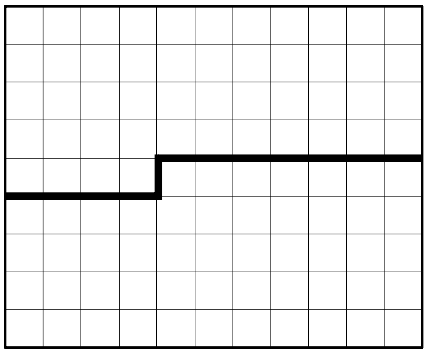


Figure 6-7 – A picture with 11 by 9 macroblocks that is partitioned into two slices

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 6-8. Each macroblock pair consists of two macroblocks.

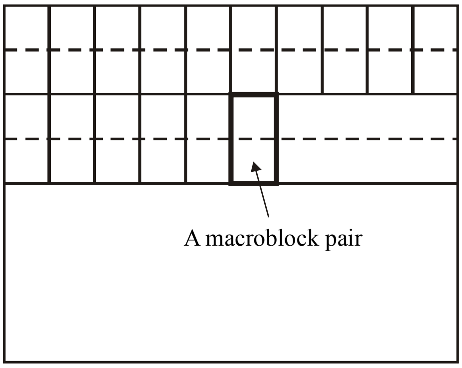


Figure 6-8 – Partitioning of the decoded frame into macroblock pairs

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,

$$x = \text{InverseRasterScan}(\ \text{mbAddr}, 16, 16, \text{PicWidthInSamples}_L, 0\ ) \qquad (6\text{-}3)$$

$$y = \text{InverseRasterScan}(\ \text{mbAddr}, 16, 16, \text{PicWidthInSamples}_L, 1\ ) \qquad (6\text{-}4)$$

– Otherwise (MbaffFrameFlag is equal to 1), the following ordered steps are specified:

1. The luma location (xO, yO) is derived by

$$\text{xO} = \text{InverseRasterScan}(\ \text{mbAddr} / 2, 16, 32, \text{PicWidthInSamples}_L, 0\ ) \quad (6\text{-}5)$$

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

2. Depending on the current macroblock the following applies:

   – If the current macroblock is a frame macroblock

$$\text{x} = \text{xO} \quad (6\text{-}7)$$

$$\text{y} = \text{yO} + (\ \text{mbAddr}\ \%\ 2\ ) * 16 \quad (6\text{-}8)$$

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

$$\text{x} = \text{xO} \quad (6\text{-}9)$$

$$\text{y} = \text{yO} + (\ \text{mbAddr}\ \%\ 2\ ) \quad (6\text{-}10)$$

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 6-9. 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 Tables 7-13, 7-14, 7-17, and 7-18. 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.

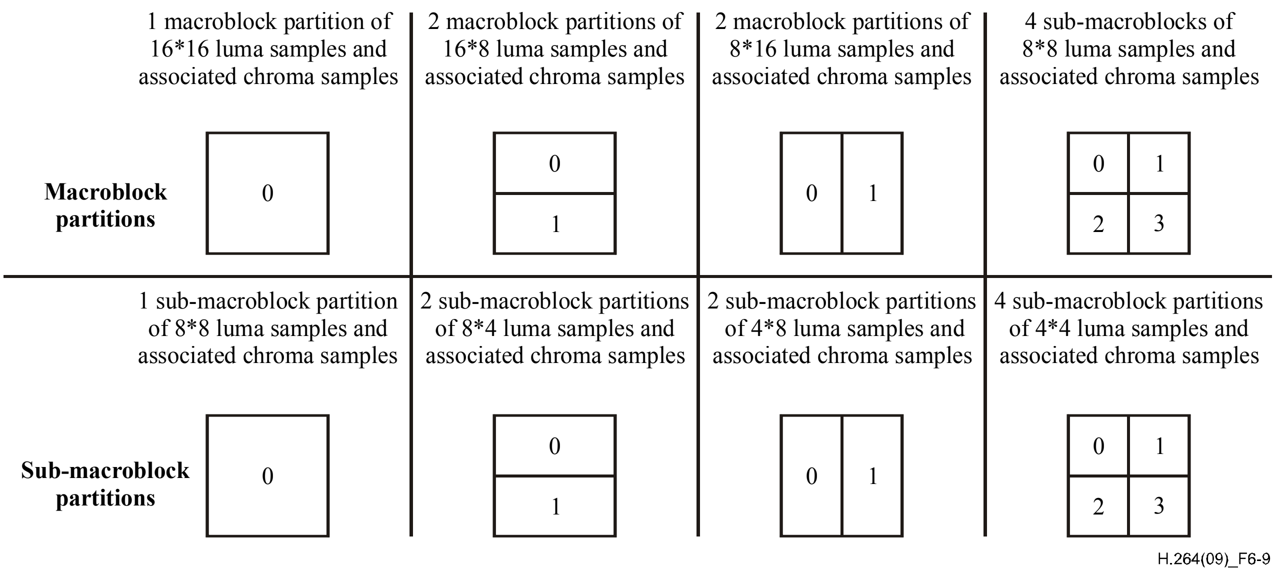


Figure 6-9 – Macroblock partitions, sub-macroblock partitions, macroblock partition scans, and sub-macroblock partition scans

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 = InverseRasterScan(mbPartIdx, MbPartWidth(mb_type), MbPartHeight(mb_type), 16, 0) (6-11)

y = InverseRasterScan(mbPartIdx, MbPartWidth(mb_type), MbPartHeight(mb_type), 16, 1) (6-12)

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 = InverseRasterScan(subMbPartIdx, SubMbPartWidth(sub_mb_type[mbPartIdx]),
SubMbPartHeight(sub_mb_type[mbPartIdx]), 8, 0) (6-13)

y = InverseRasterScan(subMbPartIdx, SubMbPartWidth(sub_mb_type[mbPartIdx]),
SubMbPartHeight(sub_mb_type[mbPartIdx]), 8, 1) (6-14)

– Otherwise (mb_type is not equal to P_8x8, P_8x8ref0, or B_8x8),

x = InverseRasterScan(subMbPartIdx, 4, 4, 8, 0) (6-15)

y = InverseRasterScan(subMbPartIdx, 4, 4, 8, 1) (6-16)

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 6-10 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 6-10 – Scan for 4x4 luma blocks

The inverse 4x4 luma block scanning process is specified by

x = InverseRasterScan(luma4x4BlkIdx / 4, 8, 8, 16, 0) +
InverseRasterScan(luma4x4BlkIdx % 4, 4, 4, 8, 0) (6-17)

y = InverseRasterScan(luma4x4BlkIdx / 4, 8, 8, 16, 1) +
InverseRasterScan(luma4x4BlkIdx % 4, 4, 4, 8, 1) (6-18)

6.4.4 Inverse 4x4 Cb or Cr block scanning process for ChromaArrayType equal to 3

This process is only invoked when ChromaArrayType is equal to 3.

The inverse 4x4 chroma block scanning process is identical to inverse 4x4 luma block scanning process as specified in subclause 6.4.3 when substituting the term "luma" with the term "Cb" or the term "Cr", and substituting the term "luma4x4BlkIdx" with the term "cb4x4BlkIdx" or the term "cr4x4BlkIdx" in all places in subclause 6.4.3.

6.4.5 Inverse 8x8 luma block scanning process

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

Output of this process is the location (x, y) of the upper-left luma sample for the 8x8 luma block with index luma8x8BlkIdx relative to the upper-left luma sample of the macroblock.

Figure 6-11 shows the scan for the 8x8 luma blocks.

0	1
2	3

Figure 6-11 – Scan for 8x8 luma blocks

The inverse 8x8 luma block scanning process is specified by:

x = InverseRasterScan(luma8x8BlkIdx, 8, 8, 16, 0) (6-19)

y = InverseRasterScan(luma8x8BlkIdx, 8, 8, 16, 1) (6-20)

6.4.6 Inverse 8x8 Cb or Cr block scanning process for ChromaArrayType equal to 3

This process is only invoked when ChromaArrayType is equal to 3.

The inverse 8x8 chroma block scanning process is identical to inverse 8x8 luma block scanning process as specified in subclause 6.4.5 when substituting the term "luma" with the term "Cb" or the term "Cr", and substituting the term "luma8x8BlkIdx" with the term "cb8x8BlkIdx" or the term "cr8x8BlkIdx" in all places in subclause 6.4.5.

6.4.7 Inverse 4x4 chroma block scanning process

Input to this process is the index of a 4x4 chroma block chroma4x4BlkIdx.

Output of this process is the location (x, y) of the upper-left chroma sample for a 4x4 chroma block with index chroma4x4BlkIdx relative to the upper-left chroma sample of the macroblock.

The inverse 4x4 chroma block scanning process is specified by

x = InverseRasterScan(chroma4x4BlkIdx, 4, 4, 8, 0) (6-21)

y = InverseRasterScan(chroma4x4BlkIdx, 4, 4, 8, 1) (6-22)

6.4.8 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 any of the following conditions are true, in which case the macroblock is marked as not available:

– mbAddr < 0,

– mbAddr > CurrMbAddr,

– the macroblock with address mbAddr belongs to a different slice than the macroblock with address CurrMbAddr.

6.4.9 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 6-12 shows the relative spatial locations of the macroblocks with mbAddrA, mbAddrB, mbAddrC, and mbAddrD relative to the current macroblock with CurrMbAddr.

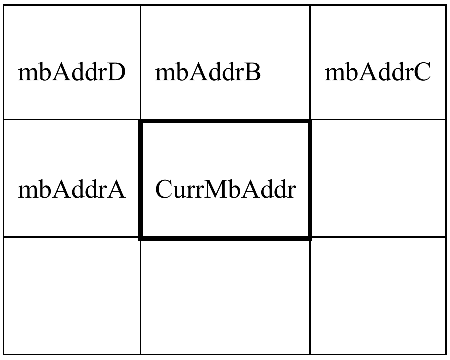


Figure 6-12 – Neighbouring macroblocks for a given macroblock

Input to the process in subclause 6.4.8 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 6.4.8 is mbAddrB = CurrMbAddr − PicWidthInMbs and the output is whether the macroblock mbAddrB is available.

Input to the process in subclause 6.4.8 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 6.4.8 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.

6.4.10 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 6-13 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.

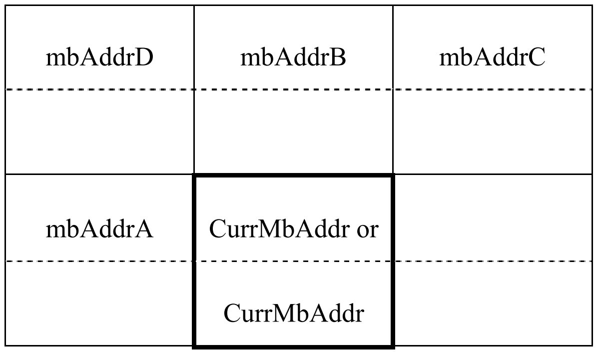


Figure 6-13 – Neighbouring macroblocks for a given macroblock in MBAFF frames

Input to the process in subclause 6.4.8 is mbAddrA = 2 * (CurrMbAddr / 2 − 1) and the output is whether the macroblock mbAddrA is available. In addition, mbAddrA is marked as not available when (CurrMbAddr / 2) % PicWidthInMbs is equal to 0.

Input to the process in subclause 6.4.8 is mbAddrB = 2 * (CurrMbAddr / 2 − PicWidthInMbs) and the output is whether the macroblock mbAddrB is available.

Input to the process in subclause 6.4.8 is mbAddrC = 2 * (CurrMbAddr / 2 − PicWidthInMbs + 1) and the output is whether the macroblock mbAddrC is available. In addition, mbAddrC is marked as not available when (CurrMbAddr / 2 + 1) % PicWidthInMbs is equal to 0.

Input to the process in subclause 6.4.8 is mbAddrD = 2 * (CurrMbAddr / 2 − PicWidthInMbs − 1) and the output is whether the macroblock mbAddrD is available. In addition, mbAddrD is marked as not available when (CurrMbAddr / 2) % PicWidthInMbs is equal to 0.

6.4.11 Derivation processes for neighbouring macroblocks, blocks, and partitions

Subclause 6.4.11.1 specifies the derivation process for neighbouring macroblocks.

Subclause 6.4.11.2 specifies the derivation process for neighbouring 8x8 luma blocks.

Subclause 6.4.11.3 specifies the derivation process for neighbouring 8x8 chroma blocks for ChromaArrayType equal to 3.

Subclause 6.4.11.4 specifies the derivation process for neighbouring 4x4 luma blocks.

Subclause 6.4.11.5 specifies the derivation process for neighbouring 4x4 chroma blocks.

Subclause 6.4.11.6 specifies the derivation process for neighbouring 4x4 chroma blocks for ChromaArrayType equal to 3.

Subclause 6.4.11.7 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, cb8x8BlkIdxN, cr8x8BlkIdxN, luma4x4BlkIdxN, cb4x4BlkIdxN, cr4x4BlkIdxN, and chroma4x4BlkIdxN for the output. These input and output assignments are used in subclauses 6.4.11.1 to 6.4.11.7. The variable predPartWidth is specified when Table 6-2 is referred to.

Table 6-2 – Specification of input and output assignments for subclauses 6.4.11.1 to 6.4.11.7

N	xD	yD
A	−1	0
B	0	−1
C	predPartWidth	−1
D	−1	−1

Figure 6-14 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.

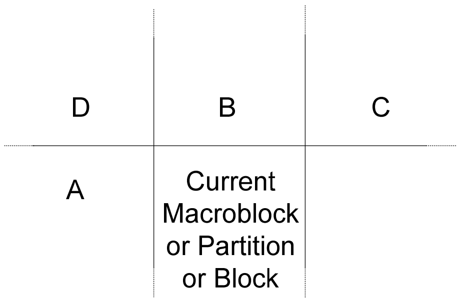


Figure 6-14 – Determination of the neighbouring macroblock, blocks, and partitions (informative)

6.4.11.1 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,

– mbAddrB: the address of the macroblock above the current macroblock and its availability status.

mbAddrN (with N being A or B) is derived as specified by the following ordered steps:

1. The difference of luma location (xD, yD) is set according to Table 6-2.

2. The derivation process for neighbouring locations as specified in subclause 6.4.12 is invoked for luma locations with (xN, yN) equal to (xD, yD), and the output is assigned to mbAddrN.

6.4.11.2 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 specified by the following ordered steps:

1. The difference of luma location (xD, yD) is set according to Table 6-2.

2. The luma location (xN, yN) is specified by

$$xN = (luma8x8BlkIdx \% 2) * 8 + xD \quad (6\text{-}23)$$

$$yN = (luma8x8BlkIdx / 2) * 8 + yD \quad (6\text{-}24)$$

3. The derivation process for neighbouring locations as specified in subclause 6.4.12 is invoked for luma locations with (xN, yN) as the input and the output is assigned to mbAddrN and (xW, yW).

4. The variable luma8x8BlkIdxN is derived as follows:

   – If mbAddrN is not available, luma8x8BlkIdxN is marked as not available.

– Otherwise (mbAddrN is available), the derivation process for 8x8 luma block indices as specified in subclause 6.4.13.3 is invoked with the luma location (xW, yW) as the input and the output is assigned to luma8x8BlkIdxN.

6.4.11.3 Derivation process for neighbouring 8x8 chroma blocks for ChromaArrayType equal to 3

This process is only invoked when ChromaArrayType is equal to 3.

The derivation process for neighbouring 8x8 chroma block is identical to the derivation process for neighbouring 8x8 luma block as specified in subclause 6.4.11.2 when substituting the term "luma" with the term "Cb" or the term "Cr", and substituting the term "luma8x8BlkIdx" with the term "cb8x8BlkIdx" or the term "cr8x8BlkIdx" in all places in subclause 6.4.11.2.

6.4.11.4 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 specified by the following ordered steps:

1. The difference of luma location (xD, yD) is set according to Table 6-2.
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.
3. The luma location (xN, yN) is specified by:

$$xN = x + xD \tag{6-25}$$

$$yN = y + yD \tag{6-26}$$

4. The derivation process for neighbouring locations as specified in subclause 6.4.12 is invoked for luma locations with (xN, yN) as the input and the output is assigned to mbAddrN and (xW, yW).
5. The variable luma4x4BlkIdxN is derived as follows:
   – If mbAddrN is not available, luma4x4BlkIdxN is marked as not available.
   – Otherwise (mbAddrN is available), the derivation process for 4x4 luma block indices as specified in subclause 6.4.13.1 is invoked with the luma location (xW, yW) as the input and the output is assigned to luma4x4BlkIdxN.

6.4.11.5 Derivation process for neighbouring 4x4 chroma blocks

This subclause is only invoked when ChromaArrayType is equal to 1 or 2.

Input to this process is a 4x4 chroma block index 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 4x4 chroma 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 4x4 chroma block with index chroma4x4BlkIdx) and its availability status.

mbAddrN and chroma4x4BlkIdxN (with N being A or B) are derived as specified by the following ordered steps:

1. The difference of chroma location (xD, yD) is set according to Table 6-2.
2. The inverse 4x4 chroma block scanning process as specified in subclause 6.4.7 is invoked with chroma4x4BlkIdx as the input and (x, y) as the output.
3. The chroma location (xN, yN) is specified by

$$xN = x + xD \tag{6-27}$$

$$yN = y + yD \tag{6-28}$$

4. The derivation process for neighbouring locations as specified in subclause 6.4.12 is invoked for chroma locations with (xN, yN) as the input and the output is assigned to mbAddrN and (xW, yW).
5. The variable chroma4x4BlkIdxN is derived as follows:
   – If mbAddrN is not available, chroma4x4BlkIdxN is marked as not available.
   – Otherwise (mbAddrN is available), the derivation process for 4x4 chroma block indices as specified in subclause 6.4.13.2 is invoked with the chroma location (xW, yW) as the input and the output is assigned to chroma4x4BlkIdxN.

6.4.11.6 Derivation process for neighbouring 4x4 chroma blocks for ChromaArrayType equal to 3

This process is only invoked when ChromaArrayType is equal to 3.

The derivation process for neighbouring 4x4 chroma block in 4:4:4 chroma format is identical to the derivation process for neighbouring 4x4 luma block as specified in subclause 6.4.11.4 when substituting the term "luma" with the term "Cb" or the term "Cr", and substituting the term "luma4x4BlkIdx" with the term "cb4x4BlkIdx" or the term "cr4x4BlkIdx" in all places in subclause 6.4.11.4.

6.4.11.7 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 subMbPartIdxN (with N being A, B, C, or D) are derived as specified by the following ordered steps:

1. 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.
2. 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).

3. 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 1 – 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).

4. The difference of luma location (xD, yD) is set according to Table 6-2.

5. The neighbouring luma location (xN, yN) is specified by

$$xN = x + xS + xD \tag{6-29}$$

$$yN = y + yS + yD \tag{6-30}$$

6. The derivation process for neighbouring locations as specified in subclause 6.4.12 is invoked for luma locations with (xN, yN) as the input and the output is assigned to mbAddrN and (xW, yW).

7. Depending on mbAddrN, the following applies:

   – If mbAddrN is not available, the macroblock or sub-macroblock partition mbAddrN\mbPartIdxN\subMbPartIdxN is marked as not available.

   – Otherwise (mbAddrN is available), the following ordered steps are specified:

     a. Let mbTypeN be the syntax element mb_type of the macroblock with macroblock address mbAddrN and, when mbTypeN is equal to P_8x8, P_8x8ref0, or B_8x8, let subMbTypeN be the syntax element list sub_mb_type of the macroblock with macroblock address mbAddrN.

     b. The derivation process for macroblock and sub-macroblock partition indices as specified in subclause 6.4.13.4 is invoked with the luma location (xW, yW), the macroblock type mbTypeN, and, when mbTypeN is equal to P_8x8, P_8x8ref0, or B_8x8, the list of sub-macroblock types subMbTypeN as the inputs and the outputs are the macroblock partition index mbPartIdxN and the sub-macroblock partition index subMbPartIdxN.

     c. 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 2 – The latter condition is, for example, the case when mbPartIdx = 2, subMbPartIdx = 3, xD = 4, yD = −1, i.e., when neighbour C of the last 4x4 luma block of the third sub-macroblock is requested.

6.4.12 Derivation process for neighbouring locations

Input to this process is a luma or chroma location (xN, yN) 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 (xN, yN) and its availability status,

– (xW, yW): the location (xN, yN) expressed relative to the upper-left corner of the macroblock mbAddrN (rather than relative to the upper-left corner of the current macroblock).

Let maxW and maxH be variables specifying maximum values of the location components xN, xW, and yN, yW, respectively. maxW and maxH are derived as follows:

– If this process is invoked for neighbouring luma locations,

$$\text{maxW} = \text{maxH} = 16 \tag{6-31}$$

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

$$\text{maxW} = \text{MbWidthC} \tag{6-32}$$

$$\text{maxH} = \text{MbHeightC} \tag{6-33}$$

Depending on the variable MbaffFrameFlag, the neighbouring locations are derived as follows:

– If MbaffFrameFlag is equal to 0, the specification for neighbouring locations in fields and non-MBAFF frames as described in subclause 6.4.12.1 is applied.

– Otherwise (MbaffFrameFlag is equal to 1), the specification for neighbouring locations in MBAFF frames as described in subclause 6.4.12.2 is applied.

6.4.12.1 Specification for neighbouring 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 6.4.9 is invoked with mbAddrA, mbAddrB, mbAddrC, and mbAddrD as well as their availability status as the output.

Table 6-3 specifies mbAddrN depending on (xN, yN).

Table 6-3 – Specification of mbAddrN

xN	**yN**	**mbAddrN**
< 0	< 0	mbAddrD
< 0	0..maxH − 1	mbAddrA
0..maxW − 1	< 0	mbAddrB
0..maxW − 1	0..maxH − 1	CurrMbAddr
> maxW − 1	< 0	mbAddrC
> maxW − 1	0..maxH − 1	not available
	> maxH − 1	not available

The neighbouring location (xW, yW) relative to the upper-left corner of the macroblock mbAddrN is derived as

$$\text{xW} = (\text{ xN} + \text{maxW}) \% \text{ maxW} \tag{6-34}$$

$$\text{yW} = (\text{ yN} + \text{maxH}) \% \text{ maxH} \tag{6-35}$$

6.4.12.2 Specification for neighbouring 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 6.4.10 is invoked with mbAddrA, mbAddrB, mbAddrC, and mbAddrD as well as their availability status as the output.

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 (i.e., CurrMbAddr % 2 is equal to 0), mbIsTopMbFlag is set equal to 1.

– Otherwise (the macroblock with address CurrMbAddr is a bottom macroblock, i.e., CurrMbAddr % 2 is equal to 1), mbIsTopMbFlag is set equal to 0.

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

1. Specification of a macroblock address mbAddrX depending on (xN, yN) and the variables currMbFrameFlag and mbIsTopMbFlag:

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 (xN, yN), 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

xN	yN	currMbFrameFlag	mbIsTopMbFlag	mbAddrX	mbAddrXFrameFlag	additional condition	mbAddrN	yM
< 0	< 0	1	1	mbAddrD			mbAddrD + 1	yN
			0	mbAddrA	1		mbAddrA	yN
					0		mbAddrA + 1	(yN + maxH) >> 1
		0	1	mbAddrD	1		mbAddrD + 1	2*yN
					0		mbAddrD	yN
			0	mbAddrD			mbAddrD + 1	yN
< 0	0..maxH − 1	1	1	mbAddrA	1		mbAddrA	yN
					0	yN % 2 = = 0	mbAddrA	yN >> 1
						yN % 2 != 0	mbAddrA + 1	yN >> 1
			0	mbAddrA	1		mbAddrA + 1	yN
					0	yN % 2 = = 0	mbAddrA	(yN + maxH) >> 1
						yN % 2 != 0	mbAddrA + 1	(yN + maxH) >> 1
		0	1	mbAddrA	1	yN < (maxH / 2)	mbAddrA	yN <<1
						yN >= (maxH / 2)	mbAddrA + 1	(yN <<1) − maxH
					0		mbAddrA	yN
			0	mbAddrA	1	yN < (maxH / 2)	mbAddrA	(yN <<1) + 1
						yN >= (maxH / 2)	mbAddrA + 1	(yN <<1) + 1 − maxH
					0		mbAddrA + 1	yN
0..maxW − 1	< 0	1	1	mbAddrB			mbAddrB + 1	yN
			0	CurrMbAddr			CurrMbAddr − 1	yN
		0	1	mbAddrB	1		mbAddrB + 1	2 * yN
					0		mbAddrB	yN
			0	mbAddrB			mbAddrB + 1	yN
0..maxW − 1	0..maxH − 1			CurrMbAddr			CurrMbAddr	yN
> maxW − 1	<0	1	1	mbAddrC			mbAddrC + 1	yN
			0	not available			not available	na
		0	1	mbAddrC	1		mbAddrC + 1	2 * yN
					0		mbAddrC	yN
			0	mbAddrC			mbAddrC + 1	yN
> maxW − 1	0..maxH − 1			not available			not available	na
	> maxH − 1			not available			not available	na

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

$$xW = (xN + maxW) \% maxW \tag{6-36}$$

$$yW = (yM + maxH) \% maxH \tag{6-37}$$

6.4.13 Derivation processes for block and partition indices

Subclause 6.4.13.1 specifies the derivation process for 4x4 luma block indices.

Subclause 6.4.13.2 specifies the derivation process for 4x4 chroma block indices.

Subclause 6.4.13.3 specifies the derivation process for 8x8 luma block indices.

Subclause 6.4.13.4 specifies the derivation process for macroblock and sub-macroblock partition indices.

6.4.13.1 Derivation process for 4x4 luma block indices

Input to this process is a luma location (xP, yP) relative to the upper-left luma sample of a macroblock.

Output of this process is a 4x4 luma block index luma4x4BlkIdx.

The 4x4 luma block index luma4x4BlkIdx is derived by

$$\text{luma4x4BlkIdx} = 8 * (yP / 8) + 4 * (xP / 8) + 2 * ((yP \% 8) / 4) + ((xP \% 8) / 4) \quad (6\text{-}38)$$

6.4.13.2 Derivation process for 4x4 chroma block indices

This subclause is only invoked when ChromaArrayType is equal to 1 or 2.

Input to this process is a chroma location (xP, yP) relative to the upper-left chroma sample of a macroblock.

Output of this process is a 4x4 chroma block index chroma4x4BlkIdx.

The 4x4 chroma block index chroma4x4BlkIdx is derived by

$$\text{chroma4x4BlkIdx} = 2 * (yP / 4) + (xP / 4) \quad (6\text{-}39)$$

6.4.13.3 Derivation process for 8x8 luma block indices

Input to this process is a luma location (xP, yP) relative to the upper-left luma sample of a macroblock.

Outputs of this process is an 8x8 luma block index luma8x8BlkIdx.

The 8x8 luma block index luma8x8BlkIdx is derived by

$$\text{luma8x8BlkIdx} = 2 * (yP / 8) + (xP / 8) \quad (6\text{-}40)$$

6.4.13.4 Derivation process for macroblock and sub-macroblock partition indices

Inputs to this process are:

– a luma location (xP, yP) relative to the upper-left luma sample of a macroblock,
– a macroblock type mbType,
– when mbType is equal to P_8x8, P_8x8ref0, or B_8x8, a list of sub-macroblock types subMbType with 4 elements.

Outputs of this process are:

– a macroblock partition index mbPartIdx,
– a sub-macroblock partition index subMbPartIdx.

The macroblock partition index mbPartIdx is derived as follows:

– If mbType specifies an I macroblock type, mbPartIdx is set equal to 0.
– Otherwise (mbType does not specify an I macroblock type), mbPartIdx is derived by

$$\begin{aligned}\text{mbPartIdx} = {} & (16 / \text{MbPartWidth}(\text{mbType})) * (yP / \text{MbPartHeight}(\text{mbType})) + \\ & (xP / \text{MbPartWidth}(\text{mbType}))\end{aligned} \quad (6\text{-}41)$$

The sub-macroblock partition index subMbPartIdx is derived as follows:

– If mbType is not equal to P_8x8, P_8x8ref0, B_8x8, B_Skip, or B_Direct_16x16, subMbPartIdx is set equal to 0.
– Otherwise, if mbType is equal to B_Skip or B_Direct_16x16, subMbPartIdx is derived by

$$\text{subMbPartIdx} = 2 * ((yP \% 8) / 4) + ((xP \% 8) / 4) \quad (6\text{-}42)$$

– Otherwise (mbType is equal to P_8x8, P_8x8ref0, or B_8x8), subMbPartIdx is derived by

$$\begin{aligned}\text{subMbPartIdx} = {} & (8 / \text{SubMbPartWidth}(\text{subMbType}[\text{mbPartIdx}])) * \\ & ((yP \% 8) / \text{SubMbPartHeight}(\text{subMbType}[\text{mbPartIdx}])) + \\ & ((xP \% 8) / \text{SubMbPartWidth}(\text{subMbType}[\text{mbPartIdx}]))\end{aligned} \quad (6\text{-}43)$$

7 Syntax and semantics

7.1 Method of specifying syntax in tabular form

The syntax tables specify a superset of the syntax of all allowed bitstreams. Additional constraints on the syntax may be specified, either directly or indirectly, in other clauses.

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

The following table lists examples of pseudo code used to describe the syntax. When **syntax_element** appears, it specifies that a syntax element is parsed from the bitstream and the bitstream pointer is advanced to the next position beyond the syntax element in the bitstream parsing process.

	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 no more data in the RBSP, the return value of more_rbsp_data( ) is equal to FALSE.

– Otherwise, the RBSP data is searched for the last (least significant, right-most) bit equal to 1 that is present in the RBSP. Given the position of this bit, which is the first bit (rbsp_stop_one_bit) of the rbsp_trailing_bits( ) syntax structure, the following applies:

  – If there is more data in an RBSP before the rbsp_trailing_bits( ) syntax structure, 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

nal_unit(NumBytesInNALunit) {	**C**	**Descriptor**
forbidden_zero_bit	All	f(1)
nal_ref_idc	All	u(2)
nal_unit_type	All	u(5)
NumBytesInRBSP = 0		
nalUnitHeaderBytes = 1		
if(nal_unit_type = = 14 || nal_unit_type = = 20) {		
svc_extension_flag	All	u(1)
if(svc_extension_flag)		
nal_unit_header_svc_extension( ) /* specified in Annex G */	All	
else		
nal_unit_header_mvc_extension( ) /* specified in Annex H */	All	
nalUnitHeaderBytes += 3		
}		
for(i = nalUnitHeaderBytes; i < NumBytesInNALunit; i++) {		
if(i + 2 < NumBytesInNALunit && next_bits(24) = = 0x000003) {		
rbsp_byte[NumBytesInRBSP++ **]**	All	b(8)
rbsp_byte[NumBytesInRBSP++ **]**	All	b(8)
i += 2		
emulation_prevention_three_byte /* equal to 0x03 */	All	f(8)
} else		
rbsp_byte[NumBytesInRBSP++ **]**	All	b(8)
}		
}		

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**
seq_parameter_set_data( )	0	
rbsp_trailing_bits( )	0	
}		

7.3.2.1.1 Sequence parameter set data syntax

seq_parameter_set_data( ) {	**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)
constraint_set3_flag	0	u(1)
constraint_set4_flag	0	u(1)
constraint_set5_flag	0	u(1)
reserved_zero_2bits /* equal to 0 */	0	u(2)
level_idc	0	u(8)
seq_parameter_set_id	0	ue(v)
if(profile_idc = = 100 || profile_idc = = 110 || profile_idc = = 122 || profile_idc = = 244 || profile_idc = = 44 || profile_idc = = 83 || profile_idc = = 86 || profile_idc = = 118 || profile_idc = = 128) {		
chroma_format_idc	0	ue(v)
if(chroma_format_idc = = 3)		
separate_colour_plane_flag	0	u(1)
bit_depth_luma_minus8	0	ue(v)
bit_depth_chroma_minus8	0	ue(v)
qpprime_y_zero_transform_bypass_flag	0	u(1)
seq_scaling_matrix_present_flag	0	u(1)
if(seq_scaling_matrix_present_flag)		
for(i = 0; i < ((chroma_format_idc != 3) ? 8 : 12); i++) {		
seq_scaling_list_present_flag[i]	0	u(1)
if(seq_scaling_list_present_flag[i])		
if(i < 6)		
scaling_list(ScalingList4x4[i], 16, UseDefaultScalingMatrix4x4Flag[i])	0	
else		
scaling_list(ScalingList8x8[i − 6], 64, UseDefaultScalingMatrix8x8Flag[i − 6])	0	
}		
}		
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)
}		
max_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	
}		

7.3.2.1.1.1 Scaling list syntax

scaling_list(scalingList, sizeOfScalingList, useDefaultScalingMatrixFlag) {	**C**	**Descriptor**
lastScale = 8		
nextScale = 8		
for(j = 0; j < sizeOfScalingList; j++) {		
if(nextScale != 0) {		
delta_scale	0 | 1	se(v)
nextScale = (lastScale + delta_scale + 256) % 256		
useDefaultScalingMatrixFlag = (j = = 0 && nextScale = = 0)		
}		
scalingList[j] = (nextScale = = 0) ? lastScale : nextScale		
lastScale = scalingList[j]		
}		
}		

7.3.2.1.2 Sequence parameter set extension RBSP syntax

seq_parameter_set_extension_rbsp( ) {	C	Descriptor
seq_parameter_set_id	10	ue(v)
aux_format_idc	10	ue(v)
if(aux_format_idc != 0) {		
bit_depth_aux_minus8	10	ue(v)
alpha_incr_flag	10	u(1)
alpha_opaque_value	10	u(v)
alpha_transparent_value	10	u(v)
}		
additional_extension_flag	10	u(1)
rbsp_trailing_bits( )	10	
}		

7.3.2.1.3 Subset sequence parameter set RBSP syntax

subset_seq_parameter_set_rbsp( ) {	C	Descriptor
seq_parameter_set_data( )	0	
if(profile_idc = = 83 || profile_idc = = 86) {		
seq_parameter_set_svc_extension( ) /* specified in Annex G */	0	
svc_vui_parameters_present_flag	0	u(1)
if(svc_vui_parameters_present_flag = = 1)		
svc_vui_parameters_extension( ) /* specified in Annex G */	0	
} else if(profile_idc = = 118 || profile_idc = = 128) {		
bit_equal_to_one /* equal to 1 */	0	f(1)
seq_parameter_set_mvc_extension( ) /* specified in Annex H */	0	
mvc_vui_parameters_present_flag	0	u(1)
if(mvc_vui_parameters_present_flag = = 1)		
mvc_vui_parameters_extension( ) /* specified in Annex H */	0	
}		
additional_extension2_flag	0	u(1)
if(additional_extension2_flag = = 1)		
while(more_rbsp_data( ))		
additional_extension2_data_flag	0	u(1)
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)
bottom_field_pic_order_in_frame_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_default_active_minus1	1	ue(v)
num_ref_idx_l1_default_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)
if(more_rbsp_data( )) {		
transform_8x8_mode_flag	1	u(1)
pic_scaling_matrix_present_flag	1	u(1)
if(pic_scaling_matrix_present_flag)		
for(i = 0; i < 6 + ((chroma_format_idc != 3) ? 2 : 6) * transform_8x8_mode_flag; i++) {		
pic_scaling_list_present_flag[i **]**	1	u(1)
if(pic_scaling_list_present_flag[i])		
if(i < 6)		

scaling_list(ScalingList4x4[i], 16, UseDefaultScalingMatrix4x4Flag[i])	1	
else		
scaling_list(ScalingList8x8[i − 6], 64, UseDefaultScalingMatrix8x8Flag[i − 6])	1	
}		
second_chroma_qp_index_offset	1	se(v)
}		
rbsp_trailing_bits( )	1	
}		

7.3.2.3 Supplemental enhancement information RBSP syntax

sei_rbsp( ) {	**C**	**Descriptor**
do		
sei_message( )	5	
while(more_rbsp_data( ))		
rbsp_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)
rbsp_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( ) {	**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	All	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	All	ue(v)
if(separate_colour_plane_flag = = 1)		
colour_plane_id	All	u(2)
if(redundant_pic_cnt_present_flag)		
redundant_pic_cnt	All	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	All	ue(v)
if(separate_colour_plane_flag = = 1)		
colour_plane_id	All	u(2)
if(redundant_pic_cnt_present_flag)		
redundant_pic_cnt	All	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)
}		

7.3.2.12 Prefix NAL unit RBSP syntax

prefix_nal_unit_rbsp( ) {	C	Descriptor
if(svc_extension_flag)		
prefix_nal_unit_svc( ) /* specified in Annex G */	2	
}		

7.3.2.13 Slice layer extension RBSP syntax

slice_layer_extension_rbsp( ) {	C	Descriptor
if(svc_extension_flag) {		
slice_header_in_scalable_extension( ) /* specified in Annex G */	2	
if(!slice_skip_flag)		
slice_data_in_scalable_extension( ) /* specified in Annex G */	2 | 3 | 4	
} else {		
slice_header( )	2	
slice_data( )	2 | 3 | 4	
}		
rbsp_slice_trailing_bits( )	2	
}		

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)
if(separate_colour_plane_flag = = 1)		
colour_plane_id	2	u(2)
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(IdrPicFlag)		
idr_pic_id	2	ue(v)
if(pic_order_cnt_type = = 0) {		
pic_order_cnt_lsb	2	u(v)
if(bottom_field_pic_order_in_frame_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(bottom_field_pic_order_in_frame_present_flag && !field_pic_flag)		
delta_pic_order_cnt[1 **]**	2	se(v)

<table>
<tr><td>   }</td><td></td><td></td></tr>
<tr><td>   if(redundant_pic_cnt_present_flag)</td><td></td><td></td></tr>
<tr><td>   redundant_pic_cnt</td><td>2</td><td>ue(v)</td></tr>
<tr><td>   if(slice_type = = B)</td><td></td><td></td></tr>
<tr><td>   direct_spatial_mv_pred_flag</td><td>2</td><td>u(1)</td></tr>
<tr><td>   if(slice_type = = P | | slice_type = = SP | | slice_type = = B) {</td><td></td><td></td></tr>
<tr><td>   num_ref_idx_active_override_flag</td><td>2</td><td>u(1)</td></tr>
<tr><td>   if(num_ref_idx_active_override_flag) {</td><td></td><td></td></tr>
<tr><td>   num_ref_idx_l0_active_minus1</td><td>2</td><td>ue(v)</td></tr>
<tr><td>   if(slice_type = = B)</td><td></td><td></td></tr>
<tr><td>   num_ref_idx_l1_active_minus1</td><td>2</td><td>ue(v)</td></tr>
<tr><td>   }</td><td></td><td></td></tr>
<tr><td>   }</td><td></td><td></td></tr>
<tr><td>   if(nal_unit_type = = 20)</td><td></td><td></td></tr>
<tr><td>   ref_pic_list_mvc_modification( ) /* specified in Annex H */</td><td>2</td><td></td></tr>
<tr><td>   else</td><td></td><td></td></tr>
<tr><td>   ref_pic_list_modification( )</td><td>2</td><td></td></tr>
<tr><td>   if((weighted_pred_flag && (slice_type = = P | | slice_type = = SP)) | |
        (weighted_bipred_idc = = 1 && slice_type = = B))</td><td></td><td></td></tr>
<tr><td>   pred_weight_table( )</td><td>2</td><td></td></tr>
<tr><td>   if(nal_ref_idc != 0)</td><td></td><td></td></tr>
<tr><td>   dec_ref_pic_marking( )</td><td>2</td><td></td></tr>
<tr><td>   if(entropy_coding_mode_flag && slice_type != I && slice_type != SI)</td><td></td><td></td></tr>
<tr><td>   cabac_init_idc</td><td>2</td><td>ue(v)</td></tr>
<tr><td>   slice_qp_delta</td><td>2</td><td>se(v)</td></tr>
<tr><td>   if(slice_type = = SP | | slice_type = = SI) {</td><td></td><td></td></tr>
<tr><td>   if(slice_type = = SP)</td><td></td><td></td></tr>
<tr><td>   sp_for_switch_flag</td><td>2</td><td>u(1)</td></tr>
<tr><td>   slice_qs_delta</td><td>2</td><td>se(v)</td></tr>
<tr><td>   }</td><td></td><td></td></tr>
<tr><td>   if(deblocking_filter_control_present_flag) {</td><td></td><td></td></tr>
<tr><td>   disable_deblocking_filter_idc</td><td>2</td><td>ue(v)</td></tr>
<tr><td>   if(disable_deblocking_filter_idc != 1) {</td><td></td><td></td></tr>
<tr><td>   slice_alpha_c0_offset_div2</td><td>2</td><td>se(v)</td></tr>
<tr><td>   slice_beta_offset_div2</td><td>2</td><td>se(v)</td></tr>
<tr><td>   }</td><td></td><td></td></tr>
<tr><td>   }</td><td></td><td></td></tr>
<tr><td>   if(num_slice_groups_minus1 > 0 &&
        slice_group_map_type >= 3 && slice_group_map_type <= 5)</td><td></td><td></td></tr>
<tr><td>   slice_group_change_cycle</td><td>2</td><td>u(v)</td></tr>
<tr><td>}</td><td></td><td></td></tr>
</table>

7.3.3.1 Reference picture list modification syntax

ref_pic_list_modification( ) {	C	**Descriptor**
if(slice_type % 5 != 2 && slice_type % 5 != 4) {		
ref_pic_list_modification_flag_l0	2	u(1)
if(ref_pic_list_modification_flag_l0)		
do {		
modification_of_pic_nums_idc	2	ue(v)
if(modification_of_pic_nums_idc = = 0 || modification_of_pic_nums_idc = = 1)		
abs_diff_pic_num_minus1	2	ue(v)
else if(modification_of_pic_nums_idc = = 2)		
long_term_pic_num	2	ue(v)
} while(modification_of_pic_nums_idc != 3)		
}		
if(slice_type % 5 = = 1) {		
ref_pic_list_modification_flag_l1	2	u(1)
if(ref_pic_list_modification_flag_l1)		
do {		
modification_of_pic_nums_idc	2	ue(v)
if(modification_of_pic_nums_idc = = 0 || modification_of_pic_nums_idc = = 1)		
abs_diff_pic_num_minus1	2	ue(v)
else if(modification_of_pic_nums_idc = = 2)		
long_term_pic_num	2	ue(v)
} while(modification_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)
if(ChromaArrayType != 0)		
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)
}		
if (ChromaArrayType != 0) {		
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 % 5 = = 1)		
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)
}		
if(ChromaArrayType != 0) {		
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)
}		
}		
}		
}		

7.3.3.3 Decoded reference picture marking syntax

dec_ref_pic_marking( ) {	**C**	**Descriptor**
if(IdrPicFlag) {		
no_output_of_prior_pics_flag	2 | 5	u(1)
long_term_reference_flag	2 | 5	u(1)
} else {		
adaptive_ref_pic_marking_mode_flag	2 | 5	u(1)
if(adaptive_ref_pic_marking_mode_flag)		
do {		
memory_management_control_operation	2 | 5	ue(v)
if(memory_management_control_operation = = 1 || memory_management_control_operation = = 3)		
difference_of_pic_nums_minus1	2 | 5	ue(v)
if(memory_management_control_operation = = 2)		
long_term_pic_num	2 | 5	ue(v)
if(memory_management_control_operation = = 3 || memory_management_control_operation = = 6)		
long_term_frame_idx	2 | 5	ue(v)
if(memory_management_control_operation = = 4)		
max_long_term_frame_idx_plus1	2 | 5	ue(v)
} while(memory_management_control_operation != 0)		
}		
}		

7.3.4 Slice data syntax

slice_data( ) {	C	Descriptor
if(entropy_coding_mode_flag)		
while(!byte_aligned( ))		
cabac_alignment_one_bit	2	f(1)
CurrMbAddr = first_mb_in_slice * (1 + MbaffFrameFlag)		
moreDataFlag = 1		
prevMbSkipped = 0		
do {		
if(slice_type != I && slice_type != SI)		
if(!entropy_coding_mode_flag) {		
mb_skip_run	2	ue(v)
prevMbSkipped = (mb_skip_run > 0)		
for(i=0; i<mb_skip_run; i++)		
CurrMbAddr = NextMbAddress(CurrMbAddr)		
if(mb_skip_run > 0)		
moreDataFlag = more_rbsp_data( )		
} else {		
mb_skip_flag	2	ae(v)
moreDataFlag = !mb_skip_flag		
}		
if(moreDataFlag) {		
if(MbaffFrameFlag && (CurrMbAddr % 2 = = 0 || (CurrMbAddr % 2 = = 1 && prevMbSkipped)))		
mb_field_decoding_flag	2	u(1) | ae(v)
macroblock_layer( )	2 | 3 | 4	
}		
if(!entropy_coding_mode_flag)		
moreDataFlag = more_rbsp_data( )		
else {		
if(slice_type != I && slice_type != SI)		
prevMbSkipped = mb_skip_flag		
if(MbaffFrameFlag && CurrMbAddr % 2 = = 0)		
moreDataFlag = 1		
else {		
end_of_slice_flag	2	ae(v)
moreDataFlag = !end_of_slice_flag		
}		
}		
CurrMbAddr = NextMbAddress(CurrMbAddr)		
} while(moreDataFlag)		
}		

7.3.5 Macroblock layer syntax

macroblock_layer( ) {	C	Descriptor
mb_type	2	ue(v) | ae(v)
if(mb_type = = I_PCM) {		
while(!byte_aligned( ))		
pcm_alignment_zero_bit	3	f(1)
for(i = 0; i < 256; i++)		
pcm_sample_luma[i]	3	u(v)
for(i = 0; i < 2 * MbWidthC * MbHeightC; i++)		
pcm_sample_chroma[i]	3	u(v)
} else {		
noSubMbPartSizeLessThan8x8Flag = 1		
if(mb_type != I_NxN && MbPartPredMode(mb_type, 0) != Intra_16x16 && NumMbPart(mb_type) = = 4) {		
sub_mb_pred(mb_type)	2	
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8) {		
if(NumSubMbPart(sub_mb_type[mbPartIdx]) > 1)		
noSubMbPartSizeLessThan8x8Flag = 0		
} else if(!direct_8x8_inference_flag)		
noSubMbPartSizeLessThan8x8Flag = 0		
} else {		
if(transform_8x8_mode_flag && mb_type = = I_NxN)		
transform_size_8x8_flag	2	u(1) | ae(v)
mb_pred(mb_type)	2	
}		
if(MbPartPredMode(mb_type, 0) != Intra_16x16) {		
coded_block_pattern	2	me(v) | ae(v)
if(CodedBlockPatternLuma > 0 && transform_8x8_mode_flag && mb_type != I_NxN && noSubMbPartSizeLessThan8x8Flag && (mb_type != B_Direct_16x16 || direct_8x8_inference_flag))		
transform_size_8x8_flag	2	u(1) | ae(v)
}		
if(CodedBlockPatternLuma > 0 || CodedBlockPatternChroma > 0 || MbPartPredMode(mb_type, 0) = = Intra_16x16) {		
mb_qp_delta	2	se(v) | ae(v)
residual(0, 15)	3 | 4	
}		
}		
}		

7.3.5.1 Macroblock prediction syntax

mb_pred(mb_type) {	C	Descriptor

if(MbPartPredMode(mb_type, 0) = = Intra_4x4 ||
     MbPartPredMode(mb_type, 0) = = Intra_8x8 ||
     MbPartPredMode(mb_type, 0) = = Intra_16x16) {		

if(MbPartPredMode(mb_type, 0) = = Intra_4x4)		
for(luma4x4BlkIdx=0; luma4x4BlkIdx<16; luma4x4BlkIdx++) {		
prev_intra4x4_pred_mode_flag[luma4x4BlkIdx **]**	2	u(1) | ae(v)
if(!prev_intra4x4_pred_mode_flag**[** luma4x4BlkIdx **]**)		
rem_intra4x4_pred_mode[luma4x4BlkIdx **]**	2	u(3) | ae(v)
}		
if(MbPartPredMode(mb_type, 0) = = Intra_8x8)		
for(luma8x8BlkIdx=0; luma8x8BlkIdx<4; luma8x8BlkIdx++) {		
prev_intra8x8_pred_mode_flag[luma8x8BlkIdx **]**	2	u(1) | ae(v)
if(!prev_intra8x8_pred_mode_flag[luma8x8BlkIdx])		
rem_intra8x8_pred_mode[luma8x8BlkIdx **]**	2	u(3) | ae(v)
}		
if(ChromaArrayType = = 1 || ChromaArrayType = = 2)		
intra_chroma_pred_mode	2	ue(v) | ae(v)
} else if(MbPartPredMode(mb_type, 0) != Direct) {		
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		

if((num_ref_idx_l0_active_minus1 > 0 ||
           mb_field_decoding_flag != field_pic_flag) &&
         MbPartPredMode(mb_type, mbPartIdx) != Pred_L1)		

ref_idx_l0[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		

if((num_ref_idx_l1_active_minus1 > 0 ||
           mb_field_decoding_flag != field_pic_flag) &&
         MbPartPredMode(mb_type, mbPartIdx) != Pred_L0)		

ref_idx_l1[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode (mb_type, mbPartIdx) != Pred_L1)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l0[mbPartIdx **][** 0 **][** compIdx **]**	2	se(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode(mb_type, mbPartIdx) != Pred_L0)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l1[mbPartIdx **][** 0 **][** compIdx **]**	2	se(v) | ae(v)
}		
}		

7.3.5.2 Sub-macroblock prediction syntax

sub_mb_pred(mb_type) {	**C**	**Descriptor**
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
sub_mb_type[mbPartIdx **]**	2	ue(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		

if((num_ref_idx_l0_active_minus1 > 0 ||
        mb_field_decoding_flag != field_pic_flag) &&
      mb_type != P_8x8ref0 &&
      sub_mb_type[mbPartIdx] != B_Direct_8x8 &&
      SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L1)		

ref_idx_l0[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		

if((num_ref_idx_l1_active_minus1 > 0 ||
        mb_field_decoding_flag != field_pic_flag) &&
      sub_mb_type[mbPartIdx] != B_Direct_8x8 &&
      SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L0)		

ref_idx_l1[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L1)		

for(subMbPartIdx = 0;
          subMbPartIdx < NumSubMbPart(sub_mb_type[mbPartIdx]);
          subMbPartIdx++)		

for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l0[mbPartIdx **][** subMbPartIdx **][** compIdx **]**	2	se(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L0)		

for(subMbPartIdx = 0;
          subMbPartIdx < NumSubMbPart(sub_mb_type[mbPartIdx]);
          subMbPartIdx++)		

for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l1[mbPartIdx **][** subMbPartIdx **][** compIdx **]**	2	se(v) | ae(v)
}		

7.3.5.3 Residual data syntax

residual(startIdx, endIdx) {	C	Descriptor
if(!entropy_coding_mode_flag)		
residual_block = residual_block_cavlc		
else		
residual_block = residual_block_cabac		
residual_luma(i16x16DClevel, i16x16AClevel, level4x4, level8x8, startIdx, endIdx)	3 | 4	
Intra16x16DCLevel = i16x16DClevel		
Intra16x16ACLevel = i16x16AClevel		
LumaLevel4x4 = level4x4		
LumaLevel8x8 = level8x8		
if(ChromaArrayType = = 1 || ChromaArrayType = = 2) {		
NumC8x8 = 4 / (SubWidthC * SubHeightC)		
for(iCbCr = 0; iCbCr < 2; iCbCr++)		
if((CodedBlockPatternChroma & 3) && startIdx = = 0) /* chroma DC residual present */		
residual_block(ChromaDCLevel[iCbCr], 0, 4 * NumC8x8 − 1, 4 * NumC8x8)	3 | 4	
else		
for(i = 0; i < 4 * NumC8x8; i++)		
ChromaDCLevel[iCbCr][i] = 0		
for(iCbCr = 0; iCbCr < 2; iCbCr++)		
for(i8x8 = 0; i8x8 < NumC8x8; i8x8++)		
for(i4x4 = 0; i4x4 < 4; i4x4++)		
if(CodedBlockPatternChroma & 2) /* chroma AC residual present */		
residual_block(ChromaACLevel[iCbCr][i8x8*4+i4x4], Max(0, startIdx − 1), endIdx − 1, 15)	3 | 4	
else		
for(i = 0; i < 15; i++)		
ChromaACLevel[iCbCr][i8x8*4+i4x4][i] = 0		
} else if(ChromaArrayType = = 3) {		
residual_luma(i16x16DClevel, i16x16AClevel, level4x4, level8x8, startIdx, endIdx)	3 | 4	
CbIntra16x16DCLevel = i16x16DClevel		
CbIntra16x16ACLevel = i16x16AClevel		
CbLevel4x4 = level4x4		
CbLevel8x8 = level8x8		
residual_luma(i16x16DClevel, i16x16AClevel, level4x4, level8x8, startIdx, endIdx)	3 | 4	
CrIntra16x16DCLevel = i16x16DClevel		
CrIntra16x16ACLevel = i16x16AClevel		
CrLevel4x4 = level4x4		
CrLevel8x8 = level8x8		
}		

7.3.5.3.1 Residual luma syntax

residual_luma(i16x16DClevel, i16x16AClevel, level4x4, level8x8, startIdx, endIdx) {	C	Descriptor
if(startIdx = = 0 && MbPartPredMode(mb_type, 0) = = Intra_16x16)		
residual_block(i16x16DClevel, 0, 15, 16)	3	
for(i8x8 = 0; i8x8 < 4; i8x8++)		
if(!transform_size_8x8_flag || !entropy_coding_mode_flag)		
for(i4x4 = 0; i4x4 < 4; i4x4++) {		
if(CodedBlockPatternLuma & (1 << i8x8))		
if(MbPartPredMode(mb_type, 0) = = Intra_16x16)		
residual_block(i16x16AClevel[i8x8*4+ i4x4], Max(0, startIdx − 1), endIdx − 1, 15)	3	
else		
residual_block(level4x4[i8x8 * 4 + i4x4], startIdx, endIdx, 16)	3 | 4	
else if(MbPartPredMode(mb_type, 0) = = Intra_16x16)		
for(i = 0; i < 15; i++)		
i16x16AClevel[i8x8 * 4 + i4x4][i] = 0		
else		
for(i = 0; i < 16; i++)		
level4x4[i8x8 * 4 + i4x4][i] = 0		
if(!entropy_coding_mode_flag && transform_size_8x8_flag)		
for(i = 0; i < 16; i++)		
level8x8[i8x8][4 * i + i4x4] = level4x4[i8x8 * 4 + i4x4][i]		
}		
else if(CodedBlockPatternLuma & (1 << i8x8))		
residual_block(level8x8[i8x8], 4 * startIdx, 4 * endIdx + 3, 64)	3 | 4	
else		
for(i = 0; i < 64; i++)		
level8x8[i8x8][i] = 0		
}		

7.3.5.3.2 Residual block CAVLC syntax

residual_block_cavlc(coeffLevel, startIdx, endIdx, maxNumCoeff) {	**C**	**Descriptor**
for(i = 0; i < maxNumCoeff; i++)		
coeffLevel[i] = 0		
coeff_token	3 | 4	ce(v)
if(TotalCoeff(coeff_token) > 0) {		
if(TotalCoeff(coeff_token) > 10 && TrailingOnes(coeff_token) < 3)		
suffixLength = 1		
else		
suffixLength = 0		
for(i = 0; i < TotalCoeff(coeff_token); i++)		
if(i < TrailingOnes(coeff_token)) {		
trailing_ones_sign_flag	3 | 4	u(1)
levelVal[i] = 1 − 2 * trailing_ones_sign_flag		
} else {		
level_prefix	3 | 4	ce(v)
levelCode = (Min(15, level_prefix) << suffixLength)		
if(suffixLength > 0 || level_prefix >= 14) {		
level_suffix	3 | 4	u(v)
levelCode += level_suffix		
}		
if(level_prefix >= 15 && suffixLength == 0)		
levelCode += 15		
if(level_prefix >= 16)		
levelCode += (1 << (level_prefix − 3)) − 4096		
if(i == TrailingOnes(coeff_token) && TrailingOnes(coeff_token) < 3)		
levelCode += 2		
if(levelCode % 2 == 0)		
levelVal[i] = (levelCode + 2) >> 1		
else		
levelVal[i] = (−levelCode − 1) >> 1		
if(suffixLength == 0)		
suffixLength = 1		
if(Abs(levelVal[i]) > (3 << (suffixLength − 1)) && suffixLength < 6)		
suffixLength++		
}		
if(TotalCoeff(coeff_token) < endIdx − startIdx + 1) {		
total_zeros	3 | 4	ce(v)
zerosLeft = total_zeros		
} else		
zerosLeft = 0		
for(i = 0; i < TotalCoeff(coeff_token) − 1; i++) {		
if(zerosLeft > 0) {		
run_before	3 | 4	ce(v)
runVal[i] = run_before		
} else		
runVal[i] = 0		

zerosLeft = zerosLeft − runVal[i]		
}		
runVal[TotalCoeff(coeff_token) − 1] = zerosLeft		
coeffNum = −1		
for(i = TotalCoeff(coeff_token) − 1; i >= 0; i− −) {		
coeffNum += runVal[i] + 1		
coeffLevel[startIdx + coeffNum] = levelVal[i]		
}		
}		
}		

7.3.5.3.3 Residual block CABAC syntax

residual_block_cabac(coeffLevel, startIdx, endIdx, maxNumCoeff) {	C	Descriptor
if(maxNumCoeff != 64 | | ChromaArrayType = = 3)		
coded_block_flag	3 | 4	ae(v)
for(i = 0; i < maxNumCoeff; i++)		
coeffLevel[i] = 0		
if(coded_block_flag) {		
numCoeff = endIdx + 1		
i = startIdx		
while(i < numCoeff − 1) {		
significant_coeff_flag[i **]**	3 | 4	ae(v)
if(significant_coeff_flag[i]) {		
last_significant_coeff_flag[i **]**	3 | 4	ae(v)
if(last_significant_coeff_flag[i])		
numCoeff = i + 1		
}		
i++		
}		
coeff_abs_level_minus1[numCoeff − 1 **]**	3 | 4	ae(v)
coeff_sign_flag[numCoeff − 1 **]**	3 | 4	ae(v)
coeffLevel[numCoeff − 1] = (coeff_abs_level_minus1[numCoeff − 1] + 1) * (1 − 2 * coeff_sign_flag[numCoeff − 1])		
for(i = numCoeff − 2; i >= startIdx; i− −)		
if(significant_coeff_flag[i]) {		
coeff_abs_level_minus1[i **]**	3 | 4	ae(v)
coeff_sign_flag[i **]**	3 | 4	ae(v)
coeffLevel[i] = (coeff_abs_level_minus1[i] + 1) * (1 − 2 * coeff_sign_flag[i])		
}		
}		
}		

7.4 Semantics

Semantics associated with the syntax structures and with the syntax elements within these structures are specified in this subclause. When the semantics of a syntax element are specified using a table or a set of tables, any values that are not specified in the table(s) shall not be present in the bitstream unless otherwise specified in this Recommendation | International Standard.

7.4.1 NAL unit semantics

NOTE 1 – The VCL is specified to efficiently represent the content of the video data. The NAL is specified to format that data and provide header information in a manner appropriate for conveyance on a variety of communication channels or storage media. All data are contained in NAL units, each of which contains an integer number of bytes. A NAL unit specifies a generic format for use in both packet-oriented and bitstream systems. The format of NAL units for both packet-oriented transport and byte stream is identical except that each NAL unit can be preceded by a start code prefix and extra padding bytes in the byte stream format.

NumBytesInNALunit specifies the size of the NAL unit in bytes. This value is required for decoding of the NAL unit. Some form of demarcation of NAL unit boundaries is necessary to enable inference of NumBytesInNALunit. One such demarcation method is specified in Annex B for the byte stream format. Other methods of demarcation may be specified outside of this Recommendation | International Standard.

forbidden_zero_bit shall be equal to 0.

nal_ref_idc not equal to 0 specifies that the content of the NAL unit contains a sequence parameter set, a sequence parameter set extension, a subset sequence parameter set, a picture parameter set, a slice of a reference picture, a slice data partition of a reference picture, or a prefix NAL unit preceding a slice of a reference picture.

For coded video sequences conforming to one or more of the profiles specified in Annex A that are decoded using the decoding process specified in clauses 2-9, nal_ref_idc equal to 0 for a NAL unit containing a slice or slice data partition indicates that the slice or slice data partition is part of a non-reference picture.

nal_ref_idc shall not be equal to 0 for sequence parameter set or sequence parameter set extension or subset sequence parameter set or picture parameter set NAL units. When nal_ref_idc is equal to 0 for one NAL unit with nal_unit_type in the range of 1 to 4, inclusive, of a particular picture, it shall be equal to 0 for all NAL units with nal_unit_type in the range of 1 to 4, inclusive, of the picture.

nal_ref_idc shall not be equal to 0 for NAL units with nal_unit_type equal to 5.

nal_ref_idc shall be equal to 0 for all NAL units having nal_unit_type equal to 6, 9, 10, 11, or 12.

nal_unit_type specifies the type of RBSP data structure contained in the NAL unit as specified in Table 7-1.

The column marked "C" in Table 7-1 lists the categories of the syntax elements that may be present in the NAL unit. In addition, syntax elements with syntax category "All" may be present, as determined by the syntax and semantics of the RBSP data structure. The presence or absence of any syntax elements of a particular listed category is determined from the syntax and semantics of the associated RBSP data structure. nal_unit_type shall not be equal to 3 or 4 unless at least one syntax element is present in the RBSP data structure having a syntax element category value equal to the value of nal_unit_type and not categorized as "All".

For coded video sequences conforming to one or more of the profiles specified in Annex A that are decoded using the decoding process specified in clauses 2-9, VCL and non-VCL NAL units are specified in Table 7-1 in the column labelled "Annex A NAL unit type class". For coded video sequences conforming to one or more of the profiles specified in Annex G that are decoded using the decoding process specified in Annex G and for coded video sequences conforming to one or more of the profiles specified in Annex H that are decoded using the decoding process specified in Annex H, VCL and non-VCL NAL units are specified in Table 7-1 in the column labelled "Annex G and Annex H NAL unit type class". The entry "suffix dependent" for nal_unit_type equal to 14 is specified as follows:

– If the NAL unit directly following in decoding order a NAL unit with nal_unit_type equal to 14 is a NAL unit with nal_unit_type equal to 1 or 5, the NAL unit with nal_unit_type equal to 14 is a VCL NAL unit.

– Otherwise (the NAL unit directly following in decoding order a NAL unit with nal_unit_type equal to 14 is a NAL unit with nal_unit_type not equal to 1 or 5), the NAL unit with nal_unit_type equal to 14 is a non-VCL NAL unit. Decoders shall ignore (remove from the bitstream and discard) the NAL unit with nal_unit_type equal to 14 and the NAL unit directly following (in decoding order) the NAL unit with nal_unit_type equal to 14.

Table 7-1 – NAL unit type codes, syntax element categories, and NAL unit type classes

nal_unit_type	Content of NAL unit and RBSP syntax structure	C	**Annex** A **NAL unit type class**	**Annex** G **and Annex** H **NAL unit type class**
0	Unspecified		non-VCL	non-VCL
1	Coded slice of a non-IDR picture slice_layer_without_partitioning_rbsp( )	2, 3, 4	VCL	VCL
2	Coded slice data partition A slice_data_partition_a_layer_rbsp( )	2	VCL	not applicable
3	Coded slice data partition B slice_data_partition_b_layer_rbsp( )	3	VCL	not applicable
4	Coded slice data partition C slice_data_partition_c_layer_rbsp( )	4	VCL	not applicable
5	Coded slice of an IDR picture slice_layer_without_partitioning_rbsp( )	2, 3	VCL	VCL
6	Supplemental enhancement information (SEI) sei_rbsp( )	5	non-VCL	non-VCL
7	Sequence parameter set seq_parameter_set_rbsp( )	0	non-VCL	non-VCL
8	Picture parameter set pic_parameter_set_rbsp( )	1	non-VCL	non-VCL
9	Access unit delimiter access_unit_delimiter_rbsp( )	6	non-VCL	non-VCL
10	End of sequence end_of_seq_rbsp( )	7	non-VCL	non-VCL
11	End of stream end_of_stream_rbsp( )	8	non-VCL	non-VCL
12	Filler data filler_data_rbsp( )	9	non-VCL	non-VCL
13	Sequence parameter set extension seq_parameter_set_extension_rbsp( )	10	non-VCL	non-VCL
14	Prefix NAL unit prefix_nal_unit_rbsp( )	2	non-VCL	suffix dependent
15	Subset sequence parameter set subset_seq_parameter_set_rbsp( )	0	non-VCL	non-VCL
16..18	Reserved		non-VCL	non-VCL
19	Coded slice of an auxiliary coded picture without partitioning slice_layer_without_partitioning_rbsp( )	2, 3, 4	non-VCL	non-VCL
20	Coded slice extension slice_layer_extension_rbsp( )	2, 3, 4	non-VCL	VCL
21..23	Reserved		non-VCL	non-VCL
24..31	Unspecified		non-VCL	non-VCL

When NAL units with nal_unit_type equal to 13 or 19 are present in a coded video sequence, decoders shall either perform the (optional) decoding process specified for these NAL units or shall ignore (remove from the bitstream and discard) the contents of these NAL units.

Decoders that conform to one or more of the profiles specified in Annex A rather than the profiles specified in Annexes G or H shall ignore (remove from the bitstream and discard) the contents of all NAL units with nal_unit_type equal to 14, 15, or 20.

NAL units that use nal_unit_type equal to 0 or in the range of 24..31, inclusive, shall not affect the decoding process specified in this Recommendation | International Standard.

NOTE 2 – NAL unit types 0 and 24..31 may be used as determined by the application. No decoding process for these values of nal_unit_type is specified in this Recommendation | International Standard. Since different applications might use NAL unit types 0 and 24..31 for different purposes, particular care must be exercised in the design of encoders that generate NAL units with nal_unit_type equal to 0 or in the range of 24 to 31, inclusive, and in the design of decoders that interpret the content of NAL units with nal_unit_type equal to 0 or in the range of 24 to 31, inclusive.

Decoders shall ignore (remove from the bitstream and discard) the contents of all NAL units that use reserved values of nal_unit_type.

NOTE 3 – This requirement allows future definition of compatible extensions to this Recommendation | International Standard.

NOTE 4 – In previous editions of this Recommendation | International Standard, the NAL unit types 13..15 and 19..20 (or a subset of these NAL unit types) were reserved and no decoding process for NAL units having these values of nal_unit_type was specified. In later editions of this Recommendation | International Standard, currently reserved values of nal_unit_type might become non-reserved and a decoding process for these values of nal_unit_type might be specified. Encoders should take into consideration that the values of nal_unit_type that were reserved in previous editions of this Recommendation | International Standard might be ignored by decoders.

In the text, coded slice NAL unit collectively refers to a coded slice of a non-IDR picture NAL unit or to a coded slice of an IDR picture NAL unit. The variable IdrPicFlag is specified as

$$\text{IdrPicFlag} = (\,(\,\text{nal\_unit\_type} == 5\,)\ ?\ 1 : 0\,) \qquad (7\text{-}1)$$

When the value of nal_unit_type is equal to 5 for a NAL unit containing a slice of a particular picture, the picture shall not contain NAL units with nal_unit_type in the range of 1 to 4, inclusive. For coded video sequences conforming to one or more of the profiles specified in Annex A that are decoded using the decoding process specified in clauses 2-9, such a picture is referred to as an IDR picture.

NOTE 5 – Slice data partitioning cannot be used for IDR pictures.

svc_extension_flag indicates whether a nal_unit_header_svc_extension( ) or nal_unit_header_mvc_extension( ) will follow next in the syntax structure.

The value of svc_extension_flag shall be equal to 1 for coded video sequences conforming to one or more profiles specified in Annex G. Decoders conforming to one or more profiles specified in Annex G shall ignore (remove from the bitstream and discard) NAL units for which nal_unit_type is equal to 14 or 20 and for which svc_extension_flag is equal to 0.

The value of svc_extension_flag shall be equal to 0 for coded video sequences conforming to one or more profiles specified in Annex H. Decoders conforming to one or more profiles specified in Annex H shall ignore (remove from the bitstream and discard) NAL units for which nal_unit_type is equal to 14 or 20 and for which svc_extension_flag is equal to 1.

rbsp_byte[i **]** is the i-th byte of an RBSP. An RBSP is specified as an ordered sequence of bytes as follows.

The RBSP contains an SODB as follows:

– If the SODB is empty (i.e., zero bits in length), the RBSP is also empty.
– Otherwise, the RBSP contains the SODB as follows:
  1) The first byte of the RBSP contains the (most significant, left-most) eight bits of the SODB; the next byte of the RBSP contains the next eight bits of the SODB, etc., until fewer than eight bits of the SODB remain.
  2) rbsp_trailing_bits( ) are present after the SODB as follows:
     i) The first (most significant, left-most) bits of the final RBSP byte contains the remaining bits of the SODB (if any).
     ii) The next bit consists of a single rbsp_stop_one_bit equal to 1.
     iii) When the rbsp_stop_one_bit is not the last bit of a byte-aligned byte, one or more rbsp_alignment_zero_bit is present to result in byte alignment.
  3) One or more cabac_zero_word 16-bit syntax elements equal to 0x0000 may be present in some RBSPs after the rbsp_trailing_bits( ) at the end of the RBSP.

Syntax structures having these RBSP properties are denoted in the syntax tables using an "_rbsp" suffix. These structures shall be carried within NAL units as the content of the rbsp_byte[i] data bytes. The association of the RBSP syntax structures to the NAL units shall be as specified in Table 7-1.

NOTE 6 – When the boundaries of the RBSP are known, the decoder can extract the SODB from the RBSP by concatenating the bits of the bytes of the RBSP and discarding the rbsp_stop_one_bit, which is the last (least significant, right-most) bit equal to 1, and discarding any following (less significant, farther to the right) bits that follow it, which are equal to 0. The data necessary for the decoding process is contained in the SODB part of the RBSP.

emulation_prevention_three_byte is a byte equal to 0x03. When an emulation_prevention_three_byte is present in the NAL unit, it shall be discarded by the decoding process.

The last byte of the NAL unit shall not be equal to 0x00.

Within the NAL unit, the following three-byte sequences shall not occur at any byte-aligned position:

– 0x000000
– 0x000001
– 0x000002

Within the NAL unit, any four-byte sequence that starts with 0x000003 other than the following sequences shall not occur at any byte-aligned position:

– 0x00000300
– 0x00000301
– 0x00000302
– 0x00000303

NOTE 7 – When nal_unit_type is equal to 0, particular care must be exercised in the design of encoders to avoid the presence of the above-listed three-byte and four-byte patterns at the beginning of the NAL unit syntax structure, as the syntax element emulation_prevention_three_byte cannot be the third byte of a NAL unit.

7.4.1.1 Encapsulation of an SODB within an RBSP (informative)

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

The form of encapsulation of an SODB within an RBSP and the use of the emulation_prevention_three_byte for encapsulation of an RBSP within a NAL unit is specified for the following purposes:

– to prevent the emulation of start codes within NAL units while allowing any arbitrary SODB to be represented within a NAL unit,
– to enable identification of the end of the SODB within the NAL unit by searching the RBSP for the rbsp_stop_one_bit starting at the end of the RBSP,
– to enable a NAL unit to have a size larger than that of the SODB under some circumstances (using one or more cabac_zero_word).

The encoder can produce a NAL unit from an RBSP by the following procedure:

1. The RBSP data is searched for byte-aligned bits of the following binary patterns:

   '00000000 00000000 000000xx' (where xx represents any 2 bit pattern: 00, 01, 10, or 11),

   and a byte equal to 0x03 is inserted to replace these bit patterns with the patterns:

   '00000000 00000000 00000011 000000xx',

   and finally, when the last byte of the RBSP data is equal to 0x00 (which can only occur when the RBSP ends in a cabac_zero_word), a final byte equal to 0x03 is appended to the end of the data. The last zero byte of a byte-aligned three-byte sequence 0x000000 in the RBSP (which is replaced by the four-byte sequence 0x00000300) is taken into account when searching the RBSP data for the next occurrence of byte-aligned bits with the binary patterns specified above.

2. The resulting sequence of bytes is then prefixed as follows:

   – If nal_unit_type is not equal to 14 or 20, the sequence of bytes is prefixed with the first byte of the NAL unit containing the syntax elements forbidden_zero_bit, nal_ref_idc, and nal_unit_type, where nal_unit_type indicates the type of RBSP data structure the NAL unit contains.

   – Otherwise (nal_unit_type is equal to 14 or 20), the sequence of bytes is prefixed with the first four bytes of the NAL unit, where the first byte contains the syntax elements forbidden_zero_bit, nal_ref_idc, and nal_unit_type and the following three bytes contain the syntax structure nal_unit_header_svc_extension( ). The syntax element nal_unit_type in the first byte indicates the presence of the syntax structure nal_unit_header_svc_extension( ) in the following three bytes and the type of RBSP data structure the NAL unit contains.

The process specified above results in the construction of the entire NAL unit.

This process can allow any SODB to be represented in a NAL unit while ensuring that

– no byte-aligned start code prefix is emulated within the NAL unit,
– no sequence of 8 zero-valued bits followed by a start code prefix, regardless of byte-alignment, is emulated within the NAL unit.

7.4.1.2 Order of NAL units and association to coded pictures, access units, and video sequences

This subclause specifies constraints on the order of NAL units in the bitstream.

Any order of NAL units in the bitstream obeying these constraints is referred to in the text as the decoding order of NAL units. Within a NAL unit, the syntax in subclauses 7.3, D.1, and E.1 specifies the decoding order of syntax elements. Decoders shall be capable of receiving NAL units and their syntax elements in decoding order.

7.4.1.2.1 Order of sequence and picture parameter set RBSPs and their activation

This subclause specifies the activation process of picture and sequence parameter sets for coded video sequences that conform to one or more of the profiles specified in Annex A and are decoded using the decoding process specified in clauses 2-9.

NOTE 1 – The sequence and picture parameter set mechanism decouples the transmission of infrequently changing information from the transmission of coded macroblock data. Sequence and picture parameter sets may, in some applications, be conveyed "out-of-band" using a reliable transport mechanism.

A picture parameter set RBSP includes parameters that can be referred to by the coded slice NAL units or coded slice data partition A NAL units of one or more coded pictures. Each picture parameter set RBSP is initially considered not active at the start of the operation of the decoding process. At most one picture parameter set RBSP is considered active at any given moment during the operation of the decoding process, and the activation of any particular picture parameter set RBSP results in the deactivation of the previously-active picture parameter set RBSP (if any).

When a picture parameter set RBSP (with a particular value of pic_parameter_set_id) is not active and it is referred to by a coded slice NAL unit or coded slice data partition A NAL unit (using that value of pic_parameter_set_id), it is activated. This picture parameter set RBSP is called the active picture parameter set RBSP until it is deactivated by the activation of another picture parameter set RBSP. A picture parameter set RBSP, with that particular value of pic_parameter_set_id, shall be available to the decoding process prior to its activation.

Any picture parameter set NAL unit containing the value of pic_parameter_set_id for the active picture parameter set RBSP for a coded picture shall have the same content as that of the active picture parameter set RBSP for the coded picture unless it follows the last VCL NAL unit of the coded picture and precedes the first VCL NAL unit of another coded picture.

When a picture parameter set NAL unit with a particular value of pic_parameter_set_id is received, its content replaces the content of the previous picture parameter set NAL unit, in decoding order, with the same value of pic_parameter_set_id (when a previous picture parameter set NAL unit with the same value of pic_parameter_set_id was present in the bitstream).

NOTE 2 – A decoder must be capable of simultaneously storing the contents of the picture parameter sets for all values of pic_parameter_set_id. The content of the picture parameter set with a particular value of pic_parameter_set_id is overwritten when a new picture parameter set NAL unit with the same value of pic_parameter_set_id is received.

A sequence parameter set RBSP includes parameters that can be referred to by one or more picture parameter set RBSPs or one or more SEI NAL units containing a buffering period SEI message. Each sequence parameter set RBSP is initially considered not active at the start of the operation of the decoding process. At most one sequence parameter set RBSP is considered active at any given moment during the operation of the decoding process, and the activation of any particular sequence parameter set RBSP results in the deactivation of the previously-active sequence parameter set RBSP (if any).

When a sequence parameter set RBSP (with a particular value of seq_parameter_set_id) is not already active and it is referred to by activation of a picture parameter set RBSP (using that value of seq_parameter_set_id) or is referred to by an SEI NAL unit containing a buffering period SEI message (using that value of seq_parameter_set_id), it is activated. This sequence parameter set RBSP is called the active sequence parameter set RBSP until it is deactivated by the activation of another sequence parameter set RBSP. A sequence parameter set RBSP, with that particular value of seq_parameter_set_id, shall be available to the decoding process prior to its activation. An activated sequence parameter set RBSP shall remain active for the entire coded video sequence.

NOTE 3 – Because an IDR access unit begins a new coded video sequence and an activated sequence parameter set RBSP must remain active for the entire coded video sequence, a sequence parameter set RBSP can only be activated by a buffering period SEI message when the buffering period SEI message is part of an IDR access unit.

Any sequence parameter set NAL unit containing the value of seq_parameter_set_id for the active sequence parameter set RBSP for a coded video sequence shall have the same content as that of the active sequence parameter set RBSP for the coded video sequence unless it follows the last access unit of the coded video sequence and precedes the first VCL NAL unit and the first SEI NAL unit containing a buffering period SEI message (when present) of another coded video sequence.

NOTE 4 – If picture parameter set RBSP or sequence parameter set RBSP are conveyed within the bitstream, these constraints impose an order constraint on the NAL units that contain the picture parameter set RBSP or sequence parameter set RBSP, respectively. Otherwise (picture parameter set RBSP or sequence parameter set RBSP are conveyed by other means not specified in this Recommendation | International Standard), they must be available to the decoding process in a timely fashion such that these constraints are obeyed.

When a sequence parameter set NAL unit with a particular value of seq_parameter_set_id is received, its content replaces the content of the previous sequence parameter set NAL unit, in decoding order, with the same value of seq_parameter_set_id (when a previous sequence parameter set NAL unit with the same value of seq_parameter_set_id was present in the bitstream).

NOTE 5 – A decoder must be capable of simultaneously storing the contents of the sequence parameter sets for all values of seq_parameter_set_id. The content of the sequence parameter set with a particular value of seq_parameter_set_id is overwritten when a new sequence parameter set NAL unit with the same value of seq_parameter_set_id is received.

When present, a sequence parameter set extension RBSP includes parameters having a similar function to those of a sequence parameter set RBSP. For purposes of establishing constraints on the syntax elements of the sequence parameter set extension RBSP and for purposes of determining activation of a sequence parameter set extension RBSP, the sequence parameter set extension RBSP shall be considered part of the preceding sequence parameter set RBSP with the same value of seq_parameter_set_id. When a sequence parameter set RBSP is present that is not followed by a sequence parameter set extension RBSP with the same value of seq_parameter_set_id prior to the activation of the sequence parameter set RBSP, the sequence parameter set extension RBSP and its syntax elements shall be considered not present for the active sequence parameter set RBSP.

All constraints that are expressed on the relationship between the values of the syntax elements (and the values of variables derived from those syntax elements) in sequence parameter sets and picture parameter sets and other syntax elements are expressions of constraints that apply only to the active sequence parameter set and the active picture parameter set. If any sequence parameter set RBSP is present that is not activated in the bitstream, its syntax elements shall have values that would conform to the specified constraints if it were activated by reference in an otherwise-conforming bitstream. If any picture parameter set RBSP is present that is not ever activated in the bitstream, its syntax elements shall have values that would conform to the specified constraints if it were activated by reference in an otherwise-conforming bitstream.

During operation of the decoding process (see clause 8), the values of parameters of the active picture parameter set and the active sequence parameter set shall be considered in effect. For interpretation of SEI messages, the values of the parameters of the picture parameter set and sequence parameter set that are active for the operation of the decoding process for the VCL NAL units of the primary coded picture in the same access unit shall be considered in effect unless otherwise specified in the SEI message semantics.

7.4.1.2.2 Order of access units and association to coded video sequences

A bitstream conforming to this Recommendation | International Standard consists of one or more coded video sequences.

A coded video sequence consists of one or more access units. For coded video sequences that conform to one or more of the profiles specified in Annex A and are decoded using the decoding process specified in clauses 2-9, the order of NAL units and coded pictures and their association to access units is described in subclause 7.4.1.2.3.

The first access unit of each coded video sequence is an IDR access unit. All subsequent access units in the coded video sequence are non-IDR access units.

It is a requirement of bitstream conformance that, when two consecutive access units in decoding order within a coded video sequence both contain non-reference pictures, the value of picture order count for each coded field or field of a coded frame in the first such access unit shall be less than or equal to the value of picture order count for each coded field or field of a coded frame in the second such access unit.

It is a requirement of bitstream conformance that, when present, an access unit following an access unit that contains an end of sequence NAL unit shall be an IDR access unit.

It is a requirement of bitstream conformance that, when an SEI NAL unit contains data that pertain to more than one access unit (for example, when the SEI NAL unit has a coded video sequence as its scope), it shall be contained in the first access unit to which it applies.

It is a requirement of bitstream conformance that, when an end of stream NAL unit is present in an access unit, this access unit shall be the last access unit in the bitstream and the end of stream NAL unit shall be the last NAL unit in that access unit.

7.4.1.2.3 Order of NAL units and coded pictures and association to access units

This subclause specifies the order of NAL units and coded pictures and association to access unit for coded video sequences that conform to one or more of the profiles specified in Annex A and are decoded using the decoding process specified in clauses 2-9.

NOTE 1 – Some bitstreams that conform to profiles specified in Annexes G or H may violate the NAL unit order specified in this subclause. Conditions under which such a violation of the NAL unit order occurs are specified in subclauses G.7.4.1.2.3 and H.7.4.1.2.3.

An access unit consists of one primary coded picture, zero or more corresponding redundant coded pictures, and zero or more non-VCL NAL units. The association of VCL NAL units to primary or redundant coded pictures is described in subclause 7.4.1.2.5.

The first access unit in the bitstream starts with the first NAL unit of the bitstream.

The first of any of the following NAL units after the last VCL NAL unit of a primary coded picture specifies the start of a new access unit:

– access unit delimiter NAL unit (when present),

– sequence parameter set NAL unit (when present),

– picture parameter set NAL unit (when present),

– SEI NAL unit (when present),

– NAL units with nal_unit_type in the range of 14 to 18, inclusive (when present),

– first VCL NAL unit of a primary coded picture (always present).

The constraints for the detection of the first VCL NAL unit of a primary coded picture are specified in subclause 7.4.1.2.4.

The following constraints shall be obeyed by the order of the coded pictures and non-VCL NAL units within an access unit:

– When an access unit delimiter NAL unit is present, it shall be the first NAL unit. There shall be at most one access unit delimiter NAL unit in any access unit.

– When any SEI NAL units are present, they shall precede the primary coded picture.

– When an SEI NAL unit containing a buffering period SEI message is present, the buffering period SEI message shall be the first SEI message payload of the first SEI NAL unit in the access unit.

– The primary coded picture shall precede the corresponding redundant coded pictures.

– When redundant coded pictures are present, they shall be ordered in ascending order of the value of redundant_pic_cnt.

– When a sequence parameter set extension NAL unit is present, it shall be the next NAL unit after a sequence parameter set NAL unit having the same value of seq_parameter_set_id as in the sequence parameter set extension NAL unit.

– When one or more coded slice of an auxiliary coded picture without partitioning NAL units is present, they shall follow the primary coded picture and all redundant coded pictures (if any).

– When an end of sequence NAL unit is present, it shall follow the primary coded picture and all redundant coded pictures (if any) and all coded slice of an auxiliary coded picture without partitioning NAL units (if any).

– When an end of stream NAL unit is present, it shall be the last NAL unit.

– NAL units having nal_unit_type equal to 0, 12, or in the range of 20 to 31, inclusive, shall not precede the first VCL NAL unit of the primary coded picture.

NOTE 2 – Sequence parameter set NAL units or picture parameter set NAL units may be present in an access unit, but cannot follow the last VCL NAL unit of the primary coded picture within the access unit, as this condition would specify the start of a new access unit.

NOTE 3 – When a NAL unit having nal_unit_type equal to 7 or 8 is present in an access unit, it may or may not be referred to in the coded pictures of the access unit in which it is present, and may be referred to in coded pictures of subsequent access units.

The structure of access units not containing any NAL units with nal_unit_type equal to 0, 7, 8, or in the range of 12 to 18, inclusive, or in the range of 20 to 31, inclusive, is shown in Figure 7-1.

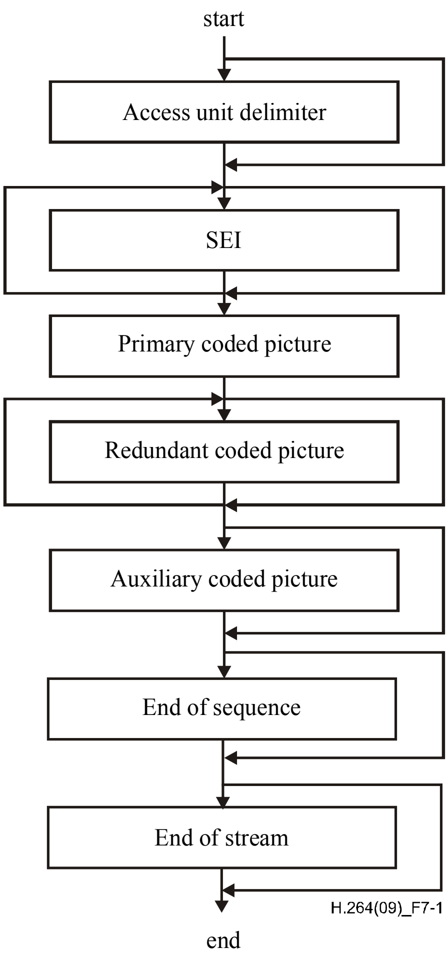


Figure 7-1 – Structure of an access unit not containing any NAL units with nal_unit_type equal to 0, 7, 8, or in the range of 12 to 18, inclusive, or in the range of 20 to 31, inclusive

7.4.1.2.4 Detection of the first VCL NAL unit of a primary coded picture

This subclause specifies constraints on VCL NAL unit syntax that are sufficient to enable the detection of the first VCL NAL unit of each primary coded picture for coded video sequences that conform to one or more of the profiles specified in Annex A and are decoded using the decoding process specified in clauses 2-9.

Any coded slice NAL unit or coded slice data partition A NAL unit of the primary coded picture of the current access unit shall be different from any coded slice NAL unit or coded slice data partition A NAL unit of the primary coded picture of the previous access unit in one or more of the following ways:

– frame_num differs in value. The value of frame_num used to test this condition is the value of frame_num that appears in the syntax of the slice header, regardless of whether that value is inferred to have been equal to 0 for subsequent use in the decoding process due to the presence of memory_management_control_operation equal to 5.

  NOTE 1 – A consequence of the above statement is that a primary coded picture having frame_num equal to 1 cannot contain a memory_management_control_operation equal to 5 unless some other condition listed below is fulfilled for the next primary coded picture that follows after it (if any).

– pic_parameter_set_id differs in value.

– field_pic_flag differs in value.

– bottom_field_flag is present in both and differs in value.

– nal_ref_idc differs in value with one of the nal_ref_idc values being equal to 0.

– pic_order_cnt_type is equal to 0 for both and either pic_order_cnt_lsb differs in value, or delta_pic_order_cnt_bottom differs in value.

– pic_order_cnt_type is equal to 1 for both and either delta_pic_order_cnt[0] differs in value, or delta_pic_order_cnt[1] differs in value.

– IdrPicFlag differs in value.

– IdrPicFlag is equal to 1 for both and idr_pic_id differs in value.

NOTE 2 – Some of the VCL NAL units in redundant coded pictures or some non-VCL NAL units (e.g., an access unit delimiter NAL unit) may also be used for the detection of the boundary between access units, and may therefore aid in the detection of the start of a new primary coded picture.

7.4.1.2.5 Order of VCL NAL units and association to coded pictures

This subclause specifies the order of VCL NAL units and association to coded pictures for coded video sequences that conform to one or more of the profiles specified in Annex A and are decoded using the decoding process specified in clauses 2-9.

Each VCL NAL unit is part of a coded picture.

The order of the VCL NAL units within a coded IDR picture is constrained as follows:

– If arbitrary slice order is allowed as specified in Annex A, coded slice of an IDR picture NAL units may have any order relative to each other.

– Otherwise (arbitrary slice order is not allowed), the following applies:

  – If separate_colour_plane_flag is equal to 0, coded slice of an IDR picture NAL units of a slice group shall not be interleaved with coded slice of an IDR picture NAL units of another slice group and the order of coded slice of an IDR picture NAL units within a slice group shall be in the order of increasing macroblock address for the first macroblock of each coded slice of an IDR picture NAL unit of the particular slice group.

  – Otherwise (separate_colour_plane_flag is equal to 1), coded slice of an IDR picture NAL units of a slice group for a particular value of colour_plane_id shall not be interleaved with coded slice of an IDR picture NAL units of another slice group with the same value of colour_plane_id and the order of coded slices of IDR picture NAL units within a slice group for a particular value of colour_plane_id shall be in the order of increasing macroblock address for the first macroblock of each coded slice of an IDR picture NAL unit of the particular slice group having the particular value of colour_plane_id.

    NOTE 1 – When separate_colour_plane_flag is equal to 1, the relative ordering of coded slices having different values of colour_plane_id is not constrained.

The order of the VCL NAL units within a coded non-IDR picture is constrained as follows:

– If arbitrary slice order is allowed as specified in Annex A, coded slice of a non-IDR picture NAL units or coded slice data partition A NAL units may have any order relative to each other. A coded slice data partition A NAL unit with a particular value of slice_id shall precede any present coded slice data partition B NAL unit with the same value of slice_id. A coded slice data partition A NAL unit with a particular value of slice_id shall precede any present coded slice data partition C NAL unit with the same value of slice_id. When a coded slice data partition B NAL unit with a particular value of slice_id is present, it shall precede any present coded slice data partition C NAL unit with the same value of slice_id.

– Otherwise (arbitrary slice order is not allowed), the following applies:

  – If separate_colour_plane_flag is equal to 0, coded slice of a non-IDR picture NAL units or coded slice data partition NAL units of a slice group shall not be interleaved with coded slice of a non-IDR picture NAL units or coded slice data partition NAL units of another slice group and the order of coded slice of a non-IDR picture NAL units or coded slice data partition A NAL units within a slice group shall be in the order of increasing macroblock address for the first macroblock of each coded slice of a non-IDR picture NAL unit or coded slice data partition A NAL unit of the particular slice group. A coded slice data partition A NAL unit with a particular value of slice_id shall immediately precede any present coded slice data partition B NAL unit with the same value of slice_id. A coded slice data partition A NAL unit with a particular value of slice_id shall immediately precede any present coded slice data partition C NAL unit with the same value of slice_id, when a coded slice data partition B NAL unit with the same value of slice_id is not present. When a coded slice data partition B NAL unit with a particular value of slice_id is present, it shall immediately precede any present coded slice data partition C NAL unit with the same value of slice_id.

– Otherwise (separate_colour_plane_flag is equal to 1), coded slice of a non-IDR picture NAL units or coded slice data partition NAL units of a slice group for a particular value of colour_plane_id shall not be interleaved with coded slice of a non-IDR picture NAL units or coded slice data partition NAL units of another slice group with the same value of colour_plane_id and the order of coded slice of a non-IDR picture NAL units or coded slice data partition A NAL units within a slice group for particular value of colour_plane_id shall be in the order of increasing macroblock address for the first macroblock of each coded slice of a non-IDR picture NAL unit or coded slice data partition A NAL unit of the particular slice group having the particular value of colour_plane_id. A coded slice data partition A NAL unit associated with a particular value of slice_id and colour_plane_id shall immediately precede any present coded slice data partition B NAL unit with the same value of slice_id and colour_plane_id. A coded slice data partition A NAL unit associated with a particular value of slice_id and colour_plane_id shall immediately precede any present coded slice data partition C NAL unit with the same value of slice_id and colour_plane_id, when a coded slice data partition B NAL unit with the same value of slice_id and colour_plane_id is not present. When a coded slice data partition B NAL unit with a particular value of slice_id and colour_plane_id is present, it shall immediately precede any present coded slice data partition C NAL unit with the same value of slice_id and colour_plane_id.

  NOTE 2 – When separate_colour_plane_flag is equal to 1, the relative ordering of coded slices having different values of colour_plane_id is not constrained.

NAL units having nal_unit_type equal to 12 may be present in the access unit but shall not precede the first VCL NAL unit of the primary coded picture within the access unit.

NAL units having nal_unit_type equal to 0 or in the range of 24 to 31, inclusive, which are unspecified, may be present in the access unit but shall not precede the first VCL NAL unit of the primary coded picture within the access unit.

NAL units having nal_unit_type in the range of 20 to 23, inclusive, shall not precede the first VCL NAL unit of the primary coded picture within the access unit.

7.4.2 Raw byte sequence payloads and RBSP trailing bits semantics

7.4.2.1 Sequence parameter set RBSP semantics

7.4.2.1.1 Sequence parameter set data semantics

profile_idc and **level_idc** indicate the profile and level to which the coded video sequence conforms.

constraint_set0_flag equal to 1 indicates that the coded video sequence obeys all constraints specified in subclause A.2.1. constraint_set0_flag equal to 0 indicates that the coded video sequence may or may not obey all constraints specified in subclause A.2.1.

constraint_set1_flag equal to 1 indicates that the coded video sequence obeys all constraints specified in subclause A.2.2. constraint_set1_flag equal to 0 indicates that the coded video sequence may or may not obey all constraints specified in subclause A.2.2.

constraint_set2_flag equal to 1 indicates that the coded video sequence obeys all constraints specified in subclause A.2.3. constraint_set2_flag equal to 0 indicates that the coded video sequence may or may not obey all constraints specified in subclause A.2.3.

NOTE 1 – When one or more than one of constraint_set0_flag, constraint_set1_flag, or constraint_set2_flag are equal to 1, the coded video sequence must obey the constraints of all of the indicated subclauses of subclause A.2. When profile_idc is equal to 44, 100, 110, 122, or 244, the values of constraint_set0_flag, constraint_set1_flag, and constraint_set2_flag must all be equal to 0.

constraint_set3_flag is specified as follows:

– If profile_idc is equal to 66, 77, or 88 and level_idc is equal to 11, constraint_set3_flag equal to 1 indicates that the coded video sequence obeys all constraints specified in Annex A for level 1b and constraint_set3_flag equal to 0 indicates that the coded video sequence obeys all constraints specified in Annex A for level 1.1.

– Otherwise, if profile_idc is equal to 100 or 110, constraint_set3_flag equal to 1 indicates that the coded video sequence obeys all constraints specified in Annex A for the High 10 Intra profile, and constraint_set3_flag equal to 0 indicates that the coded video sequence may or may not obey these corresponding constraints.

– Otherwise, if profile_idc is equal to 122, constraint_set3_flag equal to 1 indicates that the coded video sequence obeys all constraints specified in Annex A for the High 4:2:2 Intra profile, and constraint_set3_flag equal to 0 indicates that the coded video sequence may or may not obey these corresponding constraints.

– Otherwise, if profile_idc is equal to 44, constraint_set3_flag shall be equal to 1. When profile_idc is equal to 44, the value of 0 for constraint_set3_flag is forbidden.

– Otherwise, if profile_idc is equal to 244, constraint_set3_flag equal to 1 indicates that the coded video sequence obeys all constraints specified in Annex A for the High 4:4:4 Intra profile, and constraint_set3_flag equal to 0 indicates that the coded video sequence may or may not obey these corresponding constraints.

– Otherwise (profile_idc is equal to 66, 77, or 88 and level_idc is not equal to 11), the value of 1 for constraint_set3_flag is reserved for future use by ITU-T | ISO/IEC. constraint_set3_flag shall be equal to 0 for coded video sequences with profile_idc equal to 66, 77, or 88 and level_idc not equal to 11 in bitstreams conforming to this Recommendation | International Standard. Decoders shall ignore the value of constraint_set3_flag when profile_idc is equal to 66, 77, or 88 and level_idc is not equal to 11.

constraint_set4_flag is specified as follows:

– If profile_idc is equal to 77, 88, or 100, constraint_set4_flag equal to 1 indicates that the value of frame_mbs_only_flag is equal to 1. constraint_set4_flag equal to 0 indicates that the value of frame_mbs_only_flag may or may not be equal to 1.

– Otherwise, if profile_idc is equal to 118 or 128, constraint_set4_flag equal to 1 indicates that the coded video sequence obeys all constraints specified in subclause H.10.1.1. constraint_set4_flag equal to 0 indicates that the coded video sequence may or may not obey the constraints specified in subclause H.10.1.1.

– Otherwise (profile_idc is not equal to 77, 88, 100, 118, or 128), the value of 1 for constraint_set4_flag is reserved for future use by ITU-T | ISO/IEC. constraint_set4_flag shall be equal to 0 for coded video sequences with profile_idc not equal to 77, 88, 100, 118, or 128 in bitstreams conforming to this Recommendation | International Standard. Decoders shall ignore the value of constraint_set4_flag when profile_idc is not equal to 77, 88, 100, 118, or 128.

constraint_set5_flag is specified as follows:

− If profile_idc is equal to 118, constraint_set5_flag equal to 1 indicates that the coded video sequence obeys all constraints specified in subclause H.10.1.2 and constraint_set5_flag equal to 0 indicates that the coded video sequence may or may not obey all constraints specified in subclause H.10.1.2.

− Otherwise (profile_idc is not equal to 118), the value of 1 for constraint_set5_flag is reserved for future use by ITU-T | ISO/IEC. constraint_set5_flag shall be equal to 0 when profile_idc is not equal to 118 in bitstreams conforming to this Recommendation | International Standard. Decoders shall ignore the value of constraint_set5_flag when profile_idc is not equal to 118.

NOTE 2 – For a coded video sequence conforming to both Multiview High and Stereo High profiles, the profile_idc should be equal to 118 and constraint_set5 flag should be equal to 1.

reserved_zero_2bits shall be equal to 0. Other values of reserved_zero_2bits may be specified in the future by ITU-T | ISO/IEC. Decoders shall ignore the value of reserved_zero_2bits.

seq_parameter_set_id identifies the sequence parameter set that is referred to by the picture parameter set. The value of seq_parameter_set_id shall be in the range of 0 to 31, inclusive.

NOTE 3 – When feasible, encoders should use distinct values of seq_parameter_set_id when the values of other sequence parameter set syntax elements differ rather than changing the values of the syntax elements associated with a specific value of seq_parameter_set_id.

chroma_format_idc specifies the chroma sampling relative to the luma sampling as specified in subclause 6.2. The value of chroma_format_idc shall be in the range of 0 to 3, inclusive. When chroma_format_idc is not present, it shall be inferred to be equal to 1 (4:2:0 chroma format).

separate_colour_plane_flag equal to 1 specifies that the three colour components of the 4:4:4 chroma format are coded separately. separate_colour_plane_flag equal to 0 specifies that the colour components are not coded separately. When separate_colour_plane_flag is not present, it shall be inferred to be equal to 0. When separate_colour_plane_flag is equal to 1, the primary coded picture consists of three separate components, each of which consists of coded samples of one colour plane (Y, Cb or Cr) that each use the monochrome coding syntax. In this case, each colour plane is associated with a specific colour_plane_id value.

NOTE 4 – There is no dependency in decoding processes between the colour planes having different colour_plane_id values. For example, the decoding process of a monochrome picture with one value of colour_plane_id does not use any data from monochrome pictures having different values of colour_plane_id for inter prediction.

Depending on the value of separate_colour_plane_flag, the value of the variable ChromaArrayType is assigned as follows:

– If separate_colour_plane_flag is equal to 0, ChromaArrayType is set equal to chroma_format_idc.

– Otherwise (separate_colour_plane_flag is equal to 1), ChromaArrayType is set equal to 0.

bit_depth_luma_minus8 specifies the bit depth of the samples of the luma array and the value of the luma quantisation parameter range offset $QpBdOffset_Y$, as specified by

$$BitDepth_Y = 8 + bit\_depth\_luma\_minus8 \quad (7\text{-}2)$$

$$QpBdOffset_Y = 6 * bit\_depth\_luma\_minus8 \quad (7\text{-}3)$$

When bit_depth_luma_minus8 is not present, it shall be inferred to be equal to 0. bit_depth_luma_minus8 shall be in the range of 0 to 6, inclusive.

bit_depth_chroma_minus8 specifies the bit depth of the samples of the chroma arrays and the value of the chroma quantisation parameter range offset $QpBdOffset_C$, as specified by

$$BitDepth_C = 8 + bit\_depth\_chroma\_minus8 \quad (7\text{-}4)$$

$$QpBdOffset_C = 6 * bit\_depth\_chroma\_minus8 \quad (7\text{-}5)$$

When bit_depth_chroma_minus8 is not present, it shall be inferred to be equal to 0. bit_depth_chroma_minus8 shall be in the range of 0 to 6, inclusive.

NOTE 5 – The value of bit_depth_chroma_minus8 is not used in the decoding process when ChromaArrayType is equal to 0. In particular, when separate_colour_plane_flag is equal to 1, each colour plane is decoded as a distinct monochrome picture using the luma component decoding process (except for the selection of scaling matrices) and the luma bit depth is used for all three colour components.

The variable RawMbBits is derived as

$$RawMbBits = 256 * BitDepth_Y + 2 * MbWidthC * MbHeightC * BitDepth_C \quad (7\text{-}6)$$

qpprime_y_zero_transform_bypass_flag equal to 1 specifies that, when QP'_Y is equal to 0, a transform bypass operation for the transform coefficient decoding process and picture construction process prior to deblocking filter process as specified in subclause 8.5 shall be applied. qpprime_y_zero_transform_bypass_flag equal to 0 specifies that the transform coefficient decoding process and picture construction process prior to deblocking filter process shall not use the transform bypass operation. When qpprime_y_zero_transform_bypass_flag is not present, it shall be inferred to be equal to 0.

seq_scaling_matrix_present_flag equal to 1 specifies that the flags seq_scaling_list_present_flag[i] for i = 0..7 or i = 0..11 are present. seq_scaling_matrix_present_flag equal to 0 specifies that these flags are not present and the sequence-level scaling list specified by Flat_4x4_16 shall be inferred for i = 0..5 and the sequence-level scaling list specified by Flat_8x8_16 shall be inferred for i = 6..11. When seq_scaling_matrix_present_flag is not present, it shall be inferred to be equal to 0.

The scaling lists Flat_4x4_16 and Flat_8x8_16 are specified as follows:

$$Flat\_4x4\_16[\ k\ ] = 16, \quad \text{with } k = 0..15, \quad (7\text{-}7)$$

$$Flat\_8x8\_16[\ k\ ] = 16, \quad \text{with } k = 0..63. \quad (7\text{-}8)$$

seq_scaling_list_present_flag[i] equal to 1 specifies that the syntax structure for scaling list i is present in the sequence parameter set. seq_scaling_list_present_flag[i] equal to 0 specifies that the syntax structure for scaling list i is not present in the sequence parameter set and the scaling list fall-back rule set A specified in Table 7-2 shall be used to infer the sequence-level scaling list for index i.

Table 7-2 – Assignment of mnemonic names to scaling list indices and specification of fall-back rule

Value of scaling list index	Mnemonic name	Block size	MB prediction type	Component	Scaling list fall-back rule set A	Scaling list fall-back rule set B	Default scaling list
0	Sl_4x4_Intra_Y	4x4	Intra	Y	default scaling list	sequence-level scaling list	Default_4x4_Intra
1	Sl_4x4_Intra_Cb	4x4	Intra	Cb	scaling list for i = 0	scaling list for i = 0	Default_4x4_Intra
2	Sl_4x4_Intra_Cr	4x4	Intra	Cr	scaling list for i = 1	scaling list for i = 1	Default_4x4_Intra
3	Sl_4x4_Inter_Y	4x4	Inter	Y	default scaling list	sequence-level scaling list	Default_4x4_Inter
4	Sl_4x4_Inter_Cb	4x4	Inter	Cb	scaling list for i = 3	scaling list for i = 3	Default_4x4_Inter
5	Sl_4x4_Inter_Cr	4x4	Inter	Cr	scaling list for i = 4	scaling list for i = 4	Default_4x4_Inter
6	Sl_8x8_Intra_Y	8x8	Intra	Y	default scaling list	sequence-level scaling list	Default_8x8_Intra
7	Sl_8x8_Inter_Y	8x8	Inter	Y	default scaling list	sequence-level scaling list	Default_8x8_Inter
8	Sl_8x8_Intra_Cb	8x8	Intra	Cb	scaling list for i = 6	scaling list for i = 6	Default_8x8_Intra
9	Sl_8x8_Inter_Cb	8x8	Inter	Cb	scaling list for i = 7	scaling list for i = 7	Default_8x8_Inter
10	Sl_8x8_Intra_Cr	8x8	Intra	Cr	scaling list for i = 8	scaling list for i = 8	Default_8x8_Intra
11	Sl_8x8_Inter_Cr	8x8	Inter	Cr	scaling list for i = 9	scaling list for i = 9	Default_8x8_Inter

Table 7-3 specifies the default scaling lists Default_4x4_Intra and Default_4x4_Inter. Table 7-4 specifies the default scaling lists Default_8x8_Intra and Default_8x8_Inter.

Table 7-3 – Specification of default scaling lists Default_4x4_Intra and Default_4x4_Inter

idx	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Default_4x4_Intra[idx]	6	13	13	20	20	20	28	28	28	28	32	32	32	37	37	42
Default_4x4_Inter[idx]	10	14	14	20	20	20	24	24	24	24	27	27	27	30	30	34

Table 7-4 – Specification of default scaling lists Default_8x8_Intra and Default_8x8_Inter

idx	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Default_8x8_Intra[idx]	6	10	10	13	11	13	16	16	16	16	18	18	18	18	18	23
Default_8x8_Inter[idx]	9	13	13	15	13	15	17	17	17	17	19	19	19	19	19	21

Table 7-4 (continued) – Specification of default scaling lists Default_8x8_Intra and Default_8x8_Inter

idx	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
Default_8x8_Intra[idx]	23	23	23	23	23	25	25	25	25	25	25	25	27	27	27	27
Default_8x8_Inter[idx]	21	21	21	21	21	22	22	22	22	22	22	22	24	24	24	24

Table 7-4 (continued) – Specification of default scaling lists Default_8x8_Intra and Default_8x8_Inter

idx	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
Default_8x8_Intra[idx]	27	27	27	27	29	29	29	29	29	29	29	31	31	31	31	31
Default_8x8_Inter[idx]	24	24	24	24	25	25	25	25	25	25	25	27	27	27	27	27

Table 7-4 (concluded) – Specification of default scaling lists Default_8x8_Intra and Default_8x8_Inter

idx	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
Default_8x8_Intra[idx]	31	33	33	33	33	33	36	36	36	36	38	38	38	40	40	42
Default_8x8_Inter[idx]	27	28	28	28	28	28	30	30	30	30	32	32	32	33	33	35

log2_max_frame_num_minus4 specifies the value of the variable MaxFrameNum that is used in frame_num related derivations as follows:

$$\text{MaxFrameNum} = 2^{(\text{ log2\_max\_frame\_num\_minus4} + 4\text{ })} \quad (7\text{-}9)$$

The value of log2_max_frame_num_minus4 shall be in the range of 0 to 12, inclusive.

pic_order_cnt_type specifies the method to decode picture order count (as specified in subclause 8.2.1). The value of pic_order_cnt_type shall be in the range of 0 to 2, inclusive.

pic_order_cnt_type shall not be equal to 2 in a coded video sequence that contains any of the following:

– an access unit containing a non-reference frame followed immediately by an access unit containing a non-reference picture,

– two access units each containing a field with the two fields together forming a complementary non-reference field pair followed immediately by an access unit containing a non-reference picture,

– an access unit containing a non-reference field followed immediately by an access unit containing another non-reference picture that does not form a complementary non-reference field pair with the first of the two access units.

log2_max_pic_order_cnt_lsb_minus4 specifies the value of the variable MaxPicOrderCntLsb that is used in the decoding process for picture order count as specified in subclause 8.2.1 as follows:

$$\text{MaxPicOrderCntLsb} = 2^{(\text{ log2\_max\_pic\_order\_cnt\_lsb\_minus4} + 4\text{ })} \quad (7\text{-}10)$$

The value of log2_max_pic_order_cnt_lsb_minus4 shall be in the range of 0 to 12, inclusive.

delta_pic_order_always_zero_flag equal to 1 specifies that delta_pic_order_cnt[0] and delta_pic_order_cnt[1] are not present in the slice headers of the sequence and shall be inferred to be equal to 0. delta_pic_order_always_zero_flag equal to 0 specifies that delta_pic_order_cnt[0] is present in the slice headers of the sequence and delta_pic_order_cnt[1] may be present in the slice headers of the sequence.

offset_for_non_ref_pic is used to calculate the picture order count of a non-reference picture as specified in subclause 8.2.1. The value of offset_for_non_ref_pic shall be in the range of $-2^{31} + 1$ to $2^{31} - 1$, inclusive.

offset_for_top_to_bottom_field is used to calculate the picture order count of a bottom field as specified in subclause 8.2.1. The value of offset_for_top_to_bottom_field shall be in the range of $-2^{31} + 1$ to $2^{31} - 1$, inclusive.

num_ref_frames_in_pic_order_cnt_cycle is used in the decoding process for picture order count as specified in subclause 8.2.1. The value of num_ref_frames_in_pic_order_cnt_cycle shall be in the range of 0 to 255, inclusive.

offset_for_ref_frame[i] is an element of a list of num_ref_frames_in_pic_order_cnt_cycle values used in the decoding process for picture order count as specified in subclause 8.2.1. The value of offset_for_ref_frame[i] shall be in the range of $-2^{31} + 1$ to $2^{31} - 1$, inclusive.

When pic_order_cnt_type is equal to 1, the variable ExpectedDeltaPerPicOrderCntCycle is derived by

```
ExpectedDeltaPerPicOrderCntCycle = 0
for( i = 0;  i < num_ref_frames_in_pic_order_cnt_cycle;  i++ )
    ExpectedDeltaPerPicOrderCntCycle += offset_for_ref_frame[ i ]                (7-11)
```

max_num_ref_frames specifies the maximum number of short-term and long-term reference frames, complementary reference field pairs, and non-paired reference fields that may be used by the decoding process for inter prediction of any picture in the coded video sequence. max_num_ref_frames also determines the size of the sliding window operation as specified in subclause 8.2.5.3. The value of max_num_ref_frames shall be in the range of 0 to MaxDpbFrames (as specified in subclause A.3.1 or A.3.2), inclusive.

gaps_in_frame_num_value_allowed_flag specifies the allowed values of frame_num as specified in subclause 7.4.3 and the decoding process in case of an inferred gap between values of frame_num as specified in subclause 8.2.5.2.

pic_width_in_mbs_minus1 plus 1 specifies the width of each decoded picture in units of macroblocks.

The variable for the picture width in units of macroblocks is derived as

$$\text{PicWidthInMbs} = \text{pic\_width\_in\_mbs\_minus1} + 1 \tag{7-12}$$

The variable for picture width for the luma component is derived as

$$\text{PicWidthInSamples}_L = \text{PicWidthInMbs} * 16 \tag{7-13}$$

The variable for picture width for the chroma components is derived as

$$\text{PicWidthInSamples}_C = \text{PicWidthInMbs} * \text{MbWidthC} \tag{7-14}$$

pic_height_in_map_units_minus1 plus 1 specifies the height in slice group map units of a decoded frame or field.

The variables PicHeightInMapUnits and PicSizeInMapUnits are derived as

$$\text{PicHeightInMapUnits} = \text{pic\_height\_in\_map\_units\_minus1} + 1 \tag{7-15}$$

$$\text{PicSizeInMapUnits} = \text{PicWidthInMbs} * \text{PicHeightInMapUnits} \tag{7-16}$$

frame_mbs_only_flag equal to 0 specifies that coded pictures of the coded video sequence may either be coded fields or coded frames. frame_mbs_only_flag equal to 1 specifies that every coded picture of the coded video sequence is a coded frame containing only frame macroblocks.

The allowed range of values for pic_width_in_mbs_minus1, pic_height_in_map_units_minus1, and frame_mbs_only_flag is specified by constraints in Annex A.

Depending on frame_mbs_only_flag, semantics are assigned to pic_height_in_map_units_minus1 as follows:

– If frame_mbs_only_flag is equal to 0, pic_height_in_map_units_minus1 plus 1 is the height of a field in units of macroblocks.

– Otherwise (frame_mbs_only_flag is equal to 1), pic_height_in_map_units_minus1 plus 1 is the height of a frame in units of macroblocks.

The variable FrameHeightInMbs is derived as

$$\text{FrameHeightInMbs} = (2 - \text{frame\_mbs\_only\_flag}) * \text{PicHeightInMapUnits} \quad (7\text{-}17)$$

mb_adaptive_frame_field_flag equal to 0 specifies no switching between frame and field macroblocks within a picture. mb_adaptive_frame_field_flag equal to 1 specifies the possible use of switching between frame and field macroblocks within frames. When mb_adaptive_frame_field_flag is not present, it shall be inferred to be equal to 0.

direct_8x8_inference_flag specifies the method used in the derivation process for luma motion vectors for B_Skip, B_Direct_16x16 and B_Direct_8x8 as specified in subclause 8.4.1.2. When frame_mbs_only_flag is equal to 0, direct_8x8_inference_flag shall be equal to 1.

frame_cropping_flag equal to 1 specifies that the frame cropping offset parameters follow next in the sequence parameter set. frame_cropping_flag equal to 0 specifies that the frame cropping offset parameters are not present.

frame_crop_left_offset, frame_crop_right_offset, frame_crop_top_offset, frame_crop_bottom_offset specify the samples of the pictures in the coded video sequence that are output from the decoding process, in terms of a rectangular region specified in frame coordinates for output.

The variables CropUnitX and CropUnitY are derived as follows:

– If ChromaArrayType is equal to 0, CropUnitX and CropUnitY are derived as:

$$\text{CropUnitX} = 1 \quad (7\text{-}18)$$
$$\text{CropUnitY} = 2 - \text{frame\_mbs\_only\_flag} \quad (7\text{-}19)$$

– Otherwise (ChromaArrayType is equal to 1, 2, or 3), CropUnitX and CropUnitY are derived as:

$$\text{CropUnitX} = \text{SubWidthC} \quad (7\text{-}20)$$
$$\text{CropUnitY} = \text{SubHeightC} * (2 - \text{frame\_mbs\_only\_flag}) \quad (7\text{-}21)$$

The frame cropping rectangle contains luma samples with horizontal frame coordinates from CropUnitX * frame_crop_left_offset to $\text{PicWidthInSamples}_L$ − (CropUnitX * frame_crop_right_offset + 1) and vertical frame coordinates from CropUnitY * frame_crop_top_offset to (16 * FrameHeightInMbs) − (CropUnitY * frame_crop_bottom_offset + 1), inclusive. The value of frame_crop_left_offset shall be in the range of 0 to ($\text{PicWidthInSamples}_L$ / CropUnitX) − (frame_crop_right_offset + 1), inclusive; and the value of frame_crop_top_offset shall be in the range of 0 to (16 * FrameHeightInMbs / CropUnitY) − (frame_crop_bottom_offset + 1), inclusive.

When frame_cropping_flag is equal to 0, the values of frame_crop_left_offset, frame_crop_right_offset, frame_crop_top_offset, and frame_crop_bottom_offset shall be inferred to be equal to 0.

When ChromaArrayType is not equal to 0, the corresponding specified samples of the two chroma arrays are the samples having frame coordinates (x / SubWidthC, y / SubHeightC), where (x, y) are the frame coordinates of the specified luma samples.

For decoded fields, the specified samples of the decoded field are the samples that fall within the rectangle specified in frame coordinates.

vui_parameters_present_flag equal to 1 specifies that the vui_parameters( ) syntax structure as specified in Annex E is present. vui_parameters_present_flag equal to 0 specifies that the vui_parameters( ) syntax structure as specified in Annex E is not present.

7.4.2.1.1.1 Scaling list semantics

delta_scale is used to derive the j-th element of the scaling list for j in the range of 0 to sizeOfScalingList − 1, inclusive. The value of delta_scale shall be in the range of −128 to +127, inclusive.

When useDefaultScalingMatrixFlag is derived to be equal to 1, the scaling list shall be inferred to be equal to the default scaling list as specified in Table 7-2.

7.4.2.1.2 Sequence parameter set extension RBSP semantics

seq_parameter_set_id identifies the sequence parameter set associated with the sequence parameter set extension. The value of seq_parameter_set_id shall be in the range of 0 to 31, inclusive.

aux_format_idc equal to 0 indicates that there are no auxiliary coded pictures in the coded video sequence. aux_format_idc equal to 1 indicates that exactly one auxiliary coded picture is present in each access unit of the coded video sequence, and that for alpha blending purposes the decoded samples of the associated primary coded picture in each access unit should be multiplied by the interpretation sample values of the auxiliary coded picture in the access unit in the display process after output from the decoding process. aux_format_idc equal to 2 indicates that exactly one auxiliary coded picture exists in each access unit of the coded video sequence, and that for alpha blending purposes the decoded samples of the associated primary coded picture in each access unit should not be multiplied by the interpretation sample values of the auxiliary coded picture in the access unit in the display process after output from the decoding process. aux_format_idc equal to 3 indicates that exactly one auxiliary coded picture exists in each access unit of the coded video sequence, and that the usage of the auxiliary coded pictures is unspecified. The value of aux_format_idc shall be in the range of 0 to 3, inclusive. Values greater than 3 for aux_format_idc are reserved to indicate the presence of exactly one auxiliary coded picture in each access unit of the coded video sequence for purposes to be specified in the future by ITU-T | ISO/IEC. When aux_format_idc is not present, it shall be inferred to be equal to 0.

NOTE 1 – Decoders are not required to decode auxiliary coded pictures.

bit_depth_aux_minus8 specifies the bit depth of the samples of the sample array of the auxiliary coded picture. bit_depth_aux_minus8 shall be in the range of 0 to 4, inclusive.

alpha_incr_flag equal to 0 indicates that the interpretation sample value for each decoded auxiliary coded picture sample value is equal to the decoded auxiliary coded picture sample value for purposes of alpha blending. alpha_incr_flag equal to 1 indicates that, for purposes of alpha blending, after decoding the auxiliary coded picture samples, any auxiliary coded picture sample value that is greater than Min(alpha_opaque_value, alpha_transparent_value) should be increased by one to obtain the interpretation sample value for the auxiliary coded picture sample, and any auxiliary coded picture sample value that is less than or equal to Min(alpha_opaque_value, alpha_transparent_value) should be used without alteration as the interpretation sample value for the decoded auxiliary coded picture sample value.

alpha_opaque_value specifies the interpretation sample value of an auxiliary coded picture sample for which the associated luma and chroma samples of the same access unit are considered opaque for purposes of alpha blending. The number of bits used for the representation of the alpha_opaque_value syntax element is bit_depth_aux_minus8 + 9 bits.

alpha_transparent_value specifies the interpretation sample value of an auxiliary coded picture sample for which the associated luma and chroma samples of the same access unit are considered transparent for purposes of alpha blending. The number of bits used for the representation of the alpha_transparent_value syntax element is bit_depth_aux_minus8 + 9 bits.

When alpha_incr_flag is equal to 1, alpha_transparent_value shall not be equal to alpha_opaque_value and Log2(Abs(alpha_opaque_value − alpha_transparent_value)) shall have an integer value. A value of alpha_transparent_value that is equal to alpha_opaque_value indicates that the auxiliary coded picture is not intended for alpha blending purposes.

NOTE 2 – For alpha blending purposes, alpha_opaque_value may be greater than alpha_transparent_value, or it may be less than alpha_transparent_value. Interpretation sample values should be clipped to the range of alpha_opaque_value to alpha_transparent_value, inclusive.

The decoding of the sequence parameter set extension and the decoding of auxiliary coded pictures is not required for conformance with this Recommendation | International Standard.

The syntax of each coded slice of an auxiliary coded picture shall obey the same constraints as a coded slice of a redundant picture, with the following differences of constraints:

a) In regard to whether the primary coded picture is an IDR picture, the following applies:

   – If the primary coded picture is an IDR picture, the auxiliary coded slice syntax shall correspond to that of a slice having nal_unit_type equal to 5 (a slice of an IDR picture).

   – Otherwise (the primary coded picture is not an IDR picture), the auxiliary coded slice syntax shall correspond to that of a slice having nal_unit_type equal to 1 (a slice of a non-IDR picture).

b) The slices of an auxiliary coded picture (when present) shall contain all macroblocks corresponding to those of the primary coded picture.

c) redundant_pic_cnt shall be equal to 0 in all auxiliary coded slices.

The (optional) decoding process for the decoding of auxiliary coded pictures is the same as if the auxiliary coded pictures were primary coded pictures in a separate coded video stream that differs from the primary coded pictures in the current coded video stream in the following ways:

– The IDR or non-IDR status of each auxiliary coded picture shall be inferred to be the same as the IDR or non-IDR status of the primary picture in the same access unit, rather than being inferred from the value of nal_unit_type.

– The value of chroma_format_idc and the value of ChromaArrayType shall be inferred to be equal to 0 for the decoding of the auxiliary coded pictures.

– The value of bit_depth_luma_minus8 shall be inferred to be equal to bit_depth_aux_minus8 for the decoding of the auxiliary coded pictures.

NOTE 3 – Alpha blending composition is normally performed with a background picture B, a foreground picture F, and a decoded auxiliary coded picture A, all of the same size. Assume for purposes of example illustration that the chroma resolution of B and F have been upsampled to the same resolution as the luma. Denote corresponding samples of B, F and A by b, f and a, respectively. Denote luma and chroma samples by subscripts Y, Cb and Cr.

Define the variables alphaRange, alphaFwt and alphaBwt as follows:

alphaRange = Abs(alpha_opaque_value − alpha_transparent_value)

alphaFwt = Abs(a − alpha_transparent_value)

alphaBwt = Abs(a − alpha_opaque_value)

Then, in alpha blending composition, samples d of the displayed picture D may be calculated as

d_Y = (alphaFwt * f_Y + alphaBwt * b_Y + alphaRange / 2) / alphaRange

d_{Cb} = (alphaFwt * f_{Cb} + alphaBwt * b_{Cb} + alphaRange / 2) / alphaRange

d_{Cr} = (alphaFwt * f_{Cr} + alphaBwt * b_{Cr} + alphaRange / 2) / alphaRange

The samples of pictures D, F and B could also represent red, green, and blue component values (see subclause E.2.1). Here we have assumed Y, Cb and Cr component values. Each component, e.g., Y, is assumed for purposes of example illustration above to have the same bit depth in each of the pictures D, F and B. However, different components, e.g., Y and Cb, need not have the same bit depth in this example.

When aux_format_idc is equal to 1, F would be the decoded picture obtained from the decoded luma and chroma, and A would be the decoded picture obtained from the decoded auxiliary coded picture. In this case, the indicated example alpha blending composition involves multiplying the samples of F by factors obtained from the samples of A.

A picture format that is useful for editing or direct viewing, and that is commonly used, is called pre-multiplied-black video. If the foreground picture was F, then the pre-multiplied-black video S is given by

s_Y = (alphaFwt * f_Y) / alphaRange

s_{Cb} = (alphaFwt * f_{Cb}) / alphaRange

s_{Cr} = (alphaFwt * f_{Cr}) / alphaRange

Pre-multiplied-black video has the characteristic that the picture S will appear correct if displayed against a black background. For a non-black background B, the composition of the displayed picture D may be calculated as

d_Y = s_Y + (alphaBwt * b_Y + alphaRange / 2) / alphaRange

d_{Cb} = s_{Cb} + (alphaBwt * b_{Cb} + alphaRange / 2) / alphaRange

d_{Cr} = s_{Cr} + (alphaBwt * b_{Cr} + alphaRange / 2) / alphaRange

When aux_format_idc is equal to 2, S would be the decoded picture obtained from the decoded luma and chroma, and A would again be the decoded picture obtained from the decoded auxiliary coded picture. In this case, alpha blending composition does not involve multiplication of the samples of S by factors obtained from the samples of A.

additional_extension_flag equal to 0 indicates that no additional data follows within the sequence parameter set extension syntax structure prior to the RBSP trailing bits. The value of additional_extension_flag shall be equal to 0. The value of 1 for additional_extension_flag is reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore all data that follows the value of 1 for additional_extension_flag in a sequence parameter set extension NAL unit.

7.4.2.1.3 Subset sequence parameter set RBSP semantics

svc_vui_parameters_present_flag equal to 0 specifies that the syntax structure svc_vui_parameters_extension( ) is not present. svc_vui_parameters_present_flag equal to 1 specifies that the syntax structure svc_vui_parameters_extension( ) is present.

bit_equal_to_one shall be equal to 1.

mvc_vui_parameters_present_flag equal to 0 specifies that the syntax structure mvc_vui_parameters_extension( ) is not present. mvc_vui_parameters_present_flag equal to 1 specifies that the syntax structure mvc_vui_parameters_extension( ) is present.

additional_extension2_flag equal to 0 specifies that no additional_extension2_data_flag syntax elements are present in the subset sequence parameter set RBSP syntax structure. additional_extension2_flag shall be equal to 0 in bitstreams conforming to this Recommendation | International Standard. The value of 1 for additional_extension2_flag is reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore all data that follow the value 1 for additional_extension2_flag in a subset sequence parameter set NAL unit.

additional_extension2_data_flag may have any value. It shall not affect the conformance to profiles specified in Annex A, G, or H.

7.4.2.2 Picture parameter set RBSP semantics

pic_parameter_set_id identifies the picture parameter set that is referred to in the slice header. The value of pic_parameter_set_id shall be in the range of 0 to 255, inclusive.

seq_parameter_set_id refers to the active sequence parameter set. The value of seq_parameter_set_id shall be in the range of 0 to 31, inclusive.

entropy_coding_mode_flag selects the entropy decoding method to be applied for the syntax elements for which two descriptors appear in the syntax tables as follows:

– If entropy_coding_mode_flag is equal to 0, the method specified by the left descriptor in the syntax table is applied (Exp-Golomb coded, see subclause 9.1 or CAVLC, see subclause 9.2).

– Otherwise (entropy_coding_mode_flag is equal to 1), the method specified by the right descriptor in the syntax table is applied (CABAC, see subclause 9.3).

bottom_field_pic_order_in_frame_present_flag equal to 1 specifies that the syntax elements delta_pic_order_cnt_bottom (when pic_order_cnt_type is equal to 0) or delta_pic_order_cnt[1] (when pic_order_cnt_type is equal to 1), which are related to picture order counts for the bottom field of a coded frame, are present in the slice headers for coded frames as specified in subclause 7.3.3. bottom_field_pic_order_in_frame_present_flag equal to 0 specifies that the syntax elements delta_pic_order_cnt_bottom and delta_pic_order_cnt[1] are not present in the slice headers.

num_slice_groups_minus1 plus 1 specifies the number of slice groups for a picture. When num_slice_groups_minus1 is equal to 0, all slices of the picture belong to the same slice group. The allowed range of num_slice_groups_minus1 is specified in Annex A.

slice_group_map_type specifies how the mapping of slice group map units to slice groups is coded. The value of slice_group_map_type shall be in the range of 0 to 6, inclusive.

slice_group_map_type equal to 0 specifies interleaved slice groups.

slice_group_map_type equal to 1 specifies a dispersed slice group mapping.

slice_group_map_type equal to 2 specifies one or more "foreground" slice groups and a "leftover" slice group.

slice_group_map_type values equal to 3, 4, and 5 specify changing slice groups. When num_slice_groups_minus1 is not equal to 1, slice_group_map_type shall not be equal to 3, 4, or 5.

slice_group_map_type equal to 6 specifies an explicit assignment of a slice group to each slice group map unit.

Slice group map units are specified as follows:

– If frame_mbs_only_flag is equal to 0 and mb_adaptive_frame_field_flag is equal to 1 and the coded picture is a frame, the slice group map units are macroblock pair units.
– Otherwise, if frame_mbs_only_flag is equal to 1 or the coded picture is a field, the slice group map units are units of macroblocks.
– Otherwise (frame_mbs_only_flag is equal to 0 and mb_adaptive_frame_field_flag is equal to 0 and the coded picture is a frame), the slice group map units are units of two macroblocks that are vertically contiguous as in a frame macroblock pair of an MBAFF frame.

run_length_minus1[i **]** is used to specify the number of consecutive slice group map units to be assigned to the i-th slice group in raster scan order of slice group map units. The value of run_length_minus1[i] shall be in the range of 0 to PicSizeInMapUnits − 1, inclusive.

top_left[i **]** and **bottom_right[** i **]** specify the top-left and bottom-right corners of a rectangle, respectively. top_left[i] and bottom_right[i] are slice group map unit positions in a raster scan of the picture for the slice group map units. For each rectangle i, all of the following constraints shall be obeyed by the values of the syntax elements top_left[i] and bottom_right[i]:

– top_left[i] shall be less than or equal to bottom_right[i] and bottom_right[i] shall be less than PicSizeInMapUnits.
– (top_left[i] % PicWidthInMbs) shall be less than or equal to the value of (bottom_right[i] % PicWidthInMbs).

slice_group_change_direction_flag is used with slice_group_map_type to specify the refined map type when slice_group_map_type is 3, 4, or 5.

slice_group_change_rate_minus1 is used to specify the variable SliceGroupChangeRate. SliceGroupChangeRate specifies the multiple in number of slice group map units by which the size of a slice group can change from one picture to the next. The value of slice_group_change_rate_minus1 shall be in the range of 0 to PicSizeInMapUnits − 1, inclusive. The SliceGroupChangeRate variable is specified as follows:

$$\text{SliceGroupChangeRate} = \text{slice\_group\_change\_rate\_minus1} + 1 \quad (7\text{-}22)$$

pic_size_in_map_units_minus1 is used to specify the number of slice group map units in the picture. pic_size_in_map_units_minus1 shall be equal to PicSizeInMapUnits − 1.

slice_group_id[i] identifies a slice group of the i-th slice group map unit in raster scan order. The length of the slice_group_id[i] syntax element is Ceil(Log2(num_slice_groups_minus1 + 1)) bits. The value of slice_group_id[i] shall be in the range of 0 to num_slice_groups_minus1, inclusive.

num_ref_idx_l0_default_active_minus1 specifies how num_ref_idx_l0_active_minus1 is inferred for P, SP, and B slices with num_ref_idx_active_override_flag equal to 0. The value of num_ref_idx_l0_default_active_minus1 shall be in the range of 0 to 31, inclusive.

num_ref_idx_l1_default_active_minus1 specifies how num_ref_idx_l1_active_minus1 is inferred for B slices with num_ref_idx_active_override_flag equal to 0. The value of num_ref_idx_l1_default_active_minus1 shall be in the range of 0 to 31, inclusive.

weighted_pred_flag equal to 0 specifies that the default weighted prediction shall be applied to P and SP slices. weighted_pred_flag equal to 1 specifies that explicit weighted prediction shall be applied to P and SP slices.

weighted_bipred_idc equal to 0 specifies that the default weighted prediction shall be applied to B slices. weighted_bipred_idc equal to 1 specifies that explicit weighted prediction shall be applied to B slices. weighted_bipred_idc equal to 2 specifies that implicit weighted prediction shall be applied to B slices. The value of weighted_bipred_idc shall be in the range of 0 to 2, inclusive.

pic_init_qp_minus26 specifies the initial value minus 26 of SliceQP_Y for each slice. The initial value is modified at the slice layer when a non-zero value of slice_qp_delta is decoded, and is modified further when a non-zero value of mb_qp_delta is decoded at the macroblock layer. The value of pic_init_qp_minus26 shall be in the range of $-(26 + \text{QpBdOffset}_Y)$ to +25, inclusive.

pic_init_qs_minus26 specifies the initial value minus 26 of SliceQS_Y for all macroblocks in SP or SI slices. The initial value is modified at the slice layer when a non-zero value of slice_qs_delta is decoded. The value of pic_init_qs_minus26 shall be in the range of −26 to +25, inclusive.

chroma_qp_index_offset specifies the offset that shall be added to QP_Y and QS_Y for addressing the table of QP_C values for the Cb chroma component. The value of chroma_qp_index_offset shall be in the range of −12 to +12, inclusive.

deblocking_filter_control_present_flag equal to 1 specifies that a set of syntax elements controlling the characteristics of the deblocking filter is present in the slice header. deblocking_filter_control_present_flag equal to 0 specifies that the set of syntax elements controlling the characteristics of the deblocking filter is not present in the slice headers and their inferred values are in effect.

constrained_intra_pred_flag equal to 0 specifies that intra prediction allows usage of residual data and decoded samples of neighbouring macroblocks coded using Inter macroblock prediction modes for the prediction of macroblocks coded using Intra macroblock prediction modes. constrained_intra_pred_flag equal to 1 specifies constrained intra prediction, in which case prediction of macroblocks coded using Intra macroblock prediction modes only uses residual data and decoded samples from I or SI macroblock types.

redundant_pic_cnt_present_flag equal to 0 specifies that the redundant_pic_cnt syntax element is not present in slice headers, coded slice data partition B NAL units, and coded slice data partition C NAL units that refer (either directly or by association with a corresponding coded slice data partition A NAL unit) to the picture parameter set. redundant_pic_cnt_present_flag equal to 1 specifies that the redundant_pic_cnt syntax element is present in all slice headers, coded slice data partition B NAL units, and coded slice data partition C NAL units that refer (either directly or by association with a corresponding coded slice data partition A NAL unit) to the picture parameter set.

transform_8x8_mode_flag equal to 1 specifies that the 8x8 transform decoding process may be in use (see subclause 8.5). transform_8x8_mode_flag equal to 0 specifies that the 8x8 transform decoding process is not in use. When transform_8x8_mode_flag is not present, it shall be inferred to be 0.

pic_scaling_matrix_present_flag equal to 1 specifies that parameters are present to modify the scaling lists specified in the sequence parameter set. pic_scaling_matrix_present_flag equal to 0 specifies that the scaling lists used for the picture shall be inferred to be equal to those specified by the sequence parameter set. When pic_scaling_matrix_present_flag is not present, it shall be inferred to be equal to 0.

pic_scaling_list_present_flag[i] equal to 1 specifies that the scaling list syntax structure is present to specify the scaling list for index i. pic_scaling_list_present_flag[i] equal to 0 specifies that the syntax structure for scaling list i is not present in the picture parameter set and that depending on the value of seq_scaling_matrix_present_flag, the following applies:

– If seq_scaling_matrix_present_flag is equal to 0, the scaling list fall-back rule set A as specified in Table 7-2 shall be used to derive the picture-level scaling list for index i.

– Otherwise (seq_scaling_matrix_present_flag is equal to 1), the scaling list fall-back rule set B as specified in Table 7-2 shall be used to derive the picture-level scaling list for index i.

second_chroma_qp_index_offset specifies the offset that shall be added to QP_Y and QS_Y for addressing the table of QP_C values for the Cr chroma component. The value of second_chroma_qp_index_offset shall be in the range of −12 to +12, inclusive.

When second_chroma_qp_index_offset is not present, it shall be inferred to be equal to chroma_qp_index_offset.

NOTE – When ChromaArrayType is equal to 0, the values of bit_depth_chroma_minus8, chroma_qp_index_offset and second_chroma_qp_index_offset are not used in the decoding process. In particular, when separate_colour_plane_flag is equal to 1, each colour plane is decoded as a distinct monochrome picture using the luma component decoding process (except for the selection of scaling matrices), including the application of the luma quantisation parameter derivation process without application of an offset for the decoding of the pictures having colour_plane_id not equal to 0.

7.4.2.3 Supplemental enhancement information RBSP semantics

Supplemental Enhancement Information (SEI) contains information that is not necessary to decode the samples of coded pictures from VCL NAL units.

7.4.2.3.1 Supplemental enhancement information message semantics

An SEI RBSP contains one or more SEI messages. Each SEI message consists of the variables specifying the type payloadType and size payloadSize of the SEI payload. SEI payloads are specified in Annex D. The derived SEI payload size payloadSize is specified in bytes and shall be equal to the number of RBSP bytes in the SEI payload.

NOTE – The NAL unit byte sequence containing the SEI message might include one or more emulation prevention bytes (represented by emulation_prevention_three_byte syntax elements). Since the payload size of an SEI message is specified in RBSP bytes, the quantity of emulation prevention bytes is not included in the size payloadSize of an SEI payload.

ff_byte is a byte equal to 0xFF identifying a need for a longer representation of the syntax structure that it is used within.

last_payload_type_byte is the last byte of the payload type of an SEI message.

last_payload_size_byte is the last byte of the payload size of an SEI message.

7.4.2.4 Access unit delimiter RBSP semantics

The access unit delimiter may be used to indicate the type of slices present in a primary coded picture and to simplify the detection of the boundary between access units. There is no normative decoding process associated with the access unit delimiter.

primary_pic_type indicates that the slice_type values for all slices of the primary coded picture are members of the set listed in Table 7-5 for the given value of primary_pic_type.

NOTE – The value of primary_pic_type applies to the slice_type values in all slice headers of the primary coded picture, including the slice_type syntax elements in all NAL units with nal_unit_type equal to 1, 2, or 5.

Table 7-5 – Meaning of primary_pic_type

primary_pic_type	slice_type values that may be present in the primary coded picture
0	2, 7
1	0, 2, 5, 7
2	0, 1, 2, 5, 6, 7
3	4, 9
4	3, 4, 8, 9
5	2, 4, 7, 9
6	0, 2, 3, 4, 5, 7, 8, 9
7	0, 1, 2, 3, 4, 5, 6, 7, 8, 9

7.4.2.5 End of sequence RBSP semantics

The end of sequence RBSP specifies that the next subsequent access unit in the bitstream in decoding order (if any) shall be an IDR access unit. The syntax content of the SODB and RBSP for the end of sequence RBSP are empty. No normative decoding process is specified for an end of sequence RBSP.

7.4.2.6 End of stream RBSP semantics

The end of stream RBSP indicates that no additional NAL units shall be present in the bitstream that are subsequent to the end of stream RBSP in decoding order. The syntax content of the SODB and RBSP for the end of stream RBSP are empty. No normative decoding process is specified for an end of stream RBSP.

NOTE – When an end of stream NAL unit is present, the bitstream is considered to end (for purposes of the scope of this Recommendation | International Standard). In some system environments, another bitstream may follow after the bitstream that has ended, either immediately or at some time thereafter, possibly within the same communication channel. Under such circumstances, the scope of this Recommendation | International Standard applies only to the processing of each of these individual bitstreams. No requirements are specified herein regarding the transition between such bitstreams (e.g., in regard to timing, buffering operation, etc.).

7.4.2.7 Filler data RBSP semantics

The filler data RBSP contains zero or more bytes. No normative decoding process is specified for a filler data RBSP.

ff_byte is a byte. It is a requirement of bitstream conformance that the value of ff_byte shall be equal to 0xFF.

7.4.2.8 Slice layer without partitioning RBSP semantics

The slice layer without partitioning RBSP consists of a slice header and slice data.

7.4.2.9 Slice data partition RBSP semantics

7.4.2.9.1 Slice data partition A RBSP semantics

When slice data partitioning is in use, the coded data for a single slice is divided into three separate partitions. Slice data partition A contains all syntax elements of category 2.

Category 2 syntax elements include all syntax elements in the slice header and slice data syntax structures other than the syntax elements in the residual( ) syntax structure.

slice_id identifies the slice associated with the slice data partition. The value of slice_id is constrained as follows:

- If separate_colour_plane_flag is equal to 0, the following applies:
  - If arbitrary slice order is not allowed as specified in Annex A, the first slice of a coded picture, in decoding order, shall have slice_id equal to 0 and the value of slice_id shall be incremented by one for each subsequent slice of the coded picture in decoding order.
  - Otherwise (arbitrary slice order is allowed), each slice shall have a unique slice_id value within the set of slices of the coded picture.
- Otherwise (separate_colour_plane_flag is equal to 1), the following applies:
  - If arbitrary slice order is not allowed as specified in Annex A, the first slice of a coded picture having each value of colour_plane_id, in decoding order, shall have slice_id equal to 0 and the value of slice_id shall be

incremented by one for each subsequent slice of the coded picture having the same value of colour_plane_id, in decoding order.

– Otherwise (arbitrary slice order is allowed) each slice shall have a unique slice_id value within each set of slices of the coded picture that have the same value of colour_plane_id.

The range of slice_id is specified as follows:

– If MbaffFrameFlag is equal to 0, slice_id shall be in the range of 0 to PicSizeInMbs − 1, inclusive.

– Otherwise (MbaffFrameFlag is equal to 1), slice_id shall be in the range of 0 to PicSizeInMbs / 2 − 1, inclusive.

7.4.2.9.2 Slice data partition B RBSP semantics

When slice data partitioning is in use, the coded data for a single slice is divided into one to three separate partitions. Slice data partition B contains all syntax elements of category 3.

Category 3 syntax elements include all syntax elements in the residual( ) syntax structure and in syntax structures used within that syntax structure for collective macroblock types I and SI as specified in Table 7-10.

slice_id has the same semantics as specified in subclause 7.4.2.9.1.

colour_plane_id specifies the colour plane associated with the current slice RBSP when separate_colour_plane_flag is equal to 1. The value of colour_plane_id shall be in the range of 0 to 2, inclusive. colour_plane_id equal to 0, 1, and 2 correspond to the Y, Cb, and Cr planes, respectively.

NOTE – There is no dependency between the decoding processes of pictures having different values of colour_plane_id.

redundant_pic_cnt shall be equal to 0 for coded slices and coded slice data partitions belonging to the primary coded picture. The redundant_pic_cnt shall be greater than 0 for coded slices and coded slice data partitions in redundant coded pictures. When redundant_pic_cnt is not present, its value shall be inferred to be equal to 0. The value of redundant_pic_cnt shall be in the range of 0 to 127, inclusive.

The presence of a slice data partition B RBSP is specified as follows:

– If the syntax elements of a slice data partition A RBSP indicate the presence of any syntax elements of category 3 in the slice data for a slice, a slice data partition B RBSP shall be present having the same value of slice_id and redundant_pic_cnt as in the slice data partition A RBSP.

– Otherwise (the syntax elements of a slice data partition A RBSP do not indicate the presence of any syntax elements of category 3 in the slice data for a slice), no slice data partition B RBSP shall be present having the same value of slice_id and redundant_pic_cnt as in the slice data partition A RBSP.

7.4.2.9.3 Slice data partition C RBSP semantics

When slice data partitioning is in use, the coded data for a single slice is divided into three separate partitions. Slice data partition C contains all syntax elements of category 4.

Category 4 syntax elements include all syntax elements in the residual( ) syntax structure and in syntax structures used within that syntax structure for collective macroblock types P and B as specified in Table 7-10.

slice_id has the same semantics as specified in subclause 7.4.2.9.1.

colour_plane_id has the same semantics as specified in subclause 7.4.2.9.2.

redundant_pic_cnt has the same semantics as specified in subclause 7.4.2.9.2.

The presence of a slice data partition C RBSP is specified as follows:

– If the syntax elements of a slice data partition A RBSP indicate the presence of any syntax elements of category 4 in the slice data for a slice, a slice data partition C RBSP shall be present having the same value of slice_id and redundant_pic_cnt as in the slice data partition A RBSP.

– Otherwise (the syntax elements of a slice data partition A RBSP do not indicate the presence of any syntax elements of category 4 in the slice data for a slice), no slice data partition C RBSP shall be present having the same value of slice_id and redundant_pic_cnt as in the slice data partition A RBSP.

7.4.2.10 RBSP slice trailing bits semantics

cabac_zero_word is a byte-aligned sequence of two bytes equal to 0x0000.

Let NumBytesInVclNALunits be the sum of the values of NumBytesInNALunit for all VCL NAL units of a coded picture.

Let BinCountsInNALunits be the number of times that the parsing process function DecodeBin( ), specified in subclause 9.3.3.2, is invoked to decode the contents of all VCL NAL units of a coded picture. When entropy_coding_mode_flag is equal to 1, it is a requirement of bitstream conformance that BinCountsInNALunits shall not exceed (32 ÷ 3) * NumBytesInVclNALunits + (RawMbBits * PicSizeInMbs) ÷ 32.

NOTE – The constraint on the maximum number of bins resulting from decoding the contents of the slice layer NAL units can be met by inserting a number of cabac_zero_word syntax elements to increase the value of NumBytesInVclNALunits. Each cabac_zero_word is represented in a NAL unit by the three-byte sequence 0x000003 (as a result of the constraints on NAL unit contents that result in requiring inclusion of an emulation_prevention_three_byte for each cabac_zero_word).

7.4.2.11 RBSP trailing bits semantics

rbsp_stop_one_bit shall be equal to 1.

rbsp_alignment_zero_bit shall be equal to 0.

7.4.2.12 Prefix NAL unit RBSP semantics

The content of the prefix NAL unit RBSP is dependent on the value of svc_extension_flag.

7.4.2.13 Slice layer extension RBSP semantics

The content of the slice layer extension RBSP is dependent on the value of svc_extension_flag.

Coded slice extension NAL units with svc_extension_flag equal to 1 are also referred to as coded slice in scalable extension NAL units and coded slice extension NAL units with svc_extension_flag equal to 0 are also referred to as coded slice MVC extension NAL units.

7.4.3 Slice header semantics

When present, the value of the slice header syntax elements pic_parameter_set_id, frame_num, field_pic_flag, bottom_field_flag, idr_pic_id, pic_order_cnt_lsb, delta_pic_order_cnt_bottom, delta_pic_order_cnt[0], delta_pic_order_cnt[1], sp_for_switch_flag, and slice_group_change_cycle shall be the same in all slice headers of a coded picture.

first_mb_in_slice specifies the address of the first macroblock in the slice. When arbitrary slice order is not allowed as specified in Annex A, the value of first_mb_in_slice is constrained as follows:

– If separate_colour_plane_flag is equal to 0, the value of first_mb_in_slice shall not be less than the value of first_mb_in_slice for any other slice of the current picture that precedes the current slice in decoding order.

– Otherwise (separate_colour_plane_flag is equal to 1), the value of first_mb_in_slice shall not be less than the value of first_mb_in_slice for any other slice of the current picture that precedes the current slice in decoding order and has the same value of colour_plane_id.

The first macroblock address of the slice is derived as follows:

– If MbaffFrameFlag is equal to 0, first_mb_in_slice is the macroblock address of the first macroblock in the slice, and first_mb_in_slice shall be in the range of 0 to PicSizeInMbs − 1, inclusive.

– Otherwise (MbaffFrameFlag is equal to 1), first_mb_in_slice * 2 is the macroblock address of the first macroblock in the slice, which is the top macroblock of the first macroblock pair in the slice, and first_mb_in_slice shall be in the range of 0 to PicSizeInMbs / 2 − 1, inclusive.

slice_type specifies the coding type of the slice according to Table 7-6.

Table 7-6 – Name association to slice_type

slice_type	Name of slice_type
0	P (P slice)
1	B (B slice)
2	I (I slice)
3	SP (SP slice)
4	SI (SI slice)
5	P (P slice)
6	B (B slice)
7	I (I slice)
8	SP (SP slice)
9	SI (SI slice)

When slice_type has a value in the range 5..9, it is a requirement of bitstream conformance that all other slices of the current coded picture shall have a value of slice_type equal to the current value of slice_type or equal to the current value of slice_type minus 5.

NOTE 1 – Values of slice_type in the range 5..9 can be used by an encoder to indicate that all slices of a picture have the same value of (slice_type % 5). Values of slice_type in the range 5..9 are otherwise equivalent to corresponding values in the range 0..4.

When nal_unit_type is equal to 5 (IDR picture), slice_type shall be equal to 2, 4, 7, or 9.

When max_num_ref_frames is equal to 0, slice_type shall be equal to 2, 4, 7, or 9.

pic_parameter_set_id specifies the picture parameter set in use. The value of pic_parameter_set_id shall be in the range of 0 to 255, inclusive.

colour_plane_id specifies the colour plane associated with the current slice RBSP when separate_colour_plane_flag is equal to 1. The value of colour_plane_id shall be in the range of 0 to 2, inclusive. colour_plane_id equal to 0, 1, and 2 correspond to the Y, Cb, and Cr planes, respectively.

NOTE 2 – There is no dependency between the decoding processes of pictures having different values of colour_plane_id.

frame_num is used as an identifier for pictures and shall be represented by log2_max_frame_num_minus4 + 4 bits in the bitstream. frame_num is constrained as follows:

The variable PrevRefFrameNum is derived as follows:

– If the current picture is an IDR picture, PrevRefFrameNum is set equal to 0.

– Otherwise (the current picture is not an IDR picture), PrevRefFrameNum is set as follows:

  – If the decoding process for gaps in frame_num specified in subclause 8.2.5.2 was invoked by the decoding process for an access unit that contained a non-reference picture that followed the previous access unit in decoding order that contained a reference picture, PrevRefFrameNum is set equal to the value of frame_num for the last of the "non-existing" reference frames inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2.

  – Otherwise, PrevRefFrameNum is set equal to the value of frame_num for the previous access unit in decoding order that contained a reference picture.

The value of frame_num is constrained as follows:

– If the current picture is an IDR picture, frame_num shall be equal to 0.

– Otherwise (the current picture is not an IDR picture), referring to the primary coded picture in the previous access unit in decoding order that contains a reference picture as the preceding reference picture, the value of frame_num for the current picture shall not be equal to PrevRefFrameNum unless all of the following three conditions are true:

  a) The current picture and the preceding reference picture belong to consecutive access units in decoding order.

  b) The current picture and the preceding reference picture are reference fields having opposite parity.

  c) One or more of the following conditions is true:

    – The preceding reference picture is an IDR picture,

– The preceding reference picture includes a memory_management_control_operation syntax element equal to 5,

NOTE 3 – When the preceding reference picture includes a memory_management_control_operation syntax element equal to 5, PrevRefFrameNum is equal to 0.

– There is a primary coded picture that precedes the preceding reference picture and the primary coded picture that precedes the preceding reference picture does not have frame_num equal to PrevRefFrameNum,

– There is a primary coded picture that precedes the preceding reference picture and the primary coded picture that precedes the preceding reference picture is not a reference picture.

When the value of frame_num is not equal to PrevRefFrameNum, it is a requirement of bitstream conformance that the following constraints shall be obeyed:

a) There shall not be any previous field or frame in decoding order that is currently marked as "used for short-term reference" that has a value of frame_num equal to any value taken on by the variable UnusedShortTermFrameNum in the following:

```
UnusedShortTermFrameNum = ( PrevRefFrameNum + 1 ) % MaxFrameNum
while( UnusedShortTermFrameNum != frame_num )                                    (7-23)
    UnusedShortTermFrameNum = ( UnusedShortTermFrameNum + 1 ) % MaxFrameNum
```

b) The value of frame_num is constrained as follows:

– If gaps_in_frame_num_value_allowed_flag is equal to 0, the value of frame_num for the current picture shall be equal to (PrevRefFrameNum + 1) % MaxFrameNum.

– Otherwise (gaps_in_frame_num_value_allowed_flag is equal to 1), the following applies:

  – If frame_num is greater than PrevRefFrameNum, there shall not be any non-reference pictures in the bitstream that follow the previous reference picture and precede the current picture in decoding order in which either of the following conditions is true:

    – The value of frame_num for the non-reference picture is less than PrevRefFrameNum,

    – The value of frame_num for the non-reference picture is greater than the value of frame_num for the current picture.

  – Otherwise (frame_num is less than PrevRefFrameNum), there shall not be any non-reference pictures in the bitstream that follow the previous reference picture and precede the current picture in decoding order in which both of the following conditions are true:

    – The value of frame_num for the non-reference picture is less than PrevRefFrameNum,

    – The value of frame_num for the non-reference picture is greater than the value of frame_num for the current picture.

A picture including a memory_management_control_operation equal to 5 shall have frame_num constraints as described above and, after the decoding of the current picture and the processing of the memory management control operations, the picture shall be inferred to have had frame_num equal to 0 for all subsequent use in the decoding process, except as specified in subclause 7.4.1.2.4.

NOTE 4 – When the primary coded picture is not an IDR picture and does not contain memory_management_control_operation syntax element equal to 5, the value of frame_num of a corresponding redundant coded picture is the same as the value of frame_num in the primary coded picture. Alternatively, the redundant coded picture includes a memory_management_control_operation syntax element equal to 5 and the corresponding primary coded picture is an IDR picture.

field_pic_flag equal to 1 specifies that the slice is a slice of a coded field. field_pic_flag equal to 0 specifies that the slice is a slice of a coded frame. When field_pic_flag is not present it shall be inferred to be equal to 0.

The variable MbaffFrameFlag is derived as

$$\text{MbaffFrameFlag} = (\text{ mb\_adaptive\_frame\_field\_flag } \&\& \text{ !field\_pic\_flag }) \quad (7\text{-}24)$$

The variable for the picture height in units of macroblocks is derived as

$$\text{PicHeightInMbs} = \text{FrameHeightInMbs} / (1 + \text{field\_pic\_flag}) \quad (7\text{-}25)$$

The variable for picture height for the luma component is derived as

$$\text{PicHeightInSamples}_L = \text{PicHeightInMbs} * 16 \quad (7\text{-}26)$$

The variable for picture height for the chroma component is derived as

$$\text{PicHeightInSamples}_C = \text{PicHeightInMbs} * \text{MbHeightC} \tag{7-27}$$

The variable PicSizeInMbs for the current picture is derived as

$$\text{PicSizeInMbs} = \text{PicWidthInMbs} * \text{PicHeightInMbs} \tag{7-28}$$

The variable MaxPicNum is derived as follows:

– If field_pic_flag is equal to 0, MaxPicNum is set equal to MaxFrameNum.

– Otherwise (field_pic_flag is equal to 1), MaxPicNum is set equal to 2*MaxFrameNum.

The variable CurrPicNum is derived as follows:

– If field_pic_flag is equal to 0, CurrPicNum is set equal to frame_num.

– Otherwise (field_pic_flag is equal to 1), CurrPicNum is set equal to 2 * frame_num + 1.

bottom_field_flag equal to 1 specifies that the slice is part of a coded bottom field. bottom_field_flag equal to 0 specifies that the picture is a coded top field. When this syntax element is not present for the current slice, it shall be inferred to be equal to 0.

idr_pic_id identifies an IDR picture. The values of idr_pic_id in all the slices of an IDR picture shall remain unchanged. When two consecutive access units in decoding order are both IDR access units, the value of idr_pic_id in the slices of the first such IDR access unit shall differ from the idr_pic_id in the second such IDR access unit. The value of idr_pic_id shall be in the range of 0 to 65535, inclusive.

NOTE 5 – It is not prohibited for multiple IDR pictures in a bitstream to have the same value of idr_pic_id unless such pictures occur in two consecutive access units in decoding order.

pic_order_cnt_lsb specifies the picture order count modulo MaxPicOrderCntLsb for the top field of a coded frame or for a coded field. The length of the pic_order_cnt_lsb syntax element is log2_max_pic_order_cnt_lsb_minus4 + 4 bits. The value of the pic_order_cnt_lsb shall be in the range of 0 to MaxPicOrderCntLsb − 1, inclusive.

delta_pic_order_cnt_bottom specifies the picture order count difference between the bottom field and the top field of a coded frame as follows:

– If the current picture includes a memory_management_control_operation equal to 5, the value of delta_pic_order_cnt_bottom shall be in the range of (1 − MaxPicOrderCntLsb) to $2^{31} - 1$, inclusive.

– Otherwise (the current picture does not include a memory_management_control_operation equal to 5), the value of delta_pic_order_cnt_bottom shall be in the range of $-2^{31} + 1$ to $2^{31} - 1$, inclusive.

When this syntax element is not present in the bitstream for the current slice, it shall be inferred to be equal to 0.

delta_pic_order_cnt[0 **]** specifies the picture order count difference from the expected picture order count for the top field of a coded frame or for a coded field as specified in subclause 8.2.1. The value of delta_pic_order_cnt[0] shall be in the range of $-2^{31} + 1$ to $2^{31} - 1$, inclusive. When this syntax element is not present in the bitstream for the current slice, it shall be inferred to be equal to 0.

delta_pic_order_cnt[1 **]** specifies the picture order count difference from the expected picture order count for the bottom field of a coded frame specified in subclause 8.2.1. The value of delta_pic_order_cnt[1] shall be in the range of $-2^{31} + 1$ to $2^{31} - 1$, inclusive. When this syntax element is not present in the bitstream for the current slice, it shall be inferred to be equal to 0.

redundant_pic_cnt shall be equal to 0 for slices and slice data partitions belonging to the primary coded picture. The value of redundant_pic_cnt shall be greater than 0 for coded slices or coded slice data partitions of a redundant coded picture. When redundant_pic_cnt is not present in the bitstream, its value shall be inferred to be equal to 0. The value of redundant_pic_cnt shall be in the range of 0 to 127, inclusive.

NOTE 6 – Any area of the decoded primary picture and the corresponding area that would result from application of the decoding process specified in clause 8 for any redundant picture in the same access unit should be visually similar in appearance.

The value of pic_parameter_set_id in a coded slice or coded slice data partition of a redundant coded picture shall be such that the value of bottom_field_pic_order_in_frame_present_flag in the picture parameter set in use in a redundant coded picture is equal to the value of bottom_field_pic_order_in_frame_present_flag in the picture parameter set in use in the corresponding primary coded picture.

When present in the primary coded picture and any redundant coded picture, the following syntax elements shall have the same value: field_pic_flag, bottom_field_flag, and idr_pic_id.

When the value of nal_ref_idc in one VCL NAL unit of an access unit is equal to 0, the value of nal_ref_idc in all other VCL NAL units of the same access unit shall be equal to 0.

NOTE 7 – The above constraint also has the following implications. If the value of nal_ref_idc for the VCL NAL units of the primary coded picture is equal to 0, the value of nal_ref_idc for the VCL NAL units of any corresponding redundant coded picture are equal to 0; otherwise (the value of nal_ref_idc for the VCL NAL units of the primary coded picture is greater than 0), the value of nal_ref_idc for the VCL NAL units of any corresponding redundant coded picture are also greater than 0.

The marking status of reference pictures and the value of frame_num after the decoded reference picture marking process as specified in subclause 8.2.5 is invoked for the primary coded picture or any redundant coded picture of the same access unit shall be identical regardless whether the primary coded picture or any redundant coded picture (instead of the primary coded picture) of the access unit would be decoded.

NOTE 8 – The above constraint also has the following implications.

When the primary coded picture is an IDR picture and a redundant coded picture corresponding to the primary coded picture is an IDR picture, the contents of the dec_ref_pic_marking( ) syntax structure must be identical in all slice headers of the primary coded picture and the redundant coded picture corresponding to the primary coded picture.

When the primary coded picture is an IDR picture and a redundant coded picture corresponding to the primary coded picture is not an IDR picture, all slice headers of the redundant picture must contain a dec_ref_pic_marking syntax( ) structure including a memory_management_control_operation syntax element equal to 5, and the following applies:

- If the value of long_term_reference_flag in the primary coded picture is equal to 0, the dec_ref_pic_marking syntax structure of the redundant coded picture must not include a memory_management_control_operation syntax element equal to 6.
- Otherwise (the value of long_term_reference_flag in the primary coded picture is equal to 1), the dec_ref_pic_marking syntax structure of the redundant coded picture must include memory_management_control_operation syntax elements equal to 5, 4, and 6 in decoding order, and the value of max_long_term_frame_idx_plus1 must be equal to 1, and the value of long_term_frame_idx must be equal to 0.

The values of TopFieldOrderCnt and BottomFieldOrderCnt (if applicable) that result after completion of the decoding process for any redundant coded picture or the primary coded picture of the same access unit shall be identical regardless whether the primary coded picture or any redundant coded picture (instead of the primary coded picture) of the access unit would be decoded.

There is no required decoding process for a coded slice or coded slice data partition of a redundant coded picture. When the redundant_pic_cnt in the slice header of a coded slice is greater than 0, the decoder may discard the coded slice. However, a coded slice or coded slice data partition of any redundant coded picture shall obey the same constraints as a coded slice or coded slice data partition of a primary picture.

NOTE 9 – When some of the samples in the decoded primary picture cannot be correctly decoded due to errors or losses in transmission of the sequence and one or more coded slices of a redundant coded picture can be correctly decoded, the decoder should replace the samples of the decoded primary picture with the corresponding samples of the decoded slice or decoded slices of the redundant coded picture. When slices of more than one redundant coded picture cover the relevant region of the primary coded picture, the slice or slices of the redundant coded picture having the lowest value of redundant_pic_cnt should be used.

Slices and slice data partitions having the same value of redundant_pic_cnt belong to the same coded picture. If the value of redundant_pic_cnt is equal to 0, they belong to the primary coded picture; otherwise (the value of redundant_pic_cnt is greater than 0), they belong to the same redundant coded picture. Decoded slices within the same redundant coded picture need not cover the entire picture area and shall not overlap.

direct_spatial_mv_pred_flag specifies the method used in the decoding process to derive motion vectors and reference indices for inter prediction as follows:

- If direct_spatial_mv_pred_flag is equal to 1, the derivation process for luma motion vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8 in subclause 8.4.1.2 shall use spatial direct mode prediction as specified in subclause 8.4.1.2.2.
- Otherwise (direct_spatial_mv_pred_flag is equal to 0), the derivation process for luma motion vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8 in subclause 8.4.1.2 shall use temporal direct mode prediction as specified in subclause 8.4.1.2.3.

num_ref_idx_active_override_flag equal to 1 specifies that the syntax element num_ref_idx_l0_active_minus1 is present for P, SP, and B slices and that the syntax element num_ref_idx_l1_active_minus1 is present for B slices. num_ref_idx_active_override_flag equal to 0 specifies that the syntax elements num_ref_idx_l0_active_minus1 and num_ref_idx_l1_active_minus1 are not present.

When the current slice is a P, SP, or B slice and field_pic_flag is equal to 0 and the value of num_ref_idx_l0_default_active_minus1 in the picture parameter set exceeds 15, num_ref_idx_active_override_flag shall be equal to 1.

When the current slice is a B slice and field_pic_flag is equal to 0 and the value of num_ref_idx_l1_default_active_minus1 in the picture parameter set exceeds 15, num_ref_idx_active_override_flag shall be equal to 1.

num_ref_idx_l0_active_minus1 specifies the maximum reference index for reference picture list 0 that shall be used to decode the slice.

When the current slice is a P, SP, or B slice and num_ref_idx_l0_active_minus1 is not present, num_ref_idx_l0_active_minus1 shall be inferred to be equal to num_ref_idx_l0_default_active_minus1.

The range of num_ref_idx_l0_active_minus1 is specified as follows:

– If field_pic_flag is equal to 0, num_ref_idx_l0_active_minus1 shall be in the range of 0 to 15, inclusive. When MbaffFrameFlag is equal to 1, num_ref_idx_l0_active_minus1 is the maximum index value for the decoding of frame macroblocks and 2 * num_ref_idx_l0_active_minus1 + 1 is the maximum index value for the decoding of field macroblocks.

– Otherwise (field_pic_flag is equal to 1), num_ref_idx_l0_active_minus1 shall be in the range of 0 to 31, inclusive.

num_ref_idx_l1_active_minus1 specifies the maximum reference index for reference picture list 1 that shall be used to decode the slice.

When the current slice is a B slice and num_ref_idx_l1_active_minus1 is not present, num_ref_idx_l1_active_minus1 shall be inferred to be equal to num_ref_idx_l1_default_active_minus1.

The range of num_ref_idx_l1_active_minus1 is constrained as specified in the semantics for num_ref_idx_l0_active_minus1 with l0 and list 0 replaced by l1 and list 1, respectively.

cabac_init_idc specifies the index for determining the initialisation table used in the initialisation process for context variables. The value of cabac_init_idc shall be in the range of 0 to 2, inclusive.

slice_qp_delta specifies the initial value of QP_Y to be used for all the macroblocks in the slice until modified by the value of mb_qp_delta in the macroblock layer. The initial QP_Y quantisation parameter for the slice is computed as

$$\text{SliceQP}_Y = 26 + \text{pic\_init\_qp\_minus26} + \text{slice\_qp\_delta} \quad (7\text{-}29)$$

The value of slice_qp_delta shall be limited such that SliceQP_Y is in the range of $-\text{QpBdOffset}_Y$ to +51, inclusive.

sp_for_switch_flag specifies the decoding process to be used to decode P macroblocks in an SP slice as follows:

– If sp_for_switch_flag is equal to 0, the P macroblocks in the SP slice shall be decoded using the SP decoding process for non-switching pictures as specified in subclause 8.6.1.

– Otherwise (sp_for_switch_flag is equal to 1), the P macroblocks in the SP slice shall be decoded using the SP and SI decoding process for switching pictures as specified in subclause 8.6.2.

slice_qs_delta specifies the value of QS_Y for all the macroblocks in SP and SI slices. The QS_Y quantisation parameter for the slice is computed as

$$QS_Y = 26 + \text{pic\_init\_qs\_minus26} + \text{slice\_qs\_delta} \quad (7\text{-}30)$$

The value of slice_qs_delta shall be limited such that QS_Y is in the range of 0 to 51, inclusive. This value of QS_Y is used for the decoding of all macroblocks in SI slices with mb_type equal to SI and all macroblocks in SP slices that are coded in an Inter macroblock prediction mode.

disable_deblocking_filter_idc specifies whether the operation of the deblocking filter shall be disabled across some block edges of the slice and specifies for which edges the filtering is disabled. When disable_deblocking_filter_idc is not present in the slice header, the value of disable_deblocking_filter_idc shall be inferred to be equal to 0.

The value of disable_deblocking_filter_idc shall be in the range of 0 to 2, inclusive.

slice_alpha_c0_offset_div2 specifies the offset used in accessing the α and t_{C0} deblocking filter tables for filtering operations controlled by the macroblocks within the slice. From this value, the offset that shall be applied when addressing these tables shall be computed as

$$\text{FilterOffsetA} = \text{slice\_alpha\_c0\_offset\_div2} \ll 1 \quad (7\text{-}31)$$

The value of slice_alpha_c0_offset_div2 shall be in the range of −6 to +6, inclusive. When slice_alpha_c0_offset_div2 is not present in the slice header, the value of slice_alpha_c0_offset_div2 shall be inferred to be equal to 0.

slice_beta_offset_div2 specifies the offset used in accessing the β deblocking filter table for filtering operations controlled by the macroblocks within the slice. From this value, the offset that is applied when addressing the β table of the deblocking filter shall be computed as

$$\text{FilterOffsetB} = \text{slice\_beta\_offset\_div2} \ll 1 \tag{7-32}$$

The value of slice_beta_offset_div2 shall be in the range of −6 to +6, inclusive. When slice_beta_offset_div2 is not present in the slice header the value of slice_beta_offset_div2 shall be inferred to be equal to 0.

slice_group_change_cycle is used to derive the number of slice group map units in slice group 0 when slice_group_map_type is equal to 3, 4, or 5, as specified by

$$\text{MapUnitsInSliceGroup0} = \text{Min}(\ \text{slice\_group\_change\_cycle} * \text{SliceGroupChangeRate},\ \text{PicSizeInMapUnits}\ ) \tag{7-33}$$

The value of slice_group_change_cycle is represented in the bitstream by the following number of bits

$$\text{Ceil}(\ \text{Log2}(\ \text{PicSizeInMapUnits} \div \text{SliceGroupChangeRate} + 1\ )\ ) \tag{7-34}$$

The value of slice_group_change_cycle shall be in the range of 0 to Ceil(PicSizeInMapUnits÷SliceGroupChangeRate), inclusive.

7.4.3.1 Reference picture list modification semantics

The syntax elements modification_of_pic_nums_idc, abs_diff_pic_num_minus1, and long_term_pic_num specify the change from the initial reference picture lists to the reference picture lists to be used for decoding the slice.

ref_pic_list_modification_flag_l0 equal to 1 specifies that the syntax element modification_of_pic_nums_idc is present for specifying reference picture list 0. ref_pic_list_modification_flag_l0 equal to 0 specifies that this syntax element is not present.

When ref_pic_list_modification_flag_l0 is equal to 1, the number of times that modification_of_pic_nums_idc is not equal to 3 following ref_pic_list_modification_flag_l0 shall not exceed num_ref_idx_l0_active_minus1 + 1.

When RefPicList0[num_ref_idx_l0_active_minus1] in the initial reference picture list produced as specified in subclause 8.2.4.2 is equal to "no reference picture", ref_pic_list_modification_flag_l0 shall be equal to 1 and modification_of_pic_nums_idc shall not be equal to 3 until RefPicList0[num_ref_idx_l0_active_minus1] in the modified list produced as specified in subclause 8.2.4.3 is not equal to "no reference picture".

ref_pic_list_modification_flag_l1 equal to 1 specifies that the syntax element modification_of_pic_nums_idc is present for specifying reference picture list 1. ref_pic_list_modification_flag_l1 equal to 0 specifies that this syntax element is not present.

When ref_pic_list_modification_flag_l1 is equal to 1, the number of times that modification_of_pic_nums_idc is not equal to 3 following ref_pic_list_modification_flag_l1 shall not exceed num_ref_idx_l1_active_minus1 + 1.

When decoding a slice with slice_type equal to 1 or 6 and RefPicList1[num_ref_idx_l1_active_minus1] in the initial reference picture list produced as specified in subclause 8.2.4.2 is equal to "no reference picture", ref_pic_list_modification_flag_l1 shall be equal to 1 and modification_of_pic_nums_idc shall not be equal to 3 until RefPicList1[num_ref_idx_l1_active_minus1] in the modified list produced as specified in subclause 8.2.4.3 is not equal to "no reference picture".

modification_of_pic_nums_idc together with abs_diff_pic_num_minus1 or long_term_pic_num specifies which of the reference pictures are re-mapped. The values of modification_of_pic_nums_idc are specified in Table 7-7. The value of the first modification_of_pic_nums_idc that follows immediately after ref_pic_list_modification_flag_l0 or ref_pic_list_modification_flag_l1 shall not be equal to 3.

Table 7-7 – modification_of_pic_nums_idc operations for modification of reference picture lists

modification_of_pic_nums_idc	modification specified
0	abs_diff_pic_num_minus1 is present and corresponds to a difference to subtract from a picture number prediction value
1	abs_diff_pic_num_minus1 is present and corresponds to a difference to add to a picture number prediction value
2	long_term_pic_num is present and specifies the long-term picture number for a reference picture
3	End loop for modification of the initial reference picture list

abs_diff_pic_num_minus1 plus 1 specifies the absolute difference between the picture number of the picture being moved to the current index in the list and the picture number prediction value. abs_diff_pic_num_minus1 shall be in the range of 0 to MaxPicNum − 1. The allowed values of abs_diff_pic_num_minus1 are further restricted as specified in subclause 8.2.4.3.1.

long_term_pic_num specifies the long-term picture number of the picture being moved to the current index in the list. When decoding a coded frame, long_term_pic_num shall be equal to a LongTermPicNum assigned to one of the reference frames or complementary reference field pairs marked as "used for long-term reference". When decoding a coded field, long_term_pic_num shall be equal to a LongTermPicNum assigned to one of the reference fields marked as "used for long-term reference".

7.4.3.2 Prediction weight table semantics

luma_log2_weight_denom is the base 2 logarithm of the denominator for all luma weighting factors. The value of luma_log2_weight_denom shall be in the range of 0 to 7, inclusive.

chroma_log2_weight_denom is the base 2 logarithm of the denominator for all chroma weighting factors. The value of chroma_log2_weight_denom shall be in the range of 0 to 7, inclusive.

luma_weight_l0_flag equal to 1 specifies that weighting factors for the luma component of list 0 prediction are present. luma_weight_l0_flag equal to 0 specifies that these weighting factors are not present.

luma_weight_l0[i] is the weighting factor applied to the luma prediction value for list 0 prediction using RefPicList0[i]. When luma_weight_l0_flag is equal to 1, the value of luma_weight_l0[i] shall be in the range of −128 to 127, inclusive. When luma_weight_l0_flag is equal to 0, luma_weight_l0[i] shall be inferred to be equal to $2^{\text{luma\_log2\_weight\_denom}}$ for RefPicList0[i].

luma_offset_l0[i] is the additive offset applied to the luma prediction value for list 0 prediction using RefPicList0[i]. The value of luma_offset_l0[i] shall be in the range of −128 to 127, inclusive. When luma_weight_l0_flag is equal to 0, luma_offset_l0[i] shall be inferred as equal to 0 for RefPicList0[i].

chroma_weight_l0_flag equal to 1 specifies that weighting factors for the chroma prediction values of list 0 prediction are present. chroma_weight_l0_flag equal to 0 specifies that these weighting factors are not present.

chroma_weight_l0[i][j] is the weighting factor applied to the chroma prediction values for list 0 prediction using RefPicList0[i] with j equal to 0 for Cb and j equal to 1 for Cr. When chroma_weight_l0_flag is equal to 1, the value of chroma_weight_l0[i][j] shall be in the range of −128 to 127, inclusive. When chroma_weight_l0_flag is equal to 0, chroma_weight_l0[i][j] shall be inferred to be equal to $2^{\text{chroma\_log2\_weight\_denom}}$ for RefPicList0[i].

chroma_offset_l0[i][j] is the additive offset applied to the chroma prediction values for list 0 prediction using RefPicList0[i] with j equal to 0 for Cb and j equal to 1 for Cr. The value of chroma_offset_l0[i][j] shall be in the range of −128 to 127, inclusive. When chroma_weight_l0_flag is equal to 0, chroma_offset_l0[i][j] shall be inferred to be equal to 0 for RefPicList0[i].

luma_weight_l1_flag, luma_weight_l1, luma_offset_l1, chroma_weight_l1_flag, chroma_weight_l1, chroma_offset_l1 have the same semantics as luma_weight_l0_flag, luma_weight_l0, luma_offset_l0, chroma_weight_l0_flag, chroma_weight_l0, chroma_offset_l0, respectively, with l0, list 0, and List0 replaced by l1, list 1, and List1, respectively.

7.4.3.3 Decoded reference picture marking semantics

The syntax elements no_output_of_prior_pics_flag, long_term_reference_flag, adaptive_ref_pic_marking_mode_flag, memory_management_control_operation, difference_of_pic_nums_minus1, long_term_frame_idx, long_term_pic_num, and max_long_term_frame_idx_plus1 specify marking of the reference pictures.

The marking of a reference picture can be "unused for reference", "used for short-term reference", or "used for long-term reference", but only one among these three. When a reference picture is referred to as being marked as "used for reference", this collectively refers to the picture being marked as "used for short-term reference" or "used for long-term reference" (but not both). A reference picture that is marked as "used for short-term reference" is referred to as a short-term reference picture. A reference picture that is marked as "used for long-term reference" is referred to as a long-term reference picture.

The content of the decoded reference picture marking syntax structure shall be the same in all slice headers of the primary coded picture. When one or more redundant coded pictures are present, the content of the decoded reference picture marking syntax structure shall be the same in all slice headers of a redundant coded picture with a particular value of redundant_pic_cnt.

NOTE 1 – It is not required that the content of the decoded reference picture marking syntax structure in a redundant coded picture with a particular value of redundant_pic_cnt is identical to the content of the decoded reference picture marking syntax structure in the corresponding primary coded picture or a redundant coded picture with a different value of redundant_pic_cnt. However, as specified in subclause 7.4.3, the content of the decoded reference picture marking syntax structure in a redundant coded picture is constrained in the way that the marking status of reference pictures and the value of frame_num after the decoded reference picture marking process in subclause 8.2.5 must be identical regardless whether the primary coded picture or any redundant coded picture of the access unit would be decoded.

The syntax category of the decoded reference picture marking syntax structure shall be inferred as follows:

– If the decoded reference picture marking syntax structure is in a slice header, the syntax category of the decoded reference picture marking syntax structure is inferred to be equal to 2.

– Otherwise (the decoded reference picture marking syntax structure is in a decoded reference picture marking repetition SEI message as specified in Annex D), the syntax category of the decoded reference picture marking syntax structure is inferred to be equal to 5.

no_output_of_prior_pics_flag specifies how the previously-decoded pictures in the decoded picture buffer are treated after decoding of an IDR picture. See Annex C. When the IDR picture is the first IDR picture in the bitstream, the value of no_output_of_prior_pics_flag has no effect on the decoding process. When the IDR picture is not the first IDR picture in the bitstream and the value of PicWidthInMbs, FrameHeightInMbs, or max_dec_frame_buffering derived from the active sequence parameter set is different from the value of PicWidthInMbs, FrameHeightInMbs, or max_dec_frame_buffering derived from the sequence parameter set active for the preceding picture, no_output_of_prior_pics_flag equal to 1 may (but should not) be inferred by the decoder, regardless of the actual value of no_output_of_prior_pics_flag.

long_term_reference_flag equal to 0 specifies that the MaxLongTermFrameIdx variable is set equal to "no long-term frame indices" and that the IDR picture is marked as "used for short-term reference". long_term_reference_flag equal to 1 specifies that the MaxLongTermFrameIdx variable is set equal to 0 and that the current IDR picture is marked "used for long-term reference" and is assigned LongTermFrameIdx equal to 0. When max_num_ref_frames is equal to 0, long_term_reference_flag shall be equal to 0.

adaptive_ref_pic_marking_mode_flag selects the reference picture marking mode of the currently decoded picture as specified in Table 7-8. adaptive_ref_pic_marking_mode_flag shall be equal to 1 when the number of frames, complementary field pairs, and non-paired fields that are currently marked as "used for long-term reference" is equal to Max(max_num_ref_frames, 1).

Table 7-8 – Interpretation of adaptive_ref_pic_marking_mode_flag

adaptive_ref_pic_marking_mode_flag	Reference picture marking mode specified
0	Sliding window reference picture marking mode: A marking mode providing a first-in first-out mechanism for short-term reference pictures.
1	Adaptive reference picture marking mode: A reference picture marking mode providing syntax elements to specify marking of reference pictures as "unused for reference" and to assign long-term frame indices.

memory_management_control_operation specifies a control operation to be applied to affect the reference picture marking. The memory_management_control_operation syntax element is followed by data necessary for the operation specified by the value of memory_management_control_operation. The values and control operations associated with memory_management_control_operation are specified in Table 7-9. The memory_management_control_operation syntax elements are processed by the decoding process in the order in which they appear in the slice header, and the semantics constraints expressed for each memory_management_control_operation apply at the specific position in that order at which that individual memory_management_control_operation is processed.

For interpretation of memory_management_control_operation, the term reference picture is interpreted as follows:

– If the current picture is a frame, the term reference picture refers either to a reference frame or a complementary reference field pair.

– Otherwise (the current picture is a field), the term reference picture refers either to a reference field or a field of a reference frame.

memory_management_control_operation shall not be equal to 1 in a slice header unless the specified reference picture is marked as "used for short-term reference" when the memory_management_control_operation is processed by the decoding process.

memory_management_control_operation shall not be equal to 2 in a slice header unless the specified long-term picture number refers to a reference picture that is marked as "used for long-term reference" when the memory_management_control_operation is processed by the decoding process.

memory_management_control_operation shall not be equal to 3 in a slice header unless the specified reference picture is marked as "used for short-term reference" when the memory_management_control_operation is processed by the decoding process.

memory_management_control_operation shall not be equal to 3 or 6 if the value of the variable MaxLongTermFrameIdx is equal to "no long-term frame indices" when the memory_management_control_operation is processed by the decoding process.

Not more than one memory_management_control_operation equal to 4 shall be present in a slice header.

Not more than one memory_management_control_operation equal to 5 shall be present in a slice header.

Not more than one memory_management_control_operation equal to 6 shall be present in a slice header.

memory_management_control_operation shall not be equal to 5 in a slice header unless no memory_management_control_operation in the range of 1 to 3 is present in the same decoded reference picture marking syntax structure.

A memory_management_control_operation equal to 5 shall not follow a memory_management_control_operation equal to 6 in the same slice header.

When a memory_management_control_operation equal to 6 is present, any memory_management_control_operation equal to 2, 3, or 4 that follows the memory_management_control_operation equal to 6 within the same slice header shall not specify the current picture to be marked as "unused for reference".

NOTE 2 – These constraints prohibit any combination of multiple memory_management_control_operation syntax elements that would specify the current picture to be marked as "unused for reference". However, some other combinations of memory_management_control_operation syntax elements are permitted that may affect the marking status of other reference pictures more than once in the same slice header. In particular, it is permitted for a memory_management_control_operation equal to 3 that specifies a long-term frame index to be assigned to a particular short-term reference picture to be followed in the same slice header by a memory_management_control_operation equal to 2, 3, 4 or 6 that specifies the same reference picture to subsequently be marked as "unused for reference".

Table 7-9 – Memory management control operation (memory_management_control_operation) values

memory_management_control_operation	Memory Management Control Operation
0	End memory_management_control_operation syntax element loop
1	Mark a short-term reference picture as "unused for reference"
2	Mark a long-term reference picture as "unused for reference"
3	Mark a short-term reference picture as "used for long-term reference" and assign a long-term frame index to it
4	Specify the maximum long-term frame index and mark all long-term reference pictures having long-term frame indices greater than the maximum value as "unused for reference"
5	Mark all reference pictures as "unused for reference" and set the MaxLongTermFrameIdx variable to "no long-term frame indices"
6	Mark the current picture as "used for long-term reference" and assign a long-term frame index to it

When decoding a field and a memory_management_control_operation command equal to 3 is present that assigns a long-term frame index to a field that is part of a short-term reference frame or part of a complementary reference field pair, another memory_management_control_operation command (equal to 3 or 6) to assign the same long-term frame index to the other field of the same frame or complementary reference field pair shall be present in the same decoded reference picture marking syntax structure.

NOTE 3 – The above requirement must be fulfilled even when the field referred to by the memory_management_control_operation equal to 3 is subsequently marked as "unused for reference" (for example when a memory_management_control_operation equal to 2 is present in the same slice header that causes the field to be marked as "unused for reference").

NOTE 4 – The above requirement has the following implications:

- When a memory_management_control_operation equal to 3 is present that assigns a long-term frame index to a field that is part of a reference frame or complementary reference field pair with both fields marked as "used for short-term reference" (when processing the memory_management_control_operation equal to 3), another memory_management_control_operation equal to 3 must also be present in the same decoded reference picture marking syntax structure that assigns the same long-term frame index to the other field of the reference frame or complementary reference field pair.
- When the current picture is the second field (in decoding order) of a complementary reference field pair and a memory_management_control_operation equal to 3 is present in the decoded reference picture marking syntax structure of the current picture that assigns a long-term frame index to the first field (in decoding order) of the complementary reference field pair, a memory_management_control_operation equal to 6 must be present in the same decoded reference picture marking syntax structure that assigns the same long-term frame index to the second field of the complementary reference field pair.

When the first field (in decoding order) of a complementary reference field pair included a long_term_reference_flag equal to 1 or a memory_management_control_operation command equal to 6, the decoded reference picture marking syntax structure for the second field of the complementary reference field pair shall contain a memory_management_control_operation command equal to 6 that assigns the same long-term frame index to the second field.

NOTE 5 – The above requirement must be fulfilled even when the first field of the complementary reference field pair is subsequently marked as "unused for reference" (for example, when a memory_management_control_operation equal to 2 is present in the slice header of the second field that causes the first field to be marked as "unused for reference").

When the second field (in decoding order) of a complementary reference field pair includes a memory_management_control_operation command equal to 6 that assigns a long-term frame index to this field and the first field of the complementary reference field pair is marked as "used for short-term reference" when the memory_management_control_operation command equal to 6 is processed by the decoding process, the decoded reference picture marking syntax structure of that second field shall contain either a memory_management_control_operation command equal to 1 that marks the first field of the complementary field pair as "unused for reference" or a memory_management_control_operation command equal to 3 that marks the first field of the complementary field pair as "used for long-term reference" and assigns the same long-term frame index to the first field.

NOTE 6 – The above constraints specify that when both fields of a frame or a complementary field pair are marked as "used for reference" after processing all memory_management_control_operation commands of the decoded reference picture marking syntax structure, either both fields must be marked as "used for short-term reference" or both fields must be marked as "used for long-term reference". When both fields are marked as "used for long-term reference", the same long-term frame index must be assigned to both fields.

difference_of_pic_nums_minus1 is used (with memory_management_control_operation equal to 3 or 1) to assign a long-term frame index to a short-term reference picture or to mark a short-term reference picture as "unused for reference". When the associated memory_management_control_operation is processed by the decoding process, the resulting picture number derived from difference_of_pic_nums_minus1 shall be a picture number assigned to one of the reference pictures marked as "used for reference" and not previously assigned to a long-term frame index.

The resulting picture number is constrained as follows:

– If field_pic_flag is equal to 0, the resulting picture number shall be one of the set of picture numbers assigned to reference frames or complementary reference field pairs.

NOTE 7 – When field_pic_flag is equal to 0, the resulting picture number must be a picture number assigned to a complementary reference field pair in which both fields are marked as "used for reference" or a frame in which both fields are marked as "used for reference". In particular, when field_pic_flag is equal to 0, the marking of a non-paired field or a frame in which a single field is marked as "used for reference" cannot be affected by a memory_management_control_operation equal to 1.

– Otherwise (field_pic_flag is equal to 1), the resulting picture number shall be one of the set of picture numbers assigned to reference fields.

long_term_pic_num is used (with memory_management_control_operation equal to 2) to mark a long-term reference picture as "unused for reference". When the associated memory_management_control_operation is processed by the decoding process, long_term_pic_num shall be equal to a long-term picture number assigned to one of the reference pictures that is currently marked as "used for long-term reference".

The resulting long-term picture number is constrained as follows:

– If field_pic_flag is equal to 0, the resulting long-term picture number shall be one of the set of long-term picture numbers assigned to reference frames or complementary reference field pairs.

NOTE 8 – When field_pic_flag is equal to 0, the resulting long-term picture number must be a long-term picture number assigned to a complementary reference field pair in which both fields are marked as "used for reference" or a frame in which both fields are marked as "used for reference". In particular, when field_pic_flag is equal to 0, the marking of a non-paired field or a frame in which a single field is marked as "used for reference" cannot be affected by a memory_management_control_operation equal to 2.

– Otherwise (field_pic_flag is equal to 1), the resulting long-term picture number shall be one of the set of long-term picture numbers assigned to reference fields.

long_term_frame_idx is used (with memory_management_control_operation equal to 3 or 6) to assign a long-term frame index to a picture. When the associated memory_management_control_operation is processed by the decoding process, the value of long_term_frame_idx shall be in the range of 0 to MaxLongTermFrameIdx, inclusive.

max_long_term_frame_idx_plus1 minus 1 specifies the maximum value of long-term frame index allowed for long-term reference pictures (until receipt of another value of max_long_term_frame_idx_plus1). The value of max_long_term_frame_idx_plus1 shall be in the range of 0 to max_num_ref_frames, inclusive.

7.4.4 Slice data semantics

cabac_alignment_one_bit is a bit equal to 1.

mb_skip_run specifies the number of consecutive skipped macroblocks for which, when decoding a P or SP slice, mb_type shall be inferred to be P_Skip and the macroblock type is collectively referred to as a P macroblock type, or for which, when decoding a B slice, mb_type shall be inferred to be B_Skip and the macroblock type is collectively referred to as a B macroblock type. The value of mb_skip_run shall be in the range of 0 to PicSizeInMbs − CurrMbAddr, inclusive.

mb_skip_flag equal to 1 specifies that for the current macroblock, when decoding a P or SP slice, mb_type shall be inferred to be P_Skip and the macroblock type is collectively referred to as P macroblock type, or for which, when decoding a B slice, mb_type shall be inferred to be B_Skip and the macroblock type is collectively referred to as B macroblock type. mb_skip_flag equal to 0 specifies that the current macroblock is not skipped.

mb_field_decoding_flag equal to 0 specifies that the current macroblock pair is a frame macroblock pair. mb_field_decoding_flag equal to 1 specifies that the macroblock pair is a field macroblock pair. Both macroblocks of a frame macroblock pair are referred to in the text as frame macroblocks, whereas both macroblocks of a field macroblock pair are referred to in the text as field macroblocks.

When MbaffFrameFlag is equal to 0 (mb_field_decoding_flag is not present), mb_field_decoding_flag is inferred to be equal to field_pic_flag.

When MbaffFrameFlag is equal to 1 and mb_field_decoding_flag is not present for both the top and the bottom macroblock of a macroblock pair, the value of mb_field_decoding_flag shall be inferred as follows:

– If there is a neighbouring macroblock pair immediately to the left of the current macroblock pair in the same slice, the value of mb_field_decoding_flag is inferred to be equal to the value of mb_field_decoding_flag for the neighbouring macroblock pair immediately to the left of the current macroblock pair,
– Otherwise, if there is no neighbouring macroblock pair immediately to the left of the current macroblock pair in the same slice and there is a neighbouring macroblock pair immediately above the current macroblock pair in the same slice, the value of mb_field_decoding_flag is inferred to be equal to the value of mb_field_decoding_flag for the neighbouring macroblock pair immediately above the current macroblock pair,
– Otherwise (there is no neighbouring macroblock pair either immediately to the left or immediately above the current macroblock pair in the same slice), the value of mb_field_decoding_flag is inferred to be equal to 0.

NOTE – When MbaffFrameFlag is equal to 1 and mb_field_decoding_flag is not present for the top macroblock of a macroblock pair (because the top macroblock is skipped), a decoder must wait until mb_field_decoding_flag for the bottom macroblock is read (when the bottom macroblock is not skipped) or the value of mb_field_decoding_flag is inferred as specified above (when the bottom macroblock is also skipped) before it starts the decoding process for the top macroblock.

end_of_slice_flag equal to 0 specifies that another macroblock is following in the slice. end_of_slice_flag equal to 1 specifies the end of the slice and that no further macroblock follows.

The function NextMbAddress( ) used in the slice data syntax table is specified in subclause 8.2.2.

7.4.5 Macroblock layer semantics

mb_type specifies the macroblock type. The semantics of mb_type depend on the slice type.

Tables and semantics are specified for the various macroblock types for I, SI, P, SP, and B slices. Each table presents the value of mb_type, the name of mb_type, the number of macroblock partitions used (given by the NumMbPart(mb_type) function), the prediction mode of the macroblock (when it is not partitioned) or the first partition (given by the MbPartPredMode(mb_type, 0) function) and the prediction mode of the second partition (given by the MbPartPredMode(mb_type, 1) function). When a value is not applicable it is designated by "na". In the text, the value of mb_type may be referred to as the macroblock type, the value of MbPartPredMode( ) may be referred to in the text by "macroblock (partition) prediction mode", and a value X of MbPartPredMode( ) may be referred to in the text by "X macroblock (partition) prediction mode" or as "X prediction macroblocks".

Table 7-10 shows the allowed collective macroblock types for each slice_type.

NOTE 1 – There are some macroblock types with Pred_L0 macroblock (partition) prediction mode(s) that are classified as B macroblock types.

Table 7-10 – Allowed collective macroblock types for slice_type

slice_type	allowed collective macroblock types
I (slice)	I (see Table 7-11) (macroblock types)
P (slice)	P (see Table 7-13) and I (see Table 7-11) (macroblock types)
B (slice)	B (see Table 7-14) and I (see Table 7-11) (macroblock types)
SI (slice)	SI (see Table 7-12) and I (see Table 7-11) (macroblock types)
SP (slice)	P (see Table 7-13) and I (see Table 7-11) (macroblock types)

transform_size_8x8_flag equal to 1 specifies that for the current macroblock the transform coefficient decoding process and picture construction process prior to deblocking filter process for residual 8x8 blocks shall be invoked for luma samples, and when ChromaArrayType = = 3 also for Cb and Cr samples. transform_size_8x8_flag equal to 0 specifies that for the current macroblock the transform coefficient decoding process and picture construction process prior to deblocking filter process for residual 4x4 blocks shall be invoked for luma samples, and when

ChromaArrayType = = 3 also for Cb and Cr samples. When transform_size_8x8_flag is not present in the bitstream, it shall be inferred to be equal to 0.

NOTE 2 – When the current macroblock prediction mode MbPartPredMode(mb_type, 0) is equal to Intra_16x16, transform_size_8x8_flag is not present in the bitstream and then inferred to be equal to 0.

When sub_mb_type[mbPartIdx] (see subclause 7.4.5.2) is present in the bitstream for all 8x8 blocks indexed by mbPartIdx = 0..3, the variable noSubMbPartSizeLessThan8x8Flag indicates whether for each of the four 8x8 blocks the corresponding SubMbPartWidth(sub_mb_type[mbPartIdx]) and SubMbPartHeight(sub_mb_type[mbPartIdx]) are both equal to 8.

NOTE 3 – When noSubMbPartSizeLessThan8x8Flag is equal to 0 and the current macroblock type is not equal to I_NxN, transform_size_8x8_flag is not present in the bitstream and then inferred to be equal to 0.

Macroblock types that may be collectively referred to as I macroblock types are specified in Table 7-11.

The macroblock types for I slices are all I macroblock types.

Table 7-11 – Macroblock types for I slices

mb_type	Name of mb_type	transform_size_8x8_flag	MbPartPredMode (mb_type, 0)	Intra16x16PredMode	CodedBlockPatternChroma	CodedBlockPatternLuma
0	I_NxN	0	Intra_4x4	na	Equation 7-35	Equation 7-35
0	I_NxN	1	Intra_8x8	na	Equation 7-35	Equation 7-35
1	I_16x16_0_0_0	na	Intra_16x16	0	0	0
2	I_16x16_1_0_0	na	Intra_16x16	1	0	0
3	I_16x16_2_0_0	na	Intra_16x16	2	0	0
4	I_16x16_3_0_0	na	Intra_16x16	3	0	0
5	I_16x16_0_1_0	na	Intra_16x16	0	1	0
6	I_16x16_1_1_0	na	Intra_16x16	1	1	0
7	I_16x16_2_1_0	na	Intra_16x16	2	1	0
8	I_16x16_3_1_0	na	Intra_16x16	3	1	0
9	I_16x16_0_2_0	na	Intra_16x16	0	2	0
10	I_16x16_1_2_0	na	Intra_16x16	1	2	0
11	I_16x16_2_2_0	na	Intra_16x16	2	2	0
12	I_16x16_3_2_0	na	Intra_16x16	3	2	0
13	I_16x16_0_0_1	na	Intra_16x16	0	0	15
14	I_16x16_1_0_1	na	Intra_16x16	1	0	15
15	I_16x16_2_0_1	na	Intra_16x16	2	0	15
16	I_16x16_3_0_1	na	Intra_16x16	3	0	15
17	I_16x16_0_1_1	na	Intra_16x16	0	1	15
18	I_16x16_1_1_1	na	Intra_16x16	1	1	15
19	I_16x16_2_1_1	na	Intra_16x16	2	1	15
20	I_16x16_3_1_1	na	Intra_16x16	3	1	15
21	I_16x16_0_2_1	na	Intra_16x16	0	2	15
22	I_16x16_1_2_1	na	Intra_16x16	1	2	15
23	I_16x16_2_2_1	na	Intra_16x16	2	2	15
24	I_16x16_3_2_1	na	Intra_16x16	3	2	15
25	I_PCM	na	na	na	na	na

The following semantics are assigned to the macroblock types in Table 7-11:

– I_NxN: A mnemonic name for mb_type equal to 0 with MbPartPredMode(mb_type, 0) equal to Intra_4x4 or Intra_8x8.

– I_16x16_0_0_0, I_16x16_1_0_0, I_16x16_2_0_0, I_16x16_3_0_0, I_16x16_0_1_0, I_16x16_1_1_0, I_16x16_2_1_0, I_16x16_3_1_0, I_16x16_0_2_0, I_16x16_1_2_0, I_16x16_2_2_0, I_16x16_3_2_0, I_16x16_0_0_1, I_16x16_1_0_1, I_16x16_2_0_1, I_16x16_3_0_1, I_16x16_0_1_1, I_16x16_1_1_1, I_16x16_2_1_1, I_16x16_3_1_1, I_16x16_0_2_1, I_16x16_1_2_1, I_16x16_2_2_1, I_16x16_3_2_1: the macroblock is coded as an Intra_16x16 prediction macroblock.

To each Intra_16x16 prediction macroblock, an Intra16x16PredMode is assigned, which specifies the Intra_16x16 prediction mode, and values of CodedBlockPatternLuma and CodedBlockPatternChroma are assigned as specified in Table 7-11.

Intra_4x4 specifies the macroblock prediction mode and specifies that the Intra_4x4 prediction process is invoked as specified in subclause 8.3.1. Intra_4x4 is an Intra macroblock prediction mode.

Intra_8x8 specifies the macroblock prediction mode and specifies that the Intra_8x8 prediction process is invoked as specified in subclause 8.3.2. Intra_8x8 is an Intra macroblock prediction mode.

Intra_16x16 specifies the macroblock prediction mode and specifies that the Intra_16x16 prediction process is invoked as specified in subclause 8.3.3. Intra_16x16 is an Intra macroblock prediction mode.

For a macroblock coded with mb_type equal to I_PCM, the Intra macroblock prediction mode shall be inferred.

A macroblock type that may be referred to as the SI macroblock type is specified in Table 7-12.

The macroblock types for SI slices are specified in Tables 7-12 and 7-11. The mb_type value 0 is specified in Table 7-12 and the mb_type values 1 to 26 are specified in Table 7-11, indexed by subtracting 1 from the value of mb_type.

Table 7-12 – Macroblock type with value 0 for SI slices

mb_type	Name of mb_type	MbPartPredMode (mb_type, 0)	Intra16x16PredMode	CodedBlockPatternChroma	CodedBlockPatternLuma
0	SI	Intra_4x4	na	Equation 7-35	Equation 7-35

The following semantics are assigned to the macroblock type in Table 7-12:

– The SI macroblock is coded as Intra_4x4 prediction macroblock.

Macroblock types that may be collectively referred to as P macroblock types are specified in Table 7-13.

The macroblock types for P and SP slices are specified in Tables 7-13 and 7-11. mb_type values 0 to 4 are specified in Table 7-13 and mb_type values 5 to 30 are specified in Table 7-11, indexed by subtracting 5 from the value of mb_type.

Table 7-13 – Macroblock type values 0 to 4 for P and SP slices

mb_type	Name of mb_type	NumMbPart (mb_type)	MbPartPredMode (mb_type, 0)	MbPartPredMode (mb_type, 1)	MbPartWidth (mb_type)	MbPartHeight (mb_type)
0	P_L0_16x16	1	Pred_L0	na	16	16
1	P_L0_L0_16x8	2	Pred_L0	Pred_L0	16	8
2	P_L0_L0_8x16	2	Pred_L0	Pred_L0	8	16
3	P_8x8	4	na	na	8	8
4	P_8x8ref0	4	na	na	8	8
inferred	P_Skip	1	Pred_L0	na	16	16

The following semantics are assigned to the macroblock types in Table 7-13:

– P_L0_16x16: the samples of the macroblock are predicted with one luma macroblock partition of size 16x16 luma samples and associated chroma samples.

– P_L0_L0_MxN, with MxN being replaced by 16x8 or 8x16: the samples of the macroblock are predicted using two luma partitions of size MxN equal to 16x8, or two luma partitions of size MxN equal to 8x16, and associated chroma samples, respectively.

– P_8x8: for each sub-macroblock an additional syntax element (sub_mb_type[mbPartIdx] with mbPartIdx being the macroblock partition index for the corresponding sub-macroblock) is present in the bitstream that specifies the type of the corresponding sub-macroblock (see subclause 7.4.5.2).

– P_8x8ref0: has the same semantics as P_8x8 but no syntax element for the reference index (ref_idx_l0[mbPartIdx] with mbPartIdx = 0..3) is present in the bitstream and ref_idx_l0[mbPartIdx] shall be inferred to be equal to 0 for all sub-macroblocks of the macroblock (with indices mbPartIdx = 0..3).

– P_Skip: no further data is present for the macroblock in the bitstream.

The following semantics are assigned to the macroblock prediction modes (for macroblocks that are not partitioned) and macroblock partition prediction modes (for macroblocks that are partitioned) specified by MbPartPredMode( ) in Table 7-13:

– Pred_L0: specifies that the Inter prediction process is invoked using list 0 prediction. Pred_L0 is an Inter macroblock prediction mode (for macroblocks that are not partitioned) and an Inter macroblock partition prediction mode (for macroblocks that are partitioned).

When mb_type is equal to any of the values specified in Table 7-13, the macroblock is coded in an Inter macroblock prediction mode.

Macroblock types that may be collectively referred to as B macroblock types are specified in Table 7-14.

The macroblock types for B slices are specified in Tables 7-14 and 7-11. The mb_type values 0 to 22 are specified in Table 7-14 and the mb_type values 23 to 48 are specified in Table 7-11, indexed by subtracting 23 from the value of mb_type.

Table 7-14 – Macroblock type values 0 to 22 for B slices

mb_type	Name of mb_type	NumMbPart (mb_type)	MbPartPredMode (mb_type, 0)	MbPartPredMode (mb_type, 1)	MbPartWidth (mb_type)	MbPartHeight (mb_type)
0	B_Direct_16x16	na	Direct	na	8	8
1	B_L0_16x16	1	Pred_L0	na	16	16
2	B_L1_16x16	1	Pred_L1	na	16	16
3	B_Bi_16x16	1	BiPred	na	16	16
4	B_L0_L0_16x8	2	Pred_L0	Pred_L0	16	8
5	B_L0_L0_8x16	2	Pred_L0	Pred_L0	8	16
6	B_L1_L1_16x8	2	Pred_L1	Pred_L1	16	8
7	B_L1_L1_8x16	2	Pred_L1	Pred_L1	8	16
8	B_L0_L1_16x8	2	Pred_L0	Pred_L1	16	8
9	B_L0_L1_8x16	2	Pred_L0	Pred_L1	8	16
10	B_L1_L0_16x8	2	Pred_L1	Pred_L0	16	8
11	B_L1_L0_8x16	2	Pred_L1	Pred_L0	8	16
12	B_L0_Bi_16x8	2	Pred_L0	BiPred	16	8
13	B_L0_Bi_8x16	2	Pred_L0	BiPred	8	16
14	B_L1_Bi_16x8	2	Pred_L1	BiPred	16	8
15	B_L1_Bi_8x16	2	Pred_L1	BiPred	8	16
16	B_Bi_L0_16x8	2	BiPred	Pred_L0	16	8
17	B_Bi_L0_8x16	2	BiPred	Pred_L0	8	16
18	B_Bi_L1_16x8	2	BiPred	Pred_L1	16	8
19	B_Bi_L1_8x16	2	BiPred	Pred_L1	8	16
20	B_Bi_Bi_16x8	2	BiPred	BiPred	16	8
21	B_Bi_Bi_8x16	2	BiPred	BiPred	8	16
22	B_8x8	4	na	na	8	8
inferred	B_Skip	na	Direct	na	8	8

The following semantics are assigned to the macroblock types in Table 7-14:

– B_Direct_16x16: no motion vector differences or reference indices are present for the macroblock in the bitstream. The functions MbPartWidth(B_Direct_16x16), and MbPartHeight(B_Direct_16x16) are used in the derivation process for motion vectors and reference frame indices in subclause 8.4.1 for direct mode prediction.

– B_X_16x16 with X being replaced by L0, L1, or Bi: the samples of the macroblock are predicted with one luma macroblock partition of size 16x16 luma samples and associated chroma samples. For a macroblock with type B_X_16x16 with X being replaced by either L0 or L1, one motion vector difference and one reference index is present in the bitstream for the macroblock. For a macroblock with type B_X_16x16 with X being replaced by Bi, two motion vector differences and two reference indices are present in the bitstream for the macroblock.

– B_X0_X1_MxN, with X0, X1 referring to the first and second macroblock partition and being replaced by L0, L1, or Bi, and MxN being replaced by 16x8 or 8x16: the samples of the macroblock are predicted using two luma partitions of size MxN equal to 16x8, or two luma partitions of size MxN equal to 8x16, and associated chroma samples, respectively. For a macroblock partition X0 or X1 with X0 or X1 being replaced by either L0 or L1, one motion vector difference and one reference index is present in the bitstream. For a macroblock partition X0 or X1 with X0 or X1 being replaced by Bi, two motion vector differences and two reference indices are present in the bitstream for the macroblock partition.

– B_8x8: for each sub-macroblock an additional syntax element (sub_mb_type[mbPartIdx] with mbPartIdx being the macroblock partition index for the corresponding sub-macroblock) is present in the bitstream that specifies the type of the corresponding sub-macroblock (see subclause 7.4.5.2).

– B_Skip: no further data is present for the macroblock in the bitstream. The functions MbPartWidth(B_Skip), and MbPartHeight(B_Skip) are used in the derivation process for motion vectors and reference frame indices in subclause 8.4.1 for direct mode prediction.

The following semantics are assigned to the macroblock prediction modes (for macroblocks that are not partitioned) and macroblock partition prediction modes (for macroblocks that are partitioned) specified by MbPartPredMode( ) in Table 7-14:

– Direct: no motion vector differences or reference indices are present for the macroblock (in case of B_Skip or B_Direct_16x16) in the bitstream. Direct is an Inter macroblock prediction mode (for macroblocks that are not partitioned) and an Inter macroblock partition prediction mode (for macroblocks that are partitioned, see Table 7-18).

– Pred_L0: the semantics specified for Table 7-13 apply.

– Pred_L1: specifies that the Inter prediction process is invoked using list 1 prediction. Pred_L1 is an Inter macroblock prediction mode (for macroblocks that are not partitioned) and an Inter macroblock partition prediction mode (for macroblocks that are partitioned).

– BiPred: specifies that the Inter prediction process is invoked using list 0 and list 1 prediction. BiPred is an Inter macroblock prediction mode (for macroblocks that are not partitioned) and an Inter macroblock partition prediction mode (for macroblocks that are partitioned).

When mb_type is equal to any of the values specified in Table 7-14, the macroblock is coded in an Inter macroblock prediction mode.

pcm_alignment_zero_bit is a bit equal to 0.

pcm_sample_luma[i] is a sample value. The pcm_sample_luma[i] values represent luma sample values in the raster scan within the macroblock. The number of bits used to represent each of these samples is BitDepth_Y.

pcm_sample_chroma[i] is a sample value. The first MbWidthC * MbHeightC pcm_sample_chroma[i] values represent Cb sample values in the raster scan within the macroblock and the remaining MbWidthC * MbHeightC pcm_sample_chroma[i] values represent Cr sample values in the raster scan within the macroblock. The number of bits used to represent each of these samples is BitDepth_C.

coded_block_pattern specifies which of the four 8x8 luma blocks and associated chroma blocks of a macroblock may contain non-zero transform coefficient levels. When coded_block_pattern is present in the bitstream, the variables CodedBlockPatternLuma and CodedBlockPatternChroma are derived as

$$\begin{aligned}&\text{CodedBlockPatternLuma} = \text{coded\_block\_pattern} \,\%\, 16\\&\text{CodedBlockPatternChroma} = \text{coded\_block\_pattern} \,/\, 16\end{aligned} \tag{7-35}$$

When the macroblock type is not equal to P_Skip, B_Skip, or I_PCM, the following applies:

– If the macroblock prediction mode is equal Intra_16x16, the following applies:
  – If ChromaArrayType is not equal to 3, the value of CodedBlockPatternLuma specifies the following.
    – If CodedBlockPatternLuma is equal to 0, all AC transform coefficient levels of the luma component of the macroblock are equal to 0 for all 16 of the 4x4 blocks in the 16x16 luma block.
    – Otherwise (CodedBlockPatternLuma is not equal to 0), CodedBlockPatternLuma is equal to 15, at least one of the AC transform coefficient levels of the luma component of the macroblock shall be non-zero, and the AC transform coefficient levels are scanned for all 16 of the 4x4 blocks in the 16x16 block.

– Otherwise (ChromaArrayType is equal to 3), the value of CodedBlockPatternLuma specifies the following.

  – If CodedBlockPatternLuma is equal to 0, all AC transform coefficient levels of the luma, Cb, and Cr components of the macroblock are equal to 0 for all 16 of the 4x4 blocks in the luma, Cb, and Cr components of the macroblock.

  – Otherwise (CodedBlockPatternLuma is not equal to 0), CodedBlockPatternLuma is equal to 15, at least one of the AC transform coefficient levels of the luma, Cb, or Cr components of the macroblock shall be non-zero, and the AC transform coefficient levels are scanned for all 16 of the 4x4 blocks in the luma Cb, and Cr components of the macroblock.

– Otherwise (the macroblock prediction mode is not equal to Intra_16x16), coded_block_pattern is present in the bitstream, and the following applies:

  – If ChromaArrayType is not equal to 3, each of the four LSBs of CodedBlockPatternLuma specifies, for one of the four 8x8 luma blocks of the macroblock, the following.

    – If the corresponding bit of CodedBlockPatternLuma is equal to 0, all transform coefficient levels of the luma transform blocks in the 8x8 luma block are equal to zero.

    – Otherwise (the corresponding bit of CodedBlockPatternLuma is equal to 1), one or more transform coefficient levels of one or more of the luma transform blocks in the 8x8 luma block shall be non-zero valued and the transform coefficient levels of the corresponding transform blocks are scanned.

  – Otherwise (ChromaArrayType is equal to 3), each of the four LSBs of CodedBlockPatternLuma specifies, for one of the four 8x8 luma blocks of the macroblock, the following.

    – If the corresponding bit of CodedBlockPatternLuma is equal to 0, all transform coefficient levels of the luma, Cb, and Cr transform blocks in the 8x8 luma block are equal to zero.

    – Otherwise (the corresponding bit of CodedBlockPatternLuma is equal to 1), one or more transform coefficient levels of one or more of the luma, Cb, or Cr transform blocks in the 8x8 luma block shall be non-zero valued and the transform coefficient levels of the corresponding transform blocks are scanned.

When the macroblock type is not equal to P_Skip, B_Skip, or I_PCM, CodedBlockPatternChroma is interpreted as follows:

– If ChromaArrayType is not equal to 0 or 3, CodedBlockPatternChroma is specified in Table 7-15.

– Otherwise (ChromaArrayType is equal to 0 or 3), the bitstream shall not contain data that result in a derived value of CodedBlockPatternChroma that is not equal to 0.

Table 7-15 – Specification of CodedBlockPatternChroma values

CodedBlockPatternChroma	Description
0	All chroma transform coefficient levels are equal to 0.
1	One or more chroma DC transform coefficient levels shall be non-zero valued. All chroma AC transform coefficient levels are equal to 0.
2	Zero or more chroma DC transform coefficient levels are non-zero valued. One or more chroma AC transform coefficient levels shall be non-zero valued.

mb_qp_delta can change the value of QP_Y in the macroblock layer. The decoded value of mb_qp_delta shall be in the range of $-(26 + QpBdOffset_Y / 2)$ to $+(25 + QpBdOffset_Y / 2)$, inclusive. mb_qp_delta shall be inferred to be equal to 0 when it is not present for any macroblock (including P_Skip and B_Skip macroblock types).

The value of QP_Y is derived as

$$QP_Y = ((QP_{Y,PREV} + mb\_qp\_delta + 52 + 2 * QpBdOffset_Y) \% (52 + QpBdOffset_Y)) - QpBdOffset_Y \quad (7\text{-}36)$$

where $QP_{Y,PREV}$ is the luma quantisation parameter, QP_Y, of the previous macroblock in decoding order in the current slice. For the first macroblock in the slice $QP_{Y,PREV}$ is initially set equal to $SliceQP_Y$ derived in Equation 7-29 at the start of each slice.

The value of QP'_Y is derived as

$$QP'_Y = QP_Y + QpBdOffset_Y \quad (7\text{-}37)$$

The variable TransformBypassModeFlag is derived as follows:

– If qpprime_y_zero_transform_bypass_flag is equal to 1 and QP'_Y is equal to 0, TransformBypassModeFlag is set equal to 1.

– Otherwise (qpprime_y_zero_transform_bypass_flag is equal to 0 or QP'_Y is not equal to 0), TransformBypassModeFlag is set equal to 0.

7.4.5.1 Macroblock prediction semantics

All samples of the macroblock are predicted. The prediction modes are derived using the following syntax elements.

prev_intra4x4_pred_mode_flag[luma4x4BlkIdx **]** and **rem_intra4x4_pred_mode[** luma4x4BlkIdx **]** specify the Intra_4x4 prediction of the 4x4 luma block with index luma4x4BlkIdx = 0..15. When ChromaArrayType is equal to 3, prev_intra4x4_pred_mode_flag[luma4x4BlkIdx] and rem_intra4x4_pred_mode[luma4x4BlkIdx] also specify the Intra_4x4 prediction of the 4x4 Cb block with luma4x4BlkIdx equal to cb4x4BlkIdx for cb4x4BlkIdx = 0..15 and the Intra_4x4 prediction of the 4x4 Cr block with luma4x4BlkIdx equal to cr4x4BlkIdx for cr4x4BlkIdx = 0..15.

prev_intra8x8_pred_mode_flag[luma8x8BlkIdx] and **rem_intra8x8_pred_mode**[luma8x8BlkIdx] specify the Intra_8x8 prediction of the 8x8 luma block with index luma8x8BlkIdx = 0..3. When ChromaArrayType is equal to 3, prev_intra8x8_pred_mode_flag[luma8x8BlkIdx] and rem_intra8x8_pred_mode[luma8x8BlkIdx] also specify the Intra_8x8 prediction of the 8x8 Cb block with luma8x8BlkIdx equal to cb8x8BlkIdx for cb8x8BlkIdx = 0..3 and the Intra_8x8 prediction of the 8x8 Cr block with index luma8x8BlkIdx equal to cr8x8BlkIdx for cr8x8BlkIdx = 0..3.

intra_chroma_pred_mode specifies, when ChromaArrayType is equal to 1 or 2, the type of spatial prediction used for chroma in macroblocks using Intra_4x4, Intra_8x8, or Intra_16x16 prediction, as shown in Table 7-16. The value of intra_chroma_pred_mode shall be in the range of 0 to 3, inclusive.

Table 7-16 – Relationship between intra_chroma_pred_mode and spatial prediction modes

intra_chroma_pred_mode	Intra Chroma Prediction Mode
0	DC
1	Horizontal
2	Vertical
3	Plane

ref_idx_l0[mbPartIdx **]** when present, specifies the index in reference picture list 0 of the reference picture to be used for prediction.

The range of ref_idx_l0[mbPartIdx], the index in list 0 of the reference picture, and, if applicable, the parity of the field within the reference picture used for prediction are specified as follows:

– If MbaffFrameFlag is equal to 0 or mb_field_decoding_flag is equal to 0, the value of ref_idx_l0[mbPartIdx] shall be in the range of 0 to num_ref_idx_l0_active_minus1, inclusive.

– Otherwise (MbaffFrameFlag is equal to 1 and mb_field_decoding_flag is equal to 1), the value of ref_idx_l0[mbPartIdx] shall be in the range of 0 to 2 * num_ref_idx_l0_active_minus1 + 1, inclusive.

When only one reference picture is used for inter prediction, the values of ref_idx_l0[mbPartIdx] shall be inferred to be equal to 0.

ref_idx_l1[mbPartIdx **]** has the same semantics as ref_idx_l0, with l0 and list 0 replaced by l1 and list 1, respectively.

mvd_l0[mbPartIdx **][** 0 **][** compIdx **]** specifies the difference between a list 0 motion vector component to be used and its prediction. The index mbPartIdx specifies to which macroblock partition mvd_l0 is assigned. The partitioning of the macroblock is specified by mb_type. The horizontal motion vector component difference is decoded first in decoding order and is assigned compIdx = 0. The vertical motion vector component is decoded second in decoding order and is assigned compIdx = 1. The range of the components of mvd_l0[mbPartIdx][0][compIdx] is specified by constraints on the motion vector variable values derived from it as specified in Annex A.

mvd_l1[mbPartIdx **][** 0 **][** compIdx **]** has the same semantics as mvd_l0, with l0 and list 0 replaced by l1 and list 1, respectively.

7.4.5.2 Sub-macroblock prediction semantics

sub_mb_type[mbPartIdx **]** specifies the sub-macroblock types.

Tables and semantics are specified for the various sub-macroblock types for P, and B macroblock types. Each table presents the value of sub_mb_type[mbPartIdx], the name of sub_mb_type[mbPartIdx], the number of sub-macroblock partitions used (given by the NumSubMbPart(sub_mb_type[mbPartIdx]) function), and the prediction mode of the sub-macroblock (given by the SubMbPredMode(sub_mb_type[mbPartIdx]) function). In the text, the value of sub_mb_type[mbPartIdx] may be referred to by "sub-macroblock type". In the text, the value of SubMbPredMode( ) may be referred to by "sub-macroblock prediction mode" or "macroblock partition prediction mode".

The interpretation of sub_mb_type[mbPartIdx] for P macroblock types is specified in Table 7-17, where the row for "inferred" specifies values inferred when sub_mb_type[mbPartIdx] is not present.

Table 7-17 – Sub-macroblock types in P macroblocks

sub_mb_type[mbPartIdx]	Name of sub_mb_type[mbPartIdx]	NumSubMbPart (sub_mb_type[mbPartIdx])	SubMbPredMode (sub_mb_type[mbPartIdx])	SubMbPartWidth (sub_mb_type[mbPartIdx])	SubMbPartHeight (sub_mb_type[mbPartIdx])
inferred	na	na	na	na	na
0	P_L0_8x8	1	Pred_L0	8	8
1	P_L0_8x4	2	Pred_L0	8	4
2	P_L0_4x8	2	Pred_L0	4	8
3	P_L0_4x4	4	Pred_L0	4	4

The following semantics are assigned to the sub-macroblock types in Table 7-17:

– P_L0_MxN, with MxN being replaced by 8x8, 8x4, 4x8, or 4x4: the samples of the sub-macroblock are predicted using one luma partition of size MxN equal to 8x8, two luma partitions of size MxN equal to 8x4, or two luma partitions of size MxN equal to 4x8, or four luma partitions of size MxN equal to 4x4, and associated chroma samples, respectively.

The following semantics are assigned to the sub-macroblock prediction modes (or macroblock partition prediction modes) specified by SubMbPredMode( ) in Table 7-17:

– Pred_L0: see semantics for Table 7-13.

The interpretation of sub_mb_type[mbPartIdx] for B macroblock types is specified in Table 7-18, where the row for "inferred" specifies values inferred when sub_mb_type[mbPartIdx] is not present, and the inferred value "mb_type" specifies that the name of sub_mb_type[mbPartIdx] is the same as the name of mb_type for this case.

Table 7-18 – Sub-macroblock types in B macroblocks

sub_mb_type[mbPartIdx]	Name of sub_mb_type[mbPartIdx]	NumSubMbPart (sub_mb_type[mbPartIdx])	SubMbPredMode (sub_mb_type[mbPartIdx])	SubMbPartWidth (sub_mb_type[mbPartIdx])	SubMbPartHeight (sub_mb_type[mbPartIdx])
inferred	mb_type	4	Direct	4	4
0	B_Direct_8x8	4	Direct	4	4
1	B_L0_8x8	1	Pred_L0	8	8
2	B_L1_8x8	1	Pred_L1	8	8
3	B_Bi_8x8	1	BiPred	8	8
4	B_L0_8x4	2	Pred_L0	8	4
5	B_L0_4x8	2	Pred_L0	4	8
6	B_L1_8x4	2	Pred_L1	8	4
7	B_L1_4x8	2	Pred_L1	4	8
8	B_Bi_8x4	2	BiPred	8	4
9	B_Bi_4x8	2	BiPred	4	8
10	B_L0_4x4	4	Pred_L0	4	4
11	B_L1_4x4	4	Pred_L1	4	4
12	B_Bi_4x4	4	BiPred	4	4

The following semantics are assigned to the sub-macroblock types in Table 7-18:

– B_Skip and B_Direct_16x16: no motion vector differences or reference indices are present for the sub-macroblock in the bitstream. The functions SubMbPartWidth( ) and SubMbPartHeight( ) are used in the derivation process for motion vectors and reference frame indices in subclause 8.4.1 for direct mode prediction.

– B_Direct_8x8: no motion vector differences or reference indices are present for the sub-macroblock in the bitstream. The functions SubMbPartWidth(B_Direct_8x8) and SubMbPartHeight(B_Direct_8x8) are used in the derivation process for motion vectors and reference frame indices in subclause 8.4.1 for direct mode prediction.

– B_X_MxN, with X being replaced by L0, L1, or Bi, and MxN being replaced by 8x8, 8x4, 4x8 or 4x4: the samples of the sub-macroblock are predicted using one luma partition of size MxN equal to 8x8, or the samples of the sub-macroblock are predicted using two luma partitions of size MxN equal to 8x4, or the samples of the sub-macroblock are predicted using two luma partitions of size MxN equal to 4x8, or the samples of the sub-macroblock are predicted using four luma partitions of size MxN equal to 4x4, and associated chroma samples, respectively. All sub-macroblock partitions share the same reference index. For an MxN sub-macroblock partition in a sub-macroblock with sub_mb_type[mbPartIdx] being B_X_MxN with X being replaced by either L0 or L1, one motion vector difference is present in the bitstream. For an MxN sub-macroblock partition in a sub-macroblock with sub_mb_type[mbPartIdx] being B_Bi_MxN, two motion vector difference are present in the bitstream.

The following semantics are assigned to the sub-macroblock prediction modes (or macroblock partition prediction modes) specified by SubMbPredMode( ) in Table 7-18:

– Direct: see semantics for Table 7-14.

– Pred_L0: see semantics for Table 7-13.

– Pred_L1: see semantics for Table 7-14.

– BiPred: see semantics for Table 7-14.

ref_idx_l0[mbPartIdx **]** has the same semantics as ref_idx_l0 in subclause 7.4.5.1.

ref_idx_l1[mbPartIdx **]** has the same semantics as ref_idx_l1 in subclause 7.4.5.1.

mvd_l0[mbPartIdx **][** subMbPartIdx **][** compIdx **]** has the same semantics as mvd_l0 in subclause 7.4.5.1, except that it is applied to the sub-macroblock partition index with subMbPartIdx. The indices mbPartIdx and subMbPartIdx specify to which macroblock partition and sub-macroblock partition mvd_l0 is assigned.

mvd_l1[mbPartIdx **][** subMbPartIdx **][** compIdx **]** has the same semantics as mvd_l1 in subclause 7.4.5.1, except that it is applied to the sub-macroblock partition index with subMbPartIdx. The indices mbPartIdx and subMbPartIdx specify to which macroblock partition and sub-macroblock partition mvd_l1 is assigned.

7.4.5.3 Residual data semantics

The syntax structure residual_block( ), which is used for parsing the transform coefficient levels, is assigned as follows:

– If entropy_coding_mode_flag is equal to 0, residual_block is set equal to residual_block_cavlc, which is used for parsing the syntax elements for transform coefficient levels.

– Otherwise (entropy_coding_mode_flag is equal to 1), residual_block is set equal to residual_block_cabac, which is used for parsing the syntax elements for transform coefficient levels.

The syntax structure residual_luma(i16x16DClevel, i16x16AClevel, level4x4, level8x8, startIdx, endIdx) is used with the first four variables in brackets being its output and being assigned as follows.

Intra16x16DCLevel is set equal to i16x16DClevel, Intra16x16ACLevel is set equal to i16x16AClevel, LumaLevel4x4 is set equal to level4x4, and LumaLevel8x8 is set equal to level8x8.

When ChromaArrayType is equal to 1 or 2, the following applies:

– For each chroma component, indexed by iCbCr = 0..1, the DC transform coefficient levels of the 4 * NumC8x8 4x4 chroma blocks are parsed into the iCbCr-th list ChromaDCLevel[iCbCr].

– For each of the 4x4 chroma blocks, indexed by i4x4 = 0..3 and i8x8 = 0..NumC8x8 − 1, of each chroma component, indexed by iCbCr = 0..1, the 15 AC transform coefficient levels are parsed into the (i8x8*4 + i4x4)-th list of the iCbCr-th chroma component ChromaACLevel[iCbCr][i8x8*4 + i4x4].

When ChromaArrayType is equal to 3, the following applies:

– The syntax structure residual_luma(i16x16DClevel, i16x16AClevel, level4x4, level8x8, startIdx, endIdx) is used for the Cb component with the first four variables in brackets being its output and being assigned as follows. CbIntra16x16DCLevel is set equal to i16x16DClevel, CbIntra16x16ACLevel is set equal to i16x16AClevel, CbLevel4x4 is set equal to level4x4, and CbLevel8x8 is set equal to level8x8.

– The syntax structure residual_luma(i16x16DClevel, i16x16AClevel, level4x4, level8x8, startIdx, endIdx) is used for the Cr component with the first four variables in brackets being its output and being assigned as follows. CrIntra16x16DCLevel is set equal to i16x16DClevel, CrIntra16x16ACLevel is set equal to i16x16AClevel, CrLevel4x4 is set equal to level4x4, and CrLevel8x8 is set equal to level8x8.

7.4.5.3.1 Residual luma data semantics

Output of this syntax structure are the variables i16x16DClevel, i16x16AClevel, level4x4, and level8x8.

Depending on mb_type, the syntax structure residual_block(coeffLevel, startIdx, endIdx, maxNumCoeff) is used with the arguments coeffLevel, which is a list containing the maxNumCoeff transform coefficient levels that are parsed in residual_block( ), startIdx, endIdx, and maxNumCoeff as follows.

Depending on MbPartPredMode(mb_type, 0), the following applies:

– If MbPartPredMode(mb_type, 0) is equal to Intra_16x16, the transform coefficient levels are parsed into the list i16x16DClevel and into the 16 lists i16x16AClevel[i]. i16x16DClevel contains the 16 transform coefficient levels of the DC transform coefficient levels for each 4x4 luma block. For each of the 16 4x4 luma blocks indexed by i = 0..15, the 15 AC transform coefficients levels of the i-th block are parsed into the i-th list i16x16AClevel[i].

– Otherwise (MbPartPredMode(mb_type, 0) is not equal to Intra_16x16), the following applies:
  – If transform_size_8x8_flag is equal to 0, for each of the 16 4x4 luma blocks indexed by i = 0..15, the 16 transform coefficient levels of the i-th block are parsed into the i-th list level4x4[i].
  – Otherwise (transform_size_8x8_flag is equal to 1), for each of the 4 8x8 luma blocks indexed by i8x8 = 0..3, the following applies:
    – If entropy_coding_mode_flag is equal to 0, first for each of the 4 4x4 luma blocks indexed by i4x4 = 0..3, the 16 transform coefficient levels of the i4x4-th block are parsed into the (i8x8 * 4 + i4x4)-th list level4x4[i8x8 * 4 + i4x4]. Then, the 64 transform coefficient levels of the i8x8-th 8x8 luma block which are indexed by 4 * i + i4x4, where i = 0..15 and i4x4 = 0..3, are derived as level8x8[i8x8][4 * i + i4x4] = level4x4[i8x8 * 4 + i4x4][i].

      NOTE – The 4x4 luma blocks with luma4x4BlkIdx = i8x8 * 4 + i4x4 containing every fourth transform coefficient level of the corresponding i8x8-th 8x8 luma block with offset i4x4 are assumed to represent spatial locations given by the inverse 4x4 luma block scanning process in subclause 6.4.3.

    – Otherwise (entropy_coding_mode_flag is equal to 1), the 64 transform coefficient levels of the i8x8-th block are parsed into the i8x8-th list level8x8[i8x8].

7.4.5.3.2 Residual block CAVLC semantics

The function TotalCoeff(coeff_token) that is used in subclause 7.3.5.3.2 returns the number of non-zero transform coefficient levels derived from coeff_token.

The function TrailingOnes(coeff_token) that is used in subclause 7.3.5.3.2 returns the trailing ones derived from coeff_token.

coeff_token specifies the total number of non-zero transform coefficient levels and the number of trailing one transform coefficient levels in a transform coefficient level scan. A trailing one transform coefficient level is one of up to three consecutive non-zero transform coefficient levels having an absolute value equal to 1 at the end of a scan of non-zero transform coefficient levels. The range of coeff_token is specified in subclause 9.2.1.

trailing_ones_sign_flag specifies the sign of a trailing one transform coefficient level as follows:

– If trailing_ones_sign_flag is equal to 0, the corresponding transform coefficient level is decoded as +1.

– Otherwise (trailing_ones_sign_flag equal to 1), the corresponding transform coefficient level is decoded as −1.

level_prefix and **level_suffix** specify the value of a non-zero transform coefficient level. The range of level_prefix and level_suffix is specified in subclause 9.2.2.

total_zeros specifies the total number of zero-valued transform coefficient levels that are located before the position of the last non-zero transform coefficient level in a scan of transform coefficient levels. The range of total_zeros is specified in subclause 9.2.3.

run_before specifies the number of consecutive transform coefficient levels in the scan with zero value before a non-zero valued transform coefficient level. The range of run_before is specified in subclause 9.2.3.

coeffLevel contains maxNumCoeff transform coefficient levels for the current list of transform coefficient levels.

7.4.5.3.3 Residual block CABAC semantics

coded_block_flag specifies whether the transform block contains non-zero transform coefficient levels as follows:

– If coded_block_flag is equal to 0, the transform block contains no non-zero transform coefficient levels.

– Otherwise (coded_block_flag is equal to 1), the transform block contains at least one non-zero transform coefficient level.

When coded_block_flag is not present, it shall be inferred to be equal to 1.

significant_coeff_flag[i **]** specifies whether the transform coefficient level at scanning position i is non-zero as follows:

– If significant_coeff_flag[i] is equal to 0, the transform coefficient level at scanning position i is set equal to 0;

– Otherwise (significant_coeff_flag[i] is equal to 1), the transform coefficient level at scanning position i has a non-zero value.

last_significant_coeff_flag[i **]** specifies for the scanning position i whether there are non-zero transform coefficient levels for subsequent scanning positions i + 1 to maxNumCoeff − 1 as follows:

– If last_significant_coeff_flag[i] is equal to 1, all following transform coefficient levels (in scanning order) of the block have value equal to 0.

– Otherwise (last_significant_coeff_flag[i] is equal to 0), there are further non-zero transform coefficient levels along the scanning path.

coeff_abs_level_minus1[i **]** is the absolute value of a transform coefficient level minus 1. The value of coeff_abs_level_minus1 is constrained by the limits in subclause 8.5.

coeff_sign_flag[i **]** specifies the sign of a transform coefficient level as follows:

– If coeff_sign_flag is equal to 0, the corresponding transform coefficient level has a positive value.

– Otherwise (coeff_sign_flag is equal to 1), the corresponding transform coefficient level has a negative value.

coeffLevel contains maxNumCoeff transform coefficient levels for the current list of transform coefficient levels.

8 Decoding process

Outputs of this process are decoded samples of the current picture (sometimes referred to by the variable CurrPic).

Depending on the value of chroma_format_idc, the number of sample arrays of the current picture is as follows:

– If chroma_format_idc is equal to 0, the current picture consists of 1 sample array S_L.

– Otherwise (chroma_format_idc is not equal to 0), the current picture consists of 3 sample arrays S_L, S_{Cb}, S_{Cr}.

This clause describes the decoding process, given syntax elements and upper-case variables from clause 7.

The decoding process is specified such that all decoders shall produce numerically identical results. Any decoding process that produces identical results to the process described here conforms to the decoding process requirements of this Recommendation | International Standard.

Each picture referred to in this clause is a complete primary coded picture or part of a primary coded picture. Each slice referred to in this clause is a slice of a primary coded picture. Each slice data partition referred to in this clause is a slice data partition of a primary coded picture.

Depending on the value of separate_colour_plane_flag, the decoding process is structured as follows:

– If separate_colour_plane_flag is equal to 0, the decoding process is invoked a single time with the current picture being the output.

– Otherwise (separate_colour_plane_flag is equal to 1), the decoding process is invoked three times. Inputs to the decoding process are all NAL units of the primary coded picture with identical value of colour_plane_id. The decoding process of NAL units with a particular value of colour_plane_id is specified as if only a coded video sequence with monochrome colour format with that particular value of colour_plane_id would be present in the bitstream. The output of each of the three decoding processes is assigned to the 3 sample arrays of the current picture with the NAL units with colour_plane_id equal to 0 being assigned to S_L, the NAL units with colour_plane_id equal to 1 being assigned to S_{Cb}, and the NAL units with colour_plane_id equal to 2 being assigned to S_{Cr}.

  NOTE – The variable ChromaArrayType is derived as 0 when separate_colour_plane_flag is equal to 1 and chroma_format_idc is equal to 3. In the decoding process, the value of this variable is evaluated resulting in operations identical to that of monochrome pictures with chroma_format_idc being equal to 0.

An overview of the decoding process is given as follows:

1. The decoding of NAL units is specified in subclause 8.1.

2. The processes in subclause 8.2 specify decoding processes using syntax elements in the slice layer and above:

   – Variables and functions relating to picture order count are derived in subclause 8.2.1. (only needed to be invoked for one slice of a picture)

   – Variables and functions relating to the macroblock to slice group map are derived in subclause 8.2.2. (only needed to be invoked for one slice of a picture)

   – The method of combining the various slice data partitions when slice data partitioning is used is described in subclause 8.2.3.

– When the frame_num of the current picture is not equal to PrevRefFrameNum and is not equal to (PrevRefFrameNum + 1) % MaxFrameNum, the decoding process for gaps in frame_num is performed according to subclause 8.2.5.2 prior to the decoding of any slices of the current picture.

– At the beginning of the decoding process for each P, SP, or B slice, the decoding process for reference picture lists construction specified in subclause 8.2.4 is invoked for derivation of reference picture list 0 (RefPicList0), and when decoding a B slice, reference picture list 1 (RefPicList1).

– When the current picture is a reference picture and after all slices of the current picture have been decoded, the decoded reference picture marking process in subclause 8.2.5 specifies how the current picture is used in the decoding process of inter prediction in later decoded pictures.

3. The processes in subclauses 8.3, 8.4, 8.5, 8.6, and 8.7 specify decoding processes using syntax elements in the macroblock layer and above.

   – The intra prediction process for I and SI macroblocks, except for I_PCM macroblocks as specified in subclause 8.3, has intra prediction samples as its output. For I_PCM macroblocks subclause 8.3 directly specifies a picture construction process. The output are constructed samples prior to the deblocking filter process.

   – The inter prediction process for P and B macroblocks is specified in subclause 8.4 with inter prediction samples being the output.

   – The transform coefficient decoding process and picture construction process prior to deblocking filter process are specified in subclause 8.5. That process derives samples for I and B macroblocks and for P macroblocks in P slices. The output are constructed samples prior to the deblocking filter process.

   – The decoding process for P macroblocks in SP slices or SI macroblocks is specified in subclause 8.6. That process derives samples for P macroblocks in SP slices and for SI macroblocks. The output are constructed samples prior to the deblocking filter process.

   – The constructed samples prior to the deblocking filter process that are next to the edges of blocks and macroblocks are processed by a deblocking filter as specified in subclause 8.7 with the output being the decoded samples.

8.1 NAL unit decoding process

Inputs to this process are NAL units.

Outputs of this process are the RBSP syntax structures encapsulated within the NAL units.

The decoding process for each NAL unit extracts the RBSP syntax structure from the NAL unit and then operates the decoding processes specified for the RBSP syntax structure in the NAL unit as follows.

Subclause 8.2 describes the decoding process for NAL units with nal_unit_type equal to 1 through 5.

Subclause 8.3 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1, 2, and 5.

Subclause 8.4 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1 and 2.

Subclause 8.5 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1 and 3 to 5.

Subclause 8.6 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1 and 3 to 5.

Subclause 8.7 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1 to 5.

NAL units with nal_unit_type equal to 7 and 8 contain sequence parameter sets and picture parameter sets, respectively. Picture parameter sets are used in the decoding processes of other NAL units as determined by reference to a picture parameter set within the slice headers of each picture. Sequence parameter sets are used in the decoding processes of other NAL units as determined by reference to a sequence parameter set within the picture parameter sets of each sequence.

No normative decoding process is specified for NAL units with nal_unit_type equal to 6, 9, 10, 11, and 12.

8.2 Slice decoding process

8.2.1 Decoding process for picture order count

Outputs of this process are TopFieldOrderCnt (if applicable) and BottomFieldOrderCnt (if applicable).

Picture order counts are used to determine initial picture orderings for reference pictures in the decoding of B slices (see subclauses 8.2.4.2.3 and 8.2.4.2.4), to determine co-located pictures (see subclause 8.4.1.2.1) for deriving motion parameters in temporal or spatial direct mode, to represent picture order differences between frames or fields for motion vector derivation in temporal direct mode (see subclause 8.4.1.2.3), for implicit mode weighted prediction in B slices (see subclause 8.4.2.3.2), and for decoder conformance checking (see subclause C.4).

Picture order count information is derived for every frame, field (whether decoded from a coded field or as a part of a decoded frame), or complementary field pair as follows:

– Each coded frame is associated with two picture order counts, called TopFieldOrderCnt and BottomFieldOrderCnt for its top field and bottom field, respectively.

– Each coded field is associated with a picture order count, called TopFieldOrderCnt for a coded top field and BottomFieldOrderCnt for a bottom field.

– Each complementary field pair is associated with two picture order counts, which are the TopFieldOrderCnt for its coded top field and the BottomFieldOrderCnt for its coded bottom field, respectively.

TopFieldOrderCnt and BottomFieldOrderCnt indicate the picture order of the corresponding top field or bottom field relative to the first output field of the previous IDR picture or the previous reference picture including a memory_management_control_operation equal to 5 in decoding order.

TopFieldOrderCnt and BottomFieldOrderCnt are derived by invoking one of the decoding processes for picture order count type 0, 1, and 2 in subclauses 8.2.1.1, 8.2.1.2, and 8.2.1.3, respectively. When the current picture includes a memory_management_control_operation equal to 5, after the decoding of the current picture, tempPicOrderCnt is set equal to PicOrderCnt(CurrPic), TopFieldOrderCnt of the current picture (if any) is set equal to TopFieldOrderCnt − tempPicOrderCnt, and BottomFieldOrderCnt of the current picture (if any) is set equal to BottomFieldOrderCnt − tempPicOrderCnt.

NOTE 1 – When the decoding process for a picture currPic that includes a memory_management_control_operation equal to 5 refers to the values of TopFieldOrderCnt (if applicable) or BottomFieldOrderCnt (if applicable) for the picture currPic (including references to the function PicOrderCnt( ) with the picture currPic as the argument and references to the function DiffPicOrderCnt( ) with one of the arguments being currPic), the values of TopFieldOrderCnt (if applicable) and BottomFieldOrderCnt (if applicable) that are derived as specified in subclauses 8.2.1.1, 8.2.1.2, and 8.2.1.3 for the picture currPic are used. When the decoding process for a picture refers to the values TopFieldOrderCnt (if applicable) or BottomFieldOrderCnt (if applicable) of the previous picture prevMmco5Pic in decoding order that includes a memory_management_control_operation equal to 5 (including references via the functions PicOrderCnt( ) or DiffPicOrderCnt( )), the values of TopFieldOrderCnt (if applicable) and BottomFieldOrderCnt (if applicable) that are used for the picture prevMmco5Pic are the values after the modification specified in the paragraph above (resulting in TopFieldOrderCnt and/or BottomFieldOrderCnt equal to 0).

The bitstream shall not contain data that result in Min(TopFieldOrderCnt, BottomFieldOrderCnt) not equal to 0 for a coded IDR frame, TopFieldOrderCnt not equal to 0 for a coded IDR top field, or BottomFieldOrderCnt not equal to 0 for a coded IDR bottom field. Thus, at least one of TopFieldOrderCnt and BottomFieldOrderCnt shall be equal to 0 for the fields of a coded IDR frame.

When the current picture is not an IDR picture, the following applies:

1) Consider the list variable listD containing as elements the TopFieldOrderCnt and BottomFieldOrderCnt values associated with the list of pictures including all of the following:

   a. The first picture in the list is the previous picture of any of the following types:

      – an IDR picture,

      – a picture containing a memory_management_control_operation equal to 5.

   b. The following additional pictures:

      – If pic_order_cnt_type is equal to 0, all other pictures that follow in decoding order after the first picture in the list and are not "non-existing" frames inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2 and either precede the current picture in decoding order or are the current picture. When pic_order_cnt_type is equal to 0 and the current picture is not a "non-existing" frame inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2, the current picture is included in listD prior to the invoking of the decoded reference picture marking process.

– Otherwise (pic_order_cnt_type is not equal to 0), all other pictures that follow in decoding order after the first picture in the list and either precede the current picture in decoding order or are the current picture. When pic_order_cnt_type is not equal to 0, the current picture is included in listD prior to the invoking of the decoded reference picture marking process.

2) Consider the list variable listO which contains the elements of listD sorted in ascending order. listO shall not contain any of the following:

   – a pair of TopFieldOrderCnt and BottomFieldOrderCnt for a frame or complementary field pair that are not at consecutive positions in listO,

   – a TopFieldOrderCnt that has a value equal to another TopFieldOrderCnt,

   – a BottomFieldOrderCnt that has a value equal to another BottomFieldOrderCnt,

   – a BottomFieldOrderCnt that has a value equal to a TopFieldOrderCnt unless the BottomFieldOrderCnt and TopFieldOrderCnt belong to the same coded frame or complementary field pair.

The bitstream shall not contain data that result in values of TopFieldOrderCnt, BottomFieldOrderCnt, PicOrderCntMsb, or FrameNumOffset used in the decoding process as specified in subclauses 8.2.1.1 to 8.2.1.3 that exceed the range of values from -2^{31} to $2^{31} - 1$, inclusive.

The function PicOrderCnt(picX) is specified as follows:

```
if( picX is a frame or a complementary field pair )
   PicOrderCnt( picX ) = Min( TopFieldOrderCnt, BottomFieldOrderCnt ) of the frame or complementary field
pair picX
else if( picX is a top field )
   PicOrderCnt( picX ) = TopFieldOrderCnt of field picX                                        (8-1)
else if( picX is a bottom field )
   PicOrderCnt( picX ) = BottomFieldOrderCnt of field picX
```

Then DiffPicOrderCnt(picA, picB) is specified as follows:

$$\text{DiffPicOrderCnt(picA, picB)} = \text{PicOrderCnt(picA)} - \text{PicOrderCnt(picB)} \quad (8\text{-}2)$$

The bitstream shall not contain data that result in values of DiffPicOrderCnt(picA, picB) used in the decoding process that exceed the range of -2^{15} to $2^{15} - 1$, inclusive.

NOTE 2 – Let X be the current picture and Y and Z be two other pictures in the same sequence, Y and Z are considered to be in the same output order direction from X when both DiffPicOrderCnt(X, Y) and DiffPicOrderCnt(X, Z) are positive or both are negative.

NOTE 3 – Many encoders assign TopFieldOrderCnt and BottomFieldOrderCnt proportional to the sampling time of the corresponding field (which is either a coded field or a field of a coded frame) relative to the sampling time of the first output field of the previous IDR picture or the previous reference picture (in decoding order) that includes a memory_management_control_operation equal to 5.

When the current picture includes a memory_management_control_operation equal to 5, PicOrderCnt(CurrPic) shall be greater than PicOrderCnt(any other picture in listD).

8.2.1.1 Decoding process for picture order count type 0

This process is invoked when pic_order_cnt_type is equal to 0.

Input to this process is PicOrderCntMsb of the previous reference picture in decoding order as specified in this subclause.

Outputs of this process are either or both TopFieldOrderCnt or BottomFieldOrderCnt.

The variables prevPicOrderCntMsb and prevPicOrderCntLsb are derived as follows:

– If the current picture is an IDR picture, prevPicOrderCntMsb is set equal to 0 and prevPicOrderCntLsb is set equal to 0.

– Otherwise (the current picture is not an IDR picture), the following applies:

  – If the previous reference picture in decoding order included a memory_management_control_operation equal to 5, the following applies:

    – If the previous reference picture in decoding order is not a bottom field, prevPicOrderCntMsb is set equal to 0 and prevPicOrderCntLsb is set equal to the value of TopFieldOrderCnt for the previous reference picture in decoding order.

– Otherwise (the previous reference picture in decoding order is a bottom field), prevPicOrderCntMsb is set equal to 0 and prevPicOrderCntLsb is set equal to 0.

– Otherwise (the previous reference picture in decoding order did not include a memory_management_control_operation equal to 5), prevPicOrderCntMsb is set equal to PicOrderCntMsb of the previous reference picture in decoding order and prevPicOrderCntLsb is set equal to the value of pic_order_cnt_lsb of the previous reference picture in decoding order.

PicOrderCntMsb of the current picture is derived as specified by the following pseudo-code:

```
if( ( pic_order_cnt_lsb < prevPicOrderCntLsb ) &&
    ( ( prevPicOrderCntLsb − pic_order_cnt_lsb ) >= ( MaxPicOrderCntLsb / 2 ) ) )
    PicOrderCntMsb = prevPicOrderCntMsb + MaxPicOrderCntLsb                    (8-3)
else if( ( pic_order_cnt_lsb > prevPicOrderCntLsb ) &&
        ( ( pic_order_cnt_lsb − prevPicOrderCntLsb ) > ( MaxPicOrderCntLsb / 2 ) ) )
    PicOrderCntMsb = prevPicOrderCntMsb − MaxPicOrderCntLsb
else
    PicOrderCntMsb = prevPicOrderCntMsb
```

When the current picture is not a bottom field, TopFieldOrderCnt is derived as

```
TopFieldOrderCnt = PicOrderCntMsb + pic_order_cnt_lsb                          (8-4)
```

When the current picture is not a top field, BottomFieldOrderCnt is derived as specified by the following pseudo-code:

```
if( !field_pic_flag )
    BottomFieldOrderCnt = TopFieldOrderCnt + delta_pic_order_cnt_bottom
else                                                                           (8-5)
    BottomFieldOrderCnt = PicOrderCntMsb + pic_order_cnt_lsb
```

8.2.1.2 Decoding process for picture order count type 1

This process is invoked when pic_order_cnt_type is equal to 1.

Input to this process is FrameNumOffset of the previous picture in decoding order as specified in this subclause.

Outputs of this process are either or both TopFieldOrderCnt or BottomFieldOrderCnt.

The values of TopFieldOrderCnt and BottomFieldOrderCnt are derived as specified in this subclause. Let prevFrameNum be equal to the frame_num of the previous picture in decoding order.

When the current picture is not an IDR picture, the variable prevFrameNumOffset is derived as follows:

– If the previous picture in decoding order included a memory_management_control_operation equal to 5, prevFrameNumOffset is set equal to 0.

– Otherwise (the previous picture in decoding order did not include a memory_management_control_operation equal to 5), prevFrameNumOffset is set equal to the value of FrameNumOffset of the previous picture in decoding order.

NOTE – When gaps_in_frame_num_value_allowed_flag is equal to 1, the previous picture in decoding order may be a "non-existing" frame inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2.

The variable FrameNumOffset is derived as specified by the following pseudo-code:

```
if( IdrPicFlag = = 1 )
    FrameNumOffset = 0
else if( prevFrameNum > frame_num )                                            (8-6)
    FrameNumOffset = prevFrameNumOffset + MaxFrameNum
else
    FrameNumOffset = prevFrameNumOffset
```

The variable absFrameNum is derived as specified by the following pseudo-code:

```
if( num_ref_frames_in_pic_order_cnt_cycle != 0 )
    absFrameNum = FrameNumOffset + frame_num
else                                                                           (8-7)
    absFrameNum = 0
if( nal_ref_idc = = 0 && absFrameNum > 0 )
    absFrameNum = absFrameNum − 1
```

When absFrameNum > 0, picOrderCntCycleCnt and frameNumInPicOrderCntCycle are derived as

```
picOrderCntCycleCnt = ( absFrameNum − 1 ) / num_ref_frames_in_pic_order_cnt_cycle
frameNumInPicOrderCntCycle = ( absFrameNum − 1 ) % num_ref_frames_in_pic_order_cnt_cycle            (8-8)
```

The variable expectedPicOrderCnt is derived as specified by the following pseudo-code:

```
if( absFrameNum > 0 ){
    expectedPicOrderCnt = picOrderCntCycleCnt * ExpectedDeltaPerPicOrderCntCycle
    for( i = 0; i <= frameNumInPicOrderCntCycle; i++ )
        expectedPicOrderCnt = expectedPicOrderCnt + offset_for_ref_frame[ i ]
} else
    expectedPicOrderCnt = 0
if( nal_ref_idc  = =  0 )                                                                          (8-9)
    expectedPicOrderCnt = expectedPicOrderCnt + offset_for_non_ref_pic
```

The variables TopFieldOrderCnt or BottomFieldOrderCnt are derived as specified by the following pseudo-code:

```
if( !field_pic_flag ) {
    TopFieldOrderCnt = expectedPicOrderCnt + delta_pic_order_cnt[ 0 ]
    BottomFieldOrderCnt = TopFieldOrderCnt +
              offset_for_top_to_bottom_field + delta_pic_order_cnt[ 1 ]                           (8-10)
} else if( !bottom_field_flag )
    TopFieldOrderCnt = expectedPicOrderCnt + delta_pic_order_cnt[ 0 ]
else
    BottomFieldOrderCnt = expectedPicOrderCnt + offset_for_top_to_bottom_field + delta_pic_order_cnt[ 0 ]
```

8.2.1.3 Decoding process for picture order count type 2

This process is invoked when pic_order_cnt_type is equal to 2.

Outputs of this process are either or both TopFieldOrderCnt or BottomFieldOrderCnt.

Let prevFrameNum be equal to the frame_num of the previous picture in decoding order.

When the current picture is not an IDR picture, the variable prevFrameNumOffset is derived as follows:

– If the previous picture in decoding order included a memory_management_control_operation equal to 5, prevFrameNumOffset is set equal to 0.

– Otherwise (the previous picture in decoding order did not include a memory_management_control_operation equal to 5), prevFrameNumOffset is set equal to the value of FrameNumOffset of the previous picture in decoding order.

NOTE 1 – When gaps_in_frame_num_value_allowed_flag is equal to 1, the previous picture in decoding order may be a "non-existing" frame inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2.

The variable FrameNumOffset is derived as specified by the following pseudo-code:

```
if( IdrPicFlag  = =  1 )
    FrameNumOffset = 0
else if( prevFrameNum  >  frame_num )                                                              (8-11)
    FrameNumOffset = prevFrameNumOffset + MaxFrameNum
else
    FrameNumOffset = prevFrameNumOffset
```

The variable tempPicOrderCnt is derived as specified by the following pseudo-code:

```
if( IdrPicFlag  = =  1 )
    tempPicOrderCnt = 0
else if( nal_ref_idc  = =  0 )                                                                     (8-12)
    tempPicOrderCnt = 2 * ( FrameNumOffset + frame_num ) − 1
else
    tempPicOrderCnt = 2 * ( FrameNumOffset + frame_num )
```

The variables TopFieldOrderCnt or BottomFieldOrderCnt are derived as specified by the following pseudo-code:

```
if( !field_pic_flag ) {
    TopFieldOrderCnt = tempPicOrderCnt
    BottomFieldOrderCnt = tempPicOrderCnt                                    (8-13)
} else if( bottom_field_flag )
    BottomFieldOrderCnt = tempPicOrderCnt
else
    TopFieldOrderCnt = tempPicOrderCnt
```

NOTE 2 – Picture order count type 2 cannot be used in a coded video sequence that contains consecutive non-reference pictures that would result in more than one of these pictures having the same value of TopFieldOrderCnt or more than one of these pictures having the same value of BottomFieldOrderCnt.

NOTE 3 – Picture order count type 2 results in an output order that is the same as the decoding order.

8.2.2 Decoding process for macroblock to slice group map

Inputs to this process are the active picture parameter set and the slice header of the slice to be decoded.

Output of this process is a macroblock to slice group map MbToSliceGroupMap.

This process is invoked at the start of every slice.

NOTE – The output of this process is equal for all slices of a picture.

When num_slice_groups_minus1 is equal to 1 and slice_group_map_type is equal to 3, 4, or 5, slice groups 0 and 1 have a size and shape determined by slice_group_change_direction_flag as shown in Table 8-1 and specified in subclauses 8.2.2.4 to 8.2.2.6.

Table 8-1 – Refined slice group map type

slice_group_map_type	slice_group_change_direction_flag	refined slice group map type
3	0	Box-out clockwise
3	1	Box-out counter-clockwise
4	0	Raster scan
4	1	Reverse raster scan
5	0	Wipe right
5	1	Wipe left

In such a case, MapUnitsInSliceGroup0 slice group map units in the specified growth order are allocated for slice group 0 and the remaining PicSizeInMapUnits − MapUnitsInSliceGroup0 slice group map units of the picture are allocated for slice group 1.

When num_slice_groups_minus1 is equal to 1 and slice_group_map_type is equal to 4 or 5, the variable sizeOfUpperLeftGroup is defined as follows:

```
sizeOfUpperLeftGroup = ( slice_group_change_direction_flag ?
                         ( PicSizeInMapUnits − MapUnitsInSliceGroup0 ) : MapUnitsInSliceGroup0 )    (8-14)
```

The mapUnitToSliceGroupMap array is derived as follows:

– If num_slice_groups_minus1 is equal to 0, the map unit to slice group map is generated for all i ranging from 0 to PicSizeInMapUnits − 1, inclusive, as specified by

```
mapUnitToSliceGroupMap[ i ] = 0                                              (8-15)
```

– Otherwise (num_slice_groups_minus1 is not equal to 0), mapUnitToSliceGroupMap is derived as follows:
  – If slice_group_map_type is equal to 0, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.1 applies.
  – Otherwise, if slice_group_map_type is equal to 1, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.2 applies.

– Otherwise, if slice_group_map_type is equal to 2, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.3 applies.

– Otherwise, if slice_group_map_type is equal to 3, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.4 applies.

– Otherwise, if slice_group_map_type is equal to 4, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.5 applies.

– Otherwise, if slice_group_map_type is equal to 5, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.6 applies.

– Otherwise (slice_group_map_type is equal to 6), the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.7 applies.

After derivation of the mapUnitToSliceGroupMap, the process specified in subclause 8.2.2.8 is invoked to convert the map unit to slice group map mapUnitToSliceGroupMap to the macroblock to slice group map MbToSliceGroupMap. After derivation of the macroblock to slice group map as specified in subclause 8.2.2.8, the function NextMbAddress(n) is defined as the value of the variable nextMbAddress derived as specified by the following pseudo-code:

```
i = n + 1
while( i < PicSizeInMbs  &&  MbToSliceGroupMap[ i ]  !=  MbToSliceGroupMap[ n ] )
    i++;
nextMbAddress = i                                                          (8-16)
```

8.2.2.1 Specification for interleaved slice group map type

The specifications in this subclause apply when slice_group_map_type is equal to 0.

The map unit to slice group map is generated as specified by the following pseudo-code:

```
i = 0
do
    for( iGroup = 0; iGroup <= num_slice_groups_minus1 && i < PicSizeInMapUnits;
         i += run_length_minus1[ iGroup++ ] + 1 )
        for( j = 0; j <= run_length_minus1[ iGroup ] && i + j < PicSizeInMapUnits; j++ )
            mapUnitToSliceGroupMap[ i + j ] = iGroup                          (8-17)
while( i < PicSizeInMapUnits )
```

8.2.2.2 Specification for dispersed slice group map type

The specifications in this subclause apply when slice_group_map_type is equal to 1.

The map unit to slice group map is generated as specified by the following pseudo-code:

```
for( i = 0; i < PicSizeInMapUnits; i++ )
    mapUnitToSliceGroupMap[ i ] = ( ( i % PicWidthInMbs ) +
                                   ( ( ( i / PicWidthInMbs ) * ( num_slice_groups_minus1 + 1 ) ) / 2 ) )
                                  % ( num_slice_groups_minus1 + 1 )                     (8-18)
```

8.2.2.3 Specification for foreground with left-over slice group map type

The specifications in this subclause apply when slice_group_map_type is equal to 2.

The map unit to slice group map is generated as specified by the following pseudo-code:

```
for( i = 0; i < PicSizeInMapUnits; i++ )
    mapUnitToSliceGroupMap[ i ] = num_slice_groups_minus1
for( iGroup = num_slice_groups_minus1 − 1; iGroup >= 0; iGroup− − ) {
    yTopLeft = top_left[ iGroup ] / PicWidthInMbs
    xTopLeft = top_left[ iGroup ] % PicWidthInMbs
    yBottomRight = bottom_right[ iGroup ] / PicWidthInMbs
    xBottomRight = bottom_right[ iGroup ] % PicWidthInMbs
    for( y = yTopLeft; y <= yBottomRight; y++ )
        for( x = xTopLeft; x <= xBottomRight; x++ )
            mapUnitToSliceGroupMap[ y * PicWidthInMbs + x ] = iGroup                (8-19)
}
```

NOTE – The rectangles may overlap. Slice group 0 contains the macroblocks that are within the rectangle specified by top_left[0] and bottom_right[0]. A slice group having slice group ID greater than 0 and less than num_slice_groups_minus1 contains the macroblocks that are within the specified rectangle for that slice group that are not within the rectangle specified for any slice group having a smaller slice group ID. The slice group with slice group ID equal to num_slice_groups_minus1 contains the macroblocks that are not in the other slice groups.

8.2.2.4 Specification for box-out slice group map types

The specifications in this subclause apply when slice_group_map_type is equal to 3.

The map unit to slice group map is generated as specified by

```
for( i = 0; i < PicSizeInMapUnits; i++ )
    mapUnitToSliceGroupMap[ i ] = 1
x = ( PicWidthInMbs − slice_group_change_direction_flag ) / 2
y = ( PicHeightInMapUnits − slice_group_change_direction_flag ) / 2
( leftBound, topBound ) = ( x, y )
( rightBound, bottomBound ) = ( x, y )
( xDir, yDir ) = ( slice_group_change_direction_flag − 1, slice_group_change_direction_flag )
for( k = 0; k < MapUnitsInSliceGroup0; k += mapUnitVacant ) {
    mapUnitVacant = ( mapUnitToSliceGroupMap[ y * PicWidthInMbs + x ] = = 1 )
    if( mapUnitVacant )
        mapUnitToSliceGroupMap[ y * PicWidthInMbs + x ] = 0                    (8-20)
    if( xDir = = −1 && x = = leftBound ) {
        leftBound = Max( leftBound − 1, 0 )
        x = leftBound
        ( xDir, yDir ) = ( 0, 2 * slice_group_change_direction_flag − 1 )
    } else if( xDir = = 1 && x = = rightBound ) {
        rightBound = Min( rightBound + 1, PicWidthInMbs − 1 )
        x = rightBound
        ( xDir, yDir ) = ( 0, 1 − 2 * slice_group_change_direction_flag )
    } else if( yDir = = −1 && y = = topBound ) {
        topBound = Max( topBound − 1, 0 )
        y = topBound
        ( xDir, yDir ) = ( 1 − 2 * slice_group_change_direction_flag, 0 )
    } else if( yDir = = 1 && y = = bottomBound ) {
        bottomBound = Min( bottomBound + 1, PicHeightInMapUnits − 1 )
        y = bottomBound
        ( xDir, yDir ) = ( 2 * slice_group_change_direction_flag − 1, 0 )
    } else
        ( x, y ) = ( x + xDir, y + yDir )
}
```

8.2.2.5 Specification for raster scan slice group map types

The specifications in this subclause apply when slice_group_map_type is equal to 4.

The map unit to slice group map is generated as specified by

```
for( i = 0; i < PicSizeInMapUnits; i++ )
    if( i < sizeOfUpperLeftGroup )
        mapUnitToSliceGroupMap[ i ] = slice_group_change_direction_flag
    else                                                                       (8-21)
        mapUnitToSliceGroupMap[ i ] = 1 − slice_group_change_direction_flag
```

8.2.2.6 Specification for wipe slice group map types

The specifications in this subclause apply when slice_group_map_type is equal to 5.

The map unit to slice group map is generated as specified by

```
k = 0;
for( j = 0; j < PicWidthInMbs; j++ )
    for( i = 0; i < PicHeightInMapUnits; i++ )
        if( k++ < sizeOfUpperLeftGroup )
            mapUnitToSliceGroupMap[ i * PicWidthInMbs + j ] = slice_group_change_direction_flag
```

else (8-22)

mapUnitToSliceGroupMap[i * PicWidthInMbs + j] = 1 − slice_group_change_direction_flag

8.2.2.7 Specification for explicit slice group map type

The specifications in this subclause apply when slice_group_map_type is equal to 6.

The map unit to slice group map is generated as specified by

mapUnitToSliceGroupMap[i] = slice_group_id[i] (8-23)

for all i ranging from 0 to PicSizeInMapUnits − 1, inclusive.

8.2.2.8 Specification for conversion of map unit to slice group map to macroblock to slice group map

For each value of i ranging from 0 to PicSizeInMbs − 1, inclusive, the macroblock to slice group map is specified as follows:

– If frame_mbs_only_flag is equal to 1 or field_pic_flag is equal to 1, the macroblock to slice group map is specified by

MbToSliceGroupMap[i] = mapUnitToSliceGroupMap[i] (8-24)

– Otherwise, if MbaffFrameFlag is equal to 1, the macroblock to slice group map is specified by

MbToSliceGroupMap[i] = mapUnitToSliceGroupMap[i / 2] (8-25)

– Otherwise (frame_mbs_only_flag is equal to 0 and mb_adaptive_frame_field_flag is equal to 0 and field_pic_flag is equal to 0), the macroblock to slice group map is specified by

MbToSliceGroupMap[i] = mapUnitToSliceGroupMap[(i / (2 * PicWidthInMbs)) * PicWidthInMbs
+ (i % PicWidthInMbs)] (8-26)

8.2.3 Decoding process for slice data partitions

Inputs to this process are:

– a slice data partition A layer RBSP,

– when syntax elements of category 3 are present in the slice data, a slice data partition B layer RBSP having the same slice_id as in the slice data partition A layer RBSP,

– when syntax elements of category 4 are present in the slice data, a slice data partition C layer RBSP having the same slice_id as in the slice data partition A layer RBSP.

NOTE 1 – The slice data partition B layer RBSP and slice data partition C layer RBSP need not be present.

Output of this process is a coded slice.

When slice data partitioning is not used, coded slices are represented by a slice layer without partitioning RBSP that contains a slice header followed by a slice data syntax structure that contains all the syntax elements of categories 2, 3, and 4 (see category column in subclause 7.3) of the macroblock data for the macroblocks of the slice.

When slice data partitioning is used, the macroblock data of a slice is partitioned into one to three partitions contained in separate NAL units. Partition A contains a slice data partition A header, and all syntax elements of category 2. Partition B, when present, contains a slice data partition B header and all syntax elements of category 3. Partition C, when present, contains a slice data partition C header and all syntax elements of category 4.

When slice data partitioning is used, the syntax elements of each category are parsed from a separate NAL unit, which need not be present when no symbols of the respective category exist. The decoding process shall process the slice data partitions of a coded slice in a manner equivalent to processing a corresponding slice layer without partitioning RBSP by extracting each syntax element from the slice data partition in which the syntax element appears depending on the slice data partition assignment in the syntax tables in subclause 7.3.

NOTE 2 – Syntax elements of category 3 are relevant to the decoding of residual data of I and SI macroblock types. Syntax elements of category 4 are relevant to the decoding of residual data of P and B macroblock types. Category 2 encompasses all other syntax elements related to the decoding of macroblocks, and their information is often denoted as header information. The slice data partition A header contains all the syntax elements of the slice header, and additionally a slice_id that are used to associate the slice data partitions B and C with the slice data partition A. The slice data partition B and C headers contain the slice_id syntax element that establishes their association with the slice data partition A of the slice.

8.2.4 Decoding process for reference picture lists construction

This process is invoked at the beginning of the decoding process for each P, SP, or B slice.

Decoded reference pictures are marked as "used for short-term reference" or "used for long-term reference" as specified by the bitstream and specified in subclause 8.2.5. Short-term reference pictures are identified by the value of frame_num. Long-term reference pictures are assigned a long-term frame index as specified by the bitstream and specified in subclause 8.2.5.

Subclause 8.2.4.1 is invoked to specify

– the assignment of variables FrameNum, FrameNumWrap, and PicNum to each of the short-term reference pictures, and

– the assignment of variable LongTermPicNum to each of the long-term reference pictures.

Reference pictures are addressed through reference indices as specified in subclause 8.4.2.1. A reference index is an index into a reference picture list. When decoding a P or SP slice, there is a single reference picture list RefPicList0. When decoding a B slice, there is a second independent reference picture list RefPicList1 in addition to RefPicList0.

At the beginning of the decoding process for each slice, reference picture list RefPicList0, and for B slices RefPicList1, are derived as specified by the following ordered steps:

1. An initial reference picture list RefPicList0 and for B slices RefPicList1 are derived as specified in subclause 8.2.4.2.
2. When ref_pic_list_modification_flag_l0 is equal to 1 or, when decoding a B slice, ref_pic_list_modification_flag_l1 is equal to 1, the initial reference picture list RefPicList0 and, for B slices, RefPicList1 are modified as specified in subclause 8.2.4.3.

   NOTE – The modification process for reference picture lists specified in subclause 8.2.4.3 allows the contents of RefPicList0 and for B slices RefPicList1 to be modified in a flexible fashion. In particular, it is possible for a picture that is currently marked "used for reference" to be inserted into RefPicList0 and for B slices RefPicList1 even when the picture is not in the initial reference picture list derived as specified in subclause 8.2.4.2.

The number of entries in the modified reference picture list RefPicList0 is num_ref_idx_l0_active_minus1 + 1, and for B slices the number of entries in the modified reference picture list RefPicList1 is num_ref_idx_l1_active_minus1 + 1. A reference picture may appear at more than one index in the modified reference picture lists RefPicList0 or RefPicList1.

8.2.4.1 Decoding process for picture numbers

This process is invoked when the decoding process for reference picture lists construction specified in subclause 8.2.4, the decoded reference picture marking process specified in subclause 8.2.5, or the decoding process for gaps in frame_num specified in subclause 8.2.5.2 is invoked.

The variables FrameNum, FrameNumWrap, PicNum, LongTermFrameIdx, and LongTermPicNum are used for the initialisation process for reference picture lists in subclause 8.2.4.2, the modification process for reference picture lists in subclause 8.2.4.3, the decoded reference picture marking process in subclause 8.2.5, and the decoding process for gaps in frame_num in subclause 8.2.5.2.

To each short-term reference picture the variables FrameNum and FrameNumWrap are assigned as follows. First, FrameNum is set equal to the syntax element frame_num that has been decoded in the slice header(s) of the corresponding short-term reference picture. Then the variable FrameNumWrap is derived as

```
if( FrameNum > frame_num )
    FrameNumWrap = FrameNum − MaxFrameNum                                  (8-27)
else
    FrameNumWrap = FrameNum
```

where the value of frame_num used in Equation 8-27 is the frame_num in the slice header(s) for the current picture.

Each long-term reference picture has an associated value of LongTermFrameIdx (that was assigned to it as specified in subclause 8.2.5).

To each short-term reference picture a variable PicNum is assigned, and to each long-term reference picture a variable LongTermPicNum is assigned. The values of these variables depend on the value of field_pic_flag and bottom_field_flag for the current picture and they are set as follows:

– If field_pic_flag is equal to 0, the following ordered steps are specified:

1. For each short-term reference frame or complementary reference field pair:

$$\text{PicNum} = \text{FrameNumWrap} \quad (8\text{-}28)$$

2. For each long-term reference frame or long-term complementary reference field pair:

$$\text{LongTermPicNum} = \text{LongTermFrameIdx} \quad (8\text{-}29)$$

NOTE – When decoding a frame the value of MbaffFrameFlag has no influence on the derivations in subclauses 8.2.4.2, 8.2.4.3, and 8.2.5.

– Otherwise (field_pic_flag is equal to 1), the following ordered steps are specified:

1. For each short-term reference field the following applies:

   – If the reference field has the same parity as the current field

$$\text{PicNum} = 2 * \text{FrameNumWrap} + 1 \quad (8\text{-}30)$$

   – Otherwise (the reference field has the opposite parity of the current field),

$$\text{PicNum} = 2 * \text{FrameNumWrap} \quad (8\text{-}31)$$

2. For each long-term reference field the following applies:

   – If the reference field has the same parity as the current field

$$\text{LongTermPicNum} = 2 * \text{LongTermFrameIdx} + 1 \quad (8\text{-}32)$$

   – Otherwise (the reference field has the opposite parity of the current field),

$$\text{LongTermPicNum} = 2 * \text{LongTermFrameIdx} \quad (8\text{-}33)$$

8.2.4.2 Initialisation process for reference picture lists

This initialisation process is invoked when decoding a P, SP, or B slice header.

RefPicList0 and RefPicList1 have initial entries as specified in subclauses 8.2.4.2.1 through 8.2.4.2.5.

When the number of entries in the initial RefPicList0 or RefPicList1 produced as specified in subclauses 8.2.4.2.1 through 8.2.4.2.5 is greater than num_ref_idx_l0_active_minus1 + 1 or num_ref_idx_l1_active_minus1 + 1, respectively, the extra entries past position num_ref_idx_l0_active_minus1 or num_ref_idx_l1_active_minus1 are discarded from the initial reference picture list.

When the number of entries in the initial RefPicList0 or RefPicList1 produced as specified in subclauses 8.2.4.2.1 through 8.2.4.2.5 is less than num_ref_idx_l0_active_minus1 + 1 or num_ref_idx_l1_active_minus1 + 1, respectively, the remaining entries in the initial reference picture list are set equal to "no reference picture".

8.2.4.2.1 Initialisation process for the reference picture list for P and SP slices in frames

This initialisation process is invoked when decoding a P or SP slice in a coded frame.

When this process is invoked, there shall be at least one reference frame or complementary reference field pair that is currently marked as "used for reference" (i.e., as "used for short-term reference" or "used for long-term reference") and is not marked as "non-existing".

The reference picture list RefPicList0 is ordered so that short-term reference frames and short-term complementary reference field pairs have lower indices than long-term reference frames and long-term complementary reference field pairs.

The short-term reference frames and complementary reference field pairs are ordered starting with the frame or complementary field pair with the highest PicNum value and proceeding through in descending order to the frame or complementary field pair with the lowest PicNum value.

The long-term reference frames and complementary reference field pairs are ordered starting with the frame or complementary field pair with the lowest LongTermPicNum value and proceeding through in ascending order to the frame or complementary field pair with the highest LongTermPicNum value.

NOTE – A non-paired reference field is not used for inter prediction for decoding a frame, regardless of the value of MbaffFrameFlag.

For example, when three reference frames are marked as "used for short-term reference" with PicNum equal to 300, 302, and 303 and two reference frames are marked as "used for long-term reference" with LongTermPicNum equal to 0 and 3, the initial index order is:

– RefPicList0[0] is set equal to the short-term reference picture with PicNum = 303,

– RefPicList0[1] is set equal to the short-term reference picture with PicNum = 302,

– RefPicList0[2] is set equal to the short-term reference picture with PicNum = 300,

– RefPicList0[3] is set equal to the long-term reference picture with LongTermPicNum = 0,

– RefPicList0[4] is set equal to the long-term reference picture with LongTermPicNum = 3.

8.2.4.2.2 Initialisation process for the reference picture list for P and SP slices in fields

This initialisation process is invoked when decoding a P or SP slice in a coded field.

When this process is invoked, there shall be at least one reference field (which can be a field of a reference frame) that is currently marked as "used for reference" (i.e., as "used for short-term reference" or "used for long-term reference") and is not marked as "non-existing".

Each field included in the reference picture list RefPicList0 has a separate index in the reference picture list RefPicList0.

NOTE – When decoding a field, there are effectively at least twice as many pictures available for referencing as there would be when decoding a frame at the same position in decoding order.

Two ordered lists of reference frames, refFrameList0ShortTerm and refFrameList0LongTerm, are derived as follows. For purposes of the formation of this list of frames, decoded reference frames, complementary reference field pairs, non-paired reference fields and reference frames in which a single field is marked "used for short-term reference" or "used for long-term reference" are all considered reference frames.

1. All frames having one or more fields marked "used for short-term reference" are included in the list of short-term reference frames refFrameList0ShortTerm. When the current field is the second field (in decoding order) of a complementary reference field pair and the first field is marked as "used for short-term reference", the first field is included in the list of short-term reference frames refFrameList0ShortTerm. refFrameList0ShortTerm is ordered starting with the reference frame with the highest FrameNumWrap value and proceeding through in descending order to the reference frame with the lowest FrameNumWrap value.

2. All frames having one or more fields marked "used for long-term reference" are included in the list of long-term reference frames refFrameList0LongTerm. When the current field is the second field (in decoding order) of a complementary reference field pair and the first field is marked as "used for long-term reference, the first field is included in the list of long-term reference frames refFrameList0LongTerm. refFrameList0LongTerm is ordered starting with the reference frame with the lowest LongTermFrameIdx value and proceeding through in ascending order to the reference frame with the highest LongTermFrameIdx value.

The process specified in subclause 8.2.4.2.5 is invoked with refFrameList0ShortTerm and refFrameList0LongTerm given as input and the output is assigned to RefPicList0.

8.2.4.2.3 Initialisation process for reference picture lists for B slices in frames

This initialisation process is invoked when decoding a B slice in a coded frame.

For purposes of the formation of the reference picture lists RefPicList0 and RefPicList1 the term reference entry refers in the following to decoded reference frames or complementary reference field pairs.

When this process is invoked, there shall be at least one reference entry that is currently marked as "used for reference" (i.e., as "used for short-term reference" or "used for long-term reference") and is not marked as "non-existing".

For B slices, the order of short-term reference entries in the reference picture lists RefPicList0 and RefPicList1 depends on output order, as given by PicOrderCnt( ). When pic_order_cnt_type is equal to 0, reference pictures that are marked as "non-existing" as specified in subclause 8.2.5.2 are not included in either RefPicList0 or RefPicList1.

NOTE 1 – When gaps_in_frame_num_value_allowed_flag is equal to 1, encoders should use reference picture list modification to ensure proper operation of the decoding process (particularly when pic_order_cnt_type is equal to 0, in which case PicOrderCnt( ) is not inferred for "non-existing" frames).

The reference picture list RefPicList0 is ordered such that short-term reference entries have lower indices than long-term reference entries. It is ordered as follows:

1. Let entryShortTerm be a variable ranging over all reference entries that are currently marked as "used for short-term reference". When some values of entryShortTerm are present having PicOrderCnt(entryShortTerm) less than PicOrderCnt(CurrPic), these values of entryShortTerm are placed at the beginning of refPicList0 in descending order of PicOrderCnt(entryShortTerm). All of the remaining values of entryShortTerm (when present) are then appended to refPicList0 in ascending order of PicOrderCnt(entryShortTerm).

2. The long-term reference entries are ordered starting with the long-term reference entry that has the lowest LongTermPicNum value and proceeding through in ascending order to the long-term reference entry that has the highest LongTermPicNum value.

The reference picture list RefPicList1 is ordered so that short-term reference entries have lower indices than long-term reference entries. It is ordered as follows:

1. Let entryShortTerm be a variable ranging over all reference entries that are currently marked as "used for short-term reference". When some values of entryShortTerm are present having PicOrderCnt(entryShortTerm) greater than PicOrderCnt(CurrPic), these values of entryShortTerm are placed at the beginning of refPicList1 in ascending order of PicOrderCnt(entryShortTerm). All of the remaining values of entryShortTerm (when present) are then appended to refPicList1 in descending order of PicOrderCnt(entryShortTerm).

2. Long-term reference entries are ordered starting with the long-term reference frame or complementary reference field pair that has the lowest LongTermPicNum value and proceeding through in ascending order to the long-term reference entry that has the highest LongTermPicNum value.

3. When the reference picture list RefPicList1 has more than one entry and RefPicList1 is identical to the reference picture list RefPicList0, the first two entries RefPicList1[0] and RefPicList1[1] are switched.

NOTE 2 – A non-paired reference field is not used for inter prediction of frames (independent of the value of MbaffFrameFlag).

8.2.4.2.4 Initialisation process for reference picture lists for B slices in fields

This initialisation process is invoked when decoding a B slice in a coded field.

When this process is invoked, there shall be at least one reference field (which can be a field of a reference frame) that is currently marked as "used for reference" (i.e., as "used for short-term reference" or "used for long-term reference") and is not marked as "non-existing".

When decoding a field, each field of a stored reference frame is identified as a separate reference picture with a unique index. The order of short-term reference pictures in the reference picture lists RefPicList0 and RefPicList1 depend on output order, as given by PicOrderCnt( ). When pic_order_cnt_type is equal to 0, reference pictures that are marked as "non-existing" as specified in subclause 8.2.5.2 are not included in either RefPicList0 or RefPicList1.

NOTE 1 – When gaps_in_frame_num_value_allowed_flag is equal to 1, encoders should use reference picture list modification to ensure proper operation of the decoding process (particularly when pic_order_cnt_type is equal to 0, in which case PicOrderCnt( ) is not inferred for "non-existing" frames).

NOTE 2 – When decoding a field, there are effectively at least twice as many pictures available for referencing as there would be when decoding a frame at the same position in decoding order.

Three ordered lists of reference frames, refFrameList0ShortTerm, refFrameList1ShortTerm and refFrameListLongTerm, are derived as follows. For purposes of the formation of these lists of frames the term reference entry refers in the following to decoded reference frames, complementary reference field pairs, or non-paired reference fields. When pic_order_cnt_type is equal to 0, the term reference entry does not refer to frames that are marked as "non-existing" as specified in subclause 8.2.5.2.

1. Let entryShortTerm be a variable ranging over all reference entries that are currently marked as "used for short-term reference". When some values of entryShortTerm are present having PicOrderCnt(entryShortTerm) less than or equal to PicOrderCnt(CurrPic), these values of entryShortTerm are placed at the beginning of refFrameList0ShortTerm in descending order of PicOrderCnt(entryShortTerm). All of the remaining values of entryShortTerm (when present) are then appended to refFrameList0ShortTerm in ascending order of PicOrderCnt(entryShortTerm).

   NOTE 3 – When the current field follows in decoding order a coded field fldPrev with which together it forms a complementary reference field pair, fldPrev is included into the list refFrameList0ShortTerm using PicOrderCnt(fldPrev) and the ordering method described in the previous sentence is applied.

2. Let entryShortTerm be a variable ranging over all reference entries that are currently marked as "used for short-term reference". When some values of entryShortTerm are present having

PicOrderCnt(entryShortTerm) greater than PicOrderCnt(CurrPic), these values of entryShortTerm are placed at the beginning of refFrameList1ShortTerm in ascending order of PicOrderCnt(entryShortTerm). All of the remaining values of entryShortTerm (when present) are then appended to refFrameList1ShortTerm in descending order of PicOrderCnt(entryShortTerm).

NOTE 4 – When the current field follows in decoding order a coded field fldPrev with which together it forms a complementary reference field pair, fldPrev is included into the list refFrameList1ShortTerm using PicOrderCnt(fldPrev) and the ordering method described in the previous sentence is applied.

3. refFrameListLongTerm is ordered starting with the reference entry having the lowest LongTermFrameIdx value and proceeding through in ascending order to the reference entry having highest LongTermFrameIdx value.

   NOTE 5 – When the current picture is the second field of a complementary field pair and the first field of the complementary field pair is marked as "used for long-term reference", the first field is included into the list refFrameListLongTerm. A reference entry in which only one field is marked as "used for long-term reference" is included into the list refFrameListLongTerm.

The process specified in subclause 8.2.4.2.5 is invoked with refFrameList0ShortTerm and refFrameListLongTerm given as input and the output is assigned to RefPicList0.

The process specified in subclause 8.2.4.2.5 is invoked with refFrameList1ShortTerm and refFrameListLongTerm given as input and the output is assigned to RefPicList1.

When the reference picture list RefPicList1 has more than one entry and RefPicList1 is identical to the reference picture list RefPicList0, the first two entries RefPicList1[0] and RefPicList1[1] are switched.

8.2.4.2.5 Initialisation process for reference picture lists in fields

Inputs of this process are the reference frame lists refFrameListXShortTerm (with X may be 0 or 1) and refFrameListLongTerm.

The reference picture list RefPicListX is a list ordered such that short-term reference fields have lower indices than long-term reference fields. Given the reference frame lists refFrameListXShortTerm and refFrameListLongTerm, it is derived as specified by the following ordered steps:

1. Short-term reference fields are ordered by selecting reference fields from the ordered list of frames refFrameListXShortTerm by alternating between fields of differing parity, starting with a field that has the same parity as the current field (when present). When one field of a reference frame was not decoded or is not marked as "used for short-term reference", the missing field is ignored and instead the next available stored reference field of the chosen parity from the ordered list of frames refFrameListXShortTerm is inserted into RefPicListX. When there are no more short-term reference fields of the alternate parity in the ordered list of frames refFrameListXShortTerm, the next not yet indexed fields of the available parity are inserted into RefPicListX in the order in which they occur in the ordered list of frames refFrameListXShortTerm.

2. Long-term reference fields are ordered by selecting reference fields from the ordered list of frames refFrameListLongTerm by alternating between fields of differing parity, starting with a field that has the same parity as the current field (when present). When one field of a reference frame was not decoded or is not marked as "used for long-term reference", the missing field is ignored and instead the next available stored reference field of the chosen parity from the ordered list of frames refFrameListLongTerm is inserted into RefPicListX. When there are no more long-term reference fields of the alternate parity in the ordered list of frames refFrameListLongTerm, the next not yet indexed fields of the available parity are inserted into RefPicListX in the order in which they occur in the ordered list of frames refFrameListLongTerm.

8.2.4.3 Modification process for reference picture lists

When ref_pic_list_modification_flag_l0 is equal to 1, the following applies:

1. Let refIdxL0 be an index into the reference picture list RefPicList0. It is initially set equal to 0.

2. The corresponding syntax elements modification_of_pic_nums_idc are processed in the order they occur in the bitstream. For each of these syntax elements, the following applies:

   – If modification_of_pic_nums_idc is equal to 0 or equal to 1, the process specified in subclause 8.2.4.3.1 is invoked with refIdxL0 as input, and the output is assigned to refIdxL0.

   – Otherwise, if modification_of_pic_nums_idc is equal to 2, the process specified in subclause 8.2.4.3.2 is invoked with refIdxL0 as input, and the output is assigned to refIdxL0.

   – Otherwise (modification_of_pic_nums_idc is equal to 3), the modification process for reference picture list RefPicList0 is finished.

When the current slice is a B slice and ref_pic_list_modification_flag_l1 is equal to 1, the following applies:

1. Let refIdxL1 be an index into the reference picture list RefPicList1. It is initially set equal to 0.
2. The corresponding syntax elements modification_of_pic_nums_idc are processed in the order they occur in the bitstream. For each of these syntax elements, the following applies:
   – If modification_of_pic_nums_idc is equal to 0 or equal to 1, the process specified in subclause 8.2.4.3.1 is invoked with refIdxL1 as input, and the output is assigned to refIdxL1.
   – Otherwise, if modification_of_pic_nums_idc is equal to 2, the process specified in subclause 8.2.4.3.2 is invoked with refIdxL1 as input, and the output is assigned to refIdxL1.
   – Otherwise (modification_of_pic_nums_idc is equal to 3), the modification process for reference picture list RefPicList1 is finished.

8.2.4.3.1 Modification process of reference picture lists for short-term reference pictures

Input to this process is an index refIdxLX (with X being 0 or 1).

Output of this process is an incremented index refIdxLX.

The variable picNumLXNoWrap is derived as follows:

– If modification_of_pic_nums_idc is equal to 0,

```
if( picNumLXPred − ( abs_diff_pic_num_minus1 + 1 ) < 0 )
      picNumLXNoWrap = picNumLXPred − ( abs_diff_pic_num_minus1 + 1 ) + MaxPicNum        (8-34)
else
      picNumLXNoWrap = picNumLXPred − ( abs_diff_pic_num_minus1 + 1 )
```

– Otherwise (modification_of_pic_nums_idc is equal to 1),

```
if( picNumLXPred + ( abs_diff_pic_num_minus1 + 1 )  >=  MaxPicNum )
      picNumLXNoWrap = picNumLXPred + ( abs_diff_pic_num_minus1 + 1 ) − MaxPicNum        (8-35)
else
      picNumLXNoWrap = picNumLXPred + ( abs_diff_pic_num_minus1 + 1 )
```

picNumLXPred is the prediction value for the variable picNumLXNoWrap. When the process specified in this subclause is invoked the first time for a slice (that is, for the first occurrence of modification_of_pic_nums_idc equal to 0 or 1 in the ref_pic_list_modification( ) syntax), picNumL0Pred and picNumL1Pred are initially set equal to CurrPicNum. After each assignment of picNumLXNoWrap, the value of picNumLXNoWrap is assigned to picNumLXPred.

The variable picNumLX is derived as specified by the following pseudo-code:

```
if( picNumLXNoWrap > CurrPicNum )
      picNumLX = picNumLXNoWrap − MaxPicNum        (8-36)
else
      picNumLX = picNumLXNoWrap
```

picNumLX shall be equal to the PicNum of a reference picture that is marked as "used for short-term reference" and shall not be equal to the PicNum of a short-term reference picture that is marked as "non-existing".

The following procedure is conducted to place the picture with short-term picture number picNumLX into the index position refIdxLX, shift the position of any other remaining pictures to later in the list, and increment the value of refIdxLX.

```
for( cIdx = num_ref_idx_lX_active_minus1 + 1; cIdx > refIdxLX; cIdx− − )
    RefPicListX[ cIdx ] = RefPicListX[ cIdx − 1]
RefPicListX[ refIdxLX++ ] = short-term reference picture with PicNum equal to picNumLX
nIdx = refIdxLX
for( cIdx = refIdxLX; cIdx <= num_ref_idx_lX_active_minus1 + 1; cIdx++ )        (8-37)
    if( PicNumF( RefPicListX[ cIdx ] ) != picNumLX )
        RefPicListX[ nIdx++ ] = RefPicListX[ cIdx ]
```

where the function PicNumF(RefPicListX[cIdx]) is derived as follows:

– If the picture RefPicListX[cIdx] is marked as "used for short-term reference", PicNumF(RefPicListX[cIdx]) is the PicNum of the picture RefPicListX[cIdx].
– Otherwise (the picture RefPicListX[cIdx] is not marked as "used for short-term reference"), PicNumF(RefPicListX[cIdx]) is equal to MaxPicNum.

NOTE 1 – A value of MaxPicNum can never be equal to picNumLX.

NOTE 2 – Within this pseudo-code procedure, the length of the list RefPicListX is temporarily made one element longer than the length needed for the final list. After the execution of this procedure, only elements 0 through num_ref_idx_lX_active_minus1 of the list need to be retained.

8.2.4.3.2 Modification process of reference picture lists for long-term reference pictures

Input to this process is an index refIdxLX (with X being 0 or 1).

Output of this process is an incremented index refIdxLX.

The following procedure is conducted to place the picture with long-term picture number long_term_pic_num into the index position refIdxLX, shift the position of any other remaining pictures to later in the list, and increment the value of refIdxLX.

```
for( cIdx = num_ref_idx_lX_active_minus1 + 1; cIdx > refIdxLX; cIdx− − )
    RefPicListX[ cIdx ] = RefPicListX[ cIdx − 1]
RefPicListX[ refIdxLX++ ] = long-term reference picture with LongTermPicNum equal to long_term_pic_num
nIdx = refIdxLX
for( cIdx = refIdxLX; cIdx <= num_ref_idx_lX_active_minus1 + 1; cIdx++ )                    (8-38)
    if( LongTermPicNumF( RefPicListX[ cIdx ] ) != long_term_pic_num )
        RefPicListX[ nIdx++ ] = RefPicListX[ cIdx ]
```

where the function LongTermPicNumF(RefPicListX[cIdx]) is derived as follows:

– If the picture RefPicListX[cIdx] is marked as "used for long-term reference", LongTermPicNumF(RefPicListX[cIdx]) is the LongTermPicNum of the picture RefPicListX[cIdx].
– Otherwise (the picture RefPicListX[cIdx] is not marked as "used for long-term reference"), LongTermPicNumF(RefPicListX[cIdx]) is equal to 2 * (MaxLongTermFrameIdx + 1).

NOTE 1 – A value of 2 * (MaxLongTermFrameIdx + 1) can never be equal to long_term_pic_num.

NOTE 2 – Within this pseudo-code procedure, the length of the list RefPicListX is temporarily made one element longer than the length needed for the final list. After the execution of this procedure, only elements 0 through num_ref_idx_lX_active_minus1 of the list need to be retained.

8.2.5 Decoded reference picture marking process

This process is invoked for decoded pictures when nal_ref_idc is not equal to 0.

NOTE 1 – The decoding process for gaps in frame_num that is specified in subclause 8.2.5.2 may also be invoked when nal_ref_idc is equal to 0, as specified in clause 8.

A decoded picture with nal_ref_idc not equal to 0, referred to as a reference picture, is marked as "used for short-term reference" or "used for long-term reference". For a decoded reference frame, both of its fields are marked the same as the frame. For a complementary reference field pair, the pair is marked the same as both of its fields. A picture that is marked as "used for short-term reference" is identified by its FrameNum and, when it is a field, by its parity. A picture that is marked as "used for long-term reference" is identified by its LongTermFrameIdx and, when it is a field, by its parity.

Frames or complementary field pairs marked as "used for short-term reference" or as "used for long-term reference" can be used as a reference for inter prediction when decoding a frame until the frame, the complementary field pair, or one of its constituent fields is marked as "unused for reference". A field marked as "used for short-term reference" or as "used for long-term reference" can be used as a reference for inter prediction when decoding a field until marked as "unused for reference".

NOTE 2 – The marking status of a frame or complementary field pair can always be deduced from the marking status of its two fields. If both fields of a frame or complementary field pair are marked as "used for reference", the frame or complementary field pair is also marked as "used for reference"; otherwise (one field or both fields of a frame or complementary field pair are marked as "unused for reference"), the frame or complementary field pair is marked as "unused for reference".

A picture can be marked as "unused for reference" by the sliding window reference picture marking process, a first-in, first-out mechanism specified in subclause 8.2.5.3 or by the adaptive memory control reference picture marking process, a customised adaptive marking operation specified in subclause 8.2.5.4.

A short-term reference picture is identified for use in the decoding process by its variables FrameNum and FrameNumWrap and its picture number PicNum, and a long-term reference picture is identified for use in the decoding process by its long-term picture number LongTermPicNum. When the current picture is not an IDR picture, subclause 8.2.4.1 is invoked to specify the assignment of the variables FrameNum, FrameNumWrap, PicNum and LongTermPicNum.

8.2.5.1 Sequence of operations for decoded reference picture marking process

Decoded reference picture marking proceeds in the following ordered steps:

1. All slices of the current picture are decoded.

2. Depending on whether the current picture is an IDR picture, the following applies:

   – If the current picture is an IDR picture, the following ordered steps are specified:

     a. All reference pictures are marked as "unused for reference"

     b. Depending on long_term_reference_flag, the following applies:

        – If long_term_reference_flag is equal to 0, the IDR picture is marked as "used for short-term reference" and MaxLongTermFrameIdx is set equal to "no long-term frame indices".

        – Otherwise (long_term_reference_flag is equal to 1), the IDR picture is marked as "used for long-term reference", the LongTermFrameIdx for the IDR picture is set equal to 0, and MaxLongTermFrameIdx is set equal to 0.

   – Otherwise (the current picture is not an IDR picture), the following applies:

     – If adaptive_ref_pic_marking_mode_flag is equal to 0, the process specified in subclause 8.2.5.3 is invoked.

     – Otherwise (adaptive_ref_pic_marking_mode_flag is equal to 1), the process specified in subclause 8.2.5.4 is invoked.

3. When the current picture is not an IDR picture and it was not marked as "used for long-term reference" by memory_management_control_operation equal to 6, it is marked as "used for short-term reference".

It is a requirement of bitstream conformance that, after marking the current decoded reference picture, the total number of frames with at least one field marked as "used for reference", plus the number of complementary field pairs with at least one field marked as "used for reference", plus the number of non-paired fields marked as "used for reference" shall not be greater than Max(max_num_ref_frames, 1).

8.2.5.2 Decoding process for gaps in frame_num

This process is invoked when frame_num is not equal to PrevRefFrameNum and is not equal to (PrevRefFrameNum + 1) % MaxFrameNum.

NOTE 1 – Although this process is specified as a subclause within subclause 8.2.5 (which defines a process that is invoked only when nal_ref_idc is not equal to 0), this process may also be invoked when nal_ref_idc is equal to 0 (as specified in clause 8). The reasons for the location of this subclause within the structure of this Recommendation | International Standard are historical.

NOTE 2 – This process can only be invoked for a conforming bitstream when gaps_in_frame_num_value_allowed_flag is equal to 1. When gaps_in_frame_num_value_allowed_flag is equal to 0 and frame_num is not equal to PrevRefFrameNum and is not equal to (PrevRefFrameNum + 1) % MaxFrameNum, the decoding process should infer an unintentional loss of pictures.

When this process is invoked, a set of values of frame_num pertaining to "non-existing" pictures is derived as all values taken on by UnusedShortTermFrameNum in Equation 7-23 except the value of frame_num for the current picture.

For each of the values of frame_num pertaining to "non-existing" pictures, in the order in which the values of UnusedShortTermFrameNum are generated by Equation 7-23, the following ordered steps are specified:

1. The decoding process for picture numbers as specified in subclause 8.2.4.1 is invoked.

2. The sliding window decoded reference picture marking process as specified in subclause 8.2.5.3 is invoked.

3. The decoding process generates a frame and the generated frame is marked as "non-existing" and "used for short-term reference". The sample values of the generated frame may be set to any value.

The following constraints shall be obeyed:

a) The bitstream shall not contain data that result in the derivation of a co-located picture colPic that is marked as "non-existing" in any invocation of the derivation process for the co-located 4x4 sub-macroblock partitions specified in subclause 8.4.1.2.1.

b) The bitstream shall not contain data that result in the derivation of a reference picture that is marked as "non-existing" in any invocation of the reference picture selection process specified in subclause 8.4.2.1.

c) The bitstream shall not contain data that result in a variable picNumLX that is equal to the PicNum of a picture marked as "non-existing" in any invocation of the modification process for reference picture lists for short-term reference pictures specified in subclause 8.2.4.3.1.

d) The bitstream shall not contain data that result in a variable picNumLX that is equal to the PicNum of a picture marked as "non-existing" in any invocation of the assignment process of a LongTermFrameIdx to a short-term reference picture specified in subclause 8.2.5.4.3.

NOTE 3 – The above constraints specify that frames that are marked as "non-existing" by the process specified in this subclause must not be referenced in the inter prediction process (subclause 8.4, including the derivation process for co-located 4x4 sub-macroblock partitions in subclause 8.4.1.2.1), the modification commands for reference picture lists for short-term reference pictures (subclause 8.2.4.3.1), or the assignment process of a LongTermFrameIdx to a short-term reference picture (subclause 8.2.5.4.3).

When pic_order_cnt_type is not equal to 0, TopFieldOrderCnt and BottomFieldOrderCnt are derived for each of the "non-existing" frames by invoking the decoding process for picture order count in subclause 8.2.1. When invoking the process in subclause 8.2.1 for a particular "non-existing" frame, the current picture is considered to be a picture considered having frame_num inferred to be equal to UnusedShortTermFrameNum, nal_ref_idc inferred to be not equal to 0, nal_unit_type inferred to be not equal to 5, IdrPicFlag inferred to be equal to 0, field_pic_flag inferred to be equal to 0, adaptive_ref_pic_marking_mode_flag inferred to be equal to 0, delta_pic_order_cnt[0] (if needed) inferred to be equal to 0, and delta_pic_order_cnt[1] (if needed) inferred to be equal to 0.

NOTE 4 – The decoding process should infer an unintentional picture loss when any of these values of frame_num pertaining to "non-existing" pictures is referred to in the inter prediction process (subclause 8.4, including the derivation process for the co-located 4x4 sub-macroblock partitions in subclause 8.4.1.2.1), is referred to in the modification commands for reference picture lists for short-term reference pictures (subclause 8.2.4.3.1), or is referred to in the assignment process of a LongTermFrameIdx to a short-term reference picture (subclause 8.2.5.4.3). The decoding process should not infer an unintentional picture loss when a memory management control operation not equal to 3 is applied to a frame marked as "non-existing".

8.2.5.3 Sliding window decoded reference picture marking process

This process is invoked when adaptive_ref_pic_marking_mode_flag is equal to 0.

Depending on the properties of the current picture as specified below, the following applies:

– If the current picture is a coded field that is the second field in decoding order of a complementary reference field pair, and the first field has been marked as "used for short-term reference", the current picture and the complementary reference field pair are also marked as "used for short-term reference".

– Otherwise, the following applies:

1. Let numShortTerm be the total number of reference frames, complementary reference field pairs and non-paired reference fields for which at least one field is marked as "used for short-term reference". Let numLongTerm be the total number of reference frames, complementary reference field pairs and non-paired reference fields for which at least one field is marked as "used for long-term reference".

2. When numShortTerm + numLongTerm is equal to Max(max_num_ref_frames, 1), the condition that numShortTerm is greater than 0 shall be fulfilled, and the short-term reference frame, complementary reference field pair or non-paired reference field that has the smallest value of FrameNumWrap is marked as "unused for reference". When it is a frame or a complementary field pair, both of its fields are also marked as "unused for reference".

8.2.5.4 Adaptive memory control decoded reference picture marking process

This process is invoked when adaptive_ref_pic_marking_mode_flag is equal to 1.

The memory_management_control_operation commands with values of 1 to 6 are processed in the order they occur in the bitstream after the current picture has been decoded. For each of these memory_management_control_operation commands, one of the processes specified in subclauses 8.2.5.4.1 to 8.2.5.4.6 is invoked depending on the value of memory_management_control_operation. The memory_management_control_operation command with value of 0 specifies the end of memory_management_control_operation commands.

Memory management control operations are applied to pictures as follows:

– If field_pic_flag is equal to 0, memory_management_control_operation commands are applied to the frames or complementary reference field pairs specified.

– Otherwise (field_pic_flag is equal to 1), memory_management_control_operation commands are applied to the individual reference fields specified.

8.2.5.4.1 Marking process of a short-term reference picture as "unused for reference"

This process is invoked when memory_management_control_operation is equal to 1.

Let picNumX be specified by

$$picNumX = CurrPicNum - (difference\_of\_pic\_nums\_minus1 + 1). \quad (8\text{-}39)$$

Depending on field_pic_flag the value of picNumX is used to mark a short-term reference picture as "unused for reference" as follows:

– If field_pic_flag is equal to 0, the short-term reference frame or short-term complementary reference field pair specified by picNumX and both of its fields are marked as "unused for reference".

– Otherwise (field_pic_flag is equal to 1), the short-term reference field specified by picNumX is marked as "unused for reference". When that reference field is part of a reference frame or a complementary reference field pair, the frame or complementary field pair is also marked as "unused for reference", but the marking of the other field in the same reference frame or complementary reference field pair is not changed.

8.2.5.4.2 Marking process of a long-term reference picture as "unused for reference"

This process is invoked when memory_management_control_operation is equal to 2.

Depending on field_pic_flag the value of LongTermPicNum is used to mark a long-term reference picture as "unused for reference" as follows:

– If field_pic_flag is equal to 0, the long-term reference frame or long-term complementary reference field pair having LongTermPicNum equal to long_term_pic_num and both of its fields are marked as "unused for reference".

– Otherwise (field_pic_flag is equal to 1), the long-term reference field specified by LongTermPicNum equal to long_term_pic_num is marked as "unused for reference". When that reference field is part of a reference frame or a complementary reference field pair, the frame or complementary field pair is also marked as "unused for reference", but the marking of the other field in the same reference frame or complementary reference field pair is not changed.

8.2.5.4.3 Assignment process of a LongTermFrameIdx to a short-term reference picture

This process is invoked when memory_management_control_operation is equal to 3.

Given the syntax element difference_of_pic_nums_minus1, the variable picNumX is obtained as specified in subclause 8.2.5.4.1. picNumX shall refer to a frame or complementary reference field pair or non-paired reference field marked as "used for short-term reference" and not marked as "non-existing".

When LongTermFrameIdx equal to long_term_frame_idx is already assigned to a long-term reference frame or a long-term complementary reference field pair, that frame or complementary field pair and both of its fields are marked as "unused for reference". When LongTermFrameIdx is already assigned to a reference field, and that reference field is not part of a complementary field pair that includes the picture specified by picNumX, that field is marked as "unused for reference".

Depending on field_pic_flag the value of LongTermFrameIdx is used to mark a picture from "used for short-term reference" to "used for long-term reference" as follows:

– If field_pic_flag is equal to 0, the marking of the short-term reference frame or short-term complementary reference field pair specified by picNumX and both of its fields are changed from "used for short-term reference" to "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.

– Otherwise (field_pic_flag is equal to 1), the marking of the short-term reference field specified by picNumX is changed from "used for short-term reference" to "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx. When the field is part of a reference frame or a complementary reference field pair, and the other field of the same reference frame or complementary reference field pair is also marked as "used for long-term reference", the reference frame or complementary reference field pair is also marked as "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.

8.2.5.4.4 Decoding process for MaxLongTermFrameIdx

This process is invoked when memory_management_control_operation is equal to 4.

All pictures for which LongTermFrameIdx is greater than max_long_term_frame_idx_plus1 − 1 and that are marked as "used for long-term reference" are marked as "unused for reference".

The variable MaxLongTermFrameIdx is derived as follows:

– If max_long_term_frame_idx_plus1 is equal to 0, MaxLongTermFrameIdx is set equal to "no long-term frame indices".

– Otherwise (max_long_term_frame_idx_plus1 is greater than 0), MaxLongTermFrameIdx is set equal to max_long_term_frame_idx_plus1 − 1.

NOTE – The memory_management_control_operation command equal to 4 can be used to mark long-term reference pictures as "unused for reference". The frequency of transmitting max_long_term_frame_idx_plus1 is not specified by this Recommendation | International Standard. However, the encoder should send a memory_management_control_operation command equal to 4 upon receiving an error message, such as an intra refresh request message.

8.2.5.4.5 Marking process of all reference pictures as "unused for reference" and setting MaxLongTermFrameIdx to "no long-term frame indices"

This process is invoked when memory_management_control_operation is equal to 5.

All reference pictures are marked as "unused for reference" and the variable MaxLongTermFrameIdx is set equal to "no long-term frame indices".

8.2.5.4.6 Process for assigning a long-term frame index to the current picture

This process is invoked when memory_management_control_operation is equal to 6.

When a variable LongTermFrameIdx equal to long_term_frame_idx is already assigned to a long-term reference frame or a long-term complementary reference field pair, that frame or complementary field pair and both of its fields are marked as "unused for reference". When LongTermFrameIdx is already assigned to a reference field, and that reference field is not part of a complementary field pair that includes the current picture, that field is marked as "unused for reference".

The current picture is marked as "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.

When field_pic_flag is equal to 0, both its fields are also marked as "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.

When field_pic_flag is equal to 1 and the current picture is the second field (in decoding order) of a complementary reference field pair, and the first field of the complementary reference field pair is also currently marked as "used for long-term reference", the complementary reference field pair is also marked as "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.

After marking the current decoded reference picture, the total number of frames with at least one field marked as "used for reference", plus the number of complementary field pairs with at least one field marked as "used for reference", plus the number of non-paired fields marked as "used for reference" shall not be greater than Max(max_num_ref_frames, 1).

NOTE – Under some circumstances, the above statement may impose a constraint on the order in which a memory_management_control_operation syntax element equal to 6 can appear in the decoded reference picture marking syntax relative to a memory_management_control_operation syntax element equal to 1, 2, 3, or 4.

8.3 Intra prediction process

This process is invoked for I and SI macroblock types.

Inputs to this process are constructed samples prior to the deblocking filter process and, for Intra_NxN prediction modes (where NxN is equal to 4x4 or 8x8), the values of IntraNxNPredMode from neighbouring macroblocks.

Outputs of this process are specified as follows:

– If the macroblock prediction mode is Intra_4x4 or Intra_8x8, the outputs are constructed luma samples prior to the deblocking filter process and (when ChromaArrayType is not equal to 0) chroma prediction samples of the macroblock $pred_C$, where C is equal to Cb and Cr.

– Otherwise, if mb_type is not equal to I_PCM, the outputs are luma prediction samples of the macroblock $pred_L$ and (when ChromaArrayType is not equal to 0) chroma prediction samples of the macroblock $pred_C$, where C is equal to Cb and Cr.

– Otherwise (mb_type is equal to I_PCM), the outputs are constructed luma and (when ChromaArrayType is not equal to 0) chroma samples prior to the deblocking filter process.

The variable MvCnt is set equal to 0.

Depending on the value of mb_type the following applies:

– If mb_type is equal to I_PCM, the sample construction process for I_PCM macroblocks as specified in subclause 8.3.5 is invoked.

– Otherwise (mb_type is not equal to I_PCM), the following applies:

1. The decoding processes for Intra prediction modes are described for the luma component as follows:

   – If the macroblock prediction mode is equal to Intra_4x4, the Intra_4x4 prediction process for luma samples as specified in subclause 8.3.1 is invoked.

   – Otherwise, if the macroblock prediction mode is equal to Intra_8x8, the Intra_8x8 prediction process as specified in subclause 8.3.2 is invoked.

   – Otherwise (the macroblock prediction mode is equal to Intra_16x16), the Intra_16x16 prediction process as specified in subclause 8.3.3 is invoked with S'_L as the input and the outputs are luma prediction samples of the macroblock $pred_L$.

2. When ChromaArrayType is not equal to 0, the Intra prediction process for chroma samples as specified in subclause 8.3.4 is invoked with S'_{Cb}, and S'_{Cr} as the inputs and the outputs are chroma prediction samples of the macroblock $pred_{Cb}$ and $pred_{Cr}$.

Samples used in the Intra prediction process are the sample values prior to alteration by any deblocking filter operation.

8.3.1 Intra_4x4 prediction process for luma samples

This process is invoked when the macroblock prediction mode is equal to Intra_4x4.

Inputs to this process are the values of Intra4x4PredMode (if available) or Intra8x8PredMode (if available) from neighbouring macroblocks or macroblock pairs.

The luma component of a macroblock consists of 16 blocks of 4x4 luma samples. These blocks are inverse scanned using the 4x4 luma block inverse scanning process as specified in subclause 6.4.3.

For all 4x4 luma blocks of the luma component of a macroblock with luma4x4BlkIdx = 0..15, the derivation process for the Intra4x4PredMode as specified in subclause 8.3.1.1 is invoked with luma4x4BlkIdx as well as Intra4x4PredMode and Intra8x8PredMode that are previously (in decoding order) derived for adjacent macroblocks as the input and the variable Intra4x4PredMode[luma4x4BlkIdx] as the output.

For each luma block of 4x4 samples indexed using luma4x4BlkIdx = 0..15, the following ordered steps are specified:

1. The Intra_4x4 sample prediction process in subclause 8.3.1.2 is invoked with luma4x4BlkIdx and the array S'_L containing constructed luma samples prior to the deblocking filter process from adjacent luma blocks as the inputs and the outputs are the Intra_4x4 luma prediction samples $pred4x4_L[x, y]$ with x, y = 0..3.

2. The position of the upper-left sample of a 4x4 luma block with index luma4x4BlkIdx inside the current macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (xO, yO).

3. The values of the prediction samples $pred_L[xO + x, yO + y]$ with x, y = 0..3 are derived by

$$pred_L[xO + x, yO + y] = pred4x4_L[x, y] \quad (8\text{-}40)$$

4. The transform coefficient decoding process and picture construction process prior to deblocking filter process in subclause 8.5 is invoked with $pred_L$ and luma4x4BlkIdx as the input and the constructed samples for the current 4x4 luma block S'_L as the output.

8.3.1.1 Derivation process for Intra4x4PredMode

Inputs to this process are the index of the 4x4 luma block luma4x4BlkIdx and variable arrays Intra4x4PredMode (if available) and Intra8x8PredMode (if available) that are previously (in decoding order) derived for adjacent macroblocks.

Output of this process is the variable Intra4x4PredMode[luma4x4BlkIdx].

Table 8-2 specifies the values for Intra4x4PredMode[luma4x4BlkIdx] and the associated names.

Table 8-2 – Specification of Intra4x4PredMode[luma4x4BlkIdx] and associated names

Intra4x4PredMode[luma4x4BlkIdx]	Name of Intra4x4PredMode[luma4x4BlkIdx]
0	Intra_4x4_Vertical (prediction mode)
1	Intra_4x4_Horizontal (prediction mode)
2	Intra_4x4_DC (prediction mode)
3	Intra_4x4_Diagonal_Down_Left (prediction mode)
4	Intra_4x4_Diagonal_Down_Right (prediction mode)
5	Intra_4x4_Vertical_Right (prediction mode)
6	Intra_4x4_Horizontal_Down (prediction mode)
7	Intra_4x4_Vertical_Left (prediction mode)
8	Intra_4x4_Horizontal_Up (prediction mode)

Intra4x4PredMode[luma4x4BlkIdx] labelled 0, 1, 3, 4, 5, 6, 7, and 8 represent directions of predictions as illustrated in Figure 8-1.

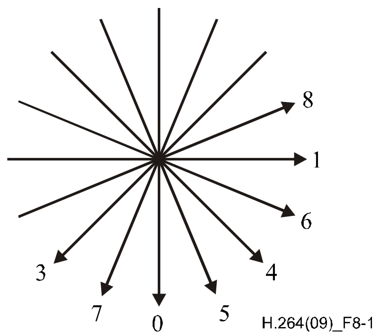


Figure 8-1 – Intra_4x4 prediction mode directions (informative)

Intra4x4PredMode[luma4x4BlkIdx] is derived as specified by the following ordered steps:

1. The process specified in subclause 6.4.11.4 is invoked with luma4x4BlkIdx given as input and the output is assigned to mbAddrA, luma4x4BlkIdxA, mbAddrB, and luma4x4BlkIdxB.
2. The variable dcPredModePredictedFlag is derived as follows:
   – If any of the following conditions are true, dcPredModePredictedFlag is set equal to 1
     – the macroblock with address mbAddrA is not available
     – the macroblock with address mbAddrB is not available
     – the macroblock with address mbAddrA is available and coded in an Inter macroblock prediction mode and constrained_intra_pred_flag is equal to 1
     – the macroblock with address mbAddrB is available and coded in an Inter macroblock prediction mode and constrained_intra_pred_flag is equal to 1
   – Otherwise, dcPredModePredictedFlag is set equal to 0.
3. For N being either replaced by A or B, the variables intraMxMPredModeN are derived as follows:
   – If dcPredModePredictedFlag is equal to 1 or the macroblock with address mbAddrN is not coded in Intra_4x4 or Intra_8x8 macroblock prediction mode, intraMxMPredModeN is set equal to 2 (Intra_4x4_DC prediction mode).

– Otherwise (dcPredModePredictedFlag is equal to 0 and the macroblock with address mbAddrN is coded in Intra_4x4 or Intra_8x8 macroblock prediction mode), the following applies:

  – If the macroblock with address mbAddrN is coded in Intra_4x4 macroblock prediction mode, intraMxMPredModeN is set equal to Intra4x4PredMode[luma4x4BlkIdxN], where Intra4x4PredMode is the variable array assigned to the macroblock mbAddrN.

  – Otherwise (the macroblock with address mbAddrN is coded in Intra_8x8 macroblock prediction mode), intraMxMPredModeN is set equal to Intra8x8PredMode[luma4x4BlkIdxN >> 2], where Intra8x8PredMode is the variable array assigned to the macroblock mbAddrN.

4. Intra4x4PredMode[luma4x4BlkIdx] is derived by applying the following procedure:

```
predIntra4x4PredMode = Min( intraMxMPredModeA, intraMxMPredModeB )
if( prev_intra4x4_pred_mode_flag[ luma4x4BlkIdx ] )
  Intra4x4PredMode[ luma4x4BlkIdx ] = predIntra4x4PredMode
else                                                                    (8-41)
  if( rem_intra4x4_pred_mode[ luma4x4BlkIdx ] < predIntra4x4PredMode )
    Intra4x4PredMode[ luma4x4BlkIdx ] = rem_intra4x4_pred_mode[ luma4x4BlkIdx ]
  else
    Intra4x4PredMode[ luma4x4BlkIdx ] = rem_intra4x4_pred_mode[ luma4x4BlkIdx ] + 1
```

8.3.1.2 Intra_4x4 sample prediction

This process is invoked for each 4x4 luma block of a macroblock with macroblock prediction mode equal to Intra_4x4 followed by the transform decoding process and picture construction process prior to deblocking for each 4x4 luma block.

Inputs to this process are:

– the index of a 4x4 luma block luma4x4BlkIdx,

– an $(\text{PicWidthInSamples}_L)\text{x}(\text{PicHeightInSamples}_L)$ array cS_L containing constructed luma samples prior to the deblocking filter process of neighbouring macroblocks.

Output of this process are the prediction samples $\text{pred4x4}_L[\, x, y\,]$, with x, y = 0..3, for the 4x4 luma block with index luma4x4BlkIdx.

The position of the upper-left sample of a 4x4 luma block with index luma4x4BlkIdx inside the current macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (xO, yO).

The 13 neighbouring samples p[x, y] that are constructed luma samples prior to the deblocking filter process, with x = −1, y = −1..3 and x = 0..7, y = −1, are derived as specified by the following ordered steps:

1. The luma location (xN, yN) is specified by

$$xN = xO + x \tag{8-42}$$

$$yN = yO + y \tag{8-43}$$

2. The derivation process for neighbouring locations in subclause 6.4.12 is invoked for luma locations with (xN, yN) as input and mbAddrN and (xW, yW) as output.

3. Each sample p[x, y] with x = −1, y = −1..3 and x = 0..7, y = −1 is derived as follows:

   – If any of the following conditions are true, the sample p[x, y] is marked as "not available for Intra_4x4 prediction"

     – mbAddrN is not available,

     – the macroblock mbAddrN is coded in an Inter macroblock prediction mode and constrained_intra_pred_flag is equal to 1,

     – the macroblock mbAddrN has mb_type equal to SI and constrained_intra_pred_flag is equal to 1 and the current macroblock does not have mb_type equal to SI,

     – x is greater than 3 and luma4x4BlkIdx is equal to 3 or 11.

– Otherwise, the sample p[x, y] is marked as "available for Intra_4x4 prediction" and the value of the sample p[x, y] is derived as specified by the following ordered steps:

a. The location of the upper-left luma sample of the macroblock mbAddrN is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddrN as the input and the output is assigned to (xM, yM).

b. Depending on the variable MbaffFrameFlag and the macroblock mbAddrN, the sample value p[x, y] is derived as follows:

– If MbaffFrameFlag is equal to 1 and the macroblock mbAddrN is a field macroblock,

$$p[\,x, y\,] = cS_L[\,xM + xW, yM + 2 * yW\,] \quad (8\text{-}44)$$

– Otherwise (MbaffFrameFlag is equal to 0 or the macroblock mbAddrN is a frame macroblock),

$$p[\,x, y\,] = cS_L[\,xM + xW, yM + yW\,] \quad (8\text{-}45)$$

When samples p[x, −1], with x = 4..7, are marked as "not available for Intra_4x4 prediction," and the sample p[3, −1] is marked as "available for Intra_4x4 prediction," the sample value of p[3, −1] is substituted for sample values p[x, −1], with x = 4..7, and samples p[x, −1], with x = 4..7, are marked as "available for Intra_4x4 prediction".

NOTE – Each block is assumed to be constructed into a picture array prior to decoding of the next block.

Depending on Intra4x4PredMode[luma4x4BlkIdx], one of the Intra_4x4 prediction modes specified in subclauses 8.3.1.2.1 to 8.3.1.2.9 is invoked.

8.3.1.2.1 Specification of Intra_4x4_Vertical prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 0.

This mode shall be used only when the samples p[x, −1] with x = 0..3 are marked as "available for Intra_4x4 prediction".

The values of the prediction samples $pred4x4_L[\,x, y\,]$, with x, y = 0..3, are derived by

$$pred4x4_L[\,x, y\,] = p[\,x, -1\,], \text{ with } x, y = 0..3 \quad (8\text{-}46)$$

8.3.1.2.2 Specification of Intra_4x4_Horizontal prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 1.

This mode shall be used only when the samples p[−1, y], with y = 0..3, are marked as "available for Intra_4x4 prediction".

The values of the prediction samples $pred4x4_L[\,x, y\,]$, with x, y = 0..3, are derived by

$$pred4x4_L[\,x, y\,] = p[\,-1, y\,], \text{ with } x,y = 0..3 \quad (8\text{-}47)$$

8.3.1.2.3 Specification of Intra_4x4_DC prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 2.

The values of the prediction samples $pred4x4_L[\,x, y\,]$, with x, y = 0..3, are derived as follows:

– If all samples p[x, −1], with x = 0..3, and p[−1, y], with y = 0..3, are marked as "available for Intra_4x4 prediction", the values of the prediction samples $pred4x4_L[\,x, y\,]$, with x, y = 0..3, are derived by

$$\begin{aligned} pred4x4_L[\,x, y\,] = (\,& p[\,0, -1\,] + p[\,1, -1\,] + p[\,2, -1\,] + p[\,3, -1\,] + \\ & p[\,-1, 0\,] + p[\,-1, 1\,] + p[\,-1, 2\,] + p[\,-1, 3\,] + 4\,) >> 3 \end{aligned} \quad (8\text{-}48)$$

– Otherwise, if any samples p[x, −1], with x = 0..3, are marked as "not available for Intra_4x4 prediction" and all samples p[−1, y], with y = 0..3, are marked as "available for Intra_4x4 prediction", the values of the prediction samples $pred4x4_L[\,x, y\,]$, with x, y = 0..3, are derived by

$$pred4x4_L[\,x, y\,] = (\,p[\,-1, 0\,] + p[\,-1, 1\,] + p[\,-1, 2\,] + p[\,-1, 3\,] + 2\,) >> 2 \quad (8\text{-}49)$$

– Otherwise, if any samples p[−1, y], with y = 0..3, are marked as "not available for Intra_4x4 prediction" and all samples p[x, −1], with x = 0 .. 3, are marked as "available for Intra_4x4 prediction", the values of the prediction samples $pred4x4_L$[x, y], with x, y = 0 .. 3, are derived by

$$pred4x4_L[x, y] = (p[0, -1] + p[1, -1] + p[2, -1] + p[3, -1] + 2) >> 2 \quad (8\text{-}50)$$

– Otherwise (some samples p[x, −1], with x = 0..3, and some samples p[−1, y], with y = 0..3, are marked as "not available for Intra_4x4 prediction"), the values of the prediction samples $pred4x4_L$[x, y], with x, y = 0..3, are derived by

$$pred4x4_L[x, y] = (1 << (BitDepth_Y - 1)) \quad (8\text{-}51)$$

NOTE – A 4x4 luma block can always be predicted using this mode.

8.3.1.2.4 Specification of Intra_4x4_Diagonal_Down_Left prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 3.

This mode shall be used only when the samples p[x, −1] with x = 0..7 are marked as "available for Intra_4x4 prediction".

The values of the prediction samples $pred4x4_L$[x, y], with x, y = 0..3, are derived as follows:

– If x is equal to 3 and y is equal to 3,

$$pred4x4_L[x, y] = (p[6, -1] + 3 * p[7, -1] + 2) >> 2 \quad (8\text{-}52)$$

– Otherwise (x is not equal to 3 or y is not equal to 3),

$$pred4x4_L[x, y] = (p[x + y, -1] + 2 * p[x + y + 1, -1] + p[x + y + 2, -1] + 2) >> 2 \quad (8\text{-}53)$$

8.3.1.2.5 Specification of Intra_4x4_Diagonal_Down_Right prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 4.

This mode shall be used only when the samples p[x, −1] with x = 0..3 and p[−1, y] with y = −1..3 are marked as "available for Intra_4x4 prediction".

The values of the prediction samples $pred4x4_L$[x, y], with x, y = 0..3, are derived as follows:

– If x is greater than y,

$$pred4x4_L[x, y] = (p[x - y - 2, -1] + 2 * p[x - y - 1, -1] + p[x - y, -1] + 2) >> 2 \quad (8\text{-}54)$$

– Otherwise if x is less than y,

$$pred4x4_L[x, y] = (p[-1, y - x - 2] + 2 * p[-1, y - x - 1] + p[-1, y - x] + 2) >> 2 \quad (8\text{-}55)$$

– Otherwise (x is equal to y),

$$pred4x4_L[x, y] = (p[0, -1] + 2 * p[-1, -1] + p[-1, 0] + 2) >> 2 \quad (8\text{-}56)$$

8.3.1.2.6 Specification of Intra_4x4_Vertical_Right prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 5.

This mode shall be used only when the samples p[x, −1] with x = 0..3 and p[−1, y] with y = −1..3 are marked as "available for Intra_4x4 prediction".

Let the variable zVR be set equal to 2 * x − y.

The values of the prediction samples $pred4x4_L$[x, y], with x, y = 0..3, are derived as follows:

– If zVR is equal to 0, 2, 4, or 6,

$$pred4x4_L[x, y] = (p[x - (y >> 1) - 1, -1] + p[x - (y >> 1), -1] + 1) >> 1 \quad (8\text{-}57)$$

– Otherwise, if zVR is equal to 1, 3, or 5,

$$pred4x4_L[x, y] = (p[x - (y >> 1) - 2, -1] + 2 * p[x - (y >> 1) - 1, -1] + p[x - (y >> 1), -1] + 2) >> 2 \quad (8\text{-}58)$$

– Otherwise, if zVR is equal to −1,

$$pred4x4_L[x, y] = (p[-1, 0] + 2 * p[-1, -1] + p[0, -1] + 2) >> 2 \quad (8\text{-}59)$$

– Otherwise (zVR is equal to −2 or −3),

$$pred4x4_L[x, y] = (p[-1, y - 1] + 2 * p[-1, y - 2] + p[-1, y - 3] + 2) >> 2 \quad (8\text{-}60)$$

8.3.1.2.7 Specification of Intra_4x4_Horizontal_Down prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 6.

This mode shall be used only when the samples p[x, −1] with x = 0..3 and p[−1, y] with y = −1..3 are marked as "available for Intra_4x4 prediction".

Let the variable zHD be set equal to 2 * y − x.

The values of the prediction samples $pred4x4_L$[x, y], with x, y = 0..3, are derived as follows:

– If zHD is equal to 0, 2, 4, or 6,

$$pred4x4_L[x, y] = (p[-1, y - (x >> 1) - 1] + p[-1, y - (x >> 1)] + 1) >> 1 \quad (8\text{-}61)$$

– Otherwise, if zHD is equal to 1, 3, or 5,

$$pred4x4_L[x, y] = (p[-1, y - (x >> 1) - 2] + 2 * p[-1, y - (x >> 1) - 1] + p[-1, y - (x >> 1)] + 2) >> 2 \quad (8\text{-}62)$$

– Otherwise, if zHD is equal to −1,

$$pred4x4_L[x, y] = (p[-1, 0] + 2 * p[-1, -1] + p[0, -1] + 2) >> 2 \quad (8\text{-}63)$$

– Otherwise (zHD is equal to −2 or −3),

$$pred4x4_L[x, y] = (p[x - 1, -1] + 2 * p[x - 2, -1] + p[x - 3, -1] + 2) >> 2 \quad (8\text{-}64)$$

8.3.1.2.8 Specification of Intra_4x4_Vertical_Left prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 7.

This mode shall be used only when the samples p[x, −1] with x = 0..7 are marked as "available for Intra_4x4 prediction".

The values of the prediction samples $pred4x4_L$[x, y], with x, y = 0..3, are derived as follows:

– If y is equal to 0 or 2,

$$pred4x4_L[x, y] = (p[x + (y >> 1), -1] + p[x + (y >> 1) + 1, -1] + 1) >> 1 \quad (8\text{-}65)$$

– Otherwise (y is equal to 1 or 3),

$$pred4x4_L[x, y] = (p[x + (y >> 1), -1] + 2 * p[x + (y >> 1) + 1, -1] + p[x + (y >> 1) + 2, -1] + 2) >> 2 \quad (8\text{-}66)$$

8.3.1.2.9 Specification of Intra_4x4_Horizontal_Up prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 8.

This mode shall be used only when the samples p[−1, y] with y = 0..3 are marked as "available for Intra_4x4 prediction".

Let the variable zHU be set equal to x + 2 * y.

The values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3, are derived as follows:

– If zHU is equal to 0, 2, or 4

$$pred4x4_L[x, y] = (p[-1, y + (x >> 1)] + p[-1, y + (x >> 1) + 1] + 1) >> 1 \quad (8\text{-}67)$$

– Otherwise, if zHU is equal to 1 or 3

$$pred4x4_L[x, y] = (p[-1, y + (x >> 1)] + 2 * p[-1, y + (x >> 1) + 1] + p[-1, y + (x >> 1) + 2] + 2) >> 2 \quad (8\text{-}68)$$

– Otherwise, if zHU is equal to 5,

$$pred4x4_L[x, y] = (p[-1, 2] + 3 * p[-1, 3] + 2) >> 2 \quad (8\text{-}69)$$

– Otherwise (zHU is greater than 5),

$$pred4x4_L[x, y] = p[-1, 3] \quad (8\text{-}70)$$

8.3.2 Intra_8x8 prediction process for luma samples

This process is invoked when the macroblock prediction mode is equal to Intra_8x8.

Inputs to this process are the values of Intra4x4PredMode (if available) or Intra8x8PredMode (if available) from the neighbouring macroblocks or macroblock pairs.

Outputs of this process are 8x8 luma sample arrays as part of the 16x16 luma array of prediction samples of the macroblock $pred_L$.

The luma component of a macroblock consists of 4 blocks of 8x8 luma samples. These blocks are inverse scanned using the inverse 8x8 luma block scanning process as specified in subclause 6.4.5.

For all 8x8 luma blocks of the luma component of a macroblock with luma8x8BlkIdx = 0..3, the derivation process for Intra8x8PredMode as specified in subclause 8.3.2.1 is invoked with luma8x8BlkIdx as well as Intra4x4PredMode and Intra8x8PredMode that are previously (in decoding order) derived for adjacent macroblocks as the input and the variable Intra8x8PredMode[luma8x8BlkIdx] as the output.

For each luma block of 8x8 samples indexed using luma8x8BlkIdx = 0..3, the following ordered steps are specified:

1. The Intra_8x8 sample prediction process in subclause 8.3.2.2 is invoked with luma8x8BlkIdx and the array S'_L containing constructed samples prior to the deblocking filter process from adjacent luma blocks as the input and the output are the Intra_8x8 luma prediction samples $pred8x8_L[x, y]$ with x, y = 0..7.
2. The position of the upper-left sample of an 8x8 luma block with index luma8x8BlkIdx inside the current macroblock is derived by invoking the inverse 8x8 luma block scanning process in subclause 6.4.5 with luma8x8BlkIdx as the input and the output being assigned to (xO, yO).
3. The values of the prediction samples $pred_L[xO + x, yO + y]$ with x, y = 0..7 are derived by

$$pred_L[xO + x, yO + y] = pred8x8_L[x, y] \quad (8\text{-}71)$$

4. The transform coefficient decoding process and picture construction process prior to deblocking filter process in subclause 8.5 is invoked with $pred_L$ and luma8x8BlkIdx as the input and the constructed samples for the current 8x8 luma block S'_L as the output.

8.3.2.1 Derivation process for Intra8x8PredMode

Inputs to this process are the index of the 8x8 luma block luma8x8BlkIdx and variable arrays Intra4x4PredMode (if available) and Intra8x8PredMode (if available) that are previously (in decoding order) derived for adjacent macroblocks.

Output of this process is the variable Intra8x8PredMode[luma8x8BlkIdx].

Table 8-3 specifies the values for Intra8x8PredMode[luma8x8BlkIdx] and the associated mnemonic names.

Table 8-3 – Specification of Intra8x8PredMode[luma8x8BlkIdx] and associated names

Intra8x8PredMode[luma8x8BlkIdx]	Name of Intra8x8PredMode[luma8x8BlkIdx]
0	Intra_8x8_Vertical (prediction mode)
1	Intra_8x8_Horizontal (prediction mode)
2	Intra_8x8_DC (prediction mode)
3	Intra_8x8_Diagonal_Down_Left (prediction mode)
4	Intra_8x8_Diagonal_Down_Right (prediction mode)
5	Intra_8x8_Vertical_Right (prediction mode)
6	Intra_8x8_Horizontal_Down (prediction mode)
7	Intra_8x8_Vertical_Left (prediction mode)
8	Intra_8x8_Horizontal_Up (prediction mode)

Intra8x8PredMode[luma8x8BlkIdx] is derived as specified by the following ordered steps:

1. The process specified in subclause 6.4.11.2 is invoked with luma8x8BlkIdx given as input and the output is assigned to mbAddrA, luma8x8BlkIdxA, mbAddrB, and luma8x8BlkIdxB.
2. The variable dcPredModePredictedFlag is derived as follows:
   - If any of the following conditions are true, dcPredModePredictedFlag is set equal to 1:
     - the macroblock with address mbAddrA is not available,
     - the macroblock with address mbAddrB is not available,
     - the macroblock with address mbAddrA is available and coded in an Inter macroblock prediction mode and constrained_intra_pred_flag is equal to 1,
     - the macroblock with address mbAddrB is available and coded in an Inter macroblock prediction mode and constrained_intra_pred_flag is equal to 1.
   - Otherwise, dcPredModePredictedFlag is set equal to 0.
3. For N being either replaced by A or B, the variables intraMxMPredModeN are derived as follows:
   - If dcPredModePredictedFlag is equal to 1 or the macroblock with address mbAddrN is not coded in Intra_4x4 or Intra_8x8 macroblock prediction mode, intraMxMPredModeN is set equal to 2 (Intra_8x8_DC prediction mode).
   - Otherwise (dcPredModePredictedFlag is equal to 0 and (the macroblock with address mbAddrN is coded in Intra_4x4 macroblock prediction mode or the macroblock with address mbAddrN is coded in Intra_8x8 macroblock prediction mode)), the following applies:
     - If the macroblock with address mbAddrN is coded in Intra_8x8 macroblock prediction mode, intraMxMPredModeN is set equal to Intra8x8PredMode[luma8x8BlkIdxN], where Intra8x8PredMode is the variable array assigned to the macroblock mbAddrN.
     - Otherwise (the macroblock with address mbAddrN is coded in Intra_4x4 macroblock prediction mode), intraMxMPredModeN is derived by the following procedure, where Intra4x4PredMode is the variable array assigned to the macroblock mbAddrN.

       $$\text{intraMxMPredModeN} = \text{Intra4x4PredMode}[\ \text{luma8x8BlkIdxN} * 4 + n\ ] \quad (8\text{-}72)$$

       where the variable n is derived as follows:
       - If N is equal to A, depending on the variable MbaffFrameFlag, the variable luma8x8BlkIdx, the current macroblock, and the macroblock mbAddrN, the following applies:

– If MbaffFrameFlag is equal to 1, the current macroblock is a frame coded macroblock, the macroblock mbAddrN is a field coded macroblock, and luma8x8BlkIdx is equal to 2, n is set equal to 3.

– Otherwise (MbaffFrameFlag is equal to 0 or the current macroblock is a field coded macroblock or the macroblock mbAddrN is a frame coded macroblock or luma8x8BlkIdx is not equal to 2), n is set equal to 1.

– Otherwise (N is equal to B), n is set equal to 2.

4. Finally, given intraMxMPredModeA and intraMxMPredModeB, the variable Intra8x8PredMode[luma8x8BlkIdx] is derived by applying the following procedure.

```
predIntra8x8PredMode = Min( intraMxMPredModeA, intraMxMPredModeB )
if( prev_intra8x8_pred_mode_flag[ luma8x8BlkIdx ] )
    Intra8x8PredMode[ luma8x8BlkIdx ] = predIntra8x8PredMode
else                                                                    (8-73)
    if( rem_intra8x8_pred_mode[ luma8x8BlkIdx ] < predIntra8x8PredMode )
        Intra8x8PredMode[ luma8x8BlkIdx ] = rem_intra8x8_pred_mode[ luma8x8BlkIdx ]
    else
        Intra8x8PredMode[ luma8x8BlkIdx ] = rem_intra8x8_pred_mode[ luma8x8BlkIdx ] + 1
```

8.3.2.2 Intra_8x8 sample prediction

This process is invoked for each 8x8 luma block of a macroblock with macroblock prediction mode equal to Intra_8x8 followed by the transform decoding process and picture construction process prior to deblocking for each 8x8 luma block.

Inputs to this process are:

– the index of an 8x8 luma block luma8x8BlkIdx,

– an ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array cS_L containing constructed luma samples prior to the deblocking filter process of neighbouring macroblocks.

Output of this process are the prediction samples $pred8x8_L$[x, y], with x, y = 0..7, for the 8x8 luma block with index luma8x8BlkIdx.

The position of the upper-left sample of an 8x8 luma block with index luma8x8BlkIdx inside the current macroblock is derived by invoking the inverse 8x8 luma block scanning process in subclause 6.4.5 with luma8x8BlkIdx as the input and the output being assigned to (xO, yO).

The 25 neighbouring samples p[x, y] that are constructed luma samples prior to the deblocking filter process, with x = −1, y = −1..7 and x = 0..15, y = −1, are derived as specified by the following ordered steps:

1. The luma location (xN, yN) is specified by

$$xN = xO + x \tag{8-74}$$

$$yN = yO + y \tag{8-75}$$

2. The derivation process for neighbouring locations in subclause 6.4.12 is invoked for luma locations with (xN, yN) as input and mbAddrN and (xW, yW) as output.

3. Each sample p[x, y] with x = −1, y = −1..7 and x = 0..15, y = −1 is derived as follows:

   – If any of the following conditions are true, the sample p[x, y] is marked as "not available for Intra_8x8 prediction":

     – mbAddrN is not available,

     – the macroblock mbAddrN is coded in an Inter macroblock prediction mode and constrained_intra_pred_flag is equal to 1.

   – Otherwise, the sample p[x, y] is marked as "available for Intra_8x8 prediction" and the sample value p[x, y] is derived as specified by the following ordered steps:

     a. The location of the upper-left luma sample of the macroblock mbAddrN is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddrN as the input and the output is assigned to (xM, yM).

b. Depending on the variable MbaffFrameFlag and the macroblock mbAddrN, the sample value p[x, y] is derived as follows:

   – If MbaffFrameFlag is equal to 1 and the macroblock mbAddrN is a field macroblock,

$$p[\ x, y\ ] = cS_L[\ xM + xW, yM + 2 * yW\ ] \tag{8-76}$$

   – Otherwise (MbaffFrameFlag is equal to 0 or the macroblock mbAddrN is a frame macroblock),

$$p[\ x, y\ ] = cS_L[\ xM + xW, yM + yW\ ] \tag{8-77}$$

When samples p[x, −1], with x = 8..15, are marked as "not available for Intra_8x8 prediction," and the sample p[7, −1] is marked as "available for Intra_8x8 prediction," the sample value of p[7, −1] is substituted for sample values p[x, −1], with x = 8..15, and samples p[x, −1], with x = 8..15, are marked as "available for Intra_8x8 prediction".

NOTE – Each block is assumed to be constructed into a picture array prior to decoding of the next block.

The reference sample filtering process for Intra_8x8 sample prediction in subclause 8.3.2.2.1 is invoked with the samples p[x, y] with x = −1, y = −1..7 and x = 0..15, y = −1 (if available) as input and p′[x, y] with x = −1, y = −1..7 and x = 0..15, y = −1 as output.

Depending on Intra8x8PredMode[luma8x8BlkIdx], one of the Intra_8x8 prediction modes specified in subclauses 8.3.2.2.2 to 8.3.2.2.10 is invoked.

8.3.2.2.1 Reference sample filtering process for Intra_8x8 sample prediction

Inputs to this process are the reference samples p[x, y] with x = −1, y = −1..7 and x = 0..15, y = −1 (if available) for Intra_8x8 sample prediction.

Outputs of this process are the filtered reference samples p′[x, y] with x = −1, y = −1..7 and x = 0..15, y = −1 for Intra_8x8 sample prediction.

When all samples p[x, −1] with x = 0..15 are marked as "available for Intra_8x8 prediction", the following applies:

1. The value of p′[0, −1] is derived as follows:

   – If p[−1, −1] is marked as "available for Intra_8x8 prediction", p′[0, −1] is derived by

$$p'[\ 0, -1\ ] = (\ p[\ -1, -1\ ] + 2 * p[\ 0, -1\ ] + p[\ 1, -1\ ] + 2\ ) >> 2 \tag{8-78}$$

   – Otherwise (p[−1, −1] is marked as "not available for Intra_8x8 prediction"), p′[0, −1] is derived by

$$p'[\ 0, -1\ ] = (\ 3 * p[\ 0, -1\ ] + p[\ 1, -1\ ] + 2\ ) >> 2 \tag{8-79}$$

2. The values of p′[x, −1], with x = 1..14, are derived by

$$p'[\ x, -1\ ] = (\ p[\ x - 1, -1\ ] + 2 * p[\ x, -1\ ] + p[\ x+1, -1\ ] + 2\ ) >> 2 \tag{8-80}$$

3. The value of p′[15, −1] is derived by

$$p'[\ 15, -1\ ] = (\ p[\ 14, -1\ ] + 3 * p[\ 15, -1\ ] + 2\ ) >> 2 \tag{8-81}$$

When the sample p[−1, −1] is marked as "available for Intra_8x8 prediction", the value of p′[−1, −1] is derived as follows:

– If the sample p[0, −1] is marked as "not available for Intra_8x8 prediction" or the sample p[−1, 0] is marked as "not available for Intra_8x8 prediction", the following applies:

  – If the sample p[0, −1] is marked as "available for Intra_8x8 prediction", p′[−1, −1] is derived by

$$p'[\ -1, -1\ ] = (\ 3 * p[\ -1, -1\ ] + p[\ 0, -1\ ] + 2\ ) >> 2 \tag{8-82}$$

  – Otherwise, if the sample p[0, −1] is marked as "not available for Intra_8x8 prediction" and the sample p[−1, 0] is marked as "available for Intra_8x8 prediction", p′[−1, −1] is derived by

$$p'[\ -1, -1\ ] = (\ 3 * p[\ -1, -1\ ] + p[\ -1, 0\ ] + 2) >> 2 \tag{8-83}$$

– Otherwise (the sample p[0, −1] is marked as "not available for Intra_8x8 prediction" and the sample p[−1, 0] is marked as "not available for Intra_8x8 prediction"), p′[−1, −1] is set equal to p[−1, −1].

NOTE – When both samples p[0, −1] and p[−1, 0] are marked as "not available for Intra_8x8 prediction", the derived sample p′[−1, −1] is not used in the intra prediction process.

– Otherwise (the sample p[0, −1] is marked as "available for Intra_8x8 prediction" and the sample p[−1, 0] is marked as "available for Intra_8x8 prediction"), p′[−1, −1] is derived by

$$p'[-1, -1] = (p[0, -1] + 2 * p[-1, -1] + p[-1, 0] + 2) >> 2 \quad (8\text{-}84)$$

When all samples p[−1, y] with y = 0..7 are marked as "available for Intra_8x8 prediction", the following applies:

1. The value of p′[−1, 0] is derived as follows:
   – If p[−1, −1] is marked as "available for Intra_8x8 prediction", p′[−1, 0] is derived by

$$p'[-1, 0] = (p[-1, -1] + 2 * p[-1, 0] + p[-1, 1] + 2) >> 2 \quad (8\text{-}85)$$

   – Otherwise (p[−1, −1] is marked as "not available for Intra_8x8 prediction"), p′[−1, 0] is derived by

$$p'[-1, 0] = (3 * p[-1, 0] + p[-1, 1] + 2) >> 2 \quad (8\text{-}86)$$

2. The values of p′[−1, y], with y = 1..6, are derived by

$$p'[-1, y] = (p[-1, y - 1] + 2 * p[-1, y] + p[-1, y+1] + 2) >> 2 \quad (8\text{-}87)$$

3. The value of p′[−1, 7] is derived by

$$p'[-1, 7] = (p[-1, 6] + 3 * p[-1, 7] + 2) >> 2 \quad (8\text{-}88)$$

8.3.2.2.2 Specification of Intra_8x8_Vertical prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 0.

This mode shall be used only when the samples p[x, −1] with x = 0..7 are marked as "available for Intra_8x8 prediction".

The values of the prediction samples $pred8x8_L$[x, y], with x, y = 0..7, are derived by

$$pred8x8_L[x, y] = p'[x, -1], \text{ with } x, y = 0..7 \quad (8\text{-}89)$$

8.3.2.2.3 Specification of Intra_8x8_Horizontal prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 1.

This mode shall be used only when the samples p[−1, y], with y = 0..7, are marked as "available for Intra_8x8 prediction".

The values of the prediction samples $pred8x8_L$[x, y], with x, y = 0..7, are derived by

$$pred8x8_L[x, y] = p'[-1, y], \text{ with } x, y = 0..7 \quad (8\text{-}90)$$

8.3.2.2.4 Specification of Intra_8x8_DC prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 2.

The values of the prediction samples $pred8x8_L$[x, y], with x, y = 0..7, are derived as follows:

– If all samples p[x, −1], with x = 0..7, and p[−1, y], with y = 0..7, are marked as "available for Intra_8x8 prediction," the values of the prediction samples $pred8x8_L$[x, y], with x, y = 0..7, are derived by

$$pred8x8_L[x, y] = (\sum_{x'=0}^{7} p'[x', -1] + \sum_{y'=0}^{7} p'[-1, y'] + 8) >> 4 \quad (8\text{-}91)$$

– Otherwise, if any samples p[x, −1], with x = 0..7, are marked as "not available for Intra_8x8 prediction" and all samples p[−1, y], with y = 0..7, are marked as "available for Intra_8x8 prediction", the values of the prediction samples $pred8x8_L$[x, y], with x, y = 0..7, are derived by

$$\text{pred8x8}_L[\,x,y\,] = (\sum_{y'=0}^{7} p'[-1,y'] + 4) >> 3 \quad (8\text{-}92)$$

– Otherwise, if any samples p[−1, y], with y = 0..7, are marked as "not available for Intra_8x8 prediction" and all samples p[x, −1], with x = 0..7, are marked as "available for Intra_8x8 prediction", the values of the prediction samples pred8x8_L[x, y], with x, y = 0..7, are derived by

$$\text{pred8x8}_L[\,x,y\,] = (\sum_{x'=0}^{7} p'[x',-1] + 4) >> 3 \quad (8\text{-}93)$$

– Otherwise (some samples p[x, −1], with x = 0..7, and some samples p[−1, y], with y = 0..7, are marked as "not available for Intra_8x8 prediction"), the values of the prediction samples pred8x8_L[x, y], with x, y = 0..7, are derived by

$$\text{pred8x8}_L[\,x,y\,] = (\,1 << (\,\text{BitDepth}_Y - 1\,)\,) \quad (8\text{-}94)$$

NOTE – An 8x8 luma block can always be predicted using this mode.

8.3.2.2.5 Specification of Intra_8x8_Diagonal_Down_Left prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 3.

This mode shall be used only when the samples p[x, −1] with x = 0..15 are marked as "available for Intra_8x8 prediction".

The values of the prediction samples pred8x8_L[x, y], with x, y = 0..7, are derived as follows:

– If x is equal to 7 and y is equal to 7,

$$\text{pred8x8}_L[\,x,y\,] = (\,p'[\,14,-1\,] + 3 * p'[15,-1\,] + 2\,) >> 2 \quad (8\text{-}95)$$

– Otherwise (x is not equal to 7 or y is not equal to 7),

$$\text{pred8x8}_L[\,x,y\,] = (\,p'[\,x+y,-1\,] + 2 * p'[\,x+y+1,-1\,] + p'[\,x+y+2,-1\,] + 2\,) >> 2 \quad (8\text{-}96)$$

8.3.2.2.6 Specification of Intra_8x8_Diagonal_Down_Right prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 4.

This mode shall be used only when the samples p[x, −1] with x = 0..7 and p[−1, y] with y = −1..7 are marked as "available for Intra_8x8 prediction".

The values of the prediction samples pred8x8_L[x, y], with x, y = 0..7, are derived as follows:

– If x is greater than y,

$$\text{pred8x8}_L[\,x,y\,] = (\,p'[\,x-y-2,-1] + 2 * p'[\,x-y-1,-1\,] + p'[\,x-y,-1\,] + 2\,) >> 2 \quad (8\text{-}97)$$

– Otherwise if x is less than y,

$$\text{pred8x8}_L[\,x,y\,] = (\,p'[\,-1,y-x-2\,] + 2 * p'[\,-1,y-x-1\,] + p'[\,-1,y-x\,] + 2\,) >> 2 \quad (8\text{-}98)$$

– Otherwise (x is equal to y),

$$\text{pred8x8}_L[\,x,y\,] = (\,p'[\,0,-1\,] + 2 * p'[\,-1,-1\,] + p'[\,-1,0\,] + 2\,) >> 2 \quad (8\text{-}99)$$

8.3.2.2.7 Specification of Intra_8x8_Vertical_Right prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 5.

This mode shall be used only when the samples p[x, −1] with x = 0..7 and p[−1, y] with y = −1..7 are marked as "available for Intra_8x8 prediction".

Let the variable zVR be set equal to 2 * x − y.

The values of the prediction samples pred8x8_L[x, y], with x, y = 0..7, are derived as follows:

– If zVR is equal to 0, 2, 4, 6, 8, 10, 12, or 14

$$\text{pred8x8}_L[\,x, y\,] = (\,p'[\,x - (\,y >> 1\,) - 1, -1\,] + p'[\,x - (\,y >> 1\,), -1\,] + 1\,) >> 1 \tag{8-100}$$

– Otherwise, if zVR is equal to 1, 3, 5, 7, 9, 11, or 13

$$\begin{aligned}\text{pred8x8}_L[\,x, y\,] = (\,&p'[\,x - (\,y >> 1\,) - 2, -1] + 2 * p'[\,x - (\,y >> 1\,) - 1, -1\,] + \\ &p'[\,x - (\,y >> 1\,), -1\,] + 2\,) >> 2\end{aligned} \tag{8-101}$$

– Otherwise, if zVR is equal to −1,

$$\text{pred8x8}_L[\,x, y\,] = (\,p'[\,-1, 0\,] + 2 * p'[\,-1, -1\,] + p'[\,0, -1\,] + 2\,) >> 2 \tag{8-102}$$

– Otherwise (zVR is equal to −2, −3, −4, −5, −6, or −7),

$$\text{pred8x8}_L[\,x, y\,] = (\,p'[\,-1, y - 2*x - 1\,] + 2 * p'[\,-1, y - 2*x - 2\,] + p'[\,-1, y - 2*x - 3\,] + 2\,) >> 2 \tag{8-103}$$

8.3.2.2.8 Specification of Intra_8x8_Horizontal_Down prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 6.

This mode shall be used only when the samples p[x, −1] with x = 0..7 and p[−1, y] with y = −1..7 are marked as "available for Intra_8x8 prediction".

Let the variable zHD be set equal to 2 * y − x.

The values of the prediction samples pred8x8_L[x, y], with x, y = 0..7, are derived as follows:

– If zHD is equal to 0, 2, 4, 6, 8, 10, 12, or 14

$$\text{pred8x8}_L[\,x, y\,] = (\,p'[\,-1, y - (\,x >> 1\,) - 1\,] + p'[\,-1, y - (\,x >> 1\,)\,] + 1\,) >> 1 \tag{8-104}$$

– Otherwise, if zHD is equal to 1, 3, 5, 7, 9, 11, or 13

$$\begin{aligned}\text{pred8x8}_L[\,x, y\,] = (\,&p'[\,-1, y - (\,x >> 1\,) - 2\,] + 2 * p'[\,-1, y - (\,x >> 1\,) - 1\,] + \\ &p'[\,-1, y - (\,x >> 1\,)\,] + 2\,) >> 2\end{aligned} \tag{8-105}$$

– Otherwise, if zHD is equal to −1,

$$\text{pred8x8}_L[\,x, y\,] = (\,p'[\,-1, 0\,] + 2 * p'[\,-1, -1\,] + p'[\,0, -1\,] + 2\,) >> 2 \tag{8-106}$$

– Otherwise (zHD is equal to −2, −3, −4, −5, −6, −7),

$$\text{pred8x8}_L[\,x, y\,] = (\,p'[\,x - 2*y - 1, -1\,] + 2 * p'[\,x - 2*y - 2, -1\,] + p'[\,x - 2*y - 3, -1\,] + 2\,) >> 2 \tag{8-107}$$

8.3.2.2.9 Specification of Intra_8x8_Vertical_Left prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 7.

This mode shall be used only when the samples p[x, −1] with x = 0..15 are marked as "available for Intra_8x8 prediction".

The values of the prediction samples pred8x8_L[x, y], with x, y = 0..7, are derived as follows:

– If y is equal to 0, 2, 4 or 6

$$\text{pred8x8}_L[\,x, y\,] = (\,p'[\,x + (\,y >> 1\,), -1\,] + p'[\,x + (\,y >> 1\,) + 1, -1\,] + 1) >> 1 \tag{8-108}$$

– Otherwise (y is equal to 1, 3, 5, 7),

$$\begin{aligned}\text{pred8x8}_L[\,x, y\,] = (\,&p'[\,x + (\,y >> 1\,), -1\,] + 2 * p'[\,x + (\,y >> 1\,) + 1, -1\,] + \\ &p'[\,x + (\,y >> 1\,) + 2, -1\,] + 2\,) >> 2\end{aligned} \tag{8-109}$$

8.3.2.2.10 Specification of Intra_8x8_Horizontal_Up prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 8.

This mode shall be used only when the samples p[−1, y] with y = 0..7 are marked as "available for Intra_8x8 prediction".

Let the variable zHU be set equal to x + 2 * y.

The values of the prediction samples $pred8x8_L[x, y]$, with x, y = 0..7, are derived as follows:

– If zHU is equal to 0, 2, 4, 6, 8, 10, or 12

$$pred8x8_L[x, y] = (p'[-1, y + (x >> 1)] + p'[-1, y + (x >> 1) + 1] + 1) >> 1 \tag{8-110}$$

– Otherwise, if zHU is equal to 1, 3, 5, 7, 9, or 11

$$pred8x8_L[x, y] = (p'[-1, y + (x >> 1)] + 2 * p'[-1, y + (x >> 1) + 1] + p'[-1, y + (x >> 1) + 2] + 2) >> 2 \tag{8-111}$$

– Otherwise, if zHU is equal to 13,

$$pred8x8_L[x, y] = (p'[-1, 6] + 3 * p'[-1, 7] + 2) >> 2 \tag{8-112}$$

– Otherwise (zHU is greater than 13),

$$pred8x8_L[x, y] = p'[-1, 7] \tag{8-113}$$

8.3.3 Intra_16x16 prediction process for luma samples

This process is invoked when the macroblock prediction mode is equal to Intra_16x16. It specifies how the Intra prediction luma samples for the current macroblock are derived.

Input to this process is a $(PicWidthInSamples_L)x(PicHeightInSamples_L)$ array cS_L containing constructed luma samples prior to the deblocking filter process of neighbouring macroblocks.

Outputs of this process are Intra prediction luma samples for the current macroblock $pred_L[x, y]$.

The 33 neighbouring samples p[x, y] that are constructed luma samples prior to the deblocking filter process, with x = −1, y = −1..15 and with x = 0..15, y = −1, are derived as specified by the following ordered steps:

1. The derivation process for neighbouring locations in subclause 6.4.12 is invoked for luma locations with (x, y) assigned to (xN, yN) as input and mbAddrN and (xW, yW) as output.
2. Each sample p[x, y] with x = −1, y = −1..15 and with x = 0..15, y = −1 is derived as follows:
   – If any of the following conditions are true, the sample p[x, y] is marked as "not available for Intra_16x16 prediction":
      – mbAddrN is not available,
      – the macroblock mbAddrN is coded in an Inter macroblock prediction mode and constrained_intra_pred_flag is equal to 1,
      – the macroblock mbAddrN has mb_type equal to SI and constrained_intra_pred_flag is equal to 1.
   – Otherwise, the sample p[x, y] is marked as "available for Intra_16x16 prediction" and the value of the sample p[x, y] is derived as specified by the following ordered steps:
      a. The location of the upper-left luma sample of the macroblock mbAddrN is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddrN as the input and the output is assigned to (xM, yM).
      b. Depending on the variable MbaffFrameFlag and the macroblock mbAddrN, the sample value p[x, y] is derived as follows:
         – If MbaffFrameFlag is equal to 1 and the macroblock mbAddrN is a field macroblock,

$$p[x, y] = cS_L[xM + xW, yM + 2 * yW] \tag{8-114}$$

         – Otherwise (MbaffFrameFlag is equal to 0 or the macroblock mbAddrN is a frame macroblock),

$$p[x, y] = cS_L[xM + xW, yM + yW] \tag{8-115}$$

Let $pred_L[x, y]$ with x, y = 0..15 denote the prediction samples for the 16x16 luma block samples.

Intra_16x16 prediction modes are specified in Table 8-4.

Table 8-4 – Specification of Intra16x16PredMode and associated names

Intra16x16PredMode	Name of Intra16x16PredMode
0	Intra_16x16_Vertical (prediction mode)
1	Intra_16x16_Horizontal (prediction mode)
2	Intra_16x16_DC (prediction mode)
3	Intra_16x16_Plane (prediction mode)

Depending on Intra16x16PredMode, one of the Intra_16x16 prediction modes specified in subclauses 8.3.3.1 to 8.3.3.4 is invoked.

8.3.3.1 Specification of Intra_16x16_Vertical prediction mode

This Intra_16x16 prediction mode shall be used only when the samples p[x, −1] with x = 0..15 are marked as "available for Intra_16x16 prediction".

The values of the prediction samples $pred_L$[x, y], with x, y = 0..15, are derived by

$$pred_L[\,x, y\,] = p[\,x, -1\,],\ \text{with } x, y = 0..15 \qquad (8\text{-}116)$$

8.3.3.2 Specification of Intra_16x16_Horizontal prediction mode

This Intra_16x16 prediction mode shall be used only when the samples p[−1, y] with y = 0..15 are marked as "available for Intra_16x16 prediction".

The values of the prediction samples $pred_L$[x, y], with x, y = 0..15, are derived by

$$pred_L[\,x, y\,] = p[\,-1, y\,],\ \text{with } x, y = 0..15 \qquad (8\text{-}117)$$

8.3.3.3 Specification of Intra_16x16_DC prediction mode

This Intra_16x16 prediction mode operates, depending on whether the neighbouring samples are marked as "available for Intra_16x16 prediction", as follows:

– If all neighbouring samples p[x, −1], with x = 0..15, and p[−1, y], with y = 0..15, are marked as "available for Intra_16x16 prediction", the prediction for all luma samples in the macroblock is given by:

$$pred_L[\,x, y\,] = \left(\sum_{x'=0}^{15} p[x',-1] + \sum_{y'=0}^{15} p[-1,y'] + 16\right) >> 5,\ \text{with } x, y = 0..15 \qquad (8\text{-}118)$$

– Otherwise, if any of the neighbouring samples p[x, −1], with x = 0..15, are marked as "not available for Intra_16x16 prediction" and all of the neighbouring samples p[−1, y], with y = 0..15, are marked as "available for Intra_16x16 prediction", the prediction for all luma samples in the macroblock is given by:

$$pred_L[\,x, y\,] = \left(\sum_{y'=0}^{15} p[-1,y'] + 8\right) >> 4,\ \text{with } x, y = 0..15 \qquad (8\text{-}119)$$

– Otherwise, if any of the neighbouring samples p[−1, y], with y = 0..15, are marked as "not available for Intra_16x16 prediction" and all of the neighbouring samples p[x, −1], with x = 0..15, are marked as "available for Intra_16x16 prediction", the prediction for all luma samples in the macroblock is given by:

$$pred_L[\,x, y\,] = \left(\sum_{x'=0}^{15} p[x',-1] + 8\right) >> 4,\ \text{with } x, y = 0..15 \qquad (8\text{-}120)$$

– Otherwise (some of the neighbouring samples p[x, −1], with x = 0..15, and some of the neighbouring samples p[−1, y], with y = 0..15, are marked as "not available for Intra_16x16 prediction"), the prediction for all luma samples in the macroblock is given by:

$$pred_L[\,x, y\,] = (\,1 << (\,BitDepth_Y - 1\,)\,),\ \text{with } x, y = 0..15 \qquad (8\text{-}121)$$

8.3.3.4 Specification of Intra_16x16_Plane prediction mode

This Intra_16x16 prediction mode shall be used only when the samples p[x, −1] with x = −1..15 and p[−1, y] with y = 0..15 are marked as "available for Intra_16x16 prediction".

The values of the prediction samples $pred_L$[x, y], with x, y = 0..15, are derived by

$$pred_L[x, y] = Clip1_Y((a + b * (x - 7) + c * (y - 7) + 16) >> 5), \text{ with } x, y = 0..15, \quad (8\text{-}122)$$

where

$$a = 16 * (p[-1, 15] + p[15, -1]) \quad (8\text{-}123)$$

$$b = (5 * H + 32) >> 6 \quad (8\text{-}124)$$

$$c = (5 * V + 32) >> 6 \quad (8\text{-}125)$$

and H and V are specified as

$$H = \sum_{x'=0}^{7} (x'+1) * (p[8 + x', -1] - p[6 - x', -1]) \quad (8\text{-}126)$$

$$V = \sum_{y'=0}^{7} (y'+1) * (p[-1, 8 + y'] - p[-1, 6 - y']) \quad (8\text{-}127)$$

8.3.4 Intra prediction process for chroma samples

This process is invoked for I and SI macroblock types. It specifies how the Intra prediction chroma samples for the current macroblock are derived.

Inputs to this process are two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays cS_{Cb} and cS_{Cr} containing constructed chroma samples prior to the deblocking filter process of neighbouring macroblocks.

Outputs of this process are Intra prediction chroma samples for the current macroblock $pred_{Cb}$[x, y] and $pred_{Cr}$[x, y].

Depending on the value of ChromaArrayType, the following applies:

– If ChromaArrayType is equal to 3, the Intra prediction chroma samples for the current macroblock $pred_{Cb}$[x, y] and $pred_{Cr}$[x, y] are derived using the Intra prediction process for chroma samples with ChromaArrayType equal to 3 as specified in subclause 8.3.4.5.

– Otherwise (ChromaArrayType is equal to 1 or 2), the following text specifies the Intra prediction chroma samples for the current macroblock $pred_{Cb}$[x, y] and $pred_{Cr}$[x, y].

Both chroma blocks (Cb and Cr) of the macroblock use the same prediction mode. The prediction mode is applied to each of the chroma blocks separately. The process specified in this subclause is invoked for each chroma block. In the remainder of this subclause, chroma block refers to one of the two chroma blocks and the subscript C is used as a replacement of the subscript Cb or Cr.

The neighbouring samples p[x, y] that are constructed chroma samples prior to the deblocking filter process, with x = −1, y = −1..MbHeightC − 1 and with x = 0..MbWidthC − 1, y = −1, are derived as specified by the following ordered steps:

1. The derivation process for neighbouring locations in subclause 6.4.12 is invoked for chroma locations with (x, y) assigned to (xN, yN) as input and mbAddrN and (xW, yW) as output.

2. Each sample p[x, y] is derived as follows:

   – If any of the following conditions are true, the sample p[x, y] is marked as "not available for Intra chroma prediction":

     – mbAddrN is not available,

     – the macroblock mbAddrN is coded in an Inter macroblock prediction mode and constrained_intra_pred_flag is equal to 1,

     – the macroblock mbAddrN has mb_type equal to SI and constrained_intra_pred_flag is equal to 1 and the current macroblock does not have mb_type equal to SI.

– Otherwise, the sample p[x, y] is marked as "available for Intra chroma prediction" and the value of the sample p[x, y] is derived as specified by the following ordered steps:

a. The location of the upper-left luma sample of the macroblock mbAddrN is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddrN as the input and the output is assigned to (xL, yL).

b. The location (xM, yM) of the upper-left chroma sample of the macroblock mbAddr is derived by:

$$xM = (xL >> 4) * MbWidthC \quad (8\text{-}128)$$
$$yM = ((yL >> 4)* MbHeightC) + (yL \% 2) \quad (8\text{-}129)$$

c. Depending on the variable MbaffFrameFlag and the macroblock mbAddrN, the sample value p[x, y] is derived as follows:

– If MbaffFrameFlag is equal to 1 and the macroblock mbAddrN is a field macroblock,

$$p[x, y] = cS_C[xM + xW, yM + 2 * yW] \quad (8\text{-}130)$$

– Otherwise (MbaffFrameFlag is equal to 0 or the macroblock mbAddrN is a frame macroblock),

$$p[x, y] = cS_C[xM + xW, yM + yW] \quad (8\text{-}131)$$

Let $pred_C$[x, y] with x = 0..MbWidthC − 1, y = 0..MbHeightC − 1 denote the prediction samples for the chroma block samples.

Intra chroma prediction modes are specified in Table 8-5.

Table 8-5 – Specification of Intra chroma prediction modes and associated names

intra_chroma_pred_mode	Name of intra_chroma_pred_mode
0	Intra_Chroma_DC (prediction mode)
1	Intra_Chroma_Horizontal (prediction mode)
2	Intra_Chroma_Vertical (prediction mode)
3	Intra_Chroma_Plane (prediction mode)

Depending on intra_chroma_pred_mode, one of the Intra chroma prediction modes specified in subclauses 8.3.4.1 to 8.3.4.4 is invoked.

8.3.4.1 Specification of Intra_Chroma_DC prediction mode

This Intra chroma prediction mode is invoked when intra_chroma_pred_mode is equal to 0.

For each chroma block of 4x4 samples indexed by chroma4x4BlkIdx = 0..(1 << (ChromaArrayType + 1)) − 1, the following applies:

– The position of the upper-left sample of a 4x4 chroma block with index chroma4x4BlkIdx inside the current macroblock is derived by invoking the inverse 4x4 chroma block scanning process in subclause 6.4.7 with chroma4x4BlkIdx as the input and the output being assigned to (xO, yO).

– Depending on the values of xO and yO, the following applies:

– If (xO, yO) is equal to (0, 0) or xO and yO are greater than 0, the values of the prediction samples $pred_C$[x + xO, y + yO] with x, y = 0..3 are derived as follows:

– If all samples p[x + xO, −1], with x = 0..3, and p[−1, y +yO], with y = 0..3, are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C$[x + xO, y + yO], with x, y = 0..3, are derived as:

$$pred_C[x + xO, y + yO] = \left(\sum_{x'=0}^{3} p[x'+xO,-1] + \sum_{y'=0}^{3} p[-1, y'+yO] + 4 \right) >> 3 \text{, with x, y = 0..3.} \quad (8\text{-}132)$$

– Otherwise, if any samples p[x + xO, −1], with x = 0..3, are marked as "not available for Intra chroma prediction" and all samples p[−1, y +yO], with y = 0..3, are marked as "available for Intra chroma prediction", the values of the prediction samples pred_C[x + xO, y + yO], with x, y = 0..3, are derived as:

$$\text{pred}_C[\, x + xO, y + yO \,] = \left(\sum_{y'=0}^{3} p[-1, y'+yO] + 2 \right) >> 2 \text{, with x, y = 0..3.} \tag{8-133}$$

– Otherwise, if any samples p[−1, y +yO], with y = 0..3, are marked as "not available for Intra chroma prediction" and all samples p[x + xO, −1], with x = 0..3, are marked as "available for Intra chroma prediction", the values of the prediction samples pred_C[x + xO, y + yO], with x, y = 0..3, are derived as:

$$\text{pred}_C[\, x + xO, y + yO \,] = \left(\sum_{x'=0}^{3} p[x'+xO, -1] + 2 \right) >> 2 \text{, with x, y = 0..3.} \tag{8-134}$$

– Otherwise (some samples p[x + xO, −1], with x = 0..3, and some samples p[−1, y +yO], with y = 0..3, are marked as "not available for Intra chroma prediction"), the values of the prediction samples pred_C[x + xO, y + yO], with x, y = 0..3, are derived as:

$$\text{pred}_C[\, x + xO, y + yO \,] = (\, 1 << (\, \text{BitDepth}_C - 1 \,)\,) \text{, with x, y = 0..3.} \tag{8-135}$$

– Otherwise, if xO is greater than 0 and yO is equal to 0, the values of the prediction samples pred_C[x + xO, y + yO] with x, y = 0..3 are derived as follows:

– If all samples p[x + xO, −1], with x = 0..3, are marked as "available for Intra chroma prediction", the values of the prediction samples pred_C[x + xO, y + yO], with x, y = 0..3, are derived as:

$$\text{pred}_C[\, x + xO, y + yO \,] = \left(\sum_{x'=0}^{3} p[x'+xO, -1] + 2 \right) >> 2 \text{, with x, y = 0..3.} \tag{8-136}$$

– Otherwise, if all samples p[−1, y +yO], with y = 0..3, are marked as "available for Intra chroma prediction", the values of the prediction samples pred_C[x + xO, y + yO], with x, y = 0..3, are derived as:

$$\text{pred}_C[\, x + xO, y + yO \,] = \left(\sum_{y'=0}^{3} p[-1, y'+yO] + 2 \right) >> 2 \text{, with x, y = 0..3.} \tag{8-137}$$

– Otherwise (some samples p[x + xO, −1], with x = 0..3, and some samples p[−1, y +yO], with y = 0..3, are marked as "not available for Intra chroma prediction"), the values of the prediction samples pred_C[x + xO, y + yO], with x, y = 0..3, are derived as:

$$\text{pred}_C[\, x + xO, y + yO \,] = (\, 1 << (\, \text{BitDepth}_C - 1 \,)\,) \text{, with x, y = 0..3.} \tag{8-138}$$

– Otherwise (xO is equal to 0 and yO is greater than 0), the values of the prediction samples pred_C[x + xO, y + yO] with x, y = 0..3 are derived as follows:

– If all samples p[−1, y +yO], with y = 0..3, are marked as "available for Intra chroma prediction", the values of the prediction samples pred_C[x + xO, y + yO], with x, y = 0..3, are derived as:

$$\text{pred}_C[\, x + xO, y + yO \,] = \left(\sum_{y'=0}^{3} p[-1, y'+yO] + 2 \right) >> 2 \text{, with x, y = 0..3.} \tag{8-139}$$

– Otherwise, if all samples p[x + xO, −1], with x = 0..3, are marked as "available for Intra chroma prediction", the values of the prediction samples pred_C[x + xO, y + yO], with x, y = 0..3, are derived as:

$$\text{pred}_C[\, x + xO, y + yO \,] = \left(\sum_{x'=0}^{3} p[x'+xO, -1] + 2 \right) >> 2 \text{, with x, y = 0..3.} \tag{8-140}$$

– Otherwise (some samples p[x + xO, −1], with x = 0..3, and some samples p[−1, y +yO], with y = 0..3, are marked as "not available for Intra chroma prediction"), the values of the prediction samples $pred_C$[x + xO, y + yO], with x, y = 0..3, are derived as:

$$pred_C[\, x + xO, y + yO \,] = (1 << (BitDepth_C - 1)), \text{ with } x, y = 0..3. \qquad (8\text{-}141)$$

8.3.4.2 Specification of Intra_Chroma_Horizontal prediction mode

This Intra chroma prediction mode is invoked when intra_chroma_pred_mode is equal to 1.

This mode shall be used only when the samples p[−1, y] with y = 0..MbHeightC − 1 are marked as "available for Intra chroma prediction".

The values of the prediction samples $pred_C$[x, y] are derived as:

$$pred_C[\, x, y \,] = p[\, -1, y \,], \text{ with } x = 0..MbWidthC - 1 \text{ and } y = 0..MbHeightC - 1 \qquad (8\text{-}142)$$

8.3.4.3 Specification of Intra_Chroma_Vertical prediction mode

This Intra chroma prediction mode is invoked when intra_chroma_pred_mode is equal to 2.

This mode shall be used only when the samples p[x, −1] with x = 0..MbWidthC − 1 are marked as "available for Intra chroma prediction".

The values of the prediction samples $pred_C$[x, y] are derived as:

$$pred_C[\, x, y \,] = p[\, x, -1 \,], \text{ with } x = 0..MbWidthC - 1 \text{ and } y = 0..MbHeightC - 1 \qquad (8\text{-}143)$$

8.3.4.4 Specification of Intra_Chroma_Plane prediction mode

This Intra chroma prediction mode is invoked when intra_chroma_pred_mode is equal to 3.

This mode shall be used only when the samples p[x, −1], with x = 0..MbWidthC − 1 and p[−1, y], with y = −1..MbHeightC − 1 are marked as "available for Intra chroma prediction".

Let the variable xCF be set equal to ((ChromaArrayType = = 3) ? 4 : 0) and let the variable yCF be set equal to ((ChromaArrayType != 1) ? 4 : 0).

The values of the prediction samples $pred_C$[x, y] are derived by:

$$pred_C[\, x, y \,] = Clip1_C((a + b * (x - 3 - xCF) + c * (y - 3 - yCF) + 16) >> 5),$$
$$\text{with } x = 0..MbWidthC - 1 \text{ and } y = 0..MbHeightC - 1 \qquad (8\text{-}144)$$

where

$$a = 16 * (p[\, -1, MbHeightC - 1 \,] + p[\, MbWidthC - 1, -1 \,]) \qquad (8\text{-}145)$$

$$b = ((34 - 29 * (ChromaArrayType == 3)) * H + 32) >> 6 \qquad (8\text{-}146)$$

$$c = ((34 - 29 * (ChromaArrayType != 1)) * V + 32) >> 6 \qquad (8\text{-}147)$$

and H and V are specified as:

$$H = \sum_{x'=0}^{3+xCF} (x'+1) * (p[4 + xCF + x', -1] - p[2 + xCF - x', -1]) \qquad (8\text{-}148)$$

$$V = \sum_{y'=0}^{3+yCF} (y'+1) * (p[-1, 4 + yCF + y'] - p[-1, 2 + yCF - y']) \qquad (8\text{-}149)$$

8.3.4.5 Intra prediction for chroma samples with ChromaArrayType equal to 3

This process is invoked when ChromaArrayType is equal to 3. This process is invoked for I and SI macroblock types. It specifies how the Intra prediction chroma samples for the current macroblock are derived when ChromaArrayType is equal to 3.

Inputs to this process are constructed samples prior to the deblocking filter process from neighbouring Cb and Cr blocks and for Intra_NxN (where NxN is equal to 4x4 or 8x8) prediction mode, the associated values of IntraNxNPredMode from neighbouring macroblocks.

Outputs of this process are the Intra prediction samples of the Cb and Cr components of the macroblock or in case of the Intra_NxN prediction process, the outputs are NxN Cb sample arrays as part of the 16x16 Cb array of prediction samples of the macroblock, and NxN Cb sample arrays as part of the 16x16 Cb array of prediction samples of the macroblock.

Each Cb, Cr, and luma block with the same block index of the macroblock use the same prediction mode. The prediction mode is applied to each of the Cb and Cr blocks separately. The process specified in this subclause is invoked for each Cb and Cr block.

Depending on the macroblock prediction mode, the following applies:

– If the macroblock prediction mode is equal to Intra_4x4, the following applies:
  – The same process described in subclause 8.3.1 is also applied to Cb or Cr samples, substituting luma with Cb or Cr, substituting luma4x4BlkIdx with cb4x4BlkIdx or cr4x4BlkIdx, substituting pred4x4_L with pred4x4_{Cb} or pred4x4_{Cr}, and substituting BitDepth_Y with BitDepth_C.
  – The output variable Intra4x4PredMode[luma4x4BlkIdx] from the process described in subclause 8.3.1.1 is also used for the 4x4 Cb or 4x4 Cr blocks with index luma4x4BlkIdx equal to index cb4x4BlkIdx or cr4x4BlkIdx.
  – The process to derive prediction Cb or Cr samples is identical to the process described in subclause 8.3.1.2 and its subsequent subclauses when substituting luma with Cb or Cr, substituting pred4x4_L with pred4x4_{Cb} or pred4x4_{Cr}, and substituting BitDepth_Y with BitDepth_C.
– Otherwise, if the macroblock prediction mode is equal to Intra_8x8, the following applies:
  – The same process described in subclause 8.3.2 is also applied to Cb or Cr samples, substituting luma with Cb or Cr, substituting luma8x8BlkIdx with cb8x8BlkIdx or cr8x8BlkIdx, substituting pred8x8_L with pred8x8_{Cb} or pred8x8_{Cr}, and substituting BitDepth_Y with BitDepth_C.
  – The output variable Intra8x8PredMode[luma8x8BlkIdx] from the process described in subclause 8.3.2.1 is used for the 8x8 Cb or 8x8 Cr blocks with index luma8x8BlkIdx equal to index cb8x8BlkIdx or cr8x8BlkIdx.
  – The process to derive prediction Cb or Cr samples is identical to the process described in subclause 8.3.2.2 and its subsequent subclauses when substituting luma with Cb or Cr, substituting pred8x8_L with pred8x8_{Cb} or pred8x8_{Cr}, and substituting BitDepth_Y with BitDepth_C.
– Otherwise (the macroblock prediction mode is equal to Intra_16x16), the same process described in subclause 8.3.3 and in the subsequent subclause 8.3.3.1 to 8.3.3.4 is also applied to Cb or Cr samples, substituting luma with Cb or Cr, substituting pred_L with pred_{Cb} or pred_{Cr}, and substituting BitDepth_Y with BitDepth_C.

8.3.5 Sample construction process for I_PCM macroblocks

This process is invoked when mb_type is equal to I_PCM.

The variable dy is derived as follows:

– If MbaffFrameFlag is equal to 1 and the current macroblock is a field macroblock, dy is set equal to 2.
– Otherwise (MbaffFrameFlag is equal to 0 or the current macroblock is a frame macroblock), dy is set equal to 1.

The position of the upper-left luma sample of the current macroblock is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with CurrMbAddr as input and the output being assigned to (xP, yP).

The constructed luma samples prior to the deblocking process are generated as specified by:

$$\begin{array}{l}\text{for(i = 0; i < 256; i++)}\\ \quad S'_L[\ xP + (\ i\ \%\ 16\ ),\ yP + dy * (\ i\ /\ 16\ )\ )\ ] = \text{pcm\_sample\_luma}[\ i\ ]\end{array} \qquad (8\text{-}150)$$

When ChromaArrayType is not equal to 0, the constructed chroma samples prior to the deblocking process are generated as specified by:

```
for( i = 0; i < MbWidthC * MbHeightC; i++ ) {
   S′Cb[ ( xP / SubWidthC ) + ( i % MbWidthC ),
            ( ( yP + SubHeightC − 1 ) / SubHeightC ) + dy * ( i / MbWidthC ) ] =
              pcm_sample_chroma[ i ]                                              (8-151)
   S′Cr[ ( xP / SubWidthC ) + ( i % MbWidthC ),
            ( ( yP + SubHeightC − 1 ) / SubHeightC ) + dy * ( i / MbWidthC ) ] =
              pcm_sample_chroma[ i + MbWidthC * MbHeightC ]
}
```

8.4 Inter prediction process

This process is invoked when decoding P and B macroblock types.

Outputs of this process are Inter prediction samples for the current macroblock that are a 16x16 array pred_L of luma samples and when ChromaArrayType is not equal to 0 two (MbWidthC)x(MbHeightC) arrays pred_{Cb} and pred_{Cr} of chroma samples, one for each of the chroma components Cb and Cr.

The partitioning of a macroblock is specified by mb_type. Each macroblock partition is referred to by mbPartIdx. When the macroblock partitioning consists of partitions that are equal to sub-macroblocks, each sub-macroblock can be further partitioned into sub-macroblock partitions as specified by sub_mb_type[mbPartIdx]. Each sub-macroblock partition is referred to by subMbPartIdx. When the macroblock partitioning does not consist of sub-macroblocks, subMbPartIdx is set equal to 0.

The following steps are specified for each macroblock partition or for each sub-macroblock partition.

The functions MbPartWidth( ), MbPartHeight( ), SubMbPartWidth( ), and SubMbPartHeight( ) describing the width and height of macroblock partitions and sub-macroblock partitions are specified in Tables 7-13, 7-14, 7-17, and 7-18.

The range of the macroblock partition index mbPartIdx is derived as follows:

– If mb_type is equal to B_Skip or B_Direct_16x16, mbPartIdx proceeds over values 0..3.
– Otherwise (mb_type is not equal to B_Skip or B_Direct_16x16), mbPartIdx proceeds over values 0..NumMbPart(mb_type) − 1.

For each value of mbPartIdx, the variables partWidth and partHeight for each macroblock partition or sub-macroblock partition in the macroblock are derived as follows:

– If mb_type is not equal to P_8x8, P_8x8ref0, B_Skip, B_Direct_16x16, or B_8x8, subMbPartIdx is set equal to 0, and partWidth and partHeight are derived as:

partWidth = MbPartWidth(mb_type) (8-152)

partHeight = MbPartHeight(mb_type) (8-153)

– Otherwise, if mb_type is equal to P_8x8 or P_8x8ref0, or mb_type is equal to B_8x8 and sub_mb_type[mbPartIdx] is not equal to B_Direct_8x8, subMbPartIdx proceeds over values 0..NumSubMbPart(sub_mb_type[mbPartIdx]) − 1, and partWidth and partHeight are derived as:

partWidth = SubMbPartWidth(sub_mb_type[mbPartIdx]) (8-154)

partHeight = SubMbPartHeight(sub_mb_type[mbPartIdx]). (8-155)

– Otherwise (mb_type is equal to B_Skip or B_Direct_16x16, or mb_type is equal to B_8x8 and sub_mb_type[mbPartIdx] is equal to B_Direct_8x8), subMbPartIdx proceeds over values 0..3, and partWidth and partHeight are derived as:

partWidth = 4 (8-156)

partHeight = 4 (8-157)

When ChromaArrayType is not equal to 0, the variables partWidthC and partHeightC are derived as:

$$\text{partWidthC} = \text{partWidth} / \text{SubWidthC} \quad (8\text{-}158)$$
$$\text{partHeightC} = \text{partHeight} / \text{SubHeightC} \quad (8\text{-}159)$$

Let the variable MvCnt be initially set equal to 0 before any invocation of subclause 8.4.1 for the macroblock.

The Inter prediction process for a macroblock partition mbPartIdx and a sub-macroblock partition subMbPartIdx consists of the following ordered steps:

1. The derivation process for motion vector components and reference indices as specified in subclause 8.4.1 is invoked.

   Inputs to this process are:

   – a macroblock partition mbPartIdx,

   – a sub-macroblock partition subMbPartIdx.

   Outputs of this process are:

   – luma motion vectors mvL0 and mvL1 and when ChromaArrayType is not equal to 0, the chroma motion vectors mvCL0 and mvCL1

   – reference indices refIdxL0 and refIdxL1

   – prediction list utilization flags predFlagL0 and predFlagL1

   – the sub-macroblock partition motion vector count subMvCnt.

2. The variable MvCnt is incremented by subMvCnt.

3. When (weighted_pred_flag is equal to 1 and (slice_type % 5) is equal to 0 or 3) or (weighted_bipred_idc is greater than 0 and (slice_type % 5) is equal to 1), the derivation process for prediction weights as specified in subclause 8.4.3 is invoked.

   Inputs to this process are:

   – reference indices refIdxL0 and refIdxL1

   – prediction list utilization flags predFlagL0 and predFlagL1

   Outputs of this process are variables for weighted prediction logWD_C, w_{0C}, w_{1C}, o_{0C}, o_{1C} with C being replaced by L and, when ChromaArrayType is not equal to 0, Cb and Cr.

4. The decoding process for Inter prediction samples as specified in subclause 8.4.2 is invoked.

   Inputs to this process are:

   – a macroblock partition mbPartIdx,

   – a sub-macroblock partition subMbPartIdx,

   – variables specifying partition width and height for luma and chroma (if available), partWidth, partHeight, partWidthC (if available), and partHeightC (if available),

   – luma motion vectors mvL0 and mvL1 and when ChromaArrayType is not equal to 0, the chroma motion vectors mvCL0 and mvCL1,

   – reference indices refIdxL0 and refIdxL1,

   – prediction list utilization flags predFlagL0 and predFlagL1,

   – variables for weighted prediction logWD_C, w_{0C}, w_{1C}, o_{0C}, o_{1C} with C being replaced by L and, when ChromaArrayType is not equal to 0, Cb and Cr.

   Outputs of this process are inter prediction samples (pred); which are a (partWidth)x(partHeight) array predPart_L of prediction luma samples and when ChromaArrayType is not equal to 0 two (partWidthC)x(partHeightC) arrays predPart_{Cr}, and predPart_{Cb} of prediction chroma samples, one for each of the chroma components Cb and Cr.

For use in derivation processes of variables invoked later in the decoding process, the following assignments are made:

$$\text{MvL0}[\ \text{mbPartIdx}\ ][\ \text{subMbPartIdx}\ ] = \text{mvL0} \quad (8\text{-}160)$$

$$\text{MvL1[mbPartIdx][subMbPartIdx] = mvL1} \tag{8-161}$$

$$\text{RefIdxL0[mbPartIdx] = refIdxL0} \tag{8-162}$$

$$\text{RefIdxL1[mbPartIdx] = refIdxL1} \tag{8-163}$$

$$\text{PredFlagL0[mbPartIdx] = predFlagL0} \tag{8-164}$$

$$\text{PredFlagL1[mbPartIdx] = predFlagL1} \tag{8-165}$$

The location of the upper-left sample of the macroblock partition relative to the upper-left sample of the macroblock is derived by invoking the inverse macroblock partition scanning process as described in subclause 6.4.2.1 with mbPartIdx as the input and (xP, yP) as the output.

The location of the upper-left sample of the sub-macroblock partition relative to the upper-left sample of the macroblock partition is derived by invoking the inverse sub-macroblock partition scanning process as described in subclause 6.4.2.2 with subMbPartIdx as the input and (xS, yS) as the output.

The macroblock prediction is formed by placing the macroblock or sub-macroblock partition prediction samples in their correct relative positions in the macroblock, as follows.

The variable pred_L[xP + xS + x, yP + yS + y] with x = 0..partWidth − 1, y = 0..partHeight − 1 is derived by:

$$\text{pred}_L[\text{ xP + xS + x, yP + yS + y }] = \text{predPart}_L[\text{ x, y }] \tag{8-166}$$

When ChromaArrayType is not equal to 0, the variable pred_C with x = 0..partWidthC − 1, y = 0..partHeightC − 1, and C in pred_C and predPart_C being replaced by Cb or Cr is derived by:

$$\text{pred}_C[\text{ xP / SubWidthC + xS / SubWidthC + x, yP / SubHeightC + yS / SubHeightC + y }] = \text{predPart}_C[\text{ x, y }] \tag{8-167}$$

8.4.1 Derivation process for motion vector components and reference indices

Inputs to this process are:

– a macroblock partition mbPartIdx,

– a sub-macroblock partition subMbPartIdx.

Outputs of this process are:

– luma motion vectors mvL0 and mvL1 and when ChromaArrayType is not equal to 0, the chroma motion vectors mvCL0 and mvCL1,

– reference indices refIdxL0 and refIdxL1,

– prediction list utilization flags predFlagL0 and predFlagL1,

– a motion vector count variable subMvCnt.

For the derivation of the variables mvL0 and mvL1 as well as refIdxL0 and refIdxL1, the following applies:

– If mb_type is equal to P_Skip, the derivation process for luma motion vectors for skipped macroblocks in P and SP slices in subclause 8.4.1.1 is invoked with the output being the luma motion vectors mvL0 and reference indices refIdxL0, and predFlagL0 is set equal to 1. mvL1 and refIdxL1 are marked as not available and predFlagL1 is set equal to 0. The motion vector count variable subMvCnt is set equal to 1.

– Otherwise, if mb_type is equal to B_Skip or B_Direct_16x16 or sub_mb_type[mbPartIdx] is equal to B_Direct_8x8, the derivation process for luma motion vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8 in B slices in subclause 8.4.1.2 is invoked with mbPartIdx and subMbPartIdx as the input and the output being the luma motion vectors mvL0, mvL1, the reference indices refIdxL0, refIdxL1, the motion vector count variable subMvCnt, and the prediction utilization flags predFlagL0 and predFlagL1.

– Otherwise, for X being replaced by either 0 or 1 in the variables predFlagLX, mvLX, refIdxLX, and in Pred_LX and in the syntax elements ref_idx_lX and mvd_lX, the following applies:

  1. The variables refIdxLX and predFlagLX are derived as follows:

     – If MbPartPredMode(mb_type, mbPartIdx) or SubMbPredMode(sub_mb_type[mbPartIdx]) is equal to Pred_LX or to BiPred,

$$refIdxLX = ref\_idx\_lX[mbPartIdx] \quad (8\text{-}168)$$

$$predFlagLX = 1 \quad (8\text{-}169)$$

– Otherwise, the variables refIdxLX and predFlagLX are specified by

$$refIdxLX = -1 \quad (8\text{-}170)$$

$$predFlagLX = 0 \quad (8\text{-}171)$$

2. The motion vector count variable subMvCnt is set equal to predFlagL0 + predFlagL1.
3. The variable currSubMbType is derived as follows:
   – If the macroblock type is equal to B_8x8, currSubMbType is set equal to sub_mb_type[mbPartIdx].
   – Otherwise (the macroblock type is not equal to B_8x8), currSubMbType is set equal to "na".
4. When predFlagLX is equal to 1, the derivation process for luma motion vector prediction in subclause 8.4.1.3 is invoked with mbPartIdx subMbPartIdx, refIdxLX, and currSubMbType as the inputs and the output being mvpLX. The luma motion vectors are derived by

$$mvLX[0] = mvpLX[0] + mvd\_lX[mbPartIdx][subMbPartIdx][0] \quad (8\text{-}172)$$

$$mvLX[1] = mvpLX[1] + mvd\_lX[mbPartIdx][subMbPartIdx][1] \quad (8\text{-}173)$$

When ChromaArrayType is not equal to 0 and predFlagLX (with X being either 0 or 1) is equal to 1, the derivation process for chroma motion vectors in subclause 8.4.1.4 is invoked with mvLX and refIdxLX as input and the output being mvCLX.

8.4.1.1 Derivation process for luma motion vectors for skipped macroblocks in P and SP slices

This process is invoked when mb_type is equal to P_Skip.

Outputs of this process are the motion vector mvL0 and the reference index refIdxL0.

The reference index refIdxL0 for a skipped macroblock is derived as:

$$refIdxL0 = 0. \quad (8\text{-}174)$$

For the derivation of the motion vector mvL0 of a P_Skip macroblock type, the following ordered steps are specified:

1. The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx set equal to 0, subMbPartIdx set equal to 0, currSubMbType set equal to "na", and listSuffixFlag set equal to 0 as input and the output is assigned to mbAddrA, mbAddrB, mvL0A, mvL0B, refIdxL0A, and refIdxL0B.
2. The variable mvL0 is specified as follows:
   – If any of the following conditions are true, both components of the motion vector mvL0 are set equal to 0:
     – mbAddrA is not available,
     – mbAddrB is not available,
     – refIdxL0A is equal to 0 and both components of mvL0A are equal to 0,
     – refIdxL0B is equal to 0 and both components of mvL0B are equal to 0.
   – Otherwise, the derivation process for luma motion vector prediction as specified in subclause 8.4.1.3 is invoked with mbPartIdx = 0, subMbPartIdx = 0, refIdxL0, and currSubMbType = "na" as inputs and the output is assigned to mvL0.

NOTE – The output is directly assigned to mvL0, since the predictor is equal to the actual motion vector.

8.4.1.2 Derivation process for luma motion vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8

This process is invoked when mb_type is equal to B_Skip or B_Direct_16x16, or sub_mb_type[mbPartIdx] is equal to B_Direct_8x8.

Inputs to this process are mbPartIdx and subMbPartIdx.

Outputs of this process are the reference indices refIdxL0, refIdxL1, the motion vectors mvL0 and mvL1, the motion vector count variable subMvCnt, and the prediction list utilization flags, predFlagL0 and predFlagL1.

The derivation process depends on the value of direct_spatial_mv_pred_flag, which is present in the bitstream in the slice header syntax as specified in subclause 7.3.3, and is specified as follows:

– If direct_spatial_mv_pred_flag is equal to 1, the mode in which the outputs of this process are derived is referred to as spatial direct prediction mode.
– Otherwise (direct_spatial_mv_pred_flag is equal to 0), mode in which the outputs of this process are derived is referred to as temporal direct prediction mode.

Both spatial and temporal direct prediction mode use the co-located motion vectors and reference indices as specified in subclause 8.4.1.2.1.

The motion vectors and reference indices are derived as follows:

– If spatial direct prediction mode is used, the direct motion vector and reference index prediction mode specified in subclause 8.4.1.2.2 is used, with subMvCnt being an output.
– Otherwise (temporal direct prediction mode is used), the direct motion vector and reference index prediction mode specified in subclause 8.4.1.2.3 is used and the variable subMvCnt is derived as follows:
  – If subMbPartIdx is equal to 0, subMvCnt is set equal to 2.
  – Otherwise (subMbPartIdx is not equal to 0), subMvCnt is set equal to 0.

8.4.1.2.1 Derivation process for the co-located 4x4 sub-macroblock partitions

Inputs to this process are mbPartIdx and subMbPartIdx.

Outputs of this process are the picture colPic, the co-located macroblock mbAddrCol, the motion vector mvCol, the reference index refIdxCol, and the variable vertMvScale (which can be One_To_One, Frm_To_Fld or Fld_To_Frm).

When RefPicList1[0] is a frame or a complementary field pair, let firstRefPicL1Top and firstRefPicL1Bottom be the top and bottom fields of RefPicList1[0], respectively, and let the following variables be specified as

$$\text{topAbsDiffPOC} = \text{Abs}(\ \text{DiffPicOrderCnt}(\ \text{firstRefPicL1Top}, \text{CurrPic}\ )\ ) \tag{8-175}$$

$$\text{bottomAbsDiffPOC} = \text{Abs}(\ \text{DiffPicOrderCnt}(\ \text{firstRefPicL1Bottom}, \text{CurrPic}\ )\ ) \tag{8-176}$$

The variable colPic specifies the picture that contains the co-located macroblock as specified in Table 8-6.

Table 8-6 – Specification of the variable colPic

field_pic_flag	RefPicList1[0] is …	mb_field_decoding_flag	additional condition	colPic
1	a field of a decoded frame			the frame containing RefPicList1[0]
	a decoded field			RefPicList1[0]
0	a decoded frame			RefPicList1[0]
	a complementary field pair	0	topAbsDiffPOC < bottomAbsDiffPOC	firstRefPicL1Top
			topAbsDiffPOC >= bottomAbsDiffPOC	firstRefPicL1Bottom
		1	(CurrMbAddr & 1) = = 0	firstRefPicL1Top
			(CurrMbAddr & 1) != 0	firstRefPicL1Bottom

NOTE – The picture order count values of a complementary field pair marked as "used for long-term reference" have an impact on the decoding process when the current picture is a coded frame, the current macroblock is a frame macroblock, and the complementary field pair marked as "used for long-term reference" is the first picture in reference list 1.

Let PicCodingStruct(X) be a function with the argument X being either CurrPic or colPic. It is specified in Table 8-7.

Table 8-7 – Specification of PicCodingStruct(X)

X is coded with field_pic_flag equal to …	mb_adaptive_frame_field_flag	PicCodingStruct(X)
1		FLD
0	0	FRM
0	1	AFRM

The variable luma4x4BlkIdx is derived as follows:

– If direct_8x8_inference_flag is equal to 0, luma4x4BlkIdx is set equal to (4 * mbPartIdx + subMbPartIdx).

– Otherwise (direct_8x8_inference_flag is equal to 1), luma4x4BlkIdx is set equal to (5 * mbPartIdx).

The inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with luma4x4BlkIdx as the input and (x, y) assigned to (xCol, yCol) as the output.

Table 8-8 specifies the co-located macroblock address mbAddrCol, yM, and the variable vertMvScale in two steps:

1. Specification of a macroblock address mbAddrX depending on PicCodingStruct(CurrPic), and PicCodingStruct(colPic).

   NOTE – It is not possible for CurrPic and colPic picture coding types to be either (FRM, AFRM) or (AFRM, FRM) because these picture coding types must be separated by an IDR picture.

2. Specification of mbAddrCol, yM, and vertMvScale depending on mb_field_decoding_flag and the variable fieldDecodingFlagX, which is derived as follows:

   – If the macroblock mbAddrX in the picture colPic is a field macroblock, fieldDecodingFlagX is set equal to 1.

   – Otherwise (the macroblock mbAddrX in the picture colPic is a frame macroblock), fieldDecodingFlagX is set equal to 0.

Unspecified values in Table 8-8 indicate that the value of the corresponding variable is not relevant for the current table row.

mbAddrCol is set equal to CurrMbAddr or to one of the following values.

$$\begin{aligned}\text{mbAddrCol1} = {} & 2 * \text{PicWidthInMbs} * (\text{CurrMbAddr} / \text{PicWidthInMbs}) + \\ & (\text{CurrMbAddr} \,\%\, \text{PicWidthInMbs}) + \text{PicWidthInMbs} * (\text{yCol} / 8)\end{aligned} \quad (8\text{-}177)$$

$$\text{mbAddrCol2} = 2 * \text{CurrMbAddr} + (\text{yCol} / 8) \quad (8\text{-}178)$$

$$\text{mbAddrCol3} = 2 * \text{CurrMbAddr} + \text{bottom\_field\_flag} \quad (8\text{-}179)$$

$$\begin{aligned}\text{mbAddrCol4} = {} & \text{PicWidthInMbs} * (\text{CurrMbAddr} / (2 * \text{PicWidthInMbs})) + \\ & (\text{CurrMbAddr} \,\%\, \text{PicWidthInMbs})\end{aligned} \quad (8\text{-}180)$$

$$\text{mbAddrCol5} = \text{CurrMbAddr} / 2 \quad (8\text{-}181)$$

$$\text{mbAddrCol6} = 2 * (\text{CurrMbAddr} / 2) + ((\text{topAbsDiffPOC} < \text{bottomAbsDiffPOC}) \; ? \; 0 : 1) \quad (8\text{-}182)$$

$$\text{mbAddrCol7} = 2 * (\text{CurrMbAddr} / 2) + (\text{yCol} / 8) \quad (8\text{-}183)$$

Table 8-8 – Specification of mbAddrCol, yM, and vertMvScale

PicCodingStruct(CurrPic)	PicCodingStruct(colPic)	mbAddrX	mb_field_decoding_flag	fieldDecodingFlagX	mbAddrCol	yM	vertMvScale
FLD	FLD				CurrMbAddr	yCol	One_To_One
	FRM				mbAddrCol1	(2 * yCol) % 16	Frm_To_Fld
	AFRM	2*CurrMbAddr		0	mbAddrCol2	(2 * yCol) % 16	Frm_To_Fld
				1	mbAddrCol3	yCol	One_To_One
FRM	FLD				mbAddrCol4	8 * ((CurrMbAddr / PicWidthInMbs) % 2) + 4 * (yCol / 8)	Fld_To_Frm
	FRM				CurrMbAddr	yCol	One_To_One
AFRM	FLD		0		mbAddrCol5	8 * (CurrMbAddr % 2) +4 * (yCol / 8)	Fld_To_Frm
			1		mbAddrCol5	yCol	One_To_One
	AFRM	CurrMbAddr	0	0	CurrMbAddr	yCol	One_To_One
				1	mbAddrCol6	8 * (CurrMbAddr % 2) + 4 * (yCol / 8)	Fld_To_Frm
		CurrMbAddr	1	0	mbAddrCol7	(2 * yCol) % 16	Frm_To_Fld
				1	CurrMbAddr	yCol	One_To_One

Let mbTypeCol be the syntax element mb_type of the macroblock with address mbAddrCol inside the picture colPic and, when mbTypeCol is equal to P_8x8, P_8x8ref0, or B_8x8, let subMbTypeCol be the syntax element list sub_mb_type of the macroblock with address mbAddrCol inside the picture colPic.

Let mbPartIdxCol be the macroblock partition index of the co-located partition and subMbPartIdxCol the sub-macroblock partition index of the co-located sub-macroblock partition. The derivation process for macroblock and sub-macroblock partition indices as specified in subclause 6.4.13.4 is invoked with the luma location (xCol, yM), the macroblock type mbTypeCol, and, when mbTypeCol is equal to P_8x8, P_8x8ref0, or B_8x8, the list of sub-macroblock types subMbTypeCol as the inputs and the outputs are the macroblock partition index mbPartIdxCol and the sub-macroblock partition index subMbPartIdxCol.

The motion vector mvCol and the reference index refIdxCol are derived as follows:

– If the macroblock mbAddrCol is coded in an Intra macroblock prediction mode, both components of mvCol are set equal to 0 and refIdxCol is set equal to −1.
– Otherwise (the macroblock mbAddrCol is not coded in an Intra macroblock prediction mode), the prediction utilization flags predFlagL0Col and predFlagL1Col are set equal to PredFlagL0[mbPartIdxCol] and PredFlagL1[mbPartIdxCol], respectively, which are the prediction utilization flags that have been assigned to the macroblock partition mbAddrCol\mbPartIdxCol inside the picture colPic, and the following applies:
  – If predFlagL0Col is equal to 1, the motion vector mvCol and the reference index refIdxCol are set equal to MvL0[mbPartIdxCol][subMbPartIdxCol] and RefIdxL0[mbPartIdxCol], respectively, which are the motion vector mvL0 and the reference index refIdxL0 that have been assigned to the (sub-)macroblock partition mbAddrCol\mbPartIdxCol\subMbPartIdxCol inside the picture colPic.

– Otherwise (predFlagL0Col is equal to 0 and predFlagL1Col is equal to 1), the motion vector mvCol and the reference index refIdxCol are set equal to MvL1[mbPartIdxCol][subMbPartIdxCol] and RefIdxL1[mbPartIdxCol], respectively, which are the motion vector mvL1 and the reference index refIdxL1 that have been assigned to the (sub-)macroblock partition mbAddrCol\mbPartIdxCol\subMbPartIdxCol inside the picture colPic.

8.4.1.2.2 Derivation process for spatial direct luma motion vector and reference index prediction mode

This process is invoked when direct_spatial_mv_pred_flag is equal to 1 and any of the following conditions are true:

– mb_type is equal to B_Skip,
– mb_type is equal to B_Direct_16x16,
– sub_mb_type[mbPartIdx] is equal to B_Direct_8x8.

Inputs to this process are mbPartIdx, subMbPartIdx.

Outputs of this process are the reference indices refIdxL0, refIdxL1, the motion vectors mvL0 and mvL1, the motion vector count variable subMvCnt, and the prediction list utilization flags, predFlagL0 and predFlagL1.

The reference indices refIdxL0 and refIdxL1 and the variable directZeroPredictionFlag are derived by applying the following ordered steps.

1. Let the variable currSubMbType be set equal to sub_mb_type[mbPartIdx].
2. The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx = 0, subMbPartIdx = 0, currSubMbType, and listSuffixFlag = 0 as inputs and the output is assigned to the motion vectors mvL0N and the reference indices refIdxL0N with N being replaced by A, B, or C.
3. The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx = 0, subMbPartIdx = 0, currSubMbType, and listSuffixFlag = 1 as inputs and the output is assigned to the motion vectors mvL1N and the reference indices refIdxL1N with N being replaced by A, B, or C.

   NOTE 1 – The motion vectors mvL0N, mvL1N and the reference indices refIdxL0N, refIdxL1N are identical for all 4x4 sub-macroblock partitions of a macroblock.

4. The reference indices refIdxL0, refIdxL1, and directZeroPredictionFlag are derived by:

$$\text{refIdxL0} = \text{MinPositive}(\text{ refIdxL0A, MinPositive(refIdxL0B, refIdxL0C) }) \tag{8-184}$$
$$\text{refIdxL1} = \text{MinPositive}(\text{ refIdxL1A, MinPositive(refIdxL1B, refIdxL1C) }) \tag{8-185}$$
$$\text{directZeroPredictionFlag} = 0 \tag{8-186}$$

   where

$$\text{MinPositive(x, y)} = \begin{cases} \text{Min}(x, y) & \text{if } x >= 0 \text{ and } y >= 0 \\ \text{Max}(x, y) & \text{otherwise} \end{cases} \tag{8-187}$$

5. When both reference indices refIdxL0 and refIdxL1 are less than 0,

$$\text{refIdxL0} = 0 \tag{8-188}$$
$$\text{refIdxL1} = 0 \tag{8-189}$$
$$\text{directZeroPredictionFlag} = 1 \tag{8-190}$$

The process specified in subclause 8.4.1.2.1 is invoked with mbPartIdx, subMbPartIdx given as input and the output is assigned to refIdxCol and mvCol.

The variable colZeroFlag is derived as follows:

– If all of the following conditions are true, colZeroFlag is set equal to 1:
  – RefPicList1[0] is currently marked as "used for short-term reference",
  – refIdxCol is equal to 0,
  – both motion vector components mvCol[0] and mvCol[1] lie in the range of −1 to 1 in units specified as follows:
    – If the co-located macroblock is a frame macroblock, the units of mvCol[0] and mvCol[1] are units of quarter luma frame samples.

– Otherwise (the co-located macroblock is a field macroblock), the units of mvCol[0] and mvCol[1] are units of quarter luma field samples.

NOTE 2 – For purposes of determining the condition above, the value mvCol[1] is not scaled to use the units of a motion vector for the current macroblock in cases when the current macroblock is a frame macroblock and the co-located macroblock is a field macroblock or when the current macroblock is a field macroblock and the co-located macroblock is a frame macroblock. This aspect differs from the use of mvCol[1] in the temporal direct mode as specified in subclause 8.4.1.2.3, which applies scaling to the motion vector of the co-located macroblock to use the same units as the units of a motion vector for the current macroblock, using Equation 8-193 or Equation 8-194 in these cases.

– Otherwise, colZeroFlag is set equal to 0.

The motion vectors mvLX (with X being 0 or 1) are derived as follows:

– If any of the following conditions are true, both components of the motion vector mvLX are set equal to 0:
  – directZeroPredictionFlag is equal to 1,
  – refIdxLX is less than 0,
  – refIdxLX is equal to 0 and colZeroFlag is equal to 1.
– Otherwise, the process specified in subclause 8.4.1.3 is invoked with mbPartIdx = 0, subMbPartIdx = 0, refIdxLX, and currSubMbType as inputs and the output is assigned to mvLX.

  NOTE 3 – The motion vector mvLX returned from subclause 8.4.1.3 is identical for all 4x4 sub-macroblock partitions of a macroblock for which the process is invoked.

The prediction utilization flags predFlagL0 and predFlagL1 are derived as specified using Table 8-9.

Table 8-9 – Assignment of prediction utilization flags

refIdxL0	**refIdxL1**	**predFlagL0**	**predFlagL1**
>= 0	>= 0	1	1
>= 0	< 0	1	0
< 0	>= 0	0	1

The variable subMvCnt is derived as follows:

– If subMbPartIdx is not equal to 0, subMvCnt is set equal to 0.
– Otherwise (subMbPartIdx is equal to 0), subMvCnt is set equal to predFlagL0 + predFLagL1.

8.4.1.2.3 Derivation process for temporal direct luma motion vector and reference index prediction mode

This process is invoked when direct_spatial_mv_pred_flag is equal to 0 and any of the following conditions are true:

– mb_type is equal to B_Skip,
– mb_type is equal to B_Direct_16x16,
– sub_mb_type[mbPartIdx] is equal to B_Direct_8x8.

Inputs to this process are mbPartIdx and subMbPartIdx.

Outputs of this process are the motion vectors mvL0 and mvL1, the reference indices refIdxL0 and refIdxL1, and the prediction list utilization flags, predFlagL0 and predFlagL1.

The process specified in subclause 8.4.1.2.1 is invoked with mbPartIdx, subMbPartIdx given as input and the output is assigned to colPic, mbAddrCol, mvCol, refIdxCol, and vertMvScale.

The reference indices refIdxL0 and refIdxL1 are derived as

$$\text{refIdxL0} = ((\text{refIdxCol} < 0) \ ?\ 0 : \text{MapColToList0}(\text{refIdxCol})) \quad (8\text{-}191)$$

$$\text{refIdxL1} = 0 \quad (8\text{-}192)$$

NOTE 1 – If the current macroblock is a field macroblock, refIdxL0 and refIdxL1 index a list of fields; otherwise (the current macroblock is a frame macroblock), refIdxL0 and refIdxL1 index a list of frames or complementary reference field pairs.

Let refPicCol be a frame, a field, or a complementary field pair that was referred by the reference index refIdxCol when decoding the co-located macroblock mbAddrCol inside the picture colPic. The function MapColToList0(refIdxCol) is specified as follows:

– If vertMvScale is equal to One_To_One, the following applies:
  – If field_pic_flag is equal to 0 and the current macroblock is a field macroblock, the following applies:
    – Let refIdxL0Frm be the lowest valued reference index in the current reference picture list RefPicList0 that references the frame or complementary field pair that contains the field refPicCol. RefPicList0 shall contain a frame or complementary field pair that contains the field refPicCol. The return value of MapColToList0( ) is specified as follows:
      – If the field referred to by refIdxCol has the same parity as the current macroblock, MapColToList0(refIdxCol) returns the reference index (refIdxL0Frm << 1).
      – Otherwise (the field referred by refIdxCol has the opposite parity of the current macroblock), MapColToList0(refIdxCol) returns the reference index ((refIdxL0Frm << 1) + 1).
  – Otherwise (field_pic_flag is equal to 1 or the current macroblock is a frame macroblock), MapColToList0(refIdxCol) returns the lowest valued reference index refIdxL0 in the current reference picture list RefPicList0 that references refPicCol. RefPicList0 shall contain refPicCol.
– Otherwise, if vertMvScale is equal to Frm_To_Fld, the following applies:
  – If field_pic_flag is equal to 0, let refIdxL0Frm be the lowest valued reference index in the current reference picture list RefPicList0 that references refPicCol. MapColToList0(refIdxCol) returns the reference index (refIdxL0Frm << 1). RefPicList0 shall contain refPicCol.
  – Otherwise (field_pic_flag is equal to 1), MapColToList0(refIdxCol) returns the lowest valued reference index refIdxL0 in the current reference picture list RefPicList0 that references the field of refPicCol with the same parity as the current picture CurrPic. RefPicList0 shall contain the field of refPicCol with the same parity as the current picture CurrPic.
– Otherwise (vertMvScale is equal to Fld_To_Frm), MapColToList0(refIdxCol) returns the lowest valued reference index refIdxL0 in the current reference picture list RefPicList0 that references the frame or complementary field pair that contains refPicCol. RefPicList0 shall contain a frame or complementary field pair that contains the field refPicCol.

NOTE 2 – A decoded reference picture that was marked as "used for short-term reference" when it was referenced in the decoding process of the picture containing the co-located macroblock may have been modified to be marked as "used for long-term reference" before being used for reference for inter prediction using the direct prediction mode for the current macroblock.

Depending on the value of vertMvScale the vertical component of mvCol is modified as follows:

– If vertMvScale is equal to Frm_To_Fld

$$\text{mvCol[1] = mvCol[1] / 2} \tag{8-193}$$

– Otherwise, if vertMvScale is equal to Fld_To_Frm

$$\text{mvCol[1] = mvCol[1] * 2} \tag{8-194}$$

– Otherwise (vertMvScale is equal to One_To_One), mvCol[1] remains unchanged.

The variables currPicOrField, pic0, and pic1, are derived as follows:

– If field_pic_flag is equal to 0 and the current macroblock is a field macroblock, the following applies:
  1. currPicOrField is the field of the current picture CurrPic that has the same parity as the current macroblock.
  2. pic1 is the field of RefPicList1[0] that has the same parity as the current macroblock.
  3. The variable pic0 is derived as follows:
     – If refIdxL0 % 2 is equal to 0, pic0 is the field of RefPicList0[refIdxL0 / 2] that has the same parity as the current macroblock.
     – Otherwise (refIdxL0 % 2 is not equal to 0), pic0 is the field of RefPicList0[refIdxL0 / 2] that has the opposite parity of the current macroblock.

– Otherwise (field_pic_flag is equal to 1 or the current macroblock is a frame macroblock), currPicOrField is the current picture CurrPic, pic1 is the decoded reference picture RefPicList1[0], and pic0 is the decoded reference picture RefPicList0[refIdxL0].

The two motion vectors mvL0 and mvL1 for each 4x4 sub-macroblock partition of the current macroblock are derived as follows:

NOTE 3 – It is often the case that many of the 4x4 sub-macroblock partitions share the same motion vectors and reference pictures. In these cases, temporal direct mode motion compensation can calculate the inter prediction sample values in larger units than 4x4 luma sample blocks. For example, when direct_8x8_inference_flag is equal to 1, at least each 8x8 luma sample quadrant of the macroblock shares the same motion vectors and reference pictures.

– If the reference index refIdxL0 refers to a long-term reference picture, or DiffPicOrderCnt(pic1, pic0) is equal to 0, the motion vectors mvL0, mvL1 for the direct mode partition are derived by:

$$\text{mvL0} = \text{mvCol} \quad (8\text{-}195)$$

$$\text{mvL1} = 0 \quad (8\text{-}196)$$

– Otherwise, the motion vectors mvL0, mvL1 are derived as scaled versions of the motion vector mvCol of the co-located sub-macroblock partition as specified below (see Figure 8-2).

$$\text{tx} = (16\,384 + \text{Abs}(\text{td} / 2)) / \text{td} \quad (8\text{-}197)$$

$$\text{DistScaleFactor} = \text{Clip3}(-1024, 1023, (\text{tb} * \text{tx} + 32) >> 6) \quad (8\text{-}198)$$

$$\text{mvL0} = (\text{DistScaleFactor} * \text{mvCol} + 128) >> 8 \quad (8\text{-}199)$$

$$\text{mvL1} = \text{mvL0} - \text{mvCol} \quad (8\text{-}200)$$

where tb and td are derived as:

$$\text{tb} = \text{Clip3}(-128, 127, \text{DiffPicOrderCnt}(\text{currPicOrField}, \text{pic0})) \quad (8\text{-}201)$$

$$\text{td} = \text{Clip3}(-128, 127, \text{DiffPicOrderCnt}(\text{pic1}, \text{pic0})) \quad (8\text{-}202)$$

NOTE 4 – mvL0 and mvL1 cannot exceed the ranges specified in Annex A.

The prediction utilization flags predFlagL0 and predFlagL1 are both set equal to 1.

Figure 8-2 illustrates the temporal direct-mode motion vector inference when the current picture is temporally between the reference picture from reference picture list 0 and the reference picture from reference picture list 1.

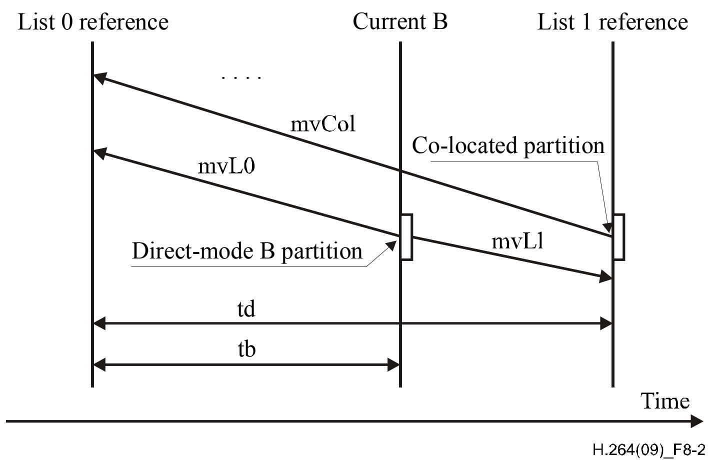


Figure 8-2 – Example for temporal direct-mode motion vector inference (informative)

8.4.1.3 Derivation process for luma motion vector prediction

Inputs to this process are:

– the macroblock partition index mbPartIdx,

– the sub-macroblock partition index subMbPartIdx,

– the reference index of the current partition refIdxLX (with X being 0 or 1),

– the variable currSubMbType.

Output of this process is the prediction mvpLX of the motion vector mvLX (with X being 0 or 1).

The derivation process for the neighbouring blocks for motion data in subclause 8.4.1.3.2 is invoked with mbPartIdx, subMbPartIdx, currSubMbType, and listSuffixFlag = X (with X being 0 or 1 for refIdxLX being refIdxL0 or refIdxL1, respectively) as the input and with mbAddrN\mbPartIdxN\subMbPartIdxN, reference indices refIdxLXN and the motion vectors mvLXN with N being replaced by A, B, or C as the output.

The motion vector predictor mvpLX is derived as follows:

– If MbPartWidth(mb_type) is equal to 16, MbPartHeight(mb_type) is equal to 8, mbPartIdx is equal to 0, and refIdxLXB is equal to refIdxLX, the motion vector predictor mvpLX is derived by:

mvpLX = mvLXB (8-203)

– Otherwise, if MbPartWidth(mb_type) is equal to 16, MbPartHeight(mb_type) is equal to 8, mbPartIdx is equal to 1, and refIdxLXA is equal to refIdxLX, the motion vector predictor mvpLX is derived by:

mvpLX = mvLXA (8-204)

– Otherwise, if MbPartWidth(mb_type) is equal to 8, MbPartHeight(mb_type) is equal to 16, mbPartIdx is equal to 0, and refIdxLXA is equal to refIdxLX, the motion vector predictor mvpLX is derived by:

mvpLX = mvLXA (8-205)

– Otherwise, if MbPartWidth(mb_type) is equal to 8, MbPartHeight(mb_type) is equal to 16, mbPartIdx is equal to 1, and refIdxLXC is equal to refIdxLX, the motion vector predictor mvpLX is derived by:

mvpLX = mvLXC (8-206)

– Otherwise, the derivation process for median luma motion vector prediction in subclause 8.4.1.3.1 is invoked with mbAddrN\mbPartIdxN\subMbPartIdxN, mvLXN, refIdxLXN with N being replaced by A, B, or C, and refIdxLX as the inputs and the output is assigned to the motion vector predictor mvpLX.

Figure 8-3 illustrates the non-median prediction as specified in Equations 8-203 to 8-206.

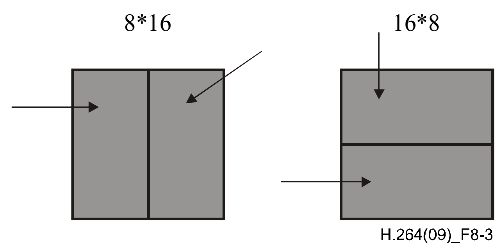


Figure 8-3 – Directional segmentation prediction (informative)

8.4.1.3.1 Derivation process for median luma motion vector prediction

Inputs to this process are:

– the neighbouring partitions mbAddrN\mbPartIdxN\subMbPartIdxN (with N being replaced by A, B, or C),

– the motion vectors mvLXN (with N being replaced by A, B, or C) of the neighbouring partitions,

– the reference indices refIdxLXN (with N being replaced by A, B, or C) of the neighbouring partitions,

– the reference index refIdxLX of the current partition.

Output of this process is the motion vector prediction mvpLX.

The variable mvpLX is derived as specified by the following ordered steps:

1. When both partitions mbAddrB\mbPartIdxB\subMbPartIdxB and mbAddrC\mbPartIdxC\subMbPartIdxC are not available and mbAddrA\mbPartIdxA\subMbPartIdxA is available,

   mvLXB = mvLXA (8-207)

   mvLXC = mvLXA (8-208)

   refIdxLXB = refIdxLXA (8-209)

   refIdxLXC = refIdxLXA (8-210)

2. Depending on reference indices refIdxLXA, refIdxLXB, or refIdxLXC, the following applies:

   – If one and only one of the reference indices refIdxLXA, refIdxLXB, or refIdxLXC is equal to the reference index refIdxLX of the current partition, the following applies. Let refIdxLXN be the reference index that is equal to refIdxLX, the motion vector mvLXN is assigned to the motion vector prediction mvpLX:

     mvpLX = mvLXN (8-211)

   – Otherwise, each component of the motion vector prediction mvpLX is given by the median of the corresponding vector components of the motion vector mvLXA, mvLXB, and mvLXC:

     mvpLX[0] = Median(mvLXA[0], mvLXB[0], mvLXC[0]) (8-212)

     mvpLX[1] = Median(mvLXA[1], mvLXB[1], mvLXC[1]) (8-213)

8.4.1.3.2 Derivation process for motion data of neighbouring partitions

Inputs to this process are:

– the macroblock partition index mbPartIdx,

– the sub-macroblock partition index subMbPartIdx,

– the current sub-macroblock type currSubMbType,

– the list suffix flag listSuffixFlag.

Outputs of this process are (with N being replaced by A, B, or C)

– mbAddrN\mbPartIdxN\subMbPartIdxN specifying neighbouring partitions,

– the motion vectors mvLXN of the neighbouring partitions,

– the reference indices refIdxLXN of the neighbouring partitions.

Variable names that include the string "LX" are interpreted with the X being equal to listSuffixFlag.

The partitions mbAddrN\mbPartIdxN\subMbPartIdxN with N being either A, B, or C are derived in the following ordered steps:

1. Let mbAddrD\mbPartIdxD\subMbPartIdxD be variables specifying an additional neighbouring partition.

2. The process in subclause 6.4.11.7 is invoked with mbPartIdx, currSubMbType, and subMbPartIdx as input and the output is assigned to mbAddrN\mbPartIdxN\subMbPartIdxN with N being replaced by A, B, C, or D.

3. When the partition mbAddrC\mbPartIdxC\subMbPartIdxC is not available, the following applies:

   mbAddrC = mbAddrD (8-214)

   mbPartIdxC = mbPartIdxD (8-215)

   subMbPartIdxC = subMbPartIdxD (8-216)

The motion vectors mvLXN and reference indices refIdxLXN (with N being A, B, or C) are derived as follows:

– If the macroblock partition or sub-macroblock partition mbAddrN\mbPartIdxN\subMbPartIdxN is not available or mbAddrN is coded in an Intra macroblock prediction mode or predFlagLX of mbAddrN\mbPartIdxN\subMbPartIdxN is equal to 0, both components of mvLXN are set equal to 0 and refIdxLXN is set equal to −1.

– Otherwise, the following ordered steps are specified:

  1. The motion vector mvLXN and reference index refIdxLXN are set equal to MvLX[mbPartIdxN][subMbPartIdxN] and RefIdxLX[mbPartIdxN], respectively, which are the motion vector mvLX and reference index refIdxLX that have been assigned to the (sub-)macroblock partition mbAddrN\mbPartIdxN\subMbPartIdxN.

  2. The variables mvLXN[1] and refIdxLXN are further processed as follows:

     – If the current macroblock is a field macroblock and the macroblock mbAddrN is a frame macroblock

$$\text{mvLXN[1]} = \text{mvLXN[1]} / 2 \qquad (8\text{-}217)$$

$$\text{refIdxLXN} = \text{refIdxLXN} * 2 \qquad (8\text{-}218)$$

     – Otherwise, if the current macroblock is a frame macroblock and the macroblock mbAddrN is a field macroblock

$$\text{mvLXN[1]} = \text{mvLXN[1]} * 2 \qquad (8\text{-}219)$$

$$\text{refIdxLXN} = \text{refIdxLXN} / 2 \qquad (8\text{-}220)$$

     – Otherwise, the vertical motion vector component mvLXN[1] and the reference index refIdxLXN remain unchanged.

8.4.1.4 Derivation process for chroma motion vectors

This process is only invoked when ChromaArrayType is not equal to 0.

Inputs to this process are a luma motion vector mvLX and a reference index refIdxLX.

Output of this process is a chroma motion vector mvCLX.

A chroma motion vector is derived from the corresponding luma motion vector.

The precision of the chroma motion vector components is 1 ÷ (4 * SubWidthC) horizontally and 1 ÷ (4 * SubHeightC) vertically.

NOTE – For example, when using the 4:2:0 chroma format, since the units of luma motion vectors are one-quarter luma sample units and chroma has half horizontal and vertical resolution compared to luma, the units of chroma motion vectors are one-eighth chroma sample units, i.e., a value of 1 for the chroma motion vector refers to a one-eighth chroma sample displacement. For example, when the luma vector applies to 8x16 luma samples, the corresponding chroma vector in 4:2:0 chroma format applies to 4x8 chroma samples and when the luma vector applies to 4x4 luma samples, the corresponding chroma vector in 4:2:0 chroma format applies to 2x2 chroma samples.

For the derivation of the motion vector mvCLX, the following applies:

– If ChromaArrayType is not equal to 1 or the current macroblock is a frame macroblock, the horizontal and vertical components of the chroma motion vector mvCLX are derived as:

$$\text{mvCLX[0]} = \text{mvLX[0]} \qquad (8\text{-}221)$$
$$\text{mvCLX[1]} = \text{mvLX[1]} \qquad (8\text{-}222)$$

– Otherwise (ChromaArrayType is equal to 1 and the current macroblock is a field macroblock), only the horizontal component of the chroma motion vector mvCLX[0] is derived using Equation 8-221. The vertical component of the chroma motion vector mvCLX[1] is dependent on the parity of the current field or the current macroblock and the reference picture, which is referred by the reference index refIdxLX. mvCLX[1] is derived from mvLX[1] according to Table 8-10.

Table 8-10 – Derivation of the vertical component of the chroma vector in field coding mode

Parity conditions		mvCLX[1]
Reference picture (refIdxLX)	Current field (picture/macroblock)	
Top field	Bottom field	mvLX[1] + 2
Bottom field	Top field	mvLX[1] − 2
Otherwise		mvLX[1]

8.4.2 Decoding process for Inter prediction samples

Inputs to this process are:

– a macroblock partition mbPartIdx,

– a sub-macroblock partition subMbPartIdx,

– variables specifying partition width and height for luma and chroma (if available), partWidth, partHeight, partWidthC (if available) and partHeightC (if available),

– luma motion vectors mvL0 and mvL1 and when ChromaArrayType is not equal to 0 chroma motion vectors mvCL0 and mvCL1,

– reference indices refIdxL0 and refIdxL1,

– prediction list utilization flags, predFlagL0 and predFlagL1,

– variables for weighted prediction logWD_C, w_{0C}, w_{1C}, o_{0C}, o_{1C} with C being replaced by L and, when ChromaArrayType is not equal to 0, Cb and Cr.

Outputs of this process are the Inter prediction samples predPart, which are a (partWidth)x(partHeight) array predPart_L of prediction luma samples, and when ChromaArrayType is not equal to 0 two (partWidthC)x(partHeightC) arrays predPart_{Cb}, predPart_{Cr} of prediction chroma samples, one for each of the chroma components Cb and Cr.

Let predPartL0_L and predPartL1_L be (partWidth)x(partHeight) arrays of predicted luma sample values and when ChromaArrayType is not equal to 0 predPartL0_{Cb}, predPartL1_{Cb}, predPartL0_{Cr}, and predPartL1_{Cr} be (partWidthC)x(partHeightC) arrays of predicted chroma sample values.

For LX being replaced by either L0 or L1 in the variables predFlagLX, RefPicListX, refIdxLX, refPicLX, predPartLX, the following is specified.

When predFlagLX is equal to 1, the following applies:

– The reference picture consisting of an ordered two-dimensional array refPicLX_L of luma samples and when ChromaArrayType is not equal to 0 two ordered two-dimensional arrays refPicLX_{Cb} and refPicLX_{Cr} of chroma samples is derived by invoking the process specified in subclause 8.4.2.1 with refIdxLX and RefPicListX given as input.

– The array predPartLX_L and when ChromaArrayType is not equal to 0 the arrays predPartLX_{Cb} and predPartLX_{Cr} are derived by invoking the process specified in subclause 8.4.2.2 with the current partition specified by mbPartIdx\subMbPartIdx, the motion vectors mvLX, mvCLX (if available), and the reference arrays with refPicLX_L, refPicLX_{Cb} (if available), and refPicLX_{Cr} (if available) given as input.

For C being replaced by L, Cb (if available), or Cr (if available), the array predPart_C of the prediction samples of component C is derived by invoking the process specified in subclause 8.4.2.3 with the current partition specified by mbPartIdx and subMbPartIdx, the prediction utilization flags predFlagL0 and predFlagL1, the arrays predPartL0_C and predPartL1_C, and the variables for weighted prediction logWD_C, w_{0C}, w_{1C}, o_{0C}, o_{1C} given as input.

8.4.2.1 Reference picture selection process

Input to this process is a reference index refIdxLX.

Output of this process is a reference picture consisting of a two-dimensional array of luma samples refPicLX_L and, when ChromaArrayType is not equal to 0, two two-dimensional arrays of chroma samples refPicLX_{Cb} and refPicLX_{Cr}.

Depending on field_pic_flag, the reference picture list RefPicListX (which has been derived as specified in subclause 8.2.4) consists of the following.

– If field_pic_flag is equal to 1, each entry of RefPicListX is a reference field or a field of a reference frame.

– Otherwise (field_pic_flag is equal to 0), each entry of RefPicListX is a reference frame or a complementary reference field pair.

For the derivation of the reference picture, the following applies:

– If field_pic_flag is equal to 1, the reference field or field of a reference frame RefPicListX[refIdxLX] is the output. The output reference field or field of a reference frame consists of a $(PicWidthInSamples_L)x(PicHeightInSamples_L)$ array of luma samples $refPicLX_L$ and, when ChromaArrayType is not equal to 0, two $(PicWidthInSamples_C)x(PicHeightInSamples_C)$ arrays of chroma samples $refPicLX_{Cb}$ and $refPicLX_{Cr}$.

– Otherwise (field_pic_flag is equal to 0), the following applies:

  – If the current macroblock is a frame macroblock, the reference frame or complementary reference field pair RefPicListX[refIdxLX] is the output. The output reference frame or complementary reference field pair consists of a $(PicWidthInSamples_L)x(PicHeightInSamples_L)$ array of luma samples $refPicLX_L$ and, when ChromaArrayType is not equal to 0, two $(PicWidthInSamples_C)x(PicHeightInSamples_C)$ arrays of chroma samples $refPicLX_{Cb}$ and $refPicLX_{Cr}$.

  – Otherwise (the current macroblock is a field macroblock), the following ordered steps are specified:

    1. Let refFrame be the reference frame or complementary reference field pair RefPicListX[refIdxLX / 2].

    2. The field of refFrame is selected as follows:

       – If refIdxLX % 2 is equal to 0, the field of refFrame that has the same parity as the current macroblock is the output.

       – Otherwise (refIdxLX % 2 is equal to 1), the field of refFrame that has the opposite parity as the current macroblock is the output.

    3. The output reference field or field of a reference frame consists of a $(PicWidthInSamples_L)x(PicHeightInSamples_L / 2)$ array of luma samples $refPicLX_L$ and, when ChromaArrayType is not equal to 0, two $(PicWidthInSamples_C)x(PicHeightInSamples_C / 2)$ arrays of chroma samples $refPicLX_{Cb}$ and $refPicLX_{Cr}$.

Depending on separate_colour_plane_flag, the following applies:

– If separate_colour_plane_flag is equal to 0, the reference picture sample arrays $refPicLX_L$, $refPicLX_{Cb}$ (if available), and $refPicLX_{Cr}$ (if available) correspond to decoded sample arrays S_L, S_{Cb} (if available), S_{Cr} (if available) derived in subclause 8.7 for a previously-decoded reference field or reference frame or complementary reference field pair or field of a reference frame.

– Otherwise (separate_colour_plane_flag is equal to 1), the following applies:

  – If colour_plane_id is equal to 0, the reference picture sample array $refPicLX_L$ corresponds to the decoded sample array S_L derived in subclause 8.7 for a previously-decoded reference field or reference frame or complementary reference field pair or field of a reference frame.

  – Otherwise, if colour_plane_id is equal to 1, the reference picture sample array $refPicLX_L$ corresponds to the decoded sample array S_{Cb} derived in subclause 8.7 for a previously-decoded reference field or reference frame or complementary reference field pair or field of a reference frame.

  – Otherwise (colour_plane_id is equal to 2), the reference picture sample array $refPicLX_L$ corresponds to the decoded sample array S_{Cr} derived in subclause 8.7 for a previously-decoded reference field or reference frame or complementary reference field pair or field of a reference frame.

8.4.2.2 Fractional sample interpolation process

Inputs to this process are:

– the current partition given by its partition index mbPartIdx and its sub-macroblock partition index subMbPartIdx,

– the width and height partWidth, partHeight of this partition in luma-sample units,

– a luma motion vector mvLX given in quarter-luma-sample units,

– when ChromaArrayType is not equal to 0, a chroma motion vector mvCLX with a precision of one-(4*SubWidthC)-th chroma-sample units horizontally and one-(4*SubHeightC)-th chroma-sample units vertically,

– the selected reference picture sample arrays $refPicLX_L$, and when ChromaArrayType is not equal to 0, $refPicLX_{Cb}$, and $refPicLX_{Cr}$.

Outputs of this process are:

– a (partWidth)x(partHeight) array $predPartLX_L$ of prediction luma sample values,

– when ChromaArrayType is not equal to 0, two (partWidthC)x(partHeightC) arrays $predPartLX_{Cb}$, and $predPartLX_{Cr}$ of prediction chroma sample values.

Let (xA_L, yA_L) be the location given in full-sample units of the upper-left luma sample of the current partition given by mbPartIdx\subMbPartIdx relative to the upper-left luma sample location of the given two-dimensional array of luma samples.

Let $(xInt_L, yInt_L)$ be a luma location given in full-sample units and $(xFrac_L, yFrac_L)$ be an offset given in quarter-sample units. These variables are used only inside this subclause for specifying general fractional-sample locations inside the reference sample arrays $refPicLX_L$, $refPicLX_{Cb}$ (if available), and $refPicLX_{Cr}$ (if available).

For each luma sample location ($0 <= x_L < partWidth$, $0 <= y_L < partHeight$) inside the prediction luma sample array $predPartLX_L$, the corresponding prediction luma sample value $predPartLX_L[x_L, y_L]$ is derived as specified by the following ordered steps:

1. The variables $xInt_L$, $yInt_L$, $xFrac_L$, and $yFrac_L$ are derived by:

$$xInt_L = xA_L + (mvLX[0] >> 2) + x_L \quad (8\text{-}223)$$
$$yInt_L = yA_L + (mvLX[1] >> 2) + y_L \quad (8\text{-}224)$$

$$xFrac_L = mvLX[0] \& 3 \quad (8\text{-}225)$$
$$yFrac_L = mvLX[1] \& 3 \quad (8\text{-}226)$$

2. The prediction luma sample value $predPartLX_L[x_L, y_L]$ is derived by invoking the process specified in subclause 8.4.2.2.1 with $(xInt_L, yInt_L)$, $(xFrac_L, yFrac_L)$ and $refPicLX_L$ given as input.

When ChromaArrayType is not equal to 0, the following applies.

Let $(xInt_C, yInt_C)$ be a chroma location given in full-sample units and $(xFrac_C, yFrac_C)$ be an offset given in one-(4*SubWidthC)-th chroma-sample units horizontally and one-(4*SubHeightC)-th chroma-sample units vertically. These variables are used only inside this subclause for specifying general fractional-sample locations inside the reference sample arrays $refPicLX_{Cb}$, and $refPicLX_{Cr}$.

For each chroma sample location ($0 <= x_C < partWidthC$, $0 <= y_C < partHeightC$) inside the prediction chroma sample arrays $predPartLX_{Cb}$ and $predPartLX_{Cr}$, the corresponding prediction chroma sample values $predPartLX_{Cb}[x_C, y_C]$ and $predPartLX_{Cr}[x_C, y_C]$ are derived as specified by the following ordered steps:

1. Depending on ChromaArrayType, the variables $xInt_C$, $yInt_C$, $xFrac_C$, and $yFrac_C$ are derived as follows:

   – If ChromaArrayType is equal to 1,

$$xInt_C = (xA_L / SubWidthC) + (mvCLX[0] >> 3) + x_C \quad (8\text{-}227)$$
$$yInt_C = (yA_L / SubHeightC) + (mvCLX[1] >> 3) + y_C \quad (8\text{-}228)$$

$$xFrac_C = mvCLX[0] \& 7 \quad (8\text{-}229)$$
$$yFrac_C = mvCLX[1] \& 7 \quad (8\text{-}230)$$

   – Otherwise, if ChromaArrayType is equal to 2,

$$xInt_C = (xA_L / SubWidthC) + (mvCLX[0] >> 3) + x_C \quad (8\text{-}231)$$
$$yInt_C = (yA_L / SubHeightC) + (mvCLX[1] >> 2) + y_C \quad (8\text{-}232)$$

$$xFrac_C = mvCLX[0] \& 7 \quad (8\text{-}233)$$
$$yFrac_C = (mvCLX[1] \& 3) << 1 \quad (8\text{-}234)$$

– Otherwise (ChromaArrayType is equal to 3),

$$xInt_C = xA_L + (mvLX[0] >> 2) + x_C \tag{8-235}$$
$$yInt_C = yA_L + (mvLX[1] >> 2) + y_C \tag{8-236}$$

$$xFrac_C = (mvCX[0] \& 3) \tag{8-237}$$
$$yFrac_C = (mvCX[1] \& 3) \tag{8-238}$$

2. Depending on ChromaArrayType, the following applies:

   – If ChromaArrayType is not equal to 3, the following applies:

     – The prediction sample value $predPartLX_{Cb}[x_C, y_C]$ is derived by invoking the process specified in subclause 8.4.2.2.2 with ($xInt_C$, $yInt_C$), ($xFrac_C$, $yFrac_C$) and $refPicLX_{Cb}$ given as input.

     – The prediction sample value $predPartLX_{Cr}[x_C, y_C]$ is derived by invoking the process specified in subclause 8.4.2.2.2 with ($xInt_C$, $yInt_C$), ($xFrac_C$, $yFrac_C$) and $refPicLX_{Cr}$ given as input.

   – Otherwise (ChromaArrayType is equal to 3), the following applies:

     – The prediction sample value $predPartLX_{Cb}[x_C, y_C]$ is derived by invoking the process specified in subclause 8.4.2.2.1 with ($xInt_C$, $yInt_C$), ($xFrac_C$, $yFrac_C$) and $refPicLX_{Cb}$ given as input.

     – The prediction sample value $predPartLX_{Cr}[x_C, y_C]$ is derived by invoking the process specified in subclause 8.4.2.2.1 with ($xInt_C$, $yInt_C$), ($xFrac_C$, $yFrac_C$) and $refPicLX_{Cr}$ given as input.

8.4.2.2.1 Luma sample interpolation process

Inputs to this process are:

– a luma location in full-sample units ($xInt_L$, $yInt_L$),

– a luma location offset in fractional-sample units ($xFrac_L$, $yFrac_L$),

– the luma sample array of the selected reference picture $refPicLX_L$.

Output of this process is a predicted luma sample value $predPartLX_L[x_L, y_L]$.

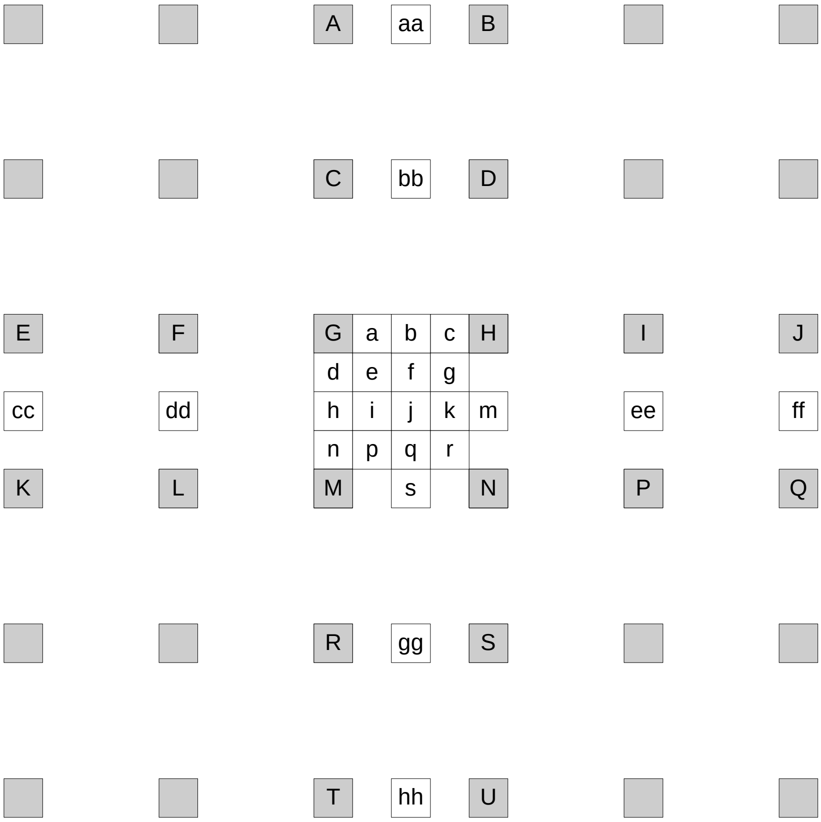


Figure 8-4 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for quarter sample luma interpolation

The variable $\text{refPicHeightEffective}_L$, which is the height of the effective reference picture luma array, is derived as follows:

– If MbaffFrameFlag is equal to 0 or mb_field_decoding_flag is equal to 0, $\text{refPicHeightEffective}_L$ is set equal to $\text{PicHeightInSamples}_L$.

– Otherwise (MbaffFrameFlag is equal to 1 and mb_field_decoding_flag is equal to 1), $\text{refPicHeightEffective}_L$ is set equal to $\text{PicHeightInSamples}_L / 2$.

In Figure 8-4, the positions labelled with upper-case letters within shaded blocks represent luma samples at full-sample locations inside the given two-dimensional array refPicLX_L of luma samples. These samples may be used for generating the predicted luma sample value $\text{predPartLX}_L[\, x_L, y_L \,]$. The locations (xZ_L, yZ_L) for each of the corresponding luma samples Z, where Z may be A, B, C, D, E, F, G, H, I, J, K, L, M, N, P, Q, R, S, T, or U, inside the given array refPicLX_L of luma samples are derived as:

$$xZ_L = \text{Clip3}(\, 0, \text{PicWidthInSamples}_L - 1, \text{xInt}_L + \text{xDZ}_L \,) \tag{8-239}$$
$$yZ_L = \text{Clip3}(\, 0, \text{refPicHeightEffective}_L - 1, \text{yInt}_L + \text{yDZ}_L \,) \tag{8-240}$$

Table 8-11 specifies (xDZ_L, yDZ_L) for different replacements of Z.

Table 8-11 – Differential full-sample luma locations

Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	P	Q	R	S	T	U
xDZ_L	0	1	0	1	−2	−1	0	1	2	3	−2	−1	0	1	2	3	0	1	0	1
yDZ_L	−2	−2	−1	−1	0	0	0	0	0	0	1	1	1	1	1	1	2	2	3	3

Given the luma samples 'A' to 'U' at full-sample locations (xA_L, yA_L) to (xU_L, yU_L), the luma samples 'a' to 's' at fractional sample positions are derived by the following rules. The luma prediction values at half sample positions are derived by applying a 6-tap filter with tap values (1, −5, 20, 20, −5, 1). The luma prediction values at quarter sample positions are derived by averaging samples at full and half sample positions. The process for each fractional position is described below.

– The samples at half sample positions labelled b are derived by first calculating intermediate values denoted as b_1 by applying the 6-tap filter to the nearest integer position samples in the horizontal direction. The samples at half sample positions labelled h are derived by first calculating intermediate values denoted as h_1 by applying the 6-tap filter to the nearest integer position samples in the vertical direction:

$$b_1 = (E - 5 * F + 20 * G + 20 * H - 5 * I + J) \quad (8\text{-}241)$$
$$h_1 = (A - 5 * C + 20 * G + 20 * M - 5 * R + T) \quad (8\text{-}242)$$

The final prediction values b and h are derived using

$$b = \text{Clip1}_Y((b_1 + 16) >> 5) \quad (8\text{-}243)$$
$$h = \text{Clip1}_Y((h_1 + 16) >> 5) \quad (8\text{-}244)$$

– The samples at half sample position labelled as j are derived by first calculating intermediate value denoted as j_1 by applying the 6-tap filter to the intermediate values of the closest half sample positions in either the horizontal or vertical direction because these yield an equal result:

$$j_1 = cc - 5 * dd + 20 * h_1 + 20 * m_1 - 5 * ee + ff, \text{ or} \quad (8\text{-}245)$$
$$j_1 = aa - 5 * bb + 20 * b_1 + 20 * s_1 - 5 * gg + hh \quad (8\text{-}246)$$

where intermediate values denoted as aa, bb, gg, s_1 and hh are derived by applying the 6-tap filter horizontally in the same manner as the derivation of b_1 and intermediate values denoted as cc, dd, ee, m_1 and ff are derived by applying the 6-tap filter vertically in the same manner as the derivation of h_1. The final prediction value j are derived using

$$j = \text{Clip1}_Y((j_1 + 512) >> 10) \quad (8\text{-}247)$$

– The final prediction values s and m are derived from s_1 and m_1 in the same manner as the derivation of b and h, as given by

$$s = \text{Clip1}_Y((s_1 + 16) >> 5) \quad (8\text{-}248)$$
$$m = \text{Clip1}_Y((m_1 + 16) >> 5) \quad (8\text{-}249)$$

– The samples at quarter sample positions labelled as a, c, d, n, f, i, k, and q are derived by averaging with upward rounding of the two nearest samples at integer and half sample positions using

$$a = (G + b + 1) >> 1 \quad (8\text{-}250)$$
$$c = (H + b + 1) >> 1 \quad (8\text{-}251)$$
$$d = (G + h + 1) >> 1 \quad (8\text{-}252)$$
$$n = (M + h + 1) >> 1 \quad (8\text{-}253)$$
$$f = (b + j + 1) >> 1 \quad (8\text{-}254)$$
$$i = (h + j + 1) >> 1 \quad (8\text{-}255)$$
$$k = (j + m + 1) >> 1 \quad (8\text{-}256)$$
$$q = (j + s + 1) >> 1 \quad (8\text{-}257)$$

– The samples at quarter sample positions labelled as e, g, p, and r are derived by averaging with upward rounding of the two nearest samples at half sample positions in the diagonal direction using

$$e = (b + h + 1) >> 1 \quad (8\text{-}258)$$
$$g = (b + m + 1) >> 1 \quad (8\text{-}259)$$
$$p = (h + s + 1) >> 1 \quad (8\text{-}260)$$
$$r = (m + s + 1) >> 1. \quad (8\text{-}261)$$

The luma location offset in fractional-sample units ($xFrac_L$, $yFrac_L$) specifies which of the generated luma samples at full-sample and fractional-sample locations is assigned to the predicted luma sample value $predPartLX_L[x_L, y_L]$. This assignment is done according to Table 8-12. The value of $predPartLX_L[x_L, y_L]$ is the output.

Table 8-12 – Assignment of the luma prediction sample $predPartLX_L[x_L, y_L]$

$xFrac_L$	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
$yFrac_L$	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
$predPartLX_L[x_L, y_L]$	G	d	h	n	a	e	i	p	b	f	j	q	c	g	k	r

8.4.2.2.2 Chroma sample interpolation process

This process is only invoked when ChromaArrayType is equal to 1 or 2.

Inputs to this process are:

– a chroma location in full-sample units ($xInt_C$, $yInt_C$),

– a chroma location offset in fractional-sample units ($xFrac_C$, $yFrac_C$),

– chroma component samples from the selected reference picture $refPicLX_C$.

Output of this process is a predicted chroma sample value $predPartLX_C[x_C, y_C]$.

In Figure 8-5, the positions labelled with A, B, C, and D represent chroma samples at full-sample locations inside the given two-dimensional array $refPicLX_C$ of chroma samples.

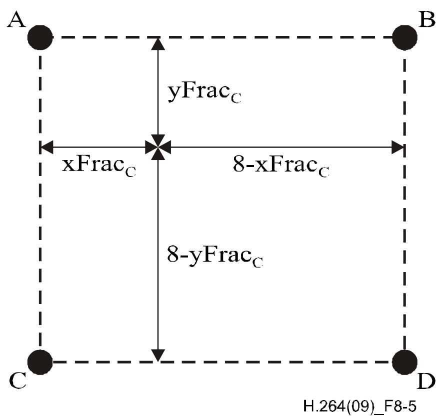


Figure 8-5 – Fractional sample position dependent variables in chroma interpolation and surrounding integer position samples A, B, C, and D

The variable $refPicHeightEffective_C$, which is the height of the effective reference picture chroma array, is derived as follows:

– If MbaffFrameFlag is equal to 0 or mb_field_decoding_flag is equal to 0, $refPicHeightEffective_C$ is set equal to $PicHeightInSamples_C$.

– Otherwise (MbaffFrameFlag is equal to 1 and mb_field_decoding_flag is equal to 1), $refPicHeightEffective_C$ is set equal to $PicHeightInSamples_C$ / 2.

The sample coordinates specified in Equations 8-262 through 8-269 are used for generating the predicted chroma sample value $predPartLX_C[x_C, y_C]$.

$$xA_C = Clip3(0, PicWidthInSamples_C - 1, xInt_C) \quad (8\text{-}262)$$
$$xB_C = Clip3(0, PicWidthInSamples_C - 1, xInt_C + 1) \quad (8\text{-}263)$$
$$xC_C = Clip3(0, PicWidthInSamples_C - 1, xInt_C) \quad (8\text{-}264)$$
$$xD_C = Clip3(0, PicWidthInSamples_C - 1, xInt_C + 1) \quad (8\text{-}265)$$

$$yA_C = Clip3(0, refPicHeightEffective_C - 1, yInt_C) \quad (8\text{-}266)$$
$$yB_C = Clip3(0, refPicHeightEffective_C - 1, yInt_C) \quad (8\text{-}267)$$

$$yC_C = \text{Clip3}(\ 0,\ \text{refPicHeightEffective}_C - 1,\ \text{yInt}_C + 1\ ) \quad (8\text{-}268)$$
$$yD_C = \text{Clip3}(\ 0,\ \text{refPicHeightEffective}_C - 1,\ \text{yInt}_C + 1\ ) \quad (8\text{-}269)$$

Given the chroma samples A, B, C, and D at full-sample locations specified in Equations 8-262 through 8-269, the predicted chroma sample value $\text{predPartLX}_C[\ x_C, y_C\ ]$ is derived as:

$$\begin{aligned}\text{predPartLX}_C[\ x_C, y_C\ ] = (\ (\ 8 - \text{xFrac}_C\ ) * (\ 8 - \text{yFrac}_C\ ) * A + \text{xFrac}_C * (\ 8 - \text{yFrac}_C\ ) * B +\\ (\ 8 - \text{xFrac}_C\ ) * \text{yFrac}_C * C \quad + \text{xFrac}_C * \text{yFrac}_C * D \quad + 32\ ) >> 6\end{aligned} \quad (8\text{-}270)$$

8.4.2.3 Weighted sample prediction process

Inputs to this process are:

- mbPartIdx: the current partition given by the partition index,
- subMbPartIdx: the sub-macroblock partition index,
- predFlagL0 and predFlagL1: prediction list utilization flags,
- predPartLX_L: a (partWidth)x(partHeight) array of prediction luma samples (with LX being replaced by L0 or L1 depending on predFlagL0 and predFlagL1),
- when ChromaArrayType is not equal to 0, predPartLX_{Cb} and predPartLX_{Cr}: (partWidthC)x(partHeightC) arrays of prediction chroma samples, one for each of the chroma components Cb and Cr (with LX being replaced by L0 or L1 depending on predFlagL0 and predFlagL1),
- variables for weighted prediction logWD_C, w_{0C}, w_{1C}, o_{0C}, o_{1C} with C being replaced by L and, when ChromaArrayType is not equal to 0, Cb and Cr.

Outputs of this process are:

- predPart_L: a (partWidth)x(partHeight) array of prediction luma samples,
- when ChromaArrayType is not equal to 0, predPart_{Cb}, and predPart_{Cr}: (partWidthC)x(partHeightC) arrays of prediction chroma samples, one for each of the chroma components Cb and Cr.

For macroblocks or partitions with predFlagL0 equal to 1 in P and SP slices, the following applies:

- If weighted_pred_flag is equal to 0, the default weighted sample prediction process as described in subclause 8.4.2.3.1 is invoked with the same inputs and outputs as the process described in this subclause.
- Otherwise (weighted_pred_flag is equal to 1), the explicit weighted sample prediction process as described in subclause 8.4.2.3.2 is invoked with the same inputs and outputs as the process described in this subclause.

For macroblocks or partitions with predFlagL0 or predFlagL1 equal to 1 in B slices, the following applies:

- If weighted_bipred_idc is equal to 0, the default weighted sample prediction process as described in subclause 8.4.2.3.1 is invoked with the same inputs and outputs as the process described in this subclause.
- Otherwise, if weighted_bipred_idc is equal to 1, the explicit weighted sample prediction process as described in subclause 8.4.2.3.2 is invoked with the same inputs and outputs as the process described in this subclause.
- Otherwise (weighted_bipred_idc is equal to 2), the following applies:
  - If predFlagL0 is equal to 1 and predFlagL1 is equal to 1, the implicit weighted sample prediction process as described in subclause 8.4.2.3.2 is invoked with the same inputs and outputs as the process described in this subclause.
  - Otherwise (predFlagL0 or predFlagL1 are equal to 1 but not both), the default weighted sample prediction process as described in subclause 8.4.2.3.1 is invoked with the same inputs and outputs as the process described in this subclause.

8.4.2.3.1 Default weighted sample prediction process

Input to this process are the same as specified in subclause 8.4.2.3.

Output of this process are the same as specified in subclause 8.4.2.3.

Depending on the available component for which the prediction block is derived, the following applies:

- If the luma sample prediction values $\text{predPart}_L[\ x, y\ ]$ are derived, the following applies with C set equal to L, x set equal to 0..partWidth − 1, and y set equal to 0..partHeight − 1.

– Otherwise, if the chroma Cb component sample prediction values $predPart_{Cb}[x, y]$ are derived, the following applies with C set equal to Cb, x set equal to 0..partWidthC − 1, and y set equal to 0..partHeightC − 1.

– Otherwise (the chroma Cr component sample prediction values $predPart_{Cr}[x, y]$ are derived), the following applies with C set equal to Cr, x set equal to 0..partWidthC − 1, and y set equal to 0..partHeightC − 1.

The prediction sample values are derived as follows:

– If predFlagL0 is equal to 1 and predFlagL1 is equal to 0,

$$predPart_C[x, y] = predPartL0_C[x, y] \tag{8-271}$$

– Otherwise, if predFlagL0 is equal to 0 and predFlagL1 is equal to 1,

$$predPart_C[x, y]= predPartL1_C[x, y] \tag{8-272}$$

– Otherwise (predFlagL0 and predFlagL1 are equal to 1),

$$predPart_C[x, y] = (predPartL0_C[x, y] + predPartL1_C[x, y] + 1) >> 1. \tag{8-273}$$

8.4.2.3.2 Weighted sample prediction process

Inputs to this process are the same as specified in subclause 8.4.2.3.

Outputs of this process are the same as specified in subclause 8.4.2.3.

Depending on the available component for which the prediction block is derived, the following applies:

– If the luma sample prediction values $predPart_L[x, y]$ are derived, the following applies with C set equal to L, x set equal to 0..partWidth − 1, y set equal to 0..partHeight − 1, and Clip1( ) being substituted with $Clip1_Y( )$.

– Otherwise, if the chroma Cb component sample prediction values $predPart_{Cb}[x, y]$ are derived, the following applies with C set equal to Cb, x set equal to 0..partWidthC − 1, y set equal to 0..partHeightC − 1, and Clip1( ) being substituted with $Clip1_C( )$.

– Otherwise (the chroma Cr component sample prediction values $predPart_{Cr}[x, y]$ are derived), the following applies with C set equal to Cr, x set equal to 0..partWidthC − 1, y set equal to 0..partHeightC − 1, and Clip1( ) being substituted with $Clip1_C( )$.

The prediction sample values are derived as follows:

– If the predFlagL0 is equal to 1 and predFlagL1 is equal to 0, the final predicted sample values $predPart_C[x, y]$ are derived by

$$\begin{aligned}&\text{if}(logWD_C >= 1)\\&\quad predPart_C[x, y] = Clip1(((predPartL0_C[x, y] * w_{0C} + 2^{logWD_C - 1}) >> logWD_C) + o_{0C})\\&\text{else}\\&\quad predPart_C[x, y] = Clip1(predPartL0_C[x, y] * w_{0C} + o_{0C})\end{aligned} \tag{8-274}$$

– Otherwise, if the predFlagL0 is equal to 0 and predFlagL1 is equal to 1, the final predicted sample values $predPart_C[x, y]$ are derived by

$$\begin{aligned}&\text{if}(logWD_C >= 1)\\&\quad predPart_C[x, y] = Clip1(((predPartL1_C[x, y] * w_{1C} + 2^{logWD_C - 1}) >> logWD_C) + o_{1C})\\&\text{else}\\&\quad predPart_C[x, y] = Clip1(predPartL1_C[x, y] * w_{1C} + o_{1C})\end{aligned} \tag{8-275}$$

– Otherwise (both predFlagL0 and predFlagL1 are equal to 1), the final predicted sample values $predPart_C[x, y]$ are derived by

$$\begin{aligned}predPart_C[x, y] = Clip1(((predPartL0_C[x, y] * w_{0C} + predPartL1_C[x, y] * w_{1C} + 2^{logWD_C}) >>\\ (logWD_C + 1)) + ((o_{0C} + o_{1C} + 1) >> 1))\end{aligned} \tag{8-276}$$

8.4.3 Derivation process for prediction weights

Inputs to this process are:

– the reference indices refIdxL0 and refIdxL1,

– the prediction utilization flags predFlagL0 and predFlagL1.

Outputs of this process are variables for weighted prediction $logWD_C$, w_{0C}, w_{1C}, o_{0C}, o_{1C} with C being replaced by L and, when ChromaArrayType is not equal to 0, Cb and Cr.

The variables implicitModeFlag and explicitModeFlag are derived as follows:

– If weighted_bipred_idc is equal to 2, (slice_type % 5) is equal to 1, predFlagL0 is equal to 1, and predFlagL1 is equal to 1, implicitModeFlag is set equal to 1 and explicitModeFlag is set equal to 0.

– Otherwise, if weighted_bipred_idc is equal to 1, (slice_type % 5) is equal to 1, and predFlagL0 + predFlagL1 is equal to 1 or 2, implicitModeFlag is set equal to 0 and explicitModeFlag is set equal to 1.

– Otherwise, if weighted_pred_flag is equal to 1, (slice_type % 5) is equal to 0 or 3, and predFlagL0 is equal to 1, implicitModeFlag is set equal to 0 and explicitModeFlag is set equal to 1.

– Otherwise, implicitModeFlag is set equal to 0 and explicitModeFlag is set equal to 0.

For C being replaced by L and, when ChromaArrayType is not equal to 0, Cb and Cr, the variables $logWD_C$, w_{0C}, w_{1C}, o_{0C}, o_{1C} are derived as follows:

– If implicitModeFlag is equal to 1, implicit mode weighted prediction is used as follows:

$$logWD_C = 5 \tag{8-277}$$

$$o_{0C} = 0 \tag{8-278}$$

$$o_{1C} = 0 \tag{8-279}$$

and w_{0C} and w_{1C} are derived as specified in the following ordered steps:

1. The variables currPicOrField, pic0, and pic1 are derived as follows:

   – If field_pic_flag is equal to 0 and the current macroblock is a field macroblock, the following applies:

     a. currPicOrField is the field of the current picture CurrPic that has the same parity as the current macroblock.

     b. The variable pic0 is derived as follows:

        – If refIdxL0 % 2 is equal to 0, pic0 is the field of RefPicList0[refIdxL0 / 2] that has the same parity as the current macroblock.

        – Otherwise (refIdxL0 % 2 is not equal to 0), pic0 is the field of RefPicList0[refIdxL0 / 2] that has the opposite parity of the current macroblock.

     c. The variable pic1 is derived as follows:

        – If refIdxL1 % 2 is equal to 0, pic1 is the field of RefPicList1[refIdxL1 / 2] that has the same parity as the current macroblock.

        – Otherwise (refIdxL1 % 2 is not equal to 0), pic1 is the field of RefPicList1[refIdxL1 / 2] that has the opposite parity of the current macroblock.

   – Otherwise (field_pic_flag is equal to 1 or the current macroblock is a frame macroblock), currPicOrField is the current picture CurrPic, pic1 is RefPicList1[refIdxL1], and pic0 is RefPicList0[refIdxL0].

2. The variables w_{0C} and w_{1C} are derived as follows:

   – If DiffPicOrderCnt(pic1, pic0) is equal to 0 or one or both of pic1 and pic0 is marked as "used for long-term reference" or (DistScaleFactor >> 2) < −64 or (DistScaleFactor >> 2) > 128, w_{0C} and w_{1C} are derived as:

$$w_{0C} = 32 \tag{8-280}$$

$$w_{1C} = 32 \tag{8-281}$$

– Otherwise, the variables tb, td, tx, and DistScaleFactor are derived from the values of currPicOrField, pic0, and pic1 using Equations 8-201, 8-202, 8-197, and 8-198, respectively, and the weights w_{0C} and w_{1C} are derived as

$$w_{0C} = 64 - (\text{DistScaleFactor} >> 2) \quad (8\text{-}282)$$

$$w_{1C} = \text{DistScaleFactor} >> 2 \quad (8\text{-}283)$$

– Otherwise, if explicitModeFlag is equal to 1, explicit mode weighted prediction is used as specified by the following ordered steps:

1. The variables refIdxL0WP and refIdxL1WP are derived as follows:

   – If MbaffFrameFlag is equal to 1 and the current macroblock is a field macroblock

$$\text{refIdxL0WP} = \text{refIdxL0} >> 1 \quad (8\text{-}284)$$

$$\text{refIdxL1WP} = \text{refIdxL1} >> 1 \quad (8\text{-}285)$$

   – Otherwise (MbaffFrameFlag is equal to 0 or the current macroblock is a frame macroblock),

$$\text{refIdxL0WP} = \text{refIdxL0} \quad (8\text{-}286)$$

$$\text{refIdxL1WP} = \text{refIdxL1} \quad (8\text{-}287)$$

2. The variables logWD_C, w_{0C}, w_{1C}, o_{0C}, and o_{1C} are derived as follows:

   – If C is equal to L for luma samples

$$\text{logWD}_C = \text{luma\_log2\_weight\_denom} \quad (8\text{-}288)$$

$$w_{0C} = \text{luma\_weight\_l0}[\ \text{refIdxL0WP}\ ] \quad (8\text{-}289)$$

$$w_{1C} = \text{luma\_weight\_l1}[\ \text{refIdxL1WP}\ ] \quad (8\text{-}290)$$

$$o_{0C} = \text{luma\_offset\_l0}[\ \text{refIdxL0WP}\ ] * (\ 1 << (\ \text{BitDepth}_Y - 8\ )\ ) \quad (8\text{-}291)$$

$$o_{1C} = \text{luma\_offset\_l1}[\ \text{refIdxL1WP}\ ] * (\ 1 << (\ \text{BitDepth}_Y - 8\ )\ ) \quad (8\text{-}292)$$

   – Otherwise (C is equal to Cb or Cr for chroma samples, with iCbCr = 0 for Cb, iCbCr = 1 for Cr),

$$\text{logWD}_C = \text{chroma\_log2\_weight\_denom} \quad (8\text{-}293)$$

$$w_{0C} = \text{chroma\_weight\_l0}[\ \text{refIdxL0WP}\ ][\ \text{iCbCr}\ ] \quad (8\text{-}294)$$

$$w_{1C} = \text{chroma\_weight\_l1}[\ \text{refIdxL1WP}\ ][\ \text{iCbCr}\ ] \quad (8\text{-}295)$$

$$o_{0C} = \text{chroma\_offset\_l0}[\ \text{refIdxL0WP}\ ][\ \text{iCbCr}\ ] * (\ 1 << (\ \text{BitDepth}_C - 8\ )\ ) \quad (8\text{-}296)$$

$$o_{1C} = \text{chroma\_offset\_l1}[\ \text{refIdxL1WP}\ ][\ \text{iCbCr}\ ] * (\ 1 << (\ \text{BitDepth}_C - 8\ )\ ) \quad (8\text{-}297)$$

– Otherwise (implicitModeFlag is equal to 0 and explicitModeFlag is equal to 0), the variables logWD_C, w_{0C}, w_{1C}, o_{0C}, o_{1C} are not used in the reconstruction process for the current macroblock.

When explicitModeFlag is equal to 1 and predFlagL0 and predFlagL1 are equal to 1, the following constraint shall be obeyed for C equal to L and, when ChromaArrayType is not equal to 0, Cb and Cr:

$$-128 <= w_{0C} + w_{1C} <= (\ (\ \text{logWD}_C\ ==\ 7\ )\ ?\ 127 : 128\ ) \quad (8\text{-}298)$$

NOTE – For implicitModeFlag equal to 1, weights w_{0C} and w_{1C} are each guaranteed to be in the range of −64..128 and the constraint expressed in Equation 8-298, although not explicitly imposed, will always be met. For explicitModeFlag equal to 1 with logWD_C equal to 7, when one of the two weights w_{0C} or w_{1C} is inferred to be equal to 128 (as a consequence of luma_weight_l0_flag, luma_weight_l1_flag, chroma_weight_l0_flag, or chroma_weight_l1_flag equal to 0), the other weight (w_{1C} or w_{0C}) must have a negative value in order for the constraint expressed in Equation 8-298 to hold (and therefore the other flag luma_weight_l0_flag, luma_weight_l1_flag, chroma_weight_l0_flag, or chroma_weight_l1_flag must be equal to 1).

8.5 Transform coefficient decoding process and picture construction process prior to deblocking filter process

Inputs to this process are Intra16x16DCLevel (if available), Intra16x16ACLevel (if available), CbIntra16x16DCLevel (if available), CbIntra16x16ACLevel (if available), CrIntra16x16DCLevel (if available), CrIntra16x16ACLevel (if available), LumaLevel4x4 (if available), LumaLevel8x8 (if available), ChromaDCLevel (if available), ChromaACLevel (if available), CbLevel4x4 (if available), CrLevel4x4 (if available), CbLevel8x8 (if available), CrLevel8x8 (if available), and available Inter or Intra prediction sample arrays for the current macroblock for the applicable components $pred_L$, $pred_{Cb}$, or $pred_{Cr}$.

NOTE 1 – When decoding a macroblock in Intra_4x4 (or Intra_8x8) macroblock prediction mode, the luma component of the macroblock prediction array may not be complete, since for each 4x4 (or 8x8) luma block, the Intra_4x4 (or Intra_8x8) prediction process for luma samples as specified in subclause 8.3.1 (or 8.3.2) and the process specified in this subclause are iterated. When ChromaArrayType is equal to 3, the Cb and Cr component of the macroblock prediction array may not be complete for the same reason.

Outputs of this process are the constructed sample arrays prior to the deblocking filter process for the applicable components S'_L, S'_{Cb}, or S'_{Cr}.

NOTE 2 – When decoding a macroblock in Intra_4x4 (or Intra_8x8) macroblock prediction mode, the luma component of the macroblock constructed sample arrays prior to the deblocking filter process may not be complete, since for each 4x4 (or 8x8) luma block, the Intra_4x4 (or Intra_8x8) prediction process for luma samples as specified in subclause 8.3.1 (or 8.3.2) and the process specified in this subclause are iterated. When ChromaArrayType is equal to 3, the Cb and Cr component of the macroblock constructed sample arrays prior to the deblocking filter process may not be complete for the same reason.

This subclause specifies transform coefficient decoding and picture construction prior to the deblocking filter process.

When the current macroblock is coded as P_Skip or B_Skip, all values of LumaLevel4x4, LumaLevel8x8, CbLevel4x4, CbLevel8x8, CrLevel4x4, CrLevel8x8, ChromaDCLevel, ChromaACLevel are set equal to 0 for the current macroblock.

8.5.1 Specification of transform decoding process for 4x4 luma residual blocks

This specification applies when transform_size_8x8_flag is equal to 0.

When the current macroblock prediction mode is not equal to Intra_16x16, the variable LumaLevel4x4 contains the levels for the luma transform coefficients. For a 4x4 luma block indexed by luma4x4BlkIdx = 0..15, the following ordered steps are specified:

1. The inverse scanning process for 4x4 transform coefficients and scaling lists as specified in subclause 8.5.6 is invoked with LumaLevel4x4[luma4x4BlkIdx] as the input and the two-dimensional array c as the output.

2. The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.12 is invoked with c as the input and r as the output.

3. When TransformBypassModeFlag is equal to 1, the macroblock prediction mode is equal to Intra_4x4, and Intra4x4PredMode[luma4x4BlkIdx] is equal to 0 or 1, the intra residual transform-bypass decoding process as specified in subclause 8.5.15 is invoked with nW set equal to 4, nH set equal to 4, horPredFlag set equal to Intra4x4PredMode[luma4x4BlkIdx], and the 4x4 array r as the inputs, and the output is a modified version of the 4x4 array r.

4. The position of the upper-left sample of a 4x4 luma block with index luma4x4BlkIdx inside the macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (xO, yO).

5. The 4x4 array u with elements u_{ij} for i, j = 0..3 is derived as:

   $$u_{ij} = \text{Clip1}_Y(pred_L[xO + j, yO + i] + r_{ij}) \qquad (8\text{-}299)$$

   When TransformBypassModeFlag is equal to 1, the bitstream shall not contain data that result in a value of u_{ij} as computed by Equation 8-299 that is not equal to $pred_L[xO + j, yO + i] + r_{ij}$.

6. The picture construction process prior to deblocking filter process in subclause 8.5.14 is invoked with u and luma4x4BlkIdx as the inputs.

8.5.2 Specification of transform decoding process for luma samples of Intra_16x16 macroblock prediction mode

When the current macroblock prediction mode is equal to Intra_16x16, the variables Intra16x16DCLevel and Intra16x16ACLevel contain the levels for the luma transform coefficients. The transform coefficient decoding proceeds in the following ordered steps:

1. The 4x4 luma DC transform coefficients of all 4x4 luma blocks of the macroblock are decoded.

   a. The inverse scanning process for 4x4 transform coefficients and scaling lists as specified in subclause 8.5.6 is invoked with Intra16x16DCLevel as the input and the two-dimensional array c as the output.

   b. The scaling and transformation process for luma DC transform coefficients for Intra_16x16 macroblock type as specified in subclause 8.5.10 is invoked with $\mathrm{BitDepth}_Y$, $\mathrm{QP'}_Y$, and c as the input and dcY as the output.

2. The 16x16 array rMb is derived by processing the 4x4 luma blocks indexed by luma4x4BlkIdx = 0..15, and for each 4x4 luma block, the following ordered steps are specified:

   a. The variable lumaList, which is a list of 16 entries, is derived. The first entry of lumaList is the corresponding value from the array dcY. Figure 8-6 shows the assignment of the indices of the array dcY to the luma4x4BlkIdx. The two numbers in the small squares refer to indices i and j in dcY_{ij}, and the numbers in large squares refer to luma4x4BlkIdx.

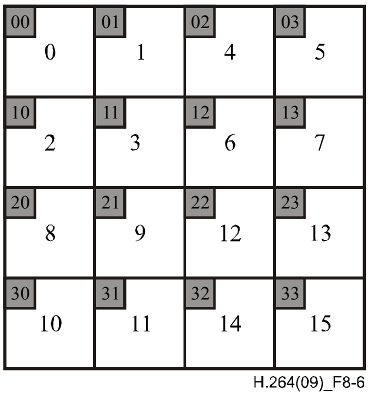


Figure 8-6 – Assignment of the indices of dcY to luma4x4BlkIdx

   The elements in lumaList with index k = 1..15 are specified as:

   $$\mathrm{lumaList}[\ k\ ] = \mathrm{Intra16x16ACLevel}[\ \mathrm{luma4x4BlkIdx}\ ][\ k - 1\ ] \tag{8-300}$$

   b. The inverse scanning process for 4x4 transform coefficients and scaling lists as specified in subclause 8.5.6 is invoked with lumaList as the input and the two-dimensional array c as the output.

   c. The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.12 is invoked with c as the input and r as the output.

   d. The position of the upper-left sample of a 4x4 luma block with index luma4x4BlkIdx inside the macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (xO, yO).

   e. The elements rMb[x, y] of the 16x16 array rMb with x = xO..xO + 3 and y = yO..yO + 3 are derived by

   $$\mathrm{rMb}[\ xO + j, yO + i\ ] = r_{ij} \tag{8-301}$$

3. When TransformBypassModeFlag is equal to 1 and Intra16x16PredMode is equal to 0 or 1, the intra residual transform-bypass decoding process as specified in subclause 8.5.15 is invoked with nW set equal to 16, nH set equal to 16, horPredFlag set equal to Intra16x16PredMode, and the 16x16 array rMb as the inputs, and the output is a modified version of the 16x16 array rMb.

4. The 16x16 array u with elements u_{ij} for i, j = 0..15 is derived as

$$u_{ij} = \text{Clip1}_Y(\ \text{pred}_L[\ j, i\ ] + \text{rMb}[\ j, i\ ]\ ) \tag{8-302}$$

   When TransformBypassModeFlag is equal to 1, the bitstream shall not contain data that result in a value of u_{ij} as computed by Equation 8-302 that is not equal to $\text{pred}_L[\ j, i\ ] + \text{rMb}[\ j, i\ ]$.

5. The picture construction process prior to deblocking filter process in subclause 8.5.14 is invoked with u as the input.

8.5.3 Specification of transform decoding process for 8x8 luma residual blocks

This specification applies when transform_size_8x8_flag is equal to 1.

The variable LumaLevel8x8[luma8x8BlkIdx] with luma8x8BlkIdx = 0..3 contains the levels for the luma transform coefficients for the luma 8x8 block with index luma8x8BlkIdx.

For an 8x8 luma block indexed by luma8x8BlkIdx = 0..3, the following ordered steps are specified:

1. The inverse scanning process for 8x8 transform coefficients and scaling lists as specified in subclause 8.5.7 is invoked with LumaLevel8x8[luma8x8BlkIdx] as the input and the two-dimensional array c as the output.
2. The scaling and transformation process for residual 8x8 blocks as specified in subclause 8.5.13 is invoked with c as the input and r as the output.
3. When TransformBypassModeFlag is equal to 1, the macroblock prediction mode is equal to Intra_8x8, and Intra8x8PredMode[luma8x8BlkIdx] is equal to 0 or 1, the intra residual transform-bypass decoding process as specified in subclause 8.5.15 is invoked with nW set equal to 8, nH set equal to 8, horPredFlag set equal to Intra8x8PredMode[luma8x8BlkIdx], and the 8x8 array r as the inputs, and the output is a modified version of the 8x8 array r.
4. The position of the upper-left sample of an 8x8 luma block with index luma8x8BlkIdx inside the macroblock is derived by invoking the inverse 8x8 luma block scanning process in subclause 6.4.5 with luma8x8BlkIdx as the input and the output being assigned to (xO, yO).
5. The 8x8 array u with elements u_{ij} for i, j = 0..7 is derived as:

$$u_{ij} = \text{Clip1}_Y(\ \text{pred}_L[\ xO + j, yO + i\ ] + r_{ij}\ ) \tag{8-303}$$

   When TransformBypassModeFlag is equal to 1, the bitstream shall not contain data that result in a value of u_{ij} as computed by Equation 8-303 that is not equal to $\text{pred}_L[\ xO + j, yO + i\ ] + r_{ij}$.

6. The picture construction process prior to deblocking filter process in subclause 8.5.14 is invoked with u and luma8x8BlkIdx as the inputs.

8.5.4 Specification of transform decoding process for chroma samples

This process is invoked for each chroma component Cb and Cr separately when ChromaArrayType is not equal to 0.

Depending on ChromaArrayType, the following applies:

– If ChromaArrayType is equal to 3, the transform decoding process for chroma samples with ChromaArrayType equal to 3 as specified in subclause 8.5.5 is invoked.

– Otherwise (ChromaArrayType is not equal to 3), the following text specifies the transform decoding process for chroma samples.

For each chroma component, the variables ChromaDCLevel[iCbCr] and ChromaACLevel[iCbCr], with iCbCr set equal to 0 for Cb and iCbCr set equal to 1 for Cr, contain the levels for both components of the chroma transform coefficients.

Let the variable numChroma4x4Blks be set equal to (MbWidthC / 4) * (MbHeightC / 4).

For each chroma component, the transform decoding proceeds separately in the following ordered steps:

1. The numChroma4x4Blks chroma DC transform coefficients of the 4x4 chroma blocks of the component indexed by iCbCr of the macroblock are decoded as specified in the following ordered steps:

   a. Depending on the variable ChromaArrayType, the following applies:

      – If ChromaArrayType is equal to 1, the 2x2 array c is derived using the inverse raster scanning process applied to ChromaDCLevel as follows:

$$c = \begin{bmatrix} \text{ChromaDCLevel[iCbCr][0]} & \text{ChromaDCLevel[iCbCr][1]} \\ \text{ChromaDCLevel[iCbCr][2]} & \text{ChromaDCLevel[iCbCr][3]} \end{bmatrix} \quad (8\text{-}304)$$

      – Otherwise (ChromaArrayType is equal to 2), the 2x4 array c is derived using the inverse raster scanning process applied to ChromaDCLevel as follows:

$$c = \begin{bmatrix} \text{ChromaDCLevel[iCbCr][0]} & \text{ChromaDCLevel[iCbCr][2]} \\ \text{ChromaDCLevel[iCbCr][1]} & \text{ChromaDCLevel[iCbCr][5]} \\ \text{ChromaDCLevel[iCbCr][3]} & \text{ChromaDCLevel[iCbCr][6]} \\ \text{ChromaDCLevel[iCbCr][4]} & \text{ChromaDCLevel[iCbCr][7]} \end{bmatrix} \quad (8\text{-}305)$$

   b. The scaling and transformation process for chroma DC transform coefficients as specified in subclause 8.5.11 is invoked with c as the input and dcC as the output.

2. The (MbWidthC)x(MbHeightC) array rMb is derived by processing the 4x4 chroma blocks indexed by chroma4x4BlkIdx = 0..numChroma4x4Blks − 1 of the component indexed by iCbCr, and for each 4x4 chroma block, the following ordered steps are specified:

   a. The variable chromaList, which is a list of 16 entries, is derived. The first entry of chromaList is the corresponding value from the array dcC. Figure 8-7 shows the assignment of the indices of the array dcC to the chroma4x4BlkIdx. The two numbers in the small squares refer to indices i and j in dcC_{ij}, and the numbers in large squares refer to chroma4x4BlkIdx.

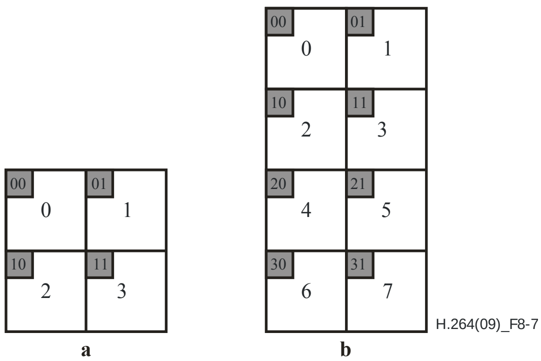


Figure 8-7 – Assignment of the indices of dcC to chroma4x4BlkIdx: (a) ChromaArrayType equal to 1, (b) ChromaArrayType equal to 2

   The elements in chromaList with index k = 1..15 are specified as:

$$\text{chromaList[k] = ChromaACLevel[chroma4x4BlkIdx][k − 1]} \quad (8\text{-}306)$$

   b. The inverse scanning process for 4x4 transform coefficients and scaling lists as specified in subclause 8.5.6 is invoked with chromaList as the input and the two-dimensional array c as the output.

   c. The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.12 is invoked with c as the input and r as the output.

d. The position of the upper-left sample of a 4x4 chroma block with index chroma4x4BlkIdx inside the current macroblock is derived by invoking the inverse 4x4 chroma block scanning process as specified in subclause 6.4.7 with chroma4x4BlkIdx as the input and the output being assigned to (xO, yO).

e. The elements rMb[x, y] of the (MbWidthC)x(MbHeightC) array rMb with x = xO..xO + 3 and y = yO..yO + 3 are derived by:

$$rMb[\ xO + j, yO + i\ ] = r_{ij} \qquad (8\text{-}307)$$

3. When TransformBypassModeFlag is equal to 1, the macroblock prediction mode is equal to Intra_4x4, Intra_8x8, or Intra_16x16, and intra_chroma_pred_mode is equal to 1 or 2, the intra residual transform-bypass decoding process as specified in subclause 8.5.15 is invoked with nW set equal to MbWidthC, nH set equal to MbHeightC, horPredFlag set equal to (2 − intra_chroma_pred_mode), and the (MbWidthC)x(MbHeightC) array rMb as the inputs, and the output is a modified version of the (MbWidthC)x(MbHeightC) array rMb.

4. The (MbWidthC)x(MbHeightC) array u with elements u_{ij} for i = 0..MbHeightC − 1 and j = 0..MbWidthC − 1 is derived as:

$$u_{ij} = \text{Clip1}_C(\ \text{pred}_C[\ j, i\ ] + rMb[\ j, i\ ]\ ) \qquad (8\text{-}308)$$

When TransformBypassModeFlag is equal to 1, the bitstream shall not contain data that result in a value of u_{ij} as computed by Equation 8-308 that is not equal to $\text{pred}_C[\ j, i\ ] + rMb[\ j, i\ ]$.

5. The picture construction process prior to deblocking filter process in subclause 8.5.14 is invoked with u as the input.

8.5.5 Specification of transform decoding process for chroma samples with ChromaArrayType equal to 3

This process is invoked for each chroma component Cb and Cr separately when ChromaArrayType is equal to 3.

Depending on the macroblock prediction mode and transform_size_8x8_flag, the following applies:

– If the macroblock prediction mode is equal to Intra_16x16, the transform decoding process for Cb or Cr residual blocks shall be identical to the process described in subclause 8.5.2 when substituting luma with Cb or Cr, substituting Intra16x16DCLevel with CbIntra16x16DCLevel or CrIntra16x16DCLevel, substituting Intra16x16ACLevel with CbIntra16x16ACLevel or CrIntra16x16ACLevel, and substituting pred_L with pred_{Cb} or pred_{Cr}, substituting luma4x4BlkIdx with cb4x4BlkIdx or cr4x4BlkIdx, substituting lumaList with CbList or CrList, substituting BitDepth_Y with BitDepth_C, substituting QP'_Y with QP'_C, and substituting Clip1_Y with Clip1_C. During the scaling of 4x4 block transform coefficient levels that is specified in subclause 8.5.12.1, which is invoked as part of the process specified in subclause 8.5.2, the input 4x4 array c is treated as relating to a luma residual block coded using an Intra_16x16 macroblock prediction mode.

– Otherwise, if transform_size_8x8_flag is equal to 1, the transform decoding process for 8x8 Cb or 8x8 Cr residual blocks shall be identical to the process described in subclause 8.5.3 when substituting luma with Cb or Cr, substituting LumaLevel8x8 with CbLevel8x8 or CrLevel8x8, substituting pred_L with pred_{Cb} or pred_{Cr}, substituting luma8x8BlkIdx with cb8x8BlkIdx or cr8x8BlkIdx, and substituting Clip1_Y with Clip1_C.

– Otherwise (the macroblock prediction mode is not equal to Intra_16x16 and transform_size_8x8_flag is equal to 0), the transform decoding process for 4x4 Cb or 4x4 Cr residual blocks shall be identical to the process described in subclause 8.5.1 when substituting luma with Cb or Cr, substituting LumaLevel4x4 with CbLevel4x4 or CrLevel4x4, substituting pred_L with pred_{Cb} or pred_{Cr}, substituting luma4x4BlkIdx with cb4x4BlkIdx or cr4x4BlkIdx, and substituting Clip1_Y with Clip1_C. During the scaling of 4x4 block transform coefficient levels that is specified in subclause 8.5.12.1, which is invoked as part of the process specified in subclause 8.5.1, the input 4x4 array c is treated as relating to a luma residual block not coded using an Intra_16x16 macroblock prediction mode.

8.5.6 Inverse scanning process for 4x4 transform coefficients and scaling lists

Input to this process is a list of 16 values.

Output of this process is a variable c containing a two-dimensional array of 4x4 values. In the case of transform coefficients, these 4x4 values represent levels assigned to locations in the transform block. In the case of applying the inverse scanning process to a scaling list, the output variable c contains a two-dimensional array representing a 4x4 scaling matrix.

When this subclause is invoked with a list of transform coefficient levels as the input, the sequence of transform coefficient levels is mapped to the transform coefficient level positions. Table 8-13 specifies the two mappings: inverse zig-zag scan and inverse field scan. The inverse zig-zag scan is used for transform coefficients in frame macroblocks and the inverse field scan is used for transform coefficients in field macroblocks.

When this subclause is invoked with a scaling list as the input, the sequence of scaling list entries is mapped to the positions in the corresponding scaling matrix. For this mapping, the inverse zig-zag scan is used.

Figure 8-8 illustrates the scans.

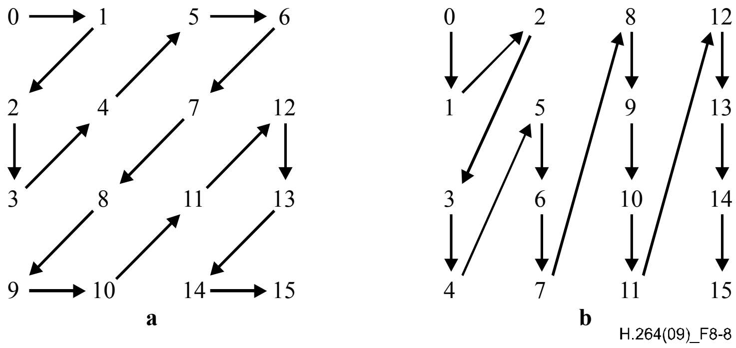


Figure 8-8 – 4x4 block scans. (a) Zig-zag scan. (b) Field scan (informative)

Table 8-13 provides the mapping from the index idx of input list of 16 elements to indices i and j of the two-dimensional array c.

Table 8-13 – Specification of mapping of idx to c_{ij} for zig-zag and field scan

idx	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
zig-zag	c_{00}	c_{01}	c_{10}	c_{20}	c_{11}	c_{02}	c_{03}	c_{12}	c_{21}	c_{30}	c_{31}	c_{22}	c_{13}	c_{23}	c_{32}	c_{33}
field	c_{00}	c_{10}	c_{01}	c_{20}	c_{30}	c_{11}	c_{21}	c_{31}	c_{02}	c_{12}	c_{22}	c_{32}	c_{03}	c_{13}	c_{23}	c_{33}

8.5.7 Inverse scanning process for 8x8 transform coefficients and scaling lists

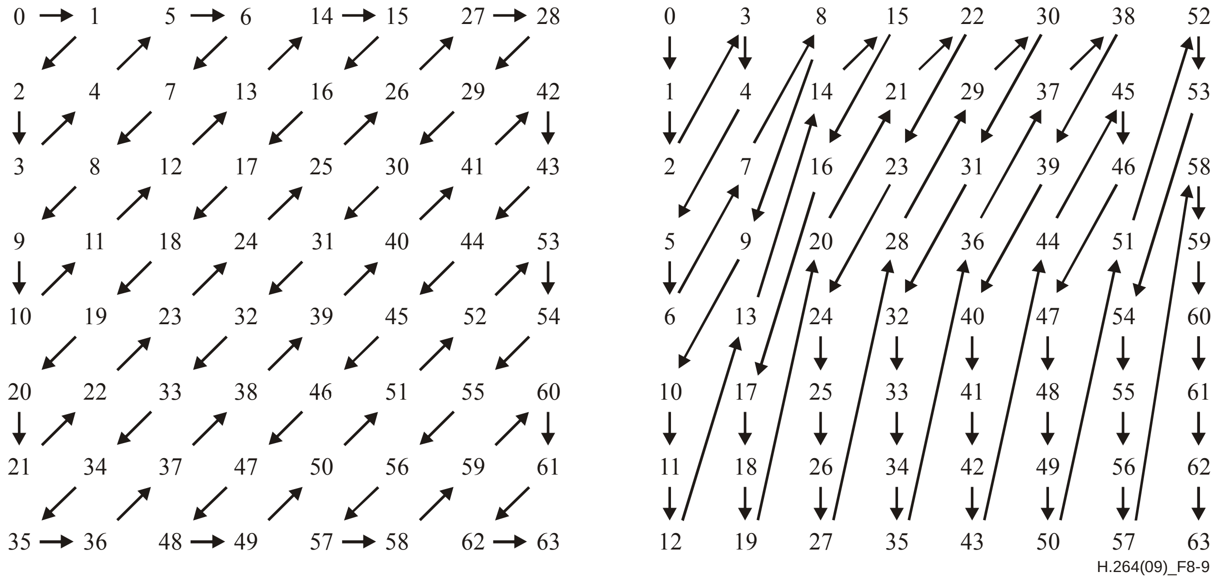
Input to this process is a list of 64 values.

Output of this process is a variable c containing a two-dimensional array of 8x8 values. In the case of transform coefficients, these 8x8 values represent levels assigned to locations in the transform block. In the case of applying the inverse scanning process to a scaling list, the output variable c contains a two-dimensional array representing an 8x8 scaling matrix.

When this subclause is invoked with a list of transform coefficient levels as the input, the sequence of transform coefficient levels is mapped to the transform coefficient level positions. Table 8-14 specifies the two mappings: inverse 8x8 zig-zag scan and inverse 8x8 field scan. The inverse 8x8 zig-zag scan is used for transform coefficient levels in frame macroblocks and the inverse 8x8 field scan is used for transform coefficient levels in field macroblocks.

When this subclause is invoked with a scaling list as the input, the sequence of scaling list entries is mapped to the positions in the corresponding scaling matrix. For this mapping, the inverse zig-zag scan is used.

Figure 8-9 illustrates the scans.



a **b**

Figure 8-9 – 8x8 block scans. (a) 8x8 zig-zag scan. (b) 8x8 field scan (informative)

Table 8-14 provides the mapping from the index idx of the input list of 64 elements to indices i and j of the two-dimensional array c.

Table 8-14 – Specification of mapping of idx to c_{ij} for 8x8 zig-zag and 8x8 field scan

idx	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
zig-zag	c_{00}	c_{01}	c_{10}	c_{20}	c_{11}	c_{02}	c_{03}	c_{12}	c_{21}	c_{30}	c_{40}	c_{31}	c_{22}	c_{13}	c_{04}	c_{05}
field	c_{00}	c_{10}	c_{20}	c_{01}	c_{11}	c_{30}	c_{40}	c_{21}	c_{02}	c_{31}	c_{50}	c_{60}	c_{70}	c_{41}	c_{12}	c_{03}

Table 8-14 (continued) – Specification of mapping of idx to cij for 8x8 zig-zag and 8x8 field scan

idx	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
zig-zag	c_{14}	c_{23}	c_{32}	c_{41}	c_{50}	c_{60}	c_{51}	c_{42}	c_{33}	c_{24}	c_{15}	c_{06}	c_{07}	c_{16}	c_{25}	c_{34}
field	c_{22}	c_{51}	c_{61}	c_{71}	c_{32}	c_{13}	c_{04}	c_{23}	c_{42}	c_{52}	c_{62}	c_{72}	c_{33}	c_{14}	c_{05}	c_{24}

Table 8-14 (continued) – Specification of mapping of idx to cij for 8x8 zig-zag and 8x8 field scan

idx	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
zig-zag	c_{43}	c_{52}	c_{61}	c_{70}	c_{71}	c_{62}	c_{53}	c_{44}	c_{35}	c_{26}	c_{17}	c_{27}	c_{36}	c_{45}	c_{54}	c_{63}
field	c_{43}	c_{53}	c_{63}	c_{73}	c_{34}	c_{15}	c_{06}	c_{25}	c_{44}	c_{54}	c_{64}	c_{74}	c_{35}	c_{16}	c_{26}	c_{45}

Table 8-14 (concluded) – Specification of mapping of idx to cij for 8x8 zig-zag and 8x8 field scan

idx	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
zig-zag	c_{72}	c_{73}	c_{64}	c_{55}	c_{46}	c_{37}	c_{47}	c_{56}	c_{65}	c_{74}	c_{75}	c_{66}	c_{57}	c_{67}	c_{76}	c_{77}
field	c_{55}	c_{65}	c_{75}	c_{36}	c_{07}	c_{17}	c_{46}	c_{56}	c_{66}	c_{76}	c_{27}	c_{37}	c_{47}	c_{57}	c_{67}	c_{77}

8.5.8 Derivation process for chroma quantisation parameters

Outputs of this process are:

– QP_C: the chroma quantisation parameter for each chroma component Cb and Cr,

– QS_C: the additional chroma quantisation parameter for each chroma component Cb and Cr required for decoding SP and SI slices (if applicable).

NOTE 1 – QP quantisation parameter values QP_Y and QS_Y are always in the range of $-QpBdOffset_Y$ to 51, inclusive. QP quantisation parameter values QP_C and QS_C are always in the range of $-QpBdOffset_C$ to 39, inclusive.

The value of QP_C for a chroma component is determined from the current value of QP_Y and the value of chroma_qp_index_offset (for Cb) or second_chroma_qp_index_offset (for Cr).

NOTE 2 – The scaling equations are specified such that the equivalent transform coefficient level scaling factor doubles for every increment of 6 in QP_Y. Thus, there is an increase in the factor used for scaling of approximately 12 % for each increase of 1 in the value of QP_Y.

The value of QP_C for each chroma component is determined as specified in Table 8-15 based on the index denoted as qP_I.

The variable qP_{Offset} for each chroma component is derived as follows:

– If the chroma component is the Cb component, qP_{Offset} is specified as:

$$qP_{Offset} = \text{chroma\_qp\_index\_offset} \tag{8-309}$$

– Otherwise (the chroma component is the Cr component), qP_{Offset} is specified as:

$$qP_{Offset} = \text{second\_chroma\_qp\_index\_offset} \tag{8-310}$$

The value of qP_I for each chroma component is derived as:

$$qP_I = \text{Clip3}(\ -QpBdOffset_C, 51, QP_Y + qP_{Offset}\ ) \tag{8-311}$$

The value of QP'_C for the chroma components is derived as:

$$QP'_C = QP_C + QpBdOffset_C \tag{8-312}$$

Table 8-15 – Specification of QP_C as a function of qP_I

qP_I	<30	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
QP_C	$= qP_I$	29	30	31	32	32	33	34	34	35	35	36	36	37	37	37	38	38	38	39	39	39	39

When the current slice is an SP or SI slice, QS_C is derived using the above process, substituting QP_Y with QS_Y and QP_C with QS_C.

8.5.9 Derivation process for scaling functions

Outputs of this process are:

– LevelScale4x4: the scaling factor for 4x4 block transform luma or chroma coefficient levels,

– LevelScale8x8: the scaling factor for 8x8 block transform luma or chroma coefficient levels.

The variable mbIsInterFlag is derived as follows:

– If the current macroblock is coded using Inter macroblock prediction modes, mbIsInterFlag is set equal to 1.

– Otherwise (the current macroblock is coded using Intra macroblock prediction modes), mbIsInterFlag is set equal to 0.

The variable iYCbCr derived as follows:

– If separate_colour_plane_flag is equal to 1, iYCbCr is set equal to colour_plane_id.

– Otherwise (separate_colour_plane_flag is equal to 0), the following applies:

  – If the scaling function LevelScale4x4 or LevelScale8x8 is derived for a luma residual block, iYCbCr is set equal to 0.

  – Otherwise, if the scaling function LevelScale4x4 or LevelScale8x8 is derived for a chroma residual block and the chroma component is equal to Cb, iYCbCr is set equal to 1.

  – Otherwise (the scaling function LevelScale4x4 or LevelScale8x8 is derived for a chroma residual block and the chroma component is equal to Cr), iYCbCr is set equal to 2.

The inverse scanning process for 4x4 transform coefficients and scaling lists as specified in subclause 8.5.6 is invoked with ScalingList4x4[iYCbCr + ((mbIsInterFlag = = 1) ? 3 : 0)] as the input and the output is assigned to the 4x4 matrix weightScale4x4.

LevelScale4x4(m, i, j) is specified by:

$$\text{LevelScale4x4}(m, i, j) = \text{weightScale4x4}(i, j) * \text{normAdjust4x4}(m, i, j) \tag{8-313}$$

where

$$\text{normAdjust4x4}(m,i,j) = \begin{cases} v_{m0} & \text{for } (i \,\%\, 2, j \,\%\, 2) \text{ equal to } (0,0), \\ v_{m1} & \text{for } (i \,\%\, 2, j \,\%\, 2) \text{ equal to } (1,1), \\ v_{m2} & \text{otherwise;} \end{cases} \tag{8-314}$$

where the first and second subscripts of v are row and column indices, respectively, of the matrix specified as:

$$v = \begin{bmatrix} 10 & 16 & 13 \\ 11 & 18 & 14 \\ 13 & 20 & 16 \\ 14 & 23 & 18 \\ 16 & 25 & 20 \\ 18 & 29 & 23 \end{bmatrix}. \tag{8-315}$$

The inverse scanning process for 8x8 transform coefficients and scaling lists as specified in subclause 8.5.7 is invoked with ScalingList8x8[2 * iYCbCr + mbIsInterFlag] as the input and the output is assigned to the 8x8 matrix weightScale8x8.

LevelScale8x8(m, i, j) is specified by:

$$\text{LevelScale8x8}(m, i, j) = \text{weightScale8x8}(i, j) * \text{normAdjust8x8}(m, i, j) \tag{8-316}$$

where

$$\text{normAdjust8x8}(m,i,j) = \begin{cases} v_{m0} & \text{for } (i \,\%\, 4, j \,\%\, 4) \text{ equal to } (0,0), \\ v_{m1} & \text{for } (i \,\%\, 2, j \,\%\, 2) \text{ equal to } (1,1), \\ v_{m2} & \text{for } (i \,\%\, 4, j \,\%\, 4) \text{ equal to } (2,2), \\ v_{m3} & \text{for } (i \,\%\, 4, j \,\%\, 2) \text{ equal to } (0,1) \text{ or } (i \,\%\, 2, j \,\%\, 4) \text{ equal to } (1,0), \\ v_{m4} & \text{for } (i \,\%\, 4, j \,\%\, 4) \text{ equal to } (0,2) \text{ or } (i \,\%\, 4, j \,\%\, 4) \text{ equal to } (2,0), \\ v_{m5} & \text{otherwise;} \end{cases} \tag{8-317}$$

where the first and second subscripts of v are row and column indices, respectively, of the matrix specified as:

$$v = \begin{bmatrix} 20 & 18 & 32 & 19 & 25 & 24 \\ 22 & 19 & 35 & 21 & 28 & 26 \\ 26 & 23 & 42 & 24 & 33 & 31 \\ 28 & 25 & 45 & 26 & 35 & 33 \\ 32 & 28 & 51 & 30 & 40 & 38 \\ 36 & 32 & 58 & 34 & 46 & 43 \end{bmatrix}. \tag{8-318}$$

8.5.10 Scaling and transformation process for DC transform coefficients for Intra_16x16 macroblock type

Inputs to this process are:

– the variables bitDepth and qP,

– transform coefficient level values for DC transform coefficients of Intra_16x16 macroblocks as a 4x4 array c with elements c_{ij}, where i and j form a two-dimensional frequency index.

Outputs of this process are 16 scaled DC values for 4x4 blocks of Intra_16x16 macroblocks as a 4x4 array dcY with elements dcY_{ij}.

Depending on the value of TransformBypassModeFlag, the following applies:

– If TransformBypassModeFlag is equal to 1, the output dcY is derived as:

$$dcY_{ij} = c_{ij} \text{ with } i, j = 0..3 \tag{8-319}$$

– Otherwise (TransformBypassModeFlag is equal to 0), the following text of this process specifies the output.

The inverse transform for the 4x4 luma DC transform coefficients is specified by:

$$f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} * \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}. \tag{8-320}$$

The bitstream shall not contain data that result in any element f_{ij} of f with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

After the inverse transform, the scaling is performed as follows:

– If qP is greater than or equal to 36, the scaled result is derived as:

$$dcY_{ij} = (f_{ij} * \text{LevelScale4x4}(qP \% 6, 0, 0)) << (qP / 6 - 6), \quad \text{with } i, j = 0...3 \tag{8-321}$$

– Otherwise (qP is less than 36), the scaled result is derived as:

$$dcY_{ij} = (f_{ij} * \text{LevelScale4x4}(qP \% 6, 0, 0) + (1 << (5 - qP / 6))) >> (6 - qP / 6), \quad \text{with } i, j = 0...3 \tag{8-322}$$

The bitstream shall not contain data that result in any element dcY_{ij} of dcY with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

NOTE 1 – When entropy_coding_mode_flag is equal to 0 and qP is less than 10 and profile_idc is equal to 66, 77, or 88, the range of values that can be represented for the elements c_{ij} of c is not sufficient to represent the full range of values of the elements dcY_{ij} of dcY that could be necessary to form a close approximation of the content of any possible source picture by use of the Intra_16x16 macroblock type.

NOTE 2 – Since the range limit imposed on the elements dcY_{ij} of dcY is imposed after the right shift in Equation 8-322, a larger range of values must be supported in the decoder prior to the right shift.

8.5.11 Scaling and transformation process for chroma DC transform coefficients

This process is only invoked when ChromaArrayType is equal to 1 or 2.

Inputs to this process are transform coefficient level values for chroma DC transform coefficients of one chroma component of the macroblock as an (MbWidthC / 4)x(MbHeightC / 4) array c with elements c_{ij}, where i and j form a two-dimensional frequency index.

Outputs of this process are the scaled DC values as an (MbWidthC / 4)x(MbHeightC / 4) array dcC with elements dcC_{ij}.

The variables bitDepth and qP are set equal to BitDepth_C and QP'_C, respectively.

Depending on the value of TransformBypassModeFlag, the following applies:

– If TransformBypassModeFlag is equal to 1, the output dcC is derived as:

$$dcC_{ij} = c_{ij} \text{ with } i = 0..(\text{MbWidthC} / 4) - 1 \text{ and } j = 0..(\text{MbHeightC} / 4) - 1. \tag{8-323}$$

– Otherwise (TransformBypassModeFlag is equal to 0), the following ordered steps are specified:

1. The transformation process for chroma DC transform coefficients as specified in subclause 8.5.11.1 is invoked with bitDepth and c as the inputs and the output is assigned to the (MbWidthC / 4)x(MbHeightC / 4) array f of chroma DC values with elements f_{ij}.

2. The scaling process for chroma DC transform coefficients as specified in subclause 8.5.11.2 is invoked with bitDepth, qP, and f as the inputs and the output is assigned to the (MbWidthC / 4)x(MbHeightC / 4) array dcC of scaled chroma DC values with elements dcC_{ij}.

8.5.11.1 Transformation process for chroma DC transform coefficients

Inputs of this process are transform coefficient level values for chroma DC transform coefficients of one chroma component of the macroblock as an (MbWidthC / 4)x(MbHeightC / 4) array c with elements c_{ij}, where i and j form a two-dimensional frequency index.

Outputs of this process are the DC values as an (MbWidthC / 4)x(MbHeightC / 4) array f with elements f_{ij}.

Depending on the variable ChromaArrayType, the inverse transform is specified as follows:

– If ChromaArrayType is equal to 1, the inverse transform for the 2x2 chroma DC transform coefficients is specified as:

$$f = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} * \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{8-324}$$

– Otherwise, (ChromaArrayType is equal to 2), the inverse transform for the 2x4 chroma DC transform coefficients is specified as:

$$f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} * \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \\ c_{20} & c_{21} \\ c_{30} & c_{31} \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{8-325}$$

8.5.11.2 Scaling process for chroma DC transform coefficients

Inputs of this process are:

– the variables bitDepth and qP,

– DC values as an (MbWidthC / 4)x(MbHeightC / 4) array f with elements f_{ij}.

Outputs of this process are scaled DC values as an (MbWidthC / 4)x(MbHeightC / 4) array dcC with elements dcC_{ij}.

The bitstream shall not contain data that result in any element f_{ij} of f with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

Scaling is performed depending on the variable ChromaArrayType as follows:

– If ChromaArrayType is equal to 1, the scaled result is derived as:

$$dcC_{ij} = ((f_{ij} * \text{LevelScale4x4}(qP \,\%\, 6, 0, 0)) << (qP / 6)) >> 5, \quad \textit{with} \quad i, j = 0, 1 \tag{8-326}$$

– Otherwise (ChromaArrayType is equal to 2), the following ordered steps are specified:

1. The variable qP_{DC} is derived as:

$$qP_{DC} = qP + 3 \tag{8-327}$$

2. Depending on the value of qP_{DC}, the following applies:

– If qP_{DC} is greater than or equal to 36, the scaled result is derived as:

$$dcC_{ij} = (f_{ij} * \text{LevelScale4x4} (qP_{DC} \,\%6, 0, 0)) << (qP_{DC} / 6 - 6), \quad \text{with } i = 0..3, j = 0, 1 \tag{8-328}$$

– Otherwise (qP_{DC} is less than 36), the scaled result is derived as:

$$dcC_{ij} = (f_{ij} * \text{LevelScale4x4}(qP_{DC} \,\%\, 6, 0, 0) + 2^{5 - qP_{DC}/6}) >> (6 - qP_{DC} / 6), \text{ with } i = 0..3, j = 0,1 \tag{8-329}$$

The bitstream shall not contain data that result in any element dcC_{ij} of dcC with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

NOTE 1 – When entropy_coding_mode_flag is equal to 0 and qP is less than 4 and profile_idc is equal to 66, 77, or 88, the range of values that can be represented for the elements c_{ij} of c in subclause 8.5.11.1 may not be sufficient to represent the full range of values of the elements dcC_{ij} of dcC that could be necessary to form a close approximation of the content of any possible source picture.

NOTE 2 – Since the range limit imposed on the elements dcC_{ij} of dcC is imposed after the right shift in Equation 8-326 or 8-329, a larger range of values must be supported in the decoder prior to the right shift.

8.5.12 Scaling and transformation process for residual 4x4 blocks

Input to this process is a 4x4 array c with elements c_{ij} which is either an array relating to a residual block of the luma component or an array relating to a residual block of a chroma component.

Outputs of this process are residual sample values as 4x4 array r with elements r_{ij}.

The variable bitDepth is derived as follows:

– If the input array c relates to a luma residual block, bitDepth is set equal to BitDepth_Y.

– Otherwise (the input array c relates to a chroma residual block), bitDepth is set equal to BitDepth_C.

The variable sMbFlag is derived as follows:

– If mb_type is equal to SI or the macroblock prediction mode is equal to Inter in an SP slice, sMbFlag is set equal to 1,

– Otherwise (mb_type not equal to SI and the macroblock prediction mode is not equal to Inter in an SP slice), sMbFlag is set equal to 0.

The variable qP is derived as follows:

– If the input array c relates to a luma residual block and sMbFlag is equal to 0,

$$qP = QP'_Y \qquad (8\text{-}330)$$

– Otherwise, if the input array c relates to a luma residual block and sMbFlag is equal to 1,

$$qP = QS_Y \qquad (8\text{-}331)$$

– Otherwise, if the input array c relates to a chroma residual block and sMbFlag is equal to 0,

$$qP = QP'_C \qquad (8\text{-}332)$$

– Otherwise (the input array c relates to a chroma residual block and sMbFlag is equal to 1),

$$qP = QS_C \qquad (8\text{-}333)$$

Depending on the value of TransformBypassModeFlag, the following applies:

– If TransformBypassModeFlag is equal to 1, the output r is derived as:

$$r_{ij} = c_{ij} \text{ with } i, j = 0..3 \qquad (8\text{-}334)$$

– Otherwise (TransformBypassModeFlag is equal to 0), the following ordered steps are specified:

1. The scaling process for residual 4x4 blocks as specified in subclause 8.5.12.1 is invoked with bitDepth, qP, and c as the inputs and the output is assigned to the 4x4 array d of scaled transform coefficients with elements d_{ij}.

2. The transformation process for residual 4x4 blocks as specified in subclause 8.5.12.2 is invoked with bitDepth and d as the inputs and the output is assigned to the 4x4 array r of residual sample values with elements r_{ij}.

8.5.12.1 Scaling process for residual 4x4 blocks

Inputs of this process are:

– the variables bitDepth and qP,

– a 4x4 array c with elements c_{ij} which is either an array relating to a residual block of luma component or an array relating to a residual block of a chroma component.

Output of this process is a 4x4 array of scaled transform coefficients d with elements d_{ij}.

The bitstream shall not contain data that result in any element c_{ij} of c with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

Scaling of 4x4 block transform coefficient levels c_{ij} proceeds as follows:

– If all of the following conditions are true:
  – i is equal to 0,
  – j is equal to 0,
  – c relates to a luma residual block coded using Intra_16x16 macroblock prediction mode or c relates to a chroma residual block.

  the variable d_{00} is derived by

$$d_{00} = c_{00} \quad (8\text{-}335)$$

– Otherwise, the following applies:
  – If qP is greater than or equal to 24, the scaled result is derived as

$$d_{ij} = (c_{ij} * \text{LevelScale4x4}(qP \% 6, i, j)) << (qP / 6 - 4), \text{ with } i, j = 0..3 \text{ except as noted above} \quad (8\text{-}336)$$

  – Otherwise (qP is less than 24), the scaled result is derived as

$$d_{ij} = (c_{ij} * \text{LevelScale4x4}(qP \% 6, i, j) + 2^{3-qP/6}) >> (4 - qP / 6), \text{ with } i, j = 0..3 \text{ except as noted above} \quad (8\text{-}337)$$

The bitstream shall not contain data that result in any element d_{ij} of d with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

8.5.12.2 Transformation process for residual 4x4 blocks

Inputs of this process are:

– the variable bitDepth,
– a 4x4 array of scaled transform coefficients d with elements d_{ij}.

Outputs of this process are residual sample values as 4x4 array r with elements r_{ij}.

The bitstream shall not contain data that result in any element d_{ij} of d with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

The transform process shall convert the block of scaled transform coefficients to a block of output samples in a manner mathematically equivalent to the following.

First, each (horizontal) row of scaled transform coefficients is transformed using a one-dimensional inverse transform as follows.

A set of intermediate values is computed as follows:

$$e_{i0} = d_{i0} + d_{i2}, \quad \text{with} \quad i = 0..3 \quad (8\text{-}338)$$

$$e_{i1} = d_{i0} - d_{i2}, \quad \text{with} \quad i = 0..3 \quad (8\text{-}339)$$

$$e_{i2} = (d_{i1} >> 1) - d_{i3}, \quad \text{with} \quad i = 0..3 \quad (8\text{-}340)$$

$$e_{i3} = d_{i1} + (d_{i3} >> 1), \quad \text{with} \quad i = 0..3 \quad (8\text{-}341)$$

The bitstream shall not contain data that result in any element e_{ij} of e with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

Then, the transformed result is computed from these intermediate values as follows:

$$f_{i0} = e_{i0} + e_{i3}, \quad \text{with} \quad i = 0..3 \quad (8\text{-}342)$$

$$f_{i1} = e_{i1} + e_{i2}, \quad \text{with} \quad i = 0..3 \quad (8\text{-}343)$$

$$f_{i2} = e_{i1} - e_{i2}, \quad \text{with} \quad i = 0..3 \quad (8\text{-}344)$$

$$f_{i3} = e_{i0} - e_{i3}, \quad \text{with} \quad i = 0..3 \tag{8-345}$$

The bitstream shall not contain data that result in any element f_{ij} of f with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

Then, each (vertical) column of the resulting matrix is transformed using the same one-dimensional inverse transform as follows.

A set of intermediate values is computed as follows:

$$g_{0j} = f_{0j} + f_{2j}, \quad \text{with} \quad j = 0..3 \tag{8-346}$$

$$g_{1j} = f_{0j} - f_{2j}, \quad \text{with} \quad j = 0..3 \tag{8-347}$$

$$g_{2j} = (f_{1j} >> 1) - f_{3j}, \quad \text{with} \quad j = 0..3 \tag{8-348}$$

$$g_{3j} = f_{1j} + (f_{3j} >> 1), \quad \text{with} \quad j = 0..3 \tag{8-349}$$

The bitstream shall not contain data that result in any element g_{ij} of g with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

Then, the transformed result is computed from these intermediate values as follows:

$$h_{0j} = g_{0j} + g_{3j}, \quad \text{with} \quad j = 0..3 \tag{8-350}$$

$$h_{1j} = g_{1j} + g_{2j}, \quad \text{with} \quad j = 0..3 \tag{8-351}$$

$$h_{2j} = g_{1j} - g_{2j}, \quad \text{with} \quad j = 0..3 \tag{8-352}$$

$$h_{3j} = g_{0j} - g_{3j}, \quad \text{with} \quad j = 0..3 \tag{8-353}$$

The bitstream shall not contain data that result in any element h_{ij} of h with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 33$, inclusive.

After performing both the one-dimensional horizontal and the one-dimensional vertical inverse transforms to produce an array of transformed samples, the final constructed residual sample values is derived as:

$$r_{ij} = (h_{ij} + 2^5) >> 6 \quad \text{with} \quad i, j = 0..3 \tag{8-354}$$

8.5.13 Scaling and transformation process for residual 8x8 blocks

Input to this process is an 8x8 array c with elements c_{ij} which is either an array relating to an 8x8 residual block of the luma component or, when ChromaArrayType is equal to 3, an array relating to an 8x8 residual block of a chroma component.

NOTE 1 – When separate_colour_plane_flag is equal to 1, all residual blocks are considered to be associated with the luma component for purposes of the decoding process of each coded picture (prior to the final assignment of the decoded picture to a particular luma or chroma picture array according to the value of colour_plane_id).

Outputs of this process are residual sample values as 8x8 array r with elements r_{ij}.

The variables bitDepth and qP are derived as follows:

– If the input array c relates to a luma residual block, bitDepth is set equal to BitDepth_Y and qP is set equal to QP'_Y.

– Otherwise (the input array c relates to a chroma residual block), bitDepth is set equal to BitDepth_C and qP is set equal to QP'_C.

NOTE 2 – When separate_colour_plane_flag is equal to 1, all residual blocks are considered to be associated with the luma component for purposes of the decoding process of each colour component of a picture.

Depending on the value of TransformBypassModeFlag, the following applies:

– If TransformBypassModeFlag is equal to 1, the output r is derived as

$$r_{ij} = c_{ij} \quad \text{with} \quad i, j = 0..7 \tag{8-355}$$

– Otherwise (TransformBypassModeFlag is equal to 0), the following ordered steps are specified:

1. The scaling process for residual 8x8 blocks as specified in subclause 8.5.13.1 is invoked with bitDepth, qP, and c as the inputs and the output is assigned to the 8x8 array d of scaled transform coefficients with elements d_{ij}.
2. The transformation process for residual 8x8 blocks as specified in subclause 8.5.13.2 is invoked with bitDepth and d as the inputs and the output is assigned to the 8x8 array r of residual sample values with elements r_{ij}.

8.5.13.1 Scaling process for residual 8x8 blocks

Inputs of this process are:

– the variables bitDepth and qP,

– an 8x8 array c with elements c_{ij} which is either an array relating to a residual block of luma component or an array relating to a residual block of a chroma component.

Output of this process is an 8x8 array of scaled transform coefficients d with elements d_{ij}.

The bitstream shall not contain data that result in any element c_{ij} of c with i, j = 0..7 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

The scaling process for 8x8 block transform coefficient levels c_{ij} proceeds as follows:

– If qP is greater than or equal to 36, the scaled result is derived as:

$$d_{ij} = (c_{ij} * \text{LevelScale8x8}(\text{ qP } \% \text{ 6, i, j})) << (\text{ qP / 6} - 6), \text{ with i, j} = 0..7 \quad (8\text{-}356)$$

– Otherwise (qP is less than 36), the scaled result is derived as:

$$d_{ij} = (c_{ij} * \text{LevelScale8x8}(\text{ qP } \% \text{ 6, i, j})) + 2^{5-\text{qP}/6}) >> (6 - \text{qP /6}), \text{ with i, j} = 0..7 \quad (8\text{-}357)$$

The bitstream shall not contain data that result in any element d_{ij} of d with i, j = 0..7 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

8.5.13.2 Transformation process for residual 8x8 blocks

Inputs of this process are:

– the variable bitDepth,

– an 8x8 array of scaled transform coefficients d with elements d_{ij}.

Outputs of this process are residual sample values as 8x8 array r with elements r_{ij}.

The bitstream shall not contain data that result in any element d_{ij} of d with i, j = 0..7 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

The transform process shall convert the block of scaled transform coefficients to a block of output samples in a manner mathematically equivalent to the following.

First, each (horizontal) row of scaled transform coefficients is transformed using a one-dimensional inverse transform as follows:

– A set of intermediate values e_{ij} is derived by:

$$e_{i0} = d_{i0} + d_{i4}, \text{ with } i = 0..7 \quad (8\text{-}358)$$

$$e_{i1} = - d_{i3} + d_{i5} - d_{i7} - (d_{i7} >> 1), \text{ with } i = 0..7 \quad (8\text{-}359)$$

$$e_{i2} = d_{i0} - d_{i4}, \text{ with } i = 0..7 \quad (8\text{-}360)$$

$$e_{i3} = d_{i1} + d_{i7} - d_{i3} - (d_{i3} >> 1), \text{ with } i = 0..7 \quad (8\text{-}361)$$

$$e_{i4} = (d_{i2} >> 1) - d_{i6}, \text{ with } i = 0..7 \quad (8\text{-}362)$$

$$e_{i5} = - d_{i1} + d_{i7} + d_{i5} + (d_{i5} >> 1), \text{ with } i = 0..7 \quad (8\text{-}363)$$

$$e_{i6} = d_{i2} + (d_{i6} >> 1), \text{ with } i = 0..7 \quad (8\text{-}364)$$

$$e_{i7} = d_{i3} + d_{i5} + d_{i1} + (d_{i1} >> 1), \text{ with } i = 0..7 \quad (8\text{-}365)$$

– A second set of intermediate results f_{ij} is computed from the intermediate values e_{ij} as:

$$f_{i0} = e_{i0} + e_{i6}, \text{ with } i = 0..7 \quad (8\text{-}366)$$

$$f_{i1} = e_{i1} + (e_{i7} >> 2), \text{ with } i = 0..7 \quad (8\text{-}367)$$

$$f_{i2} = e_{i2} + e_{i4}, \text{ with } i = 0..7 \quad (8\text{-}368)$$

$$f_{i3} = e_{i3} + (e_{i5} >> 2), \text{ with } i = 0..7 \quad (8\text{-}369)$$

$$f_{i4} = e_{i2} - e_{i4}, \text{ with } i = 0..7 \quad (8\text{-}370)$$

$$f_{i5} = (e_{i3} >> 2) - e_{i5}, \text{ with } i = 0..7 \quad (8\text{-}371)$$

$$f_{i6} = e_{i0} - e_{i6}, \text{ with } i = 0..7 \quad (8\text{-}372)$$

$$f_{i7} = e_{i7} - (e_{i1} >> 2), \text{ with } i = 0..7 \quad (8\text{-}373)$$

– Then, the transformed result g_{ij} is computed from these intermediate values f_{ij} as:

$$g_{i0} = f_{i0} + f_{i7}, \text{ with } i = 0..7 \quad (8\text{-}374)$$

$$g_{i1} = f_{i2} + f_{i5}, \text{ with } i = 0..7 \quad (8\text{-}375)$$

$$g_{i2} = f_{i4} + f_{i3}, \text{ with } i = 0..7 \quad (8\text{-}376)$$

$$g_{i3} = f_{i6} + f_{i1}, \text{ with } i = 0..7 \quad (8\text{-}377)$$

$$g_{i4} = f_{i6} - f_{i1}, \text{ with } i = 0..7 \quad (8\text{-}378)$$

$$g_{i5} = f_{i4} - f_{i3}, \text{ with } i = 0..7 \quad (8\text{-}379)$$

$$g_{i6} = f_{i2} - f_{i5}, \text{ with } i = 0..7 \quad (8\text{-}380)$$

$$g_{i7} = f_{i0} - f_{i7}, \text{ with } i = 0..7 \quad (8\text{-}381)$$

Then, each (vertical) column of the resulting matrix is transformed using the same one-dimensional inverse transform as follows:

– A set of intermediate values h_{ij} is computed from the horizontally transformed value g_{ij} as:

$$h_{0j} = g_{0j} + g_{4j}, \text{ with } j = 0..7 \quad (8\text{-}382)$$

$$h_{1j} = -g_{3j} + g_{5j} - g_{7j} - (g_{7j} >> 1), \text{ with } j = 0..7 \quad (8\text{-}383)$$

$$h_{2j} = g_{0j} - g_{4j}, \text{ with } j = 0..7 \quad (8\text{-}384)$$

$$h_{3j} = g_{1j} + g_{7j} - g_{3j} - (g_{3j} >> 1), \text{ with } j = 0..7 \quad (8\text{-}385)$$

$$h_{4j} = (g_{2j} >> 1) - g_{6j}, \text{ with } j = 0..7 \quad (8\text{-}386)$$

$$h_{5j} = -g_{1j} + g_{7j} + g_{5j} + (g_{5j} >> 1), \text{ with } j = 0..7 \quad (8\text{-}387)$$

$$h_{6j} = g_{2j} + (g_{6j} >> 1), \text{ with } j = 0..7 \quad (8\text{-}388)$$

$$h_{7j} = g_{3j} + g_{5j} + g_{1j} + (g_{1j} >> 1), \text{ with } j = 0..7 \quad (8\text{-}389)$$

– A second set of intermediate results k_{ij} is computed from the intermediate values h_{ij} as:

$$k_{0j} = h_{0j} + h_{6j}, \text{ with } j = 0..7 \quad (8\text{-}390)$$

$$k_{1j} = h_{1j} + (h_{7j} >> 2), \text{ with } j = 0..7 \quad (8\text{-}391)$$

$$k_{2j} = h_{2j} + h_{4j}, \text{ with } j = 0..7 \quad (8\text{-}392)$$

$$k_{3j} = h_{3j} + (h_{5j} >> 2), \text{ with } j = 0..7 \quad (8\text{-}393)$$

$$k_{4j} = h_{2j} - h_{4j}, \text{ with } j = 0..7 \quad (8\text{-}394)$$

$$k_{5j} = (h_{3j} >> 2) - h_{5j}, \text{ with } j = 0..7 \quad (8\text{-}395)$$

$$k_{6j} = h_{0j} - h_{6j}, \text{ with } j = 0..7 \quad (8\text{-}396)$$

$$k_{7j} = h_{7j} - (h_{1j} >> 2), \text{ with } j = 0..7 \quad (8\text{-}397)$$

– Then, the transformed result m_{ij} is computed from these intermediate values k_{ij} as:

$$m_{0j} = k_{0j} + k_{7j}, \text{ with } j = 0..7 \quad (8\text{-}398)$$

$$m_{1j} = k_{2j} + k_{5j}, \text{ with } j = 0..7 \quad (8\text{-}399)$$

$$m_{2j} = k_{4j} + k_{3j}, \text{ with } j = 0..7 \quad (8\text{-}400)$$

$$m_{3j} = k_{6j} + k_{1j}, \text{ with } j = 0..7 \quad (8\text{-}401)$$

$$m_{4j} = k_{6j} - k_{1j}, \text{ with } j = 0..7 \quad (8\text{-}402)$$

$$m_{5j} = k_{4j} - k_{3j}, \text{ with } j = 0..7 \quad (8\text{-}403)$$

$$m_{6j} = k_{2j} - k_{5j}, \text{ with } j = 0..7 \quad (8\text{-}404)$$

$$m_{7j} = k_{0j} - k_{7j}, \text{ with } j = 0..7 \quad (8\text{-}405)$$

The bitstream shall not contain data that result in any element e_{ij}, f_{ij}, g_{ij}, h_{ij}, or k_{ij} for i and j in the range of 0..7, inclusive, that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

The bitstream shall not contain data that result in any element m_{ij} for i and j in the range of 0..7, inclusive, that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 33$, inclusive.

After performing both the one-dimensional horizontal and the one-dimensional vertical inverse transforms to produce an array of transformed samples, the final constructed residual sample values are derived as

$$r_{ij} = (m_{ij} + 2^5) >> 6 \text{ with } i, j = 0..7 \quad (8\text{-}406)$$

8.5.14 Picture construction process prior to deblocking filter process

Inputs to this process are:

– a sample array u with elements u_{ij} which is a 16x16 luma block or an (MbWidthC)x(MbHeightC) chroma block or a 4x4 luma block or a 4x4 chroma block or an 8x8 luma block or, when ChromaArrayType is equal to 3, an 8x8 chroma block,

– when u is not a 16x16 luma block or an (MbWidthC)x(MbHeightC) chroma block, a block index luma4x4BlkIdx or chroma4x4BlkIdx or luma8x8BlkIdx or cb4x4BlkIdx or cr4x4BlkIdx or cb8x8BlkIdx or cr8x8BlkIdx.

The position of the upper-left luma sample of the current macroblock is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with CurrMbAddr as input and the output being assigned to (xP, yP).

When u is a luma block, for each sample u_{ij} of the luma block, the following ordered steps are specified:

1. Depending on the size of the block u, the following applies:

   – If u is a 16x16 luma block, the position (xO, yO) of the upper-left sample of the 16x16 luma block inside the macroblock is set equal to (0, 0) and the variable nE is set equal to 16.

   – Otherwise, if u is an 4x4 luma block, the position of the upper-left sample of the 4x4 luma block with index luma4x4BlkIdx inside the macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (xO, yO), and the variable nE is set equal to 4.

– Otherwise (u is an 8x8 luma block), the position of the upper-left sample of the 8x8 luma block with index luma8x8BlkIdx inside the macroblock is derived by invoking the inverse 8x8 luma block scanning process in subclause 6.4.5 with luma8x8BlkIdx as the input and the output being assigned to (xO, yO), and the variable nE is set equal to 8.

2. Depending on the variable MbaffFrameFlag and the current macroblock, the following applies:

   – If MbaffFrameFlag is equal to 1 and the current macroblock is a field macroblock,

$$S'_L[\ xP + xO + j,\ yP + 2 * (\ yO + i\ )\ ] = u_{ij} \quad \text{with } i, j = 0..nE - 1 \tag{8-407}$$

   – Otherwise (MbaffFrameFlag is equal to 0 or the current macroblock is a frame macroblock),

$$S'_L[\ xP + xO + j,\ yP + yO + i\ ] = u_{ij} \quad \text{with } i, j = 0..nE - 1 \tag{8-408}$$

When u is a chroma block, for each sample u_{ij} of the chroma block, the following ordered steps are specified:

1. The subscript C in the variable S'_C is replaced with Cb for the Cb chroma component and with Cr for the Cr chroma component.

2. Depending on the size of the block u, the following applies:

   – If u is an (MbWidthC)x(MbHeightC) Cb or Cr block, the variable nW is set equal to MbWidthC, the variable nH is set equal to MbHeightC, and the position (xO, yO) of the upper-left sample of the (nW)x(nH) Cb or Cr block inside the macroblock is set equal to (0, 0).

   – Otherwise, if u is a 4x4 Cb or Cr block, the variables nW and nH are set equal to 4 and, depending on the variable ChromaArrayType, the position of the upper-left sample of a 4x4 Cb or Cr block with index chroma4x4BlkIdx inside the macroblock is derived as follows:

     – If ChromaArrayType is equal to 1 or 2, the position of the upper-left sample of the 4x4 chroma block with index chroma4x4BlkIdx inside the macroblock is derived by invoking the inverse 4x4 chroma block scanning process in subclause 6.4.7 with chroma4x4BlkIdx as the input and the output being assigned to (xO, yO).

     – Otherwise (ChromaArrayType is equal to 3), the position of the upper-left sample of the 4x4 Cb block with index cb4x4BlkIdx or the 4x4 Cr block with index cr4x4BlkIdx inside the macroblock is derived by invoking the inverse 4x4 Cb or Cr block scanning process in subclause 6.4.4 with cb4x4BlkIdx or cr4x4BlkIdx as the input and the output being assigned to (xO, yO).

   – Otherwise (u is an 8x8 Cb or Cr block when ChromaArrayType is equal to 3), the variables nW and nH are set equal to 8 and the position of the upper-left sample of the 8x8 Cb block with index cb8x8BlkIdx or the Cr block with index cr8x8BlkIdx inside the macroblock is derived by invoking the inverse 8x8 Cb or Cr block scanning process in subclause 6.4.6 with cb8x8BlkIdx or cr8x8BlkIdx as the input and the output being assigned to (xO, yO).

3. Depending on the variable MbaffFrameFlag and the current macroblock, the following applies:

   – If MbaffFrameFlag is equal to 1 and the current macroblock is a field macroblock,

$$S'_C[\ (\ xP / subWidthC\ ) + xO + j,\ (\ (\ yP + SubHeightC - 1\ ) / SubHeightC\ ) + 2 * (\ yO + i\ )\ ] = u_{ij} \\ \text{with } i = 0..nH - 1 \text{ and } j = 0..nW - 1 \tag{8-409}$$

   – Otherwise (MbaffFrameFlag is equal to 0 or the current macroblock is a frame macroblock),

$$S'_C[\ (\ xP / subWidthC\ ) + xO + j,\ (\ yP / SubHeightC\ ) + yO + i\ ] = u_{ij} \\ \text{with } i = 0..nH - 1 \text{ and } j = 0..nW - 1 \tag{8-410}$$

8.5.15 Intra residual transform-bypass decoding process

This process is invoked when TransformBypassModeFlag is equal to 1, the macroblock prediction mode is equal to Intra_4x4, Intra_8x8, or Intra_16x16, and the applicable intra prediction mode is equal to the vertical or horizontal mode. The process for the Cb and Cr components is applied in the same way as for the luma (L or Y) component.

Inputs to this process are:

– two variables nW and nH,

– a variable horPredFlag,

– an (nW)x(nH) array r with elements r_{ij} which is either an array relating to a residual transform-bypass block of the luma component or an array relating to a residual transform-bypass block of the Cb and Cr component.

Output of this process is a modified version of the (nW)x(nH) array r with elements r_{ij} containing the result of the intra residual transform-bypass decoding process.

Let f be a temporary (nW)x(nH) array with elements f_{ij}, which are derived by:

$$f_{ij} = r_{ij} \quad \text{with } i = 0..nH - 1 \text{ and } j = 0..nW - 1 \tag{8-411}$$

Depending on horPredFlag, the following applies:

– If horPredFlag is equal to 0, the modified array r is derived by:

$$r_{ij} = \sum_{k=0}^{i} f_{kj} \quad \text{with } i = 0..nH - 1 \text{ and } j = 0..nW - 1 \tag{8-412}$$

– Otherwise (horPredFlag is equal to 1), the modified array r is derived by:

$$r_{ij} = \sum_{k=0}^{j} f_{ik} \quad \text{with } i = 0..nH - 1 \text{ and } j = 0..nW - 1 \tag{8-413}$$

8.6 Decoding process for P macroblocks in SP slices or SI macroblocks

This process is invoked when decoding P macroblock types in an SP slice type or the SI macroblock type in SI slices.

Inputs to this process are the prediction residual transform coefficient levels and the predicted samples for the current macroblock.

Outputs of this process are the decoded samples of the current macroblock prior to the deblocking filter process.

This subclause specifies the transform coefficient decoding process and picture construction process for P macroblock types in SP slices and the SI macroblock type in SI slices.

NOTE – SP slices make use of Inter predictive coding to exploit temporal redundancy in the sequence, in a similar manner to P slice coding. Unlike P slice coding, however, SP slice coding allows identical reconstruction of a slice even when different reference pictures are being used. SI slices make use of spatial prediction, in a similar manner to I slices. SI slice coding allows identical reconstruction to a corresponding SP slice. The properties of SP and SI slices aid in providing functionalities for bitstream switching, splicing, random access, fast-forward, fast reverse, and error resilience/recovery.

An SP slice consists of macroblocks coded either as I macroblock types or P macroblock types.

An SI slice consists of macroblocks coded either as I macroblock types or SI macroblock type.

The transform coefficient decoding process and picture construction process prior to deblocking filter process for I macroblock types in SI slices is invoked as specified in subclause 8.5. The SI macroblock type is decoded as described below.

When the current macroblock is coded as P_Skip, all values of LumaLevel4x4, ChromaDCLevel, ChromaACLevel are set equal to 0 for the current macroblock.

8.6.1 SP decoding process for non-switching pictures

This process is invoked, when decoding P macroblock types in SP slices in which sp_for_switch_flag is equal to 0.

Inputs to this process are Inter prediction samples for the current macroblock from subclause 8.4 and the prediction residual transform coefficient levels.

Outputs of this process are the decoded samples of the current macroblock prior to the deblocking filter process.

This subclause applies to all macroblocks in SP slices in which sp_for_switch_flag is equal to 0, except those with macroblock prediction mode equal to Intra_4x4 or Intra_16x16. It does not apply to SI slices.

8.6.1.1 Luma transform coefficient decoding process

Inputs to this process are Inter prediction luma samples for the current macroblock $pred_L$ from subclause 8.4 and the prediction residual transform coefficient levels, LumaLevel4x4, and the index of the 4x4 luma block luma4x4BlkIdx.

The position of the upper-left sample of the 4x4 luma block with index luma4x4BlkIdx inside the current macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (x, y).

Let the variable p be a 4x4 array of prediction samples with element p_{ij} being derived as:

$$p_{ij} = \text{pred}_L[\, x + j,\, y + i \,] \quad \text{with } i, j = 0..3 \tag{8-414}$$

The variable p is transformed producing transform coefficients c^p according to:

$$c^p = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} * \begin{bmatrix} p_{00} & p_{01} & p_{02} & p_{03} \\ p_{10} & p_{11} & p_{12} & p_{13} \\ p_{20} & p_{21} & p_{22} & p_{23} \\ p_{30} & p_{31} & p_{32} & p_{33} \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \tag{8-415}$$

The inverse scanning process for 4x4 transform coefficients and scaling lists as specified in subclause 8.5.6 is invoked with LumaLevel4x4[luma4x4BlkIdx] as the input and the two-dimensional array c^r with elements c_{ij}^r as the output.

The prediction residual transform coefficients c^r are scaled using quantisation parameter QP_Y, and added to the transform coefficients of the prediction block c^p with i, j = 0..3 as follows:

$$c_{ij}^s = c_{ij}^p + (((c_{ij}^r * \text{LevelScale4x4}(QP_Y \% 6, i, j) * A_{ij}) << (QP_Y / 6)) >> 10) \tag{8-416}$$

where LevelScale4x4(m, i, j) is specified in Equation 8-313, and where A_{ij} is specified as:

$$A_{ij} = \begin{cases} 16 & \text{for } (i,j) \in \{(0,0),(0,2),(2,0),(2,2)\}, \\ 25 & \text{for } (i,j) \in \{(1,1),(1,3),(3,1),(3,3)\}, \\ 20 & \text{otherwise;} \end{cases} \tag{8-417}$$

The function LevelScale2(m, i, j), used in the formulas below, is specified as

$$\text{LevelScale2}(m,i,j) = \begin{cases} w_{m0} & \text{for } (i,j) \in \{(0,0),(0,2),(2,0),(2,2)\}, \\ w_{m1} & \text{for } (i,j) \in \{(1,1),(1,3),(3,1),(3,3)\}, \\ w_{m2} & \text{otherwise;} \end{cases} \tag{8-418}$$

where the first and second subscripts of w are row and column indices, respectively, of the matrix specified as

$$w = \begin{bmatrix} 13107 & 5243 & 8066 \\ 11916 & 4660 & 7490 \\ 10082 & 4194 & 6554 \\ 9362 & 3647 & 5825 \\ 8192 & 3355 & 5243 \\ 7282 & 2893 & 4559 \end{bmatrix} \tag{8-419}$$

The resulting sum, c^s, is quantised with a quantisation parameter QS_Y and with i, j = 0..3 as follows:

$$c_{ij} = \text{Sign}(c_{ij}^s) * ((\text{Abs}(c_{ij}^s) * \text{LevelScale2}(QS_Y \% 6, i, j) + (1 << (14 + QS_Y / 6))) >> (15 + QS_Y / 6)) \tag{8-420}$$

The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.12 is invoked with c as the input and r as the output.

The 4x4 array u with elements u_{ij} is derived by:

$$u_{ij} = \text{Clip1}_Y(r_{ij}) \text{ with } i, j = 0..3 \tag{8-421}$$

The picture construction process prior to deblocking filter process in subclause 8.5.14 is invoked with luma4x4BlkIdx and u as the inputs.

8.6.1.2 Chroma transform coefficient decoding process

Inputs to this process are Inter prediction chroma samples for the current macroblock from subclause 8.4 and the prediction residual transform coefficient levels, ChromaDCLevel and ChromaACLevel.

This process is invoked twice: once for the Cb component and once for the Cr component. The component is referred to by replacing C with Cb for the Cb component and C with Cr for the Cr component. Let iCbCr select the current chroma component.

For each 4x4 block of the current chroma component indexed using chroma4x4BlkIdx with chroma4x4BlkIdx equal to 0..3, the following ordered steps are specified:

1. The position of the upper-left sample of a 4x4 chroma block with index chroma4x4BlkIdx inside the macroblock is derived by invoking the inverse 4x4 chroma block scanning process in subclause 6.4.7 with chroma4x4BlkIdx as the input and the output being assigned to (xO, yO).
2. Let p be a 4x4 array of prediction samples with elements p_{ij} being derived as

   $$p_{ij} = \text{pred}_C[\ x + j,\ y + i\ ] \quad \text{with } i, j = 0..3 \tag{8-422}$$

3. The 4x4 array p is transformed producing transform coefficients c^p(chroma4x4BlkIdx) using Equation 8-415.
4. The variable chromaList, which is a list of 16 entries, is derived. chromaList[0] is set equal to 0. chromaList[k] with index k = 1..15 are specified as follows:

   $$\text{chromaList}[\ k\ ] = \text{ChromaACLevel}[\ \text{iCbCr}\ ][\ \text{chroma4x4BlkIdx}\ ][\ k - 1\ ] \tag{8-423}$$

5. The inverse scanning process for 4x4 transform coefficients and scaling lists as specified in subclause 8.5.6 is invoked with chromaList as the input and the 4x4 array c^r as the output.
6. The prediction residual transform coefficients c^r are scaled using quantisation parameter QP_C, and added to the transform coefficients of the prediction block c^p with i, j = 0..3 except for the combination i = 0, j = 0 as follows:

   $$c_{ij}^s = c_{ij}^p(\ \text{chroma4x4BlkIdx}\ ) + (\ (\ (\ c_{ij}^r * \text{LevelScale4x4}(\ QP_C \% 6, i, j\ ) * A_{ij}\ ) << (\ QP_C / 6\ )\ ) >> 10\ ) \tag{8-424}$$

7. The resulting sum, c^s, is quantised with a quantisation parameter QS_C and with i, j = 0..3 except for the combination i = 0, j = 0 as follows. The derivation of c_{00}(chroma4x4BlkIdx) is described below in this subclause.

   $$\begin{aligned} c_{ij}(\ \text{chroma4x4BlkIdx}\ ) = (\ &\text{Sign}(\ c_{ij}^s\ ) * (\ \text{Abs}(\ c_{ij}^s\ ) * \text{LevelScale2}(\ QS_C \% 6, i, j\ ) + \\ &(\ 1 << (\ 14 + QS_C / 6\ )\ )\ )\ ) >> (\ 15 + QS_C / 6\ ) \end{aligned} \tag{8-425}$$

8. The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.12 is invoked with c(chroma4x4BlkIdx) as the input and r as the output.
9. The 4x4 array u with elements u_{ij} is derived by:

   $$u_{ij} = \text{Clip1}_C(\ r_{ij}\ ) \text{ with } i, j = 0..3 \tag{8-426}$$

10. The picture construction process prior to deblocking filter process in subclause 8.5.14 is invoked with chroma4x4BlkIdx and u as the inputs.

The derivation of the DC transform coefficient level c_{00}(chroma4x4BlkIdx) is specified as follows. The DC transform coefficients of the 4 prediction chroma 4x4 blocks of the current component of the macroblock are assembled into a 2x2 matrix with elements c_{00}^p(chroma4x4BlkIdx) and a 2x2 transform is applied to the DC transform coefficients as follows:

$$dc^p = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} * \begin{bmatrix} c_{00}^p(0) & c_{00}^p(1) \\ c_{00}^p(2) & c_{00}^p(3) \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{8-427}$$

The chroma DC prediction residual transform coefficient levels, ChromaDCLevel[iCbCr][k] with k = 0..3 are scaled using quantisation parameter QP_C, and added to the prediction DC transform coefficients as follows:

$$\begin{aligned} dc_{ij}^s = dc_{ij}^p + (\ (\ (\ &\text{ChromaDCLevel}[\ \text{iCbCr}\ ][\ j * 2 + i\ ] * \text{LevelScale4x4}(\ QP_C \% 6, 0, 0) * A_{00}\ ) << (\ QP_C / 6\ )\ ) \\ &>> 9\ ) \qquad \text{with } i, j = 0, 1 \end{aligned} \tag{8-428}$$

The 2x2 array dc^s, is quantised using the quantisation parameter QS_C as follows:

$$dc_{ij}^r = (\text{Sign}(dc_{ij}^s) * (\text{Abs}(dc_{ij}^s) * \text{LevelScale2}(QS_C \% 6, 0, 0) + (1 << (15 + QS_C / 6)))) >> (16 + QS_C / 6) \quad \text{with } i, j = 0, 1 \tag{8-429}$$

The 2x2 array f with elements f_{ij} and i, j = 0..1 is derived as:

$$f = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} * \begin{bmatrix} dc_{00}^r & dc_{01}^r \\ dc_{10}^r & dc_{11}^r \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{8-430}$$

Scaling of the elements f_{ij} of f is performed as follows:

$$c_{00}(j * 2 + i) = ((f_{ij} * \text{LevelScale4x4}(QS_C \% 6, 0, 0)) << (QS_C / 6)) >> 5 \quad \text{with } i,\ j = 0, 1 \tag{8-431}$$

8.6.2 SP and SI slice decoding process for switching pictures

This process is invoked, when decoding P macroblock types in SP slices in which sp_for_switch_flag is equal to 1 and when decoding the SI macroblock type in SI slices.

Inputs to this process are the prediction residual transform coefficient levels and the prediction sample arrays $pred_L$, $pred_{Cb}$ and $pred_{Cr}$ for the current macroblock.

8.6.2.1 Luma transform coefficient decoding process

Inputs to this process are prediction luma samples $pred_L$ and the luma prediction residual transform coefficient levels, LumaLevel4x4.

The 4x4 array p with elements p_{ij} with i, j = 0..3 is derived as in subclause 8.6.1.1, is transformed according to Equation 8-415 to produce transform coefficients c^p. These transform coefficients are then quantised with the quantisation parameter QS_Y, as follows:

$$c_{ij}^s = \text{Sign}(c_{ij}^p) * ((\text{Abs}(c_{ij}^p) * \text{LevelScale2}(QS_Y \% 6, i, j) + (1 << (14 + QS_Y / 6))) >> (15 + QS_Y / 6)) \quad \text{with } i, j = 0..3 \tag{8-432}$$

The inverse scanning process for 4x4 transform coefficients and scaling lists as specified in subclause 8.5.6 is invoked with LumaLevel4x4[luma4x4BlkIdx] as the input and the two-dimensional array c^r with elements c_{ij}^r as the output.

The 4x4 array c with elements c_{ij} with i, j = 0..3 is derived by:

$$c_{ij} = c_{ij}^r + c_{ij}^s \quad \text{with } i, j = 0..3 \tag{8-433}$$

The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.12 is invoked with c as the input and r as the output.

The 4x4 array u with elements u_{ij} is derived by:

$$u_{ij} = \text{Clip1}_Y(r_{ij}) \text{ with } i, j = 0..3 \tag{8-434}$$

The picture construction process prior to deblocking filter process in subclause 8.5.14 is invoked with luma4x4BlkIdx and u as the inputs.

8.6.2.2 Chroma transform coefficient decoding process

Inputs to this process are predicted chroma samples for the current macroblock from subclause 8.4 and the prediction residual transform coefficient levels, ChromaDCLevel and ChromaACLevel.

This process is invoked twice: once for the Cb component and once for the Cr component. The component is referred to by replacing C with Cb for the Cb component and C with Cr for the Cr component. Let iCbCr select the current chroma component.

For each 4x4 block of the current chroma component indexed using chroma4x4BlkIdx with chroma4x4BlkIdx equal to 0..3, the following ordered steps are specified:

1. The 4x4 array p with elements p_{ij} with i, j = 0..3 is derived as in subclause 8.6.1.2, is transformed according to Equation 8-415 to produce transform coefficients c^p(chroma4x4BlkIdx). These transform coefficients are then quantised with the quantisation parameter QS_C, with i, j = 0..3 except for the combination i = 0, j = 0 as follows. The processing of c_{00}^p(chroma4x4BlkIdx) is described below in this subclause.

$$c_{ij}^{s} = (\mathrm{Sign}(c_{ij}^{p}(\mathrm{chroma4x4BlkIdx})) * (\mathrm{Abs}(c_{ij}^{p}(\mathrm{chroma4x4BlkIdx})) * \mathrm{LevelScale2}(QS_C \% 6, i, j) + (1 << (14 + QS_C / 6)))) >> (15 + QS_C / 6) \quad (8\text{-}435)$$

2. The variable chromaList, which is a list of 16 entries, is derived. chromaList[0] is set equal to 0. chromaList[k] with index k = 1..15 are specified as follows:

   $$\mathrm{chromaList}[k] = \mathrm{ChromaACLevel}[\mathrm{iCbCr}][\mathrm{chroma4x4BlkIdx}][k - 1] \quad (8\text{-}436)$$

3. The inverse scanning process for 4x4 transform coefficients and scaling lists as specified in subclause 8.5.6 is invoked with chromaList as the input and the two-dimensional array c^r(chroma4x4BlkIdx) with elements c_{ij}^r(chroma4x4BlkIdx) as the output.
4. The 4x4 array c(chroma4x4BlkIdx) with elements c_{ij}(chroma4x4BlkIdx) with i, j = 0..3 except for the combination i = 0, j = 0 is derived as follows. The derivation of c_{00}(chroma4x4BlkIdx) is described below.

   $$c_{ij}(\mathrm{chroma4x4BlkIdx}) = c_{ij}^{r}(\mathrm{chroma4x4BlkIdx}) + c_{ij}^{s} \quad (8\text{-}437)$$

5. The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.12 is invoked with c(chroma4x4BlkIdx) as the input and r as the output.
6. The 4x4 array u with elements u_{ij} is derived by:

   $$u_{ij} = \mathrm{Clip1}_C(r_{ij}) \text{ with } i, j = 0..3 \quad (8\text{-}438)$$

7. The picture construction process prior to deblocking filter process in subclause 8.5.14 is invoked with chroma4x4BlkIdx and u as the inputs.

The derivation of the DC transform coefficient level c_{00}(chroma4x4BlkIdx) is specified as follows. The DC transform coefficients of the 4 prediction 4x4 chroma blocks of the current component of the macroblock, c_{00}^{p}(chroma4x4BlkIdx), are assembled into a 2x2 matrix, and a 2x2 transform is applied to the DC transform coefficients of these blocks according to Equation 8-427 resulting in DC transform coefficients dc_{ij}^{p}.

These DC transform coefficients are then quantised with the quantisation parameter QS_C, as given by:

$$dc_{ij}^{s} = (\mathrm{Sign}(dc_{ij}^{p}) * (\mathrm{Abs}(dc_{ij}^{p}) * \mathrm{LevelScale2}(QS_C \% 6, 0, 0) + (1 << (15 + QS_C / 6)))) >> (16 + QS_C / 6) \quad \text{with } i, j = 0, 1 \quad (8\text{-}439)$$

The parsed chroma DC prediction residual transform coefficients, ChromaDCLevel[iCbCr][k] with k = 0..3 are added to these quantised DC transform coefficients of the prediction block, as given by:

$$dc_{ij}^{r} = dc_{ij}^{s} + \mathrm{ChromaDCLevel}[\mathrm{iCbCr}][j * 2 + i] \text{ with } i, j = 0, 1 \quad (8\text{-}440)$$

The 2x2 array f with elements f_{ij} and i, j = 0..1 is derived using Equation 8-430.

The 2x2 array f with elements f_{ij} and i, j = 0..1 is copied as follows:

$$c_{00}(j * 2 + i) = f_{ij} \text{ with } i, j = 0, 1 \quad (8\text{-}441)$$

8.7 Deblocking filter process

A conditional filtering process is specified in this subclause that is an integral part of the decoding process which shall be applied by decoders conforming to the Baseline, Constrained Baseline, Main, Extended, High, Progressive High, High 10, High 4:2:2, and High 4:4:4 Predictive profiles. For decoders conforming to the High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, and CAVLC 4:4:4 Intra profiles, the filtering process specified in this subclause, or one similar to it, should be applied but is not required.

The conditional filtering process is applied to all NxN (where N = 4 or N = 8 for luma, N = 4 for chroma when ChromaArrayType is equal to 1 or 2, and N = 4 or N = 8 for chroma when ChromaArrayType is equal to 3) block edges of a picture, except edges at the boundary of the picture and any edges for which the deblocking filter process is disabled by disable_deblocking_filter_idc, as specified below. This filtering process is performed on a macroblock basis after the completion of the picture construction process prior to deblocking filter process (as specified in subclauses 8.5 and 8.6) for the entire decoded picture, with all macroblocks in a picture processed in order of increasing macroblock addresses.

NOTE 1 – Prior to the operation of the deblocking filter process for each macroblock, the deblocked samples of the macroblock or macroblock pair above (if any) and the macroblock or macroblock pair to the left (if any) of the current macroblock are always available because the deblocking filter process is performed after the completion of the picture construction process prior to deblocking filter process for the entire decoded picture. However, for purposes of determining which edges are to be filtered when disable_deblocking_filter_idc is equal to 2, macroblocks in different slices are considered not available during specified steps of the operation of the deblocking filter process.

The deblocking filter process is invoked for the luma and chroma components separately. For each macroblock and each component, vertical edges are filtered first, starting with the edge on the left-hand side of the macroblock proceeding through the edges towards the right-hand side of the macroblock in their geometrical order, and then horizontal edges are filtered, starting with the edge on the top of the macroblock proceeding through the edges towards the bottom of the macroblock in their geometrical order. Figure 8-10 shows edges of a macroblock which can be interpreted as luma or chroma edges.

When interpreting the edges in Figure 8-10 as luma edges, depending on the transform_size_8x8_flag, the following applies:

– If transform_size_8x8_flag is equal to 0, both types, the solid bold and dashed bold luma edges are filtered.

– Otherwise (transform_size_8x8_flag is equal to 1), only the solid bold luma edges are filtered.

When interpreting the edges in Figure 8-10 as chroma edges, depending on ChromaArrayType, the following applies:

– If ChromaArrayType is equal to 1 (4:2:0 format), only the solid bold chroma edges are filtered.

– Otherwise, if ChromaArrayType is equal to 2 (4:2:2 format), the solid bold vertical chroma edges are filtered and both types, the solid bold and dashed bold horizontal chroma edges are filtered.

– Otherwise, if ChromaArrayType is equal to 3 (4:4:4 format), the following applies:

  – If transform_size_8x8_flag is equal to 0, both types, the solid bold and dashed bold chroma edges are filtered.

  – Otherwise (transform_size_8x8_flag is equal to 1), only the solid bold chroma edges are filtered.

– Otherwise (ChromaArrayType is equal to 0), no chroma edges are filtered.

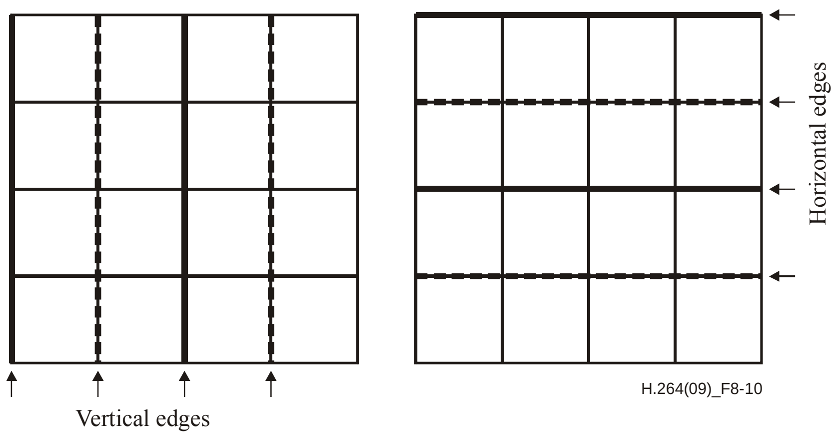


Figure 8-10 – Boundaries in a macroblock to be filtered

For the current macroblock address CurrMbAddr proceeding over values 0..PicSizeInMbs − 1, the following ordered steps are specified:

1. The derivation process for neighbouring macroblocks specified in subclause 6.4.11.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

2. The variables fieldMbInFrameFlag, filterInternalEdgesFlag, filterLeftMbEdgeFlag and filterTopMbEdgeFlag are derived as specified by the following ordered steps:

   a. The variable fieldMbInFrameFlag is derived as follows:

      – If MbaffFrameFlag is equal to 1 and mb_field_decoding_flag is equal to 1, fieldMbInFrameFlag is set equal to 1.

      – Otherwise (MbaffFrameFlag is equal to 0 or mb_field_decoding_flag is equal to 0), fieldMbInFrameFlag is set equal to 0.

b. The variable filterInternalEdgesFlag is derived as follows:

– If disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 1, the variable filterInternalEdgesFlag is set equal to 0.

– Otherwise (disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is not equal to 1), the variable filterInternalEdgesFlag is set equal to 1.

c. The variable filterLeftMbEdgeFlag is derived as follows:

– If any of the following conditions are true, the variable filterLeftMbEdgeFlag is set equal to 0:

– MbaffFrameFlag is equal to 0 and CurrMbAddr % PicWidthInMbs is equal to 0,

– MbaffFrameFlag is equal to 1 and (CurrMbAddr >> 1) % PicWidthInMbs is equal to 0,

– disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 1,

– disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 2 and the macroblock mbAddrA is not available.

– Otherwise, the variable filterLeftMbEdgeFlag is set equal to 1.

d. The variable filterTopMbEdgeFlag is derived as follows:

– If any of the following conditions are true, the variable filterTopMbEdgeFlag is set equal to 0:

– MbaffFrameFlag is equal to 0 and CurrMbAddr is less than PicWidthInMbs,

– MbaffFrameFlag is equal to 1, (CurrMbAddr >> 1) is less than PicWidthInMbs, and the macroblock CurrMbAddr is a field macroblock,

– MbaffFrameFlag is equal to 1, (CurrMbAddr >> 1) is less than PicWidthInMbs, the macroblock CurrMbAddr is a frame macroblock, and CurrMbAddr % 2 is equal to 0,

– disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 1,

– disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 2 and the macroblock mbAddrB is not available.

– Otherwise, the variable filterTopMbEdgeFlag is set equal to 1.

3. Given the variables fieldMbInFrameFlag, filterInternalEdgesFlag, filterLeftMbEdgeFlag and filterTopMbEdgeFlag the deblocking filtering is controlled as follows:

a. When filterLeftMbEdgeFlag is equal to 1, the left vertical luma edge is filtered by invoking the process specified in subclause 8.7.1 with chromaEdgeFlag = 0, verticalEdgeFlag = 1, fieldModeInFrameFilteringFlag = fieldMbInFrameFlag, and $(xE_k, yE_k) = (0, k)$ with $k = 0..15$ as the inputs and S'_L as the output.

b. When filterInternalEdgesFlag is equal to 1, the filtering of the internal vertical luma edges is specified by the following ordered steps:

i. When transform_size_8x8_flag is equal to 0, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 1, fieldModeInFrameFilteringFlag = fieldMbInFrameFlag, and $(xE_k, yE_k) = (4, k)$ with $k = 0..15$ as the inputs and S'_L as the output.

ii. The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 1, fieldModeInFrameFilteringFlag = fieldMbInFrameFlag, and $(xE_k, yE_k) = (8, k)$ with $k = 0..15$ as the inputs and S'_L as the output.

iii. When transform_size_8x8_flag is equal to 0, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 1, fieldModeInFrameFilteringFlag = fieldMbInFrameFlag, and $(xE_k, yE_k) = (12, k)$ with $k = 0..15$ as the inputs and S'_L as the output.

c. When filterTopMbEdgeFlag is equal to 1, the filtering of the top horizontal luma edge is specified as follows:

– If MbaffFrameFlag is equal to 1, (CurrMbAddr % 2) is equal to 0, CurrMbAddr is greater than or equal to 2 * PicWidthInMbs, the macroblock CurrMbAddr is a frame macroblock, and the macroblock (CurrMbAddr − 2 * PicWidthInMbs + 1) is a field macroblock, the following ordered steps are specified:

i. The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeInFrameFilteringFlag = 1, and $(xE_k, yE_k) = (k, 0)$ with k = 0..15 as the inputs and S'_L as the output.

ii. The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeInFrameFilteringFlag = 1, and $(xE_k, yE_k) = (k, 1)$ with k = 0..15 as the inputs and S'_L as the output.

– Otherwise, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeInFrameFilteringFlag = fieldMbInFrameFlag, and $(xE_k, yE_k) = (k, 0)$ with k = 0..15 as the inputs and S'_L as the output.

d. When filterInternalEdgesFlag is equal to 1, the filtering of the internal horizontal luma edges is specified by the following ordered steps:

i. When transform_size_8x8_flag is equal to 0, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeInFrameFilteringFlag = fieldMbInFrameFlag, and $(xE_k, yE_k) = (k, 4)$ with k = 0..15 as the inputs and S'_L as the output.

ii. The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeInFrameFilteringFlag = fieldMbInFrameFlag, and $(xE_k, yE_k) = (k, 8)$ with k = 0..15 as the inputs and S'_L as the output.

iii. When transform_size_8x8_flag is equal to 0, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeInFrameFilteringFlag = fieldMbInFrameFlag, and $(xE_k, yE_k) = (k, 12)$ with k = 0..15 as the inputs and S'_L as the output.

e. When ChromaArrayType is not equal to 0, for the filtering of both chroma components, with iCbCr = 0 for Cb and iCbCr = 1 for Cr, the following ordered steps are specified:

i. When filterLeftMbEdgeFlag is equal to 1, the left vertical chroma edge is filtered by invoking the process specified in subclause 8.7.1 with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 1, fieldModeInFrameFilteringFlag = fieldMbInFrameFlag, and $(xE_k, yE_k) = (0, k)$ with k = 0..MbHeightC − 1 as the inputs and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as the output.

ii. When filterInternalEdgesFlag is equal to 1, the filtering of the internal vertical chroma edge is specified by the following ordered steps:

(1) When ChromaArrayType is not equal to 3 or transform_size_8x8_flag is equal to 0, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 1, fieldModeInFrameFilteringFlag = fieldMbInFrameFlag, and $(xE_k, yE_k) = (4, k)$ with k = 0..MbHeightC − 1 as the inputs and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as the output.

(2) When ChromaArrayType is equal to 3, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 1, fieldModeInFrameFilteringFlag = fieldMbInFrameFlag, and $(xE_k, yE_k) = (8, k)$ with k = 0..MbHeightC − 1 as the inputs and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as the output.

(3) When ChromaArrayType is equal to 3 and transform_size_8x8_flag is equal to 0, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 1, fieldModeInFrameFilteringFlag = fieldMbInFrameFlag, and $(xE_k, yE_k) = (12, k)$ with k = 0..MbHeightC − 1 as the inputs and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as the output.

iii. When filterTopMbEdgeFlag is equal to 1, the filtering of the top horizontal chroma edge is specified as follows:

– If MbaffFrameFlag is equal to 1, (CurrMbAddr % 2) is equal to 0, CurrMbAddr is greater than or equal to 2 * PicWidthInMbs, the macroblock CurrMbAddr is a frame macroblock, and the macroblock (CurrMbAddr − 2 * PicWidthInMbs + 1) is a field macroblock, the following ordered steps are specified:

(1) The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeInFrameFilteringFlag = 1, and $(xE_k, yE_k) = (k, 0)$ with k = 0..MbWidthC − 1 as the inputs and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as the output.

(2) The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeInFrameFilteringFlag = 1, and $(xE_k, yE_k) = (k, 1)$ with k = 0..MbWidthC − 1 as the inputs and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as the output.

– Otherwise, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeInFrameFilteringFlag = fieldMbInFrameFlag, and $(xE_k, yE_k) = (k, 0)$ with k = 0..MbWidthC − 1 as the inputs and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as the output.

iv. When filterInternalEdgesFlag is equal to 1, the filtering of the internal horizontal chroma edge is specified by the following ordered steps:

(1) When ChromaArrayType is not equal to 3 or transform_size_8x8_flag is equal to 0, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeInFrameFilteringFlag = fieldMbInFrameFlag, and $(xE_k, yE_k) = (k, 4)$ with k = 0..MbWidthC − 1 as the inputs and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as the output.

(2) When ChromaArrayType is not equal to 1, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeInFrameFilteringFlag = fieldMbInFrameFlag, and $(xE_k, yE_k) = (k, 8)$ with k = 0..MbWidthC − 1 as the inputs and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as the output.

(3) When ChromaArrayType is equal to 2, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeInFrameFilteringFlag = fieldMbInFrameFlag, and $(xE_k, yE_k) = (k, 12)$ with k = 0..MbWidthC − 1 as the inputs and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as the output.

(4) When ChromaArrayType is equal to 3 and transform_size_8x8_flag is equal to 0, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeInFrameFilteringFlag = fieldMbInFrameFlag, and $(xE_k, yE_k) = (k, 12)$ with k = 0..MbWidthC − 1 as the inputs and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as the output.

NOTE 2 – When field mode filtering (fieldModeInFrameFilteringFlag is equal to 1) is applied across the top horizontal edges of a frame macroblock, this vertical filtering across the top or bottom macroblock boundary may involve some samples that extend across an internal block edge that is also filtered internally in frame mode.

NOTE 3 – For example, in 4:2:0 chroma format when transform_size_8x8_flag is equal to 0, the following applies. 3 horizontal luma edges, 1 horizontal chroma edge for Cb, and 1 horizontal chroma edge for Cr are filtered that are internal to a macroblock. When field mode filtering (fieldModeInFrameFilteringFlag is equal to 1) is applied to the top edges of a frame macroblock, 2 horizontal luma, 2 horizontal chroma edges for Cb, and 2 horizontal chroma edges for Cr between the frame macroblock and the above macroblock pair are filtered using field mode filtering, for a total of up to 5 horizontal luma edges, 3 horizontal chroma edges for Cb, and 3 horizontal chroma edges for Cr filtered that are considered to be controlled by the frame macroblock. In all other cases, at most 4 horizontal luma, 2 horizontal chroma edges for Cb, and 2 horizontal chroma edges for Cr are filtered that are considered to be controlled by a particular macroblock.

Depending on separate_colour_plane_flag the following applies:

– If separate_colour_plane_flag is equal to 0, the arrays S'_L, S'_{Cb}, S'_{Cr} are assigned to the arrays S_L, S_{Cb}, S_{Cr} (which represent the decoded picture), respectively.

– Otherwise (separate_colour_plane_flag is equal to 1), the following applies:

  – If colour_plane_id is equal to 0, the arrays S'_L is assigned to the array S_L (which represent the luma component of the decoded picture).

  – Otherwise, if colour_plane_id is equal to 1, the arrays S'_L is assigned to the array S_{Cb} (which represents the Cb component of the decoded picture).

  – Otherwise (colour_plane_id is equal to 2), the arrays S'_L is assigned to the array S_{Cr} (which represents the Cr component of the decoded picture).

8.7.1 Filtering process for block edges

Inputs to this process are chromaEdgeFlag, the chroma component index iCbCr (when chromaEdgeFlag is equal to 1), verticalEdgeFlag, fieldModeInFrameFilteringFlag, and a set of nE sample locations (xE_k, yE_k), with $k = 0..nE - 1$, expressed relative to the upper left corner of the macroblock CurrMbAddr. The set of sample locations (xE_k, yE_k) represent the sample locations immediately to the right of a vertical edge (when verticalEdgeFlag is equal to 1) or immediately below a horizontal edge (when verticalEdgeFlag is equal to 0).

The variable nE is derived as follows:

– If chromaEdgeFlag is equal to 0, nE is set equal to 16.

– Otherwise (chromaEdgeFlag is equal to 1), nE is set equal to (verticalEdgeFlag = = 1) ? MbHeightC : MbWidthC.

Let s′ be a variable specifying a luma or chroma sample array. s′ is derived as follows:

– If chromaEdgeFlag is equal to 0, s′ represents the luma sample array S'_L of the current picture.

– Otherwise, if chromaEdgeFlag is equal to 1 and iCbCr is equal to 0, s′ represents the chroma sample array S'_{Cb} of the chroma component Cb of the current picture.

– Otherwise (chromaEdgeFlag is equal to 1 and iCbCr is equal to 1), s′ represents the chroma sample array S'_{Cr} of the chroma component Cr of the current picture.

The variable dy is set equal to (1 + fieldModeInFrameFilteringFlag).

The position of the upper-left luma sample of the macroblock CurrMbAddr is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddr = CurrMbAddr as input and the output being assigned to (xI, yI).

The variables xP and yP are derived as follows:

– If chromaEdgeFlag is equal to 0, xP is set equal to xI and yP is set equal to yI.

– Otherwise (chromaEdgeFlag is equal to 1), xP is set equal to xI / SubWidthC and yP is set equal to (yI + SubHeightC − 1) / SubHeightC.

p_3	p_2	p_1	p_0	q_0	q_1	q_2	q_3

Figure 8-11 – Convention for describing samples across a 4x4 block horizontal or vertical boundary

For each sample location (xE_k, yE_k), $k = 0..(nE - 1)$, the following ordered steps are specified:

1. The filtering process is applied to a set of eight samples across a 4x4 block horizontal or vertical edge denoted as p_i and q_i with $i = 0..3$ as shown in Figure 8-11 with the edge lying between p_0 and q_0. p_i and q_i with $i = 0..3$ are specified as follows:

   – If verticalEdgeFlag is equal to 1,

$$q_i = s'[\ xP + xE_k + i,\ yP + dy * yE_k\ ] \quad (8\text{-}442)$$

$$p_i = s'[\ xP + xE_k - i - 1,\ yP + dy * yE_k\ ] \quad (8\text{-}443)$$

– Otherwise (verticalEdgeFlag is equal to 0),

$$q_i = s'[\ xP + xE_k,\ yP + dy * (\ yE_k + i\ ) - (yE_k \% 2\ )\ ] \quad (8\text{-}444)$$

$$p_i = s'[\ xP + xE_k,\ yP + dy * (\ yE_k - i - 1\ ) - (yE_k \% 2\ )\ ] \quad (8\text{-}445)$$

2. The process specified in subclause 8.7.2 is invoked with the sample values p_i and q_i (i = 0..3), chromaEdgeFlag, and verticalEdgeFlag as the inputs, and the output is assigned to the filtered result sample values p'_i and q'_i with i = 0..2.

3. The input sample values p_i and q_i with i = 0..2 are replaced by the corresponding filtered result sample values p'_i and q'_i with i = 0..2 inside the sample array s′ as follows:

   – If verticalEdgeFlag is equal to 1,

$$s'[\ xP + xE_k + i,\ yP + dy * yE_k\ ] = q'_i \quad (8\text{-}446)$$

$$s'[\ xP + xE_k - i - 1,\ yP + dy * yE_k\ ] = p'_i \quad (8\text{-}447)$$

   – Otherwise (verticalEdgeFlag is equal to 0),

$$s'[\ xP + xE_k,\ yP + dy * (\ yE_k + i\ ) - (\ yE_k \% 2\ )\ ] = q'_i \quad (8\text{-}448)$$

$$s'[\ xP + xE_k,\ yP + dy * (\ yE_k - i - 1\ ) - (\ yE_k \% 2\ )\ ] = p'_i \quad (8\text{-}449)$$

8.7.2 Filtering process for a set of samples across a horizontal or vertical block edge

Inputs to this process are the input sample values p_i and q_i with i in the range of 0..3 of a single set of samples across an edge that is to be filtered, chromaEdgeFlag, and verticalEdgeFlag.

Outputs of this process are the filtered result sample values p'_i and q'_i with i in the range of 0..2.

The content dependent boundary filtering strength variable bS is derived as follows:

– If chromaEdgeFlag is equal to 0, the derivation process for the content dependent boundary filtering strength specified in subclause 8.7.2.1 is invoked with p_0, q_0, and verticalEdgeFlag as input, and the output is assigned to bS.

– Otherwise (chromaEdgeFlag is equal to 1), the bS used for filtering a set of samples of a horizontal or vertical chroma edge is set equal to the value of bS for filtering the set of samples of a horizontal or vertical luma edge, respectively, that contains the luma sample at location (SubWidthC * x, SubHeightC * y) inside the luma array of the same field, where (x, y) is the location of the chroma sample q_0 inside the chroma array for that field.

Let filterOffsetA and filterOffsetB be the values of FilterOffsetA and FilterOffsetB as specified in subclause 7.4.3 for the slice that contains the macroblock containing sample q_0.

Let qP_p and qP_q be variables specifying quantisation parameter values for the macroblocks containing the samples p_0 and q_0, respectively. The variables qP_z (with z being replaced by p or q) are derived as follows:

– If chromaEdgeFlag is equal to 0, the following applies:

  – If the macroblock containing the sample z_0 is an I_PCM macroblock, qP_z is set to 0.

  – Otherwise (the macroblock containing the sample z_0 is not an I_PCM macroblock), qP_z is set to the value of QP_Y of the macroblock containing the sample z_0.

– Otherwise (chromaEdgeFlag is equal to 1), the following applies:

  – If the macroblock containing the sample z_0 is an I_PCM macroblock, qP_z is set equal to the value of QP_C that corresponds to a value of 0 for QP_Y as specified in subclause 8.5.8.

  – Otherwise (the macroblock containing the sample z_0 is not an I_PCM macroblock), qP_z is set equal to the value of QP_C that corresponds to the value QP_Y of the macroblock containing the sample z_0 as specified in subclause 8.5.8.

The process specified in subclause 8.7.2.2 is invoked with p_0, q_0, p_1, q_1, chromaEdgeFlag, bS, filterOffsetA, filterOffsetB, qP_p, and qP_q as inputs, and the outputs are assigned to filterSamplesFlag, indexA, α, and β.

The variable chromaStyleFilteringFlag is set by

$$\text{chromaStyleFilteringFlag} = \text{chromaEdgeFlag} \ \&\& \ (\text{ChromaArrayType} \ != \ 3) \tag{8-450}$$

Depending on the variable filterSamplesFlag, the following applies:

– If filterSamplesFlag is equal to 1, the following applies:

  – If bS is less than 4, the process specified in subclause 8.7.2.3 is invoked with p_i and q_i (i = 0..2), chromaEdgeFlag, chromaStyleFilteringFlag, bS, β, and indexA given as input, and the output is assigned to p'_i and q'_i (i = 0..2).

  – Otherwise (bS is equal to 4), the process specified in subclause 8.7.2.4 is invoked with p_i and q_i (i = 0..3), chromaEdgeFlag, chromaStyleFilteringFlag, α, and β given as input, and the output is assigned to p'_i and q'_i (i = 0..2).

– Otherwise (filterSamplesFlag is equal to 0), the filtered result samples p'_i and q'_i (i = 0..2) are replaced by the corresponding input samples p_i and q_i:

$$\text{for } i = 0..2, \qquad p'_i = p_i \tag{8-451}$$

$$\text{for } i = 0..2, \qquad q'_i = q_i \tag{8-452}$$

8.7.2.1 Derivation process for the luma content dependent boundary filtering strength

Inputs to this process are the input sample values p_0 and q_0 of a single set of samples across an edge that is to be filtered and verticalEdgeFlag.

Output of this process is the variable bS.

Let the variable mixedModeEdgeFlag be derived as follows:

– If MbaffFrameFlag is equal to 1 and the samples p_0 and q_0 are in different macroblock pairs, one of which is a field macroblock pair and the other is a frame macroblock pair, mixedModeEdgeFlag is set equal to 1.

– Otherwise, mixedModeEdgeFlag is set equal to 0.

The variable bS is derived as follows:

– If the block edge is also a macroblock edge and any of the following conditions are true, a value of bS equal to 4 is the output:

  – the samples p_0 and q_0 are both in frame macroblocks and either or both of the samples p_0 or q_0 is in a macroblock coded using an Intra macroblock prediction mode,

  – the samples p_0 and q_0 are both in frame macroblocks and either or both of the samples p_0 or q_0 is in a macroblock that is in a slice with slice_type equal to SP or SI,

  – MbaffFrameFlag is equal to 1 or field_pic_flag is equal to 1, and verticalEdgeFlag is equal to 1, and either or both of the samples p_0 or q_0 is in a macroblock coded using an Intra macroblock prediction mode,

  – MbaffFrameFlag is equal to 1 or field_pic_flag is equal to 1, and verticalEdgeFlag is equal to 1, and either or both of the samples p_0 or q_0 is in a macroblock that is in a slice with slice_type equal to SP or SI.

– Otherwise, if any of the following conditions are true, a value of bS equal to 3 is the output:

  – mixedModeEdgeFlag is equal to 0 and either or both of the samples p_0 or q_0 is in a macroblock coded using an Intra macroblock prediction mode,

  – mixedModeEdgeFlag is equal to 0 and either or both of the samples p_0 or q_0 is in a macroblock that is in a slice with slice_type equal to SP or SI,

  – mixedModeEdgeFlag is equal to 1, verticalEdgeFlag is equal to 0, and either or both of the samples p_0 or q_0 is in a macroblock coded using an Intra macroblock prediction mode,

  – mixedModeEdgeFlag is equal to 1, verticalEdgeFlag is equal to 0, and either or both of the samples p_0 or q_0 is in a macroblock that is in a slice with slice_type equal to SP or SI.

– Otherwise, if any of the following conditions are true, a value of bS equal to 2 is the output:

  – transform_size_8x8_flag is equal to 1 for the macroblock containing the sample p_0 and the 8x8 luma transform block associated with the 8x8 luma block containing the sample p_0 contains non-zero transform coefficient levels,

  – transform_size_8x8_flag is equal to 0 for the macroblock containing the sample p_0 and the 4x4 luma transform block associated with the 4x4 luma block containing the sample p_0 contains non-zero transform coefficient levels,

  – transform_size_8x8_flag is equal to 1 for the macroblock containing the sample q_0 and the 8x8 luma transform block associated with the 8x8 luma block containing the sample q_0 contains non-zero transform coefficient levels,

  – transform_size_8x8_flag is equal to 0 for the macroblock containing the sample q_0 and the 4x4 luma transform block associated with the 4x4 luma block containing the sample q_0 contains non-zero transform coefficient levels.

– Otherwise, if any of the following conditions are true, a value of bS equal to 1 is the output:

  – mixedModeEdgeFlag is equal to 1,

  – mixedModeEdgeFlag is equal to 0 and for the prediction of the macroblock/sub-macroblock partition containing the sample p_0 different reference pictures or a different number of motion vectors are used than for the prediction of the macroblock/sub-macroblock partition containing the sample q_0,

    NOTE 1 – The determination of whether the reference pictures used for the two macroblock/sub-macroblock partitions are the same or different is based only on which pictures are referenced, without regard to whether a prediction is formed using an index into reference picture list 0 or an index into reference picture list 1, and also without regard to whether the index position within a reference picture list is different.

    NOTE 2 – The number of motion vectors that are used for the prediction of a macroblock partition with macroblock partition index mbPartIdx, or a sub-macroblock partition contained in this macroblock partition, is equal to PredFlagL0[mbPartIdx] + PredFlagL1[mbPartIdx].

  – mixedModeEdgeFlag is equal to 0 and one motion vector is used to predict the macroblock/sub-macroblock partition containing the sample p_0 and one motion vector is used to predict the macroblock/sub-macroblock partition containing the sample q_0 and the absolute difference between the horizontal or vertical components of the motion vectors used is greater than or equal to 4 in units of quarter luma frame samples,

  – mixedModeEdgeFlag is equal to 0 and two motion vectors and two different reference pictures are used to predict the macroblock/sub-macroblock partition containing the sample p_0 and two motion vectors for the same two reference pictures are used to predict the macroblock/sub-macroblock partition containing the sample q_0 and, for either or both of the two used reference pictures, the absolute difference between the horizontal or vertical components of the two motion vectors used in the prediction of the two macroblock/sub-macroblock partitions for the particular reference picture is greater than or equal to 4 in units of quarter luma frame samples,

  – mixedModeEdgeFlag is equal to 0 and two motion vectors for the same reference picture are used to predict the macroblock/sub-macroblock partition containing the sample p_0 and two motion vectors for the same reference picture are used to predict the macroblock/sub-macroblock partition containing the sample q_0 and both of the following conditions are true:

    – The absolute difference between the horizontal or vertical components of list 0 motion vectors used in the prediction of the two macroblock/sub-macroblock partitions is greater than or equal to 4 in quarter luma frame samples or the absolute difference between the horizontal or vertical components of the list 1 motion vectors used in the prediction of the two macroblock/sub-macroblock partitions is greater than or equal to 4 in units of quarter luma frame samples,

    – The absolute difference between the horizontal or vertical components of list 0 motion vector used in the prediction of the macroblock/sub-macroblock partition containing the sample p_0 and the list 1 motion vector used in the prediction of the macroblock/sub-macroblock partition containing the sample q_0 is greater than or equal to 4 in units of quarter luma frame samples or the absolute difference between the horizontal or vertical components of the list 1 motion vector used in the prediction of the macroblock/sub-macroblock partition containing the sample p_0 and list 0 motion vector used in the prediction of the macroblock/sub-macroblock partition containing the sample q_0 is greater than or equal to 4 in units of quarter luma frame samples.

      NOTE 3 – A vertical difference of 4 in units of quarter luma frame samples is a difference of 2 in units of quarter luma field samples.

– Otherwise, a value of bS equal to 0 is the output.

8.7.2.2 Derivation process for the thresholds for each block edge

Inputs to this process are:

– the input sample values p_0, q_0, p_1 and q_1 of a single set of samples across an edge that is to be filtered,

– the variables chromaEdgeFlag and bS, for the set of input samples, as specified in subclause 8.7.2,

– the variables filterOffsetA, filterOffsetB, qP_p, and qP_q.

Outputs of this process are the variable filterSamplesFlag, which indicates whether the input samples are filtered, the value of indexA, and the values of the threshold variables α and β.

Let qP_{av} be a variable specifying an average quantisation parameter. It is derived as:

$$qP_{av} = (qP_p + qP_q + 1) >> 1 \quad (8\text{-}453)$$

NOTE – In SP and SI slices, qP_{av} is derived in the same way as in other slice types. QS_Y from Equation 7-30 is not used in the deblocking filter.

Let indexA be a variable that is used to access the α table (Table 8-16) as well as the t_{C0} table (Table 8-17), which is used in filtering of edges with bS less than 4 as specified in subclause 8.7.2.3, and let indexB be a variable that is used to access the β table (Table 8-16). The variables indexA and indexB are derived as:

$$\text{indexA} = \text{Clip3}(0, 51, qP_{av} + \text{filterOffsetA}) \quad (8\text{-}454)$$

$$\text{indexB} = \text{Clip3}(0, 51, qP_{av} + \text{filterOffsetB}) \quad (8\text{-}455)$$

The variables α' and β' depending on the values of indexA and indexB are specified in Table 8-16. Depending on chromaEdgeFlag, the corresponding threshold variables α and β are derived as follows:

– If chromaEdgeFlag is equal to 0,

$$\alpha = \alpha' * (1 << (\text{BitDepth}_Y - 8)) \quad (8\text{-}456)$$

$$\beta = \beta' * (1 << (\text{BitDepth}_Y - 8)) \quad (8\text{-}457)$$

– Otherwise (chromaEdgeFlag is equal to 1),

$$\alpha = \alpha' * (1 << (\text{BitDepth}_C - 8)) \quad (8\text{-}458)$$

$$\beta = \beta' * (1 << (\text{BitDepth}_C - 8)) \quad (8\text{-}459)$$

The variable filterSamplesFlag is derived by:

$$\text{filterSamplesFlag} = (\text{bS} \mathrel{!=} 0 \;\&\&\; \text{Abs}(p_0 - q_0) < \alpha \;\&\&\; \text{Abs}(p_1 - p_0) < \beta \;\&\&\; \text{Abs}(q_1 - q_0) < \beta) \quad (8\text{-}460)$$

Table 8-16 – Derivation of offset dependent threshold variables α' and β' from indexA and indexB

	indexA (for α') or indexB (for β')																									
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
α'	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	5	6	7	8	9	10	12	13
β'	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	3	3	3	3	4	4	4

Table 8-16 (concluded) – Derivation of indexA and indexB from offset dependent threshold variables α' and β'

	indexA (for α') or indexB (for β')																									
	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
α'	15	17	20	22	25	28	32	36	40	45	50	56	63	71	80	90	101	113	127	144	162	182	203	226	255	255
β'	6	6	7	7	8	8	9	9	10	10	11	11	12	12	13	13	14	14	15	15	16	16	17	17	18	18

8.7.2.3 Filtering process for edges with bS less than 4

Inputs to this process are the input sample values p_i and q_i ($i = 0..2$) of a single set of samples across an edge that is to be filtered, chromaEdgeFlag, chromaStyleFilteringFlag, bS, β, and indexA, for the set of input samples, as specified in subclause 8.7.2.

Outputs of this process are the filtered result sample values p'_i and q'_i ($i = 0..2$) for the set of input sample values.

Depending on the values of indexA and bS, the variable t'_{C0} is specified in Table 8-17. Depending on chromaEdgeFlag, the corresponding threshold variable t_{C0} is derived as follows:

– If chromaEdgeFlag is equal to 0,

$$t_{C0} = t'_{C0} * (1 << (\mathrm{BitDepth}_Y - 8)) \quad (8\text{-}461)$$

– Otherwise (chromaEdgeFlag is equal to 1),

$$t_{C0} = t'_{C0} * (1 << (\mathrm{BitDepth}_C - 8)) \quad (8\text{-}462)$$

Table 8-17 – Value of variable t'_{C0} as a function of indexA and bS

	indexA																									
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
bS = 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
bS = 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
bS = 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1

Table 8-17 (concluded) – Value of variable t'_{C0} as a function of indexA and bS

	indexA																									
	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
bS = 1	1	1	1	1	1	1	1	2	2	2	2	3	3	3	4	4	4	5	6	6	7	8	9	10	11	13
bS = 2	1	1	1	1	1	2	2	2	2	3	3	3	4	4	5	5	6	7	8	8	10	11	12	13	15	17
bS = 3	1	2	2	2	2	3	3	3	4	4	4	5	6	6	7	8	9	10	11	13	14	16	18	20	23	25

The threshold variables a_p and a_q are derived by:

$$a_p = \mathrm{Abs}(p_2 - p_0) \quad (8\text{-}463)$$
$$a_q = \mathrm{Abs}(q_2 - q_0) \quad (8\text{-}464)$$

The threshold variable t_C is determined as follows:

– If chromaStyleFilteringFlag is equal to 0,

$$t_C = t_{C0} + ((a_p < \beta) \mathbin{?} 1 : 0) + ((a_q < \beta) \mathbin{?} 1 : 0) \quad (8\text{-}465)$$

– Otherwise (chromaStyleFilteringFlag is equal to 1),

$$t_C = t_{C0} + 1 \tag{8-466}$$

Let Clip1( ) be a function that is replaced by $\text{Clip1}_Y(\ )$ when chromaEdgeFlag is equal to 0 and by $\text{Clip1}_C(\ )$ when chromaEdgeFlag is equal to 1.

The filtered result samples p'_0 and q'_0 are derived by:

$$\Delta = \text{Clip3}(-t_C, t_C, ((((q_0 - p_0) << 2) + (p_1 - q_1) + 4) >> 3)) \tag{8-467}$$
$$p'_0 = \text{Clip1}(p_0 + \Delta) \tag{8-468}$$
$$q'_0 = \text{Clip1}(q_0 - \Delta) \tag{8-469}$$

The filtered result sample p'_1 is derived as follows:

– If chromaStyleFilteringFlag is equal to 0 and a_p is less than β,

$$p'_1 = p_1 + \text{Clip3}(-t_{C0}, t_{C0}, (p_2 + ((p_0 + q_0 + 1) >> 1) - (p_1 << 1)) >> 1) \tag{8-470}$$

– Otherwise (chromaStyleFilteringFlag is equal to 1 or a_p is greater than or equal to β),

$$p'_1 = p_1 \tag{8-471}$$

The filtered result sample q'_1 is derived as follows:

– If chromaStyleFilteringFlag is equal to 0 and a_q is less than β,

$$q'_1 = q_1 + \text{Clip3}(-t_{C0}, t_{C0}, (q_2 + ((p_0 + q_0 + 1) >> 1) - (q_1 << 1)) >> 1) \tag{8-472}$$

– Otherwise (chromaStyleFilteringFlag is equal to 1 or a_q is greater than or equal to β),

$$q'_1 = q_1 \tag{8-473}$$

The filtered result samples p'_2 and q'_2 are always set equal to the input samples p_2 and q_2:

$$p'_2 = p_2 \tag{8-474}$$
$$q'_2 = q_2 \tag{8-475}$$

8.7.2.4 Filtering process for edges for bS equal to 4

Inputs to this process are the input sample values p_i and q_i ($i = 0..3$) of a single set of samples across an edge that is to be filtered, chromaEdgeFlag, chromaStyleFilteringFlag, and the values of the threshold variables α and β for the set of samples, as specified in subclause 8.7.2.

Outputs of this process are the filtered result sample values p'_i and q'_i ($i = 0..2$) for the set of input sample values.

Let a_p and a_q be two threshold variables as specified in Equations 8-463 and 8-464, respectively, in subclause 8.7.2.3.

The filtered result samples p'_i ($i = 0..2$) are derived as follows:

– If chromaStyleFilteringFlag is equal to 0 and the following condition holds,

$$a_p < \beta \ \&\& \ \text{Abs}(p_0 - q_0) < ((\alpha >> 2) + 2) \tag{8-476}$$

then the variables p'_0, p'_1, and p'_2 are derived by:

$$p'_0 = (p_2 + 2{*}p_1 + 2{*}p_0 + 2{*}q_0 + q_1 + 4) >> 3 \tag{8-477}$$

$$p'_1 = (p_2 + p_1 + p_0 + q_0 + 2) >> 2 \tag{8-478}$$

$$p'_2 = (2{*}p_3 + 3{*}p_2 + p_1 + p_0 + q_0 + 4) >> 3 \tag{8-479}$$

– Otherwise (chromaStyleFilteringFlag is equal to 1 or the condition in Equation 8-476 does not hold), the variables p'_0, p'_1, and p'_2 are derived by:

$$p'_0 = (2{*}p_1 + p_0 + q_1 + 2) >> 2 \tag{8-480}$$

$$p'_1 = p_1 \quad (8\text{-}481)$$

$$p'_2 = p_2 \quad (8\text{-}482)$$

The filtered result samples q'_i ($i = 0..2$) are derived as follows:

– If chromaStyleFilteringFlag is equal to 0 and the following condition holds,

$$a_q < \beta \ \&\& \ \text{Abs}(p_0 - q_0) < ((\alpha >> 2) + 2) \quad (8\text{-}483)$$

then the variables q'_0, q'_1, and q'_2 are derived by

$$q'_0 = (p_1 + 2{*}p_0 + 2{*}q_0 + 2{*}q_1 + q_2 + 4) >> 3 \quad (8\text{-}484)$$

$$q'_1 = (p_0 + q_0 + q_1 + q_2 + 2) >> 2 \quad (8\text{-}485)$$

$$q'_2 = (2{*}q_3 + 3{*}q_2 + q_1 + q_0 + p_0 + 4) >> 3 \quad (8\text{-}486)$$

– Otherwise (chromaStyleFilteringFlag is equal to 1 or the condition in Equation 8-483 does not hold), the variables q'_0, q'_1, and q'_2 are derived by:

$$q'_0 = (2{*}q_1 + q_0 + p_1 + 2) >> 2 \quad (8\text{-}487)$$

$$q'_1 = q_1 \quad (8\text{-}488)$$

$$q'_2 = q_2 \quad (8\text{-}489)$$

9 Parsing process

Inputs to this process are bits from the RBSP.

Outputs of this process are syntax element values.

This process is invoked when the descriptor of a syntax element in the syntax tables in subclause 7.3 is equal to ue(v), me(v), se(v), te(v) (see subclause 9.1), ce(v) (see subclause 9.2), or ae(v) (see subclause 9.3).

9.1 Parsing process for Exp-Golomb codes

This process is invoked when the descriptor of a syntax element in the syntax tables in subclause 7.3 is equal to ue(v), me(v), se(v), or te(v). For syntax elements in subclauses 7.3.4 and 7.3.5, this process is invoked only when entropy_coding_mode_flag is equal to 0.

Inputs to this process are bits from the RBSP.

Outputs of this process are syntax element values.

Syntax elements coded as ue(v), me(v), or se(v) are Exp-Golomb-coded. Syntax elements coded as te(v) are truncated Exp-Golomb-coded. The parsing process for these syntax elements begins with reading the bits starting at the current location in the bitstream up to and including the first non-zero bit, and counting the number of leading bits that are equal to 0. This process is specified as follows:

```
leadingZeroBits = −1
for( b = 0; !b; leadingZeroBits++ )                                    (9-1)
      b = read_bits( 1 )
```

The variable codeNum is then assigned as follows:

$$\text{codeNum} = 2^{\text{leadingZeroBits}} - 1 + \text{read\_bits(leadingZeroBits)} \quad (9\text{-}2)$$

where the value returned from read_bits(leadingZeroBits) is interpreted as a binary representation of an unsigned integer with most significant bit written first.

Table 9-1 illustrates the structure of the Exp-Golomb code by separating the bit string into "prefix" and "suffix" bits. The "prefix" bits are those bits that are parsed in the above pseudo-code for the computation of leadingZeroBits, and are shown as either 0 or 1 in the bit string column of Table 9-1. The "suffix" bits are those bits that are parsed in the computation of codeNum and are shown as x_i in Table 9-1, with i being in the range 0 to leadingZeroBits − 1, inclusive. Each x_i can take on values 0 or 1.

Table 9-1 – Bit strings with "prefix" and "suffix" bits and assignment to codeNum ranges (informative)

Bit string form	Range of codeNum
1	0
0 1 x_0	1..2
0 0 1 x_1 x_0	3..6
0 0 0 1 x_2 x_1 x_0	7..14
0 0 0 0 1 x_3 x_2 x_1 x_0	15..30
0 0 0 0 0 1 x_4 x_3 x_2 x_1 x_0	31..62
…	…

Table 9-2 illustrates explicitly the assignment of bit strings to codeNum values.

Table 9-2 – Exp-Golomb bit strings and codeNum in explicit form and used as ue(v) (informative)

Bit string	codeNum
1	0
0 1 0	1
0 1 1	2
0 0 1 0 0	3
0 0 1 0 1	4
0 0 1 1 0	5
0 0 1 1 1	6
0 0 0 1 0 0 0	7
0 0 0 1 0 0 1	8
0 0 0 1 0 1 0	9
…	…

Depending on the descriptor, the value of a syntax element is derived as follows:

– If the syntax element is coded as ue(v), the value of the syntax element is equal to codeNum.

– Otherwise, if the syntax element is coded as se(v), the value of the syntax element is derived by invoking the mapping process for signed Exp-Golomb codes as specified in subclause 9.1.1 with codeNum as the input.

– Otherwise, if the syntax element is coded as me(v), the value of the syntax element is derived by invoking the mapping process for coded block pattern as specified in subclause 9.1.2 with codeNum as the input.

– Otherwise (the syntax element is coded as te(v)), the range of possible values for the syntax element is determined first. The range of this syntax element may be between 0 and x, with x being greater than or equal to 1 and the range is used in the derivation of the value of the syntax element value as follows:

  – If x is greater than 1, codeNum and the value of the syntax element is derived in the same way as for syntax elements coded as ue(v).

– Otherwise (x is equal to 1), the parsing process for codeNum which is equal to the value of the syntax element is given by a process equivalent to:

b = read_bits(1) (9-3)
codeNum = !b

9.1.1 Mapping process for signed Exp-Golomb codes

Input to this process is codeNum as specified in subclause 9.1.

Output of this process is a value of a syntax element coded as se(v).

The syntax element is assigned to the codeNum by ordering the syntax element by its absolute value in increasing order and representing the positive value for a given absolute value with the lower codeNum. Table 9-3 provides the assignment rule.

Table 9-3 – Assignment of syntax element to codeNum for signed Exp-Golomb coded syntax elements se(v)

codeNum	syntax element value
0	0
1	1
2	−1
3	2
4	−2
5	3
6	−3
k	$(-1)^{k+1}$ Ceil(k÷2)

9.1.2 Mapping process for coded block pattern

Input to this process is codeNum as specified in subclause 9.1.

Output of this process is a value of the syntax element coded_block_pattern coded as me(v).

Table 9-4 shows the assignment of coded_block_pattern to codeNum depending on whether the macroblock prediction mode is equal to Intra_4x4, Intra_8x8 or Inter.

Table 9-4 – Assignment of codeNum to values of coded_block_pattern for macroblock prediction modes

(a) ChromaArrayType is equal to 1 or 2

codeNum	coded_block_pattern	
	Intra_4x4, Intra_8x8	Inter
0	47	0
1	31	16
2	15	1
3	0	2
4	23	4
5	27	8
6	29	32
7	30	3
8	7	5
9	11	10
10	13	12
11	14	15
12	39	47
13	43	7
14	45	11
15	46	13
16	16	14
17	3	6
18	5	9
19	10	31
20	12	35
21	19	37
22	21	42
23	26	44
24	28	33
25	35	34
26	37	36
27	42	40
28	44	39
29	1	43
30	2	45

(a) ChromaArrayType is equal to 1 or 2

codeNum	coded_block_pattern	
	Intra_4x4, Intra_8x8	Inter
31	4	46
32	8	17
33	17	18
34	18	20
35	20	24
36	24	19
37	6	21
38	9	26
39	22	28
40	25	23
41	32	27
42	33	29
43	34	30
44	36	22
45	40	25
46	38	38
47	41	41

(b) ChromaArrayType is equal to 0 or 3

codeNum	coded_block_pattern	
	Intra_4x4, Intra_8x8	Inter
0	15	0
1	0	1
2	7	2
3	11	4
4	13	8
5	14	3
6	3	5
7	5	10
8	10	12
9	12	15

(b) ChromaArrayType is equal to 0 or 3

codeNum	coded_block_pattern	
	Intra_4x4, Intra_8x8	Inter
10	1	7
11	2	11
12	4	13
13	8	14
14	6	6
15	9	9

9.2 CAVLC parsing process for transform coefficient levels

This process is invoked for the parsing of syntax elements with descriptor equal to ce(v) in subclause 7.3.5.3.2 when entropy_coding_mode_flag is equal to 0.

Inputs to this process are bits from slice data, a maximum number of non-zero transform coefficient levels maxNumCoeff, the luma block index luma4x4BlkIdx or the chroma block index chroma4x4BlkIdx, cb4x4BlkIdx or cr4x4BlkIdx of the current block of transform coefficient levels.

Output of this process is the list coeffLevel containing transform coefficient levels of the luma block with block index luma4x4BlkIdx or the chroma block with block index chroma4x4BlkIdx, cb4x4BlkIdx or cr4x4BlkIdx.

The process is specified in the following ordered steps:

1. All transform coefficient level values coeffLevel[i], with indices i ranging from 0 to maxNumCoeff − 1, in the list coeffLevel are set equal to 0.
2. The total number of non-zero transform coefficient levels TotalCoeff(coeff_token) and the number of trailing one transform coefficient levels TrailingOnes(coeff_token) are derived by parsing coeff_token as specified in subclause 9.2.1.
3. The following then applies:
   – If the number of non-zero transform coefficient levels TotalCoeff(coeff_token) is equal to 0, the list coeffLevel (in which all transform coefficient level values are equal to 0) is returned and no further steps are carried out.
   – Otherwise, the following steps are carried out:
     a. The non-zero transform coefficient levels are derived by parsing trailing_ones_sign_flag, level_prefix, and level_suffix as specified in subclause 9.2.2.
     b. The runs of zero transform coefficient levels before each non-zero transform coefficient level are derived by parsing total_zeros and run_before as specified in subclause 9.2.3.
     c. The level and run information are combined into the list coeffLevel as specified in subclause 9.2.4.

9.2.1 Parsing process for total number of non-zero transform coefficient levels and number of trailing ones

Inputs to this process are bits from slice data, a maximum number of non-zero transform coefficient levels maxNumCoeff, the luma block index luma4x4BlkIdx or the chroma block index chroma4x4BlkIdx, cb4x4BlkIdx or cr4x4BlkIdx of the current block of transform coefficient levels.

Outputs of this process are TotalCoeff(coeff_token), TrailingOnes(coeff_token), and the variable nC.

The syntax element coeff_token is decoded using one of the six VLCs specified in the six right-most columns of Table 9-5. Each VLC specifies both TotalCoeff(coeff_token) and TrailingOnes(coeff_token) for a given codeword coeff_token. The selection of the applicable column of Table 9-5 is determined by a variable nC. The value of nC is derived as follows:

– If the CAVLC parsing process is invoked for ChromaDCLevel, nC is derived as follows:

  – If ChromaArrayType is equal to 1, nC is set equal to −1,

  – Otherwise (ChromaArrayType is equal to 2), nC is set equal to −2,

– Otherwise, the following ordered steps are performed:

  1. When the CAVLC parsing process is invoked for Intra16x16DCLevel, luma4x4BlkIdx is set equal to 0.

  2. When the CAVLC parsing process is invoked for CbIntra16x16DCLevel, cb4x4BlkIdx is set equal to 0.

  3. When the CAVLC parsing process is invoked for CrIntra16x16DCLevel, cr4x4BlkIdx is set equal to 0.

  4. The variables blkA and blkB are derived as follows:

     – If the CAVLC parsing process is invoked for Intra16x16DCLevel, Intra16x16ACLevel, or LumaLevel4x4, the process specified in subclause 6.4.11.4 is invoked with luma4x4BlkIdx as the input, and the output is assigned to mbAddrA, mbAddrB, luma4x4BlkIdxA, and luma4x4BlkIdxB. The 4x4 luma block specified by mbAddrA\luma4x4BlkIdxA is assigned to blkA, and the 4x4 luma block specified by mbAddrB\luma4x4BlkIdxB is assigned to blkB.

     – Otherwise, if the CAVLC parsing process is invoked for CbIntra16x16DCLevel, CbIntra16x16ACLevel, or CbLevel4x4, the process specified in subclause 6.4.11.6 is invoked with cb4x4BlkIdx as the input, and the output is assigned to mbAddrA, mbAddrB, cb4x4BlkIdxA, and cb4x4BlkIdxB. The 4x4 Cb block specified by mbAddrA\cb4x4BlkIdxA is assigned to blkA, and the 4x4 Cb block specified by mbAddrB\cb4x4BlkIdxB is assigned to blkB.

     – Otherwise, if the CAVLC parsing process is invoked for CrIntra16x16DCLevel, CrIntra16x16ACLevel, or CrLevel4x4, the process specified in subclause 6.4.11.6 is invoked with cr4x4BlkIdx as the input, and the output is assigned to mbAddrA, mbAddrB, cr4x4BlkIdxA, and cr4x4BlkIdxB. The 4x4 Cr block specified by mbAddrA\cr4x4BlkIdxA is assigned to blkA, and the 4x4 Cr block specified by mbAddrB\cr4x4BlkIdxB is assigned to blkB.

     – Otherwise (the CAVLC parsing process is invoked for ChromaACLevel), the process specified in subclause 6.4.11.5 is invoked with chroma4x4BlkIdx as input, and the output is assigned to mbAddrA, mbAddrB, chroma4x4BlkIdxA, and chroma4x4BlkIdxB. The 4x4 chroma block specified by mbAddrA\iCbCr\chroma4x4BlkIdxA is assigned to blkA, and the 4x4 chroma block specified by mbAddrB\iCbCr\chroma4x4BlkIdxB is assigned to blkB.

  5. The variable availableFlagN with N being replaced by A and B is derived as follows:

     – If any of the following conditions are true, availableFlagN is set equal to 0:

       – mbAddrN is not available,

       – the current macroblock is coded using an Intra macroblock prediction mode, constrained_intra_pred_flag is equal to 1, mbAddrN is coded using an Inter macroblock prediction mode, and slice data partitioning is in use (nal_unit_type is in the range of 2 to 4, inclusive).

     – Otherwise, availableFlagN is set equal to 1.

  6. For N being replaced by A and B, when availableFlagN is equal to 1, the variable nN is derived as follows:

     – If any of the following conditions are true, nN is set equal to 0:

       – The macroblock mbAddrN has mb_type equal to P_Skip or B_Skip,

       – The macroblock mbAddrN has mb_type not equal to I_PCM and all AC residual transform coefficient levels of the neighbouring block blkN are equal to 0 due to the corresponding bit of CodedBlockPatternLuma or CodedBlockPatternChroma being equal to 0.

     – Otherwise, if mbAddrN is an I_PCM macroblock, nN is set equal to 16.

     – Otherwise, nN is set equal to the value TotalCoeff(coeff_token) of the neighbouring block blkN.

       NOTE 1 – The values nA and nB that are derived using TotalCoeff(coeff_token) do not include the DC transform coefficient levels in Intra_16x16 macroblocks or DC transform coefficient levels in chroma blocks, because these transform coefficient levels are decoded separately. When the block above or to the left belongs to an Intra_16x16 macroblock, nA or nB is the number of decoded non-zero AC transform coefficient levels for the adjacent 4x4 block in the Intra_16x16 macroblock. When the block above or to the left is a chroma block, nA or nB is the number of decoded non-zero AC transform coefficient levels for the adjacent chroma block.

NOTE 2 – When parsing for Intra16x16DCLevel, CbIntra16x16DCLevel, or CrIntra16x16DCLevel, the values nA and nB are based on the number of non-zero transform coefficient levels in adjacent 4x4 blocks and not on the number of non-zero DC transform coefficient levels in adjacent 16x16 blocks.

7. The variable nC is derived as follows:
   - If availableFlagA is equal to 1 and availableFlagB is equal to 1, the variable nC is set equal to (nA + nB + 1) >> 1.
   - Otherwise, if availableFlagA is equal to 1 (and availableFlagB is equal to 0), the variable nC is set equal to nA.
   - Otherwise, if availableFlagB is equal to 1 (and availableFlagA is equal to 0), the variable nC is set equal to nB.
   - Otherwise (availableFlagA is equal to 0 and availableFlagB is equal to 0), the variable nC is set equal to 0.

When maxNumCoeff is equal to 15, it is a requirement of bitstream conformance that the value of TotalCoeff(coeff_token) resulting from decoding coeff_token shall not be equal to 16.

Table 9-5 – coeff_token mapping to TotalCoeff(coeff_token) and TrailingOnes(coeff_token)

TrailingOnes (coeff_token)	TotalCoeff (coeff_token)	0 <= nC < 2	2 <= nC < 4	4 <= nC < 8	8 <= nC	nC = = −1	nC = = −2
0	0	1	11	1111	0000 11	01	1
0	1	0001 01	0010 11	0011 11	0000 00	0001 11	0001 111
1	1	01	10	1110	0000 01	1	01
0	2	0000 0111	0001 11	0010 11	0001 00	0001 00	0001 110
1	2	0001 00	0011 1	0111 1	0001 01	0001 10	0001 101
2	2	001	011	1101	0001 10	001	001
0	3	0000 0011 1	0000 111	0010 00	0010 00	0000 11	0000 0011 1
1	3	0000 0110	0010 10	0110 0	0010 01	0000 011	0001 100
2	3	0000 101	0010 01	0111 0	0010 10	0000 010	0001 011
3	3	0001 1	0101	1100	0010 11	0001 01	0000 1
0	4	0000 0001 11	0000 0111	0001 111	0011 00	0000 10	0000 0011 0
1	4	0000 0011 0	0001 10	0101 0	0011 01	0000 0011	0000 0010 1
2	4	0000 0101	0001 01	0101 1	0011 10	0000 0010	0001 010
3	4	0000 11	0100	1011	0011 11	0000 000	0000 01
0	5	0000 0000 111	0000 0100	0001 011	0100 00	-	0000 0001 11
1	5	0000 0001 10	0000 110	0100 0	0100 01	-	0000 0001 10
2	5	0000 0010 1	0000 101	0100 1	0100 10	-	0000 0010 0
3	5	0000 100	0011 0	1010	0100 11	-	0001 001
0	6	0000 0000 0111 1	0000 0011 1	0001 001	0101 00	-	0000 0000 111
1	6	0000 0000 110	0000 0110	0011 10	0101 01	-	0000 0000 110
2	6	0000 0001 01	0000 0101	0011 01	0101 10	-	0000 0001 01
3	6	0000 0100	0010 00	1001	0101 11	-	0001 000

Table 9-5 – coeff_token mapping to TotalCoeff(coeff_token) and TrailingOnes(coeff_token)

TrailingOnes (coeff_token)	TotalCoeff (coeff_token)	0 <= nC < 2	2 <= nC < 4	4 <= nC < 8	8 <= nC	nC = = −1	nC = = −2
0	7	0000 0000 0101 1	0000 0001 111	0001 000	0110 00	-	0000 0000 0111
1	7	0000 0000 0111 0	0000 0011 0	0010 10	0110 01	-	0000 0000 0110
2	7	0000 0000 101	0000 0010 1	0010 01	0110 10	-	0000 0000 101
3	7	0000 0010 0	0001 00	1000	0110 11	-	0000 0001 00
0	8	0000 0000 0100 0	0000 0001 011	0000 1111	0111 00	-	0000 0000 0011 1
1	8	0000 0000 0101 0	0000 0001 110	0001 110	0111 01	-	0000 0000 0101
2	8	0000 0000 0110 1	0000 0001 101	0001 101	0111 10	-	0000 0000 0100
3	8	0000 0001 00	0000 100	0110 1	0111 11	-	0000 0000 100
0	9	0000 0000 0011 11	0000 0000 1111	0000 1011	1000 00	-	-
1	9	0000 0000 0011 10	0000 0001 010	0000 1110	1000 01	-	-
2	9	0000 0000 0100 1	0000 0001 001	0001 010	1000 10	-	-
3	9	0000 0000 100	0000 0010 0	0011 00	1000 11	-	-
0	10	0000 0000 0010 11	0000 0000 1011	0000 0111 1	1001 00	-	-
1	10	0000 0000 0010 10	0000 0000 1110	0000 1010	1001 01	-	-
2	10	0000 0000 0011 01	0000 0000 1101	0000 1101	1001 10	-	-
3	10	0000 0000 0110 0	0000 0001 100	0001 100	1001 11	-	-
0	11	0000 0000 0001 111	0000 0000 1000	0000 0101 1	1010 00	-	-
1	11	0000 0000 0001 110	0000 0000 1010	0000 0111 0	1010 01	-	-
2	11	0000 0000 0010 01	0000 0000 1001	0000 1001	1010 10	-	-
3	11	0000 0000 0011 00	0000 0001 000	0000 1100	1010 11	-	-
0	12	0000 0000 0001 011	0000 0000 0111 1	0000 0100 0	1011 00	-	-
1	12	0000 0000 0001 010	0000 0000 0111 0	0000 0101 0	1011 01	-	-
2	12	0000 0000 0001 101	0000 0000 0110 1	0000 0110 1	1011 10	-	-
3	12	0000 0000 0010 00	0000 0000 1100	0000 1000	1011 11	-	-
0	13	0000 0000 0000 1111	0000 0000 0101 1	0000 0011 01	1100 00	-	-
1	13	0000 0000 0000 001	0000 0000 0101 0	0000 0011 1	1100 01	-	-
2	13	0000 0000 0001 001	0000 0000 0100 1	0000 0100 1	1100 10	-	-
3	13	0000 0000 0001 100	0000 0000 0110 0	0000 0110 0	1100 11	-	-
0	14	0000 0000 0000 1011	0000 0000 0011 1	0000 0010 01	1101 00	-	-
1	14	0000 0000 0000 1110	0000 0000 0010 11	0000 0011 00	1101 01	-	-
2	14	0000 0000 0000 1101	0000 0000 0011 0	0000 0010 11	1101 10	-	-
3	14	0000 0000 0001 000	0000 0000 0100 0	0000 0010 10	1101 11	-	-
0	15	0000 0000 0000 0111	0000 0000 0010 01	0000 0001 01	1110 00	-	-
1	15	0000 0000 0000 1010	0000 0000 0010 00	0000 0010 00	1110 01	-	-

Table 9-5 – coeff_token mapping to TotalCoeff(coeff_token) and TrailingOnes(coeff_token)

TrailingOnes (coeff_token)	TotalCoeff (coeff_token)	0 <= nC < 2	2 <= nC < 4	4 <= nC < 8	8 <= nC	nC = = −1	nC = = −2
2	15	0000 0000 0000 1001	0000 0000 0010 10	0000 0001 11	1110 10	-	-
3	15	0000 0000 0000 1100	0000 0000 0000 1	0000 0001 10	1110 11	-	-
0	16	0000 0000 0000 0100	0000 0000 0001 11	0000 0000 01	1111 00	-	-
1	16	0000 0000 0000 0110	0000 0000 0001 10	0000 0001 00	1111 01	-	-
2	16	0000 0000 0000 0101	0000 0000 0001 01	0000 0000 11	1111 10	-	-
3	16	0000 0000 0000 1000	0000 0000 0001 00	0000 0000 10	1111 11	-	-

9.2.2 Parsing process for level information

Inputs to this process are bits from slice data, the number of non-zero transform coefficient levels TotalCoeff(coeff_token), and the number of trailing one transform coefficient levels TrailingOnes(coeff_token).

Output of this process is a list with name levelVal containing transform coefficient levels.

Initially an index i is set equal to 0. Then, when TrailingOnes(coeff_token) is not equal to 0, the following ordered steps are applied TrailingOnes(coeff_token) times to decode the trailing one transform coefficient levels:

1. A 1-bit syntax element trailing_ones_sign_flag is decoded and evaluated as follows:
   – If trailing_ones_sign_flag is equal to 0, levelVal[i] is set equal to 1.
   – Otherwise (trailing_ones_sign_flag is equal to 1), levelVal[i] is set equal to −1.
2. The index i is incremented by 1.

Then, the variable suffixLength is initialised as follows:

– If TotalCoeff(coeff_token) is greater than 10 and TrailingOnes(coeff_token) is less than 3, suffixLength is set equal to 1.

– Otherwise (TotalCoeff(coeff_token) is less than or equal to 10 or TrailingOnes(coeff_token) is equal to 3), suffixLength is set equal to 0.

Then, when TotalCoeff(coeff_token) − TrailingOnes(coeff_token) is not equal to 0, the following ordered steps are applied TotalCoeff(coeff_token) − TrailingOnes(coeff_token) times to decode the remaining non-zero level values:

1. The syntax element level_prefix is decoded as specified in subclause 9.2.2.1.
2. The variable levelSuffixSize is set as follows:
   – If level_prefix is equal to 14 and suffixLength is equal to 0, levelSuffixSize is set equal to 4.
   – Otherwise, if level_prefix is greater than or equal to 15, levelSuffixSize is set equal to level_prefix − 3.
   – Otherwise, levelSuffixSize is set equal to suffixLength.
3. The syntax element level_suffix is decoded as follows:
   – If levelSuffixSize is greater than 0, the syntax element level_suffix is decoded as unsigned integer representation u(v) with levelSuffixSize bits.
   – Otherwise (levelSuffixSize is equal to 0), the syntax element level_suffix is inferred to be equal to 0.
4. The variable levelCode is set equal to (Min(15, level_prefix) << suffixLength) + level_suffix.
5. When level_prefix is greater than or equal to 15 and suffixLength is equal to 0, levelCode is incremented by 15.
6. When level_prefix is greater than or equal to 16, levelCode is incremented by (1<<(level_prefix − 3)) − 4096.

7. When the index i is equal to TrailingOnes(coeff_token) and TrailingOnes(coeff_token) is less than 3, levelCode is incremented by 2.

8. The variable levelVal[i] is derived as follows:

   – If levelCode is an even number, levelVal[i] is set equal to (levelCode + 2) >> 1.

   – Otherwise (levelCode is an odd number), levelVal[i] is set equal to (−levelCode − 1) >> 1.

9. When suffixLength is equal to 0, suffixLength is set equal to 1.

10. When the absolute value of levelVal[i] is greater than (3 << (suffixLength − 1)) and suffixLength is less than 6, suffixLength is incremented by 1.

11. The index i is incremented by 1.

9.2.2.1 Parsing process for level_prefix

Inputs to this process are bits from slice data.

Output of this process is level_prefix.

The parsing process for this syntax element consists in reading the bits starting at the current location in the bitstream up to and including the first non-zero bit, and counting the number of leading bits that are equal to 0. This process is specified as follows:

```
leadingZeroBits = −1
for( b = 0; !b; leadingZeroBits++ )                    (9-4)
      b = read_bits( 1 )
level_prefix = leadingZeroBits
```

Table 9-6 illustrates the codeword table for level_prefix.

NOTE – The value of level_prefix is constrained to not exceed 15 in bitstreams conforming to the Baseline, Constrained Baseline, Main, and Extended profiles, as specified in subclauses A.2.1, A.2.1.1, A.2.2, and A.2.3, respectively. In bitstreams conforming to other profiles, it has been reported that the value of level_prefix cannot exceed 11 + bitDepth with bitDepth being the variable $BitDepth_Y$ for transform coefficient blocks related to the luma component and being the variable $BitDepth_C$ for transform coefficient blocks related to a chroma component.

Table 9-6 – Codeword table for level_prefix (informative)

level_prefix	bit string
0	1
1	01
2	001
3	0001
4	0000 1
5	0000 01
6	0000 001
7	0000 0001
8	0000 0000 1
9	0000 0000 01
10	0000 0000 001
11	0000 0000 0001
12	0000 0000 0000 1
13	0000 0000 0000 01
14	0000 0000 0000 001

15	0000 0000 0000 0001
…	…

9.2.3 Parsing process for run information

Inputs to this process are bits from slice data, the number of non-zero transform coefficient levels TotalCoeff(coeff_token), and the maximum number of non-zero transform coefficient levels maxNumCoeff.

Output of this process is a list of runs of zero transform coefficient levels preceding non-zero transform coefficient levels called runVal.

Initially, an index i is set equal to 0.

The variable zerosLeft is derived as follows:

– If the number of non-zero transform coefficient levels TotalCoeff(coeff_token) is equal to the maximum number of non-zero transform coefficient levels maxNumCoeff, a variable zerosLeft is set equal to 0.

– Otherwise (the number of non-zero transform coefficient levels TotalCoeff(coeff_token) is less than the maximum number of non-zero transform coefficient levels maxNumCoeff), total_zeros is decoded and zerosLeft is set equal to its value.

The variable tzVlcIndex is set equal to TotalCoeff(coeff_token).

The VLC used to decode total_zeros is derived as follows:

– If maxNumCoeff is equal to 4, one of the VLCs specified in Table 9-9 (a) is used.

– Otherwise, if maxNumCoeff is equal to 8, one of the VLCs specified in Table 9-9 (b) is used.

– Otherwise (maxNumCoeff is not equal to 4 and not equal to 8), VLCs from Tables 9-7 and 9-8 are used.

The following ordered steps are then performed TotalCoeff(coeff_token) − 1 times:

1. The variable runVal[i] is derived as follows:

   – If zerosLeft is greater than zero, a value run_before is decoded based on Table 9-10 and zerosLeft. runVal[i] is set equal to run_before.

   – Otherwise (zerosLeft is equal to 0), runVal[i] is set equal to 0.

2. The value of runVal[i] is subtracted from zerosLeft and the result is assigned to zerosLeft. It is a requirement of bitstream conformance that the result of the subtraction shall be greater than or equal to 0.

3. The index i is incremented by 1.

Finally the value of zerosLeft is assigned to runVal[i].

Table 9-7 – total_zeros tables for 4x4 blocks with tzVlcIndex 1 to 7

total_zeros	tzVlcIndex						
	1	2	3	4	5	6	7
0	1	111	0101	0001 1	0101	0000 01	0000 01
1	011	110	111	111	0100	0000 1	0000 1
2	010	101	110	0101	0011	111	101
3	0011	100	101	0100	111	110	100
4	0010	011	0100	110	110	101	011
5	0001 1	0101	0011	101	101	100	11
6	0001 0	0100	100	100	100	011	010
7	0000 11	0011	011	0011	011	010	0001
8	0000 10	0010	0010	011	0010	0001	001
9	0000 011	0001 1	0001 1	0010	0000 1	001	0000 00
10	0000 010	0001 0	0001 0	0001 0	0001	0000 00	-
11	0000 0011	0000 11	0000 01	0000 1	0000 0	-	-
12	0000 0010	0000 10	0000 1	0000 0	-	-	-
13	0000 0001 1	0000 01	0000 00	-	-	-	-
14	0000 0001 0	0000 00	-	-	-	-	-
15	0000 0000 1	-	-	-	-	-	-

Table 9-8 – total_zeros tables for 4x4 blocks with tzVlcIndex 8 to 15

total_zeros	tzVlcIndex							
	8	9	10	11	12	13	14	15
0	0000 01	0000 01	0000 1	0000	0000	000	00	0
1	0001	0000 00	0000 0	0001	0001	001	01	1
2	0000 1	0001	001	001	01	1	1	-
3	011	11	11	010	1	01	-	-
4	11	10	10	1	001	-	-	-
5	10	001	01	011	-	-	-	-
6	010	01	0001	-	-	-	-	-
7	001	0000 1	-	-	-	-	-	-
8	0000 00	-	-	-	-	-	-	-

Table 9-9 – total_zeros tables for chroma DC 2x2 and 2x4 blocks

(a) Chroma DC 2x2 block (4:2:0 chroma sampling)

total_zeros	tzVlcIndex		
	1	2	3
0	1	1	1
1	01	01	0
2	001	00	-
3	000	-	-

(b) Chroma DC 2x4 block (4:2:2 chroma sampling)

total_zeros	tzVlcIndex						
	1	2	3	4	5	6	7
0	1	000	000	110	00	00	0
1	010	01	001	00	01	01	1
2	011	001	01	01	10	1	-
3	0010	100	10	10	11	-	-
4	0011	101	110	111	-	-	-
5	0001	110	111	-	-	-	-
6	0000 1	111	-	-	-	-	-
7	0000 0	-	-	-	-	-	-

Table 9-10 – Tables for run_before

run_before	zerosLeft						
	1	2	3	4	5	6	>6
0	1	1	11	11	11	11	111
1	0	01	10	10	10	000	110
2	-	00	01	01	011	001	101
3	-	-	00	001	010	011	100
4	-	-	-	000	001	010	011
5	-	-	-	-	000	101	010
6	-	-	-	-	-	100	001
7	-	-	-	-	-	-	0001
8		-	-	-	-	-	00001
9	-	-	-	-	-	-	000001
10	-	-	-	-	-	-	0000001
11	-	-	-	-	-	-	00000001
12	-	-	-	-	-	-	000000001
13	-	-	-	-	-	-	0000000001
14	-	-	-	-	-	-	00000000001

9.2.4 Combining level and run information

Input to this process are a list of transform coefficient levels called levelVal, a list of runs called runVal, and the number of non-zero transform coefficient levels TotalCoeff(coeff_token).

Output of this process is an list coeffLevel of transform coefficient levels.

A variable coeffNum is set equal to −1 and an index i is set equal to TotalCoeff(coeff_token) − 1. The following ordered steps are then applied TotalCoeff(coeff_token) times:

1. coeffNum is incremented by runVal[i] + 1.
2. coeffLevel[coeffNum] is set equal to levelVal[i].
3. The index i is decremented by 1.

9.3 CABAC parsing process for slice data

This process is invoked when parsing syntax elements with descriptor ae(v) in subclauses 7.3.4 and 7.3.5 when entropy_coding_mode_flag is equal to 1.

Inputs to this process are a request for a value of a syntax element and values of prior parsed syntax elements.

Output of this process is the value of the syntax element.

When starting the parsing of the slice data of a slice in subclause 7.3.4, the initialisation process of the CABAC parsing process is invoked as specified in subclause 9.3.1.

The parsing of syntax elements proceeds as follows.

For each requested value of a syntax element a binarization is derived as described in subclause 9.3.2.

The binarization for the syntax element and the sequence of parsed bins determines the decoding process flow as described in subclause 9.3.3.

For each bin of the binarization of the syntax element, which is indexed by the variable binIdx, a context index ctxIdx is derived as specified in subclause 9.3.3.1.

For each ctxIdx the arithmetic decoding process is invoked as specified in subclause 9.3.3.2.

The resulting sequence ($b_0..b_{binIdx}$) of parsed bins is compared to the set of bin strings given by the binarization process after decoding of each bin. When the sequence matches a bin string in the given set, the corresponding value is assigned to the syntax element.

In case the request for a value of a syntax element is processed for the syntax element mb_type and the decoded value of mb_type is equal to I_PCM, the decoding engine is initialised after the decoding of any pcm_alignment_zero_bit and all pcm_sample_luma and pcm_sample_chroma data as specified in subclause 9.3.1.2.

The whole CABAC parsing process is illustrated in the flowchart of Figure 9-1 with the abbreviation SE for syntax element.

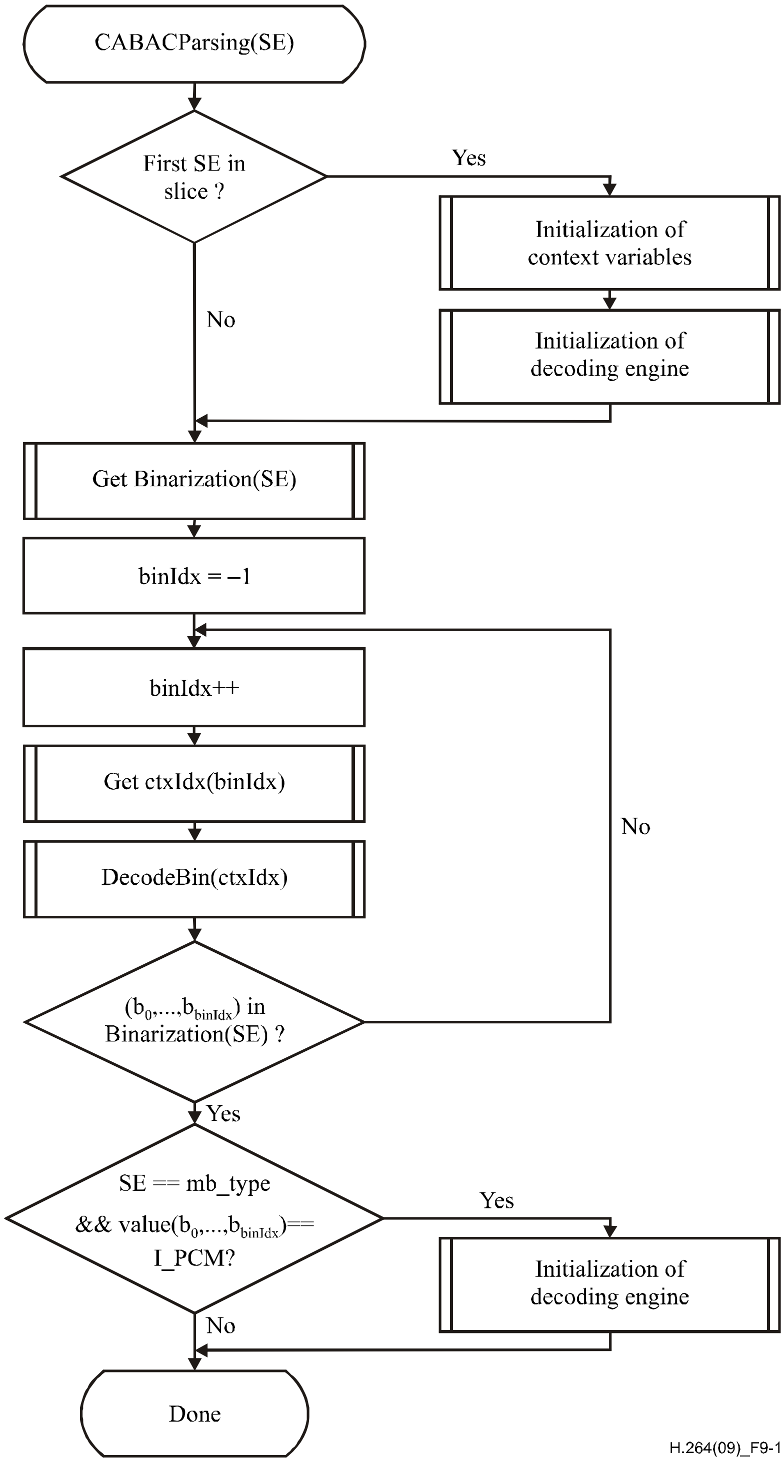


Figure 9-1 – Illustration of CABAC parsing process for a syntax element SE (informative)

9.3.1 Initialisation process

Outputs of this process are initialised CABAC internal variables.

The processes in subclauses 9.3.1.1 and 9.3.1.2 are invoked when starting the parsing of the slice data of a slice in subclause 7.3.4.

The process in subclause 9.3.1.2 is also invoked after decoding any pcm_alignment_zero_bit and all pcm_sample_luma and pcm_sample_chroma data for a macroblock of type I_PCM.

9.3.1.1 Initialisation process for context variables

Outputs of this process are the initialised CABAC context variables indexed by ctxIdx.

Tables 9-12 to 9-33 contain the values of the variables n and m used in the initialisation of context variables that are assigned to all syntax elements in subclauses 7.3.4 and 7.3.5 except for the end-of-slice flag.

For each context variable, the two variables pStateIdx and valMPS are initialised.

NOTE 1 – The variable pStateIdx corresponds to a probability state index and the variable valMPS corresponds to the value of the most probable symbol as further described in subclause 9.3.3.2.

The two values assigned to pStateIdx and valMPS for the initialisation are derived from SliceQP_Y, which is derived in Equation 7-29. Given the two table entries (m, n), the initialisation is specified by the following pseudo-code process:

```
preCtxState = Clip3( 1, 126, ( ( m * Clip3( 0, 51, SliceQPY ) ) >> 4 ) + n )
if( preCtxState <= 63 ) {
    pStateIdx = 63 − preCtxState
    valMPS = 0                                                          (9-5)
} else {
    pStateIdx = preCtxState − 64
    valMPS = 1
}
```

In Table 9-11, the ctxIdx for which initialisation is needed for each of the slice types are listed. Also listed is the table number that includes the values of m and n needed for the initialisation. For P, SP and B slice type, the initialisation depends also on the value of the cabac_init_idc syntax element. Note that the syntax element names do not affect the initialisation process.

Table 9-11 – Association of ctxIdx and syntax elements for each slice type in the initialisation process

	Syntax element	Table	Slice type			
			SI	I	P, SP	B
slice_data( )	mb_skip_flag	Table 9-13 Table 9-14			11..13	24..26
	mb_field_decoding_flag	Table 9-18	70..72	70..72	70..72	70..72
macroblock_layer( )	mb_type	Table 9-12 Table 9-13 Table 9-14	0..10	3..10	14..20	27..35
	transform_size_8x8_flag	Table 9-16	na	399..401	399..401	399..401
	coded_block_pattern (luma)	Table 9-18	73..76	73..76	73..76	73..76
	coded_block_pattern (chroma)	Table 9-18	77..84	77..84	77..84	77..84
	mb_qp_delta	Table 9-17	60..63	60..63	60..63	60..63
mb_pred( )	prev_intra4x4_pred_mode_flag	Table 9-17	68	68	68	68
	rem_intra4x4_pred_mode	Table 9-17	69	69	69	69
	prev_intra8x8_pred_mode_flag	Table 9-17	na	68	68	68
	rem_intra8x8_pred_mode	Table 9-17	na	69	69	69
	intra_chroma_pred_mode	Table 9-17	64..67	64..67	64..67	64..67
mb_pred( ) and sub_mb_pred( )	ref_idx_l0	Table 9-16			54..59	54..59
	ref_idx_l1	Table 9-16				54..59
	mvd_l0[ ][ ][0]	Table 9-15			40..46	40..46
	mvd_l1[ ][ ][0]	Table 9-15				40..46
	mvd_l0[ ][ ][1]	Table 9-15			47..53	47..53
	mvd_l1[ ][ ][1]	Table 9-15				47..53
sub_mb_pred( )	sub_mb_type[ ]	Table 9-13 Table 9-14			21..23	36..39

Table 9-11 – Association of ctxIdx and syntax elements for each slice type in the initialisation process

	Syntax element	Table	Slice type			
			SI	I	P, SP	B
residual_block_cabac( )	coded_block_flag	Table 9-18 Table 9-25 Table 9-33	85..104 460..483	85..104 460..483 1012..1023	85..104 460..483 1012..1023	85..104 460..483 1012..1023
	significant_coeff_flag[ ]	Table 9-19 Table 9-22 Table 9-24 Table 9-24 Table 9-26 Table 9-30 Table 9-28 Table 9-29	105..165 277..337	105..165 277..337 402..416 436..450 484..571 776..863 660..689 718..747	105..165 277..337 402..416 436..450 484..571 776..863 660..689 718..747	105..165 277..337 402..416 436..450 484..571 776..863 660..689 718..747
	last_significant_coeff_flag[ ]	Table 9-20 Table 9-23 Table 9-24 Table 9-24 Table 9-27 Table 9-31 Table 9-28 Table 9-29	166..226 338..398	166..226 338..398 417..425 451..459 572..659 864..951 690..707 748..765	166..226 338..398 417..425 451..459 572..659 864..951 690..707 748..765	166..226 338..398 417..425 451..459 572..659 864..951 690..707 748..765
	coeff_abs_level_minus1[ ]	Table 9-21 Table 9-24 Table 9-32 Table 9-28 Table 9-29	227..275	227..275 426..435 952..1011 708..717 766..775	227..275 426..435 952..1011 708..717 766..775	227..275 426..435 952..1011 708..717 766..775

NOTE 2 – ctxIdx equal to 276 is associated with the end_of_slice_flag and the bin of mb_type, which specifies the I_PCM macroblock type. The decoding process specified in subclause 9.3.3.2.4 applies to ctxIdx equal to 276. This decoding process, however, may also be implemented by using the decoding process specified in subclause 9.3.3.2.1. In this case, the initial values associated with ctxIdx equal to 276 are specified to be pStateIdx = 63 and valMPS = 0, where pStateIdx = 63 represents a non-adapting probability state.

Table 9-12 – Values of variables m and n for ctxIdx from 0 to 10

Initialisation variables	ctxIdx										
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
m	20	2	3	20	2	3	−28	−23	−6	−1	7
n	−15	54	74	−15	54	74	127	104	53	54	51

Table 9-13 – Values of variables m and n for ctxIdx from 11 to 23

Value of cabac_init_idc	Initialisation variables	ctxIdx												
		11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
0	**m**	23	23	21	1	0	−37	5	−13	−11	1	12	−4	17
	n	33	2	0	9	49	118	57	78	65	62	49	73	50
1	**m**	22	34	16	−2	4	−29	2	−6	−13	5	9	−3	10
	n	25	0	0	9	41	118	65	71	79	52	50	70	54
2	**m**	29	25	14	−10	−3	−27	26	−4	−24	5	6	−17	14
	n	16	0	0	51	62	99	16	85	102	57	57	73	57

Table 9-14 – Values of variables m and n for ctxIdx from 24 to 39

Value of cabac_init_idc	Initialisation variables	ctxIdx															
		24	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**
0	**m**	18	9	29	26	16	9	−46	−20	1	−13	−11	1	−6	−17	−6	9
	n	64	43	0	67	90	104	127	104	67	78	65	62	86	95	61	45
1	**m**	26	19	40	57	41	26	−45	−15	−4	−6	−13	5	6	−13	0	8
	n	34	22	0	2	36	69	127	101	76	71	79	52	69	90	52	43
2	**m**	20	20	29	54	37	12	−32	−22	−2	−4	−24	5	−6	−14	−6	4
	n	40	10	0	0	42	97	127	117	74	85	102	57	93	88	44	55

Table 9-15 – Values of variables m and n for ctxIdx from 40 to 53

Value of cabac_init_idc	Initialisation variables	ctxIdx													
		40	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**	**51**	**52**	**53**
0	**m**	−3	−6	−11	6	7	−5	2	0	−3	−10	5	4	−3	0
	n	69	81	96	55	67	86	88	58	76	94	54	69	81	88
1	**m**	−2	−5	−10	2	2	−3	−3	1	−3	−6	0	−3	−7	−5
	n	69	82	96	59	75	87	100	56	74	85	59	81	86	95
2	**m**	−11	−15	−21	19	20	4	6	1	−5	−13	5	6	−3	−1
	n	89	103	116	57	58	84	96	63	85	106	63	75	90	101

Table 9-16 – Values of variables m and n for ctxIdx from 54 to 59, and 399 to 401

Value of cabac_init_idc	Initialisation variables	ctxIdx								
		54	**55**	**56**	**57**	**58**	**59**	**399**	**400**	**401**
I slices	**m**	na	na	na	na	na	na	31	31	25
	n	na	na	na	na	na	na	21	31	50
0	**m**	−7	−5	−4	−5	−7	1	12	11	14
	n	67	74	74	80	72	58	40	51	59
1	**m**	−1	−1	1	−2	−5	0	25	21	21
	n	66	77	70	86	72	61	32	49	54
2	**m**	3	−4	−2	−12	−7	1	21	19	17
	n	55	79	75	97	50	60	33	50	61

Table 9-17 – Values of variables m and n for ctxIdx from 60 to 69

Initialisation variables	ctxIdx									
	60	**61**	**62**	**63**	**64**	**65**	**66**	**67**	**68**	**69**
m	0	0	0	0	−9	4	0	−7	13	3
n	41	63	63	63	83	86	97	72	41	62

Table 9-18 – Values of variables m and n for ctxIdx from 70 to 104

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
70	0	11	0	45	13	15	7	34	**88**	−11	115	−13	108	−4	92	5	78
71	1	55	−4	78	7	51	−9	88	**89**	−12	63	−3	46	0	39	−6	55
72	0	69	−3	96	2	80	−20	127	**90**	−2	68	−1	65	0	65	4	61
73	−17	127	−27	126	−39	127	−36	127	**91**	−15	84	−1	57	−15	84	−14	83
74	−13	102	−28	98	−18	91	−17	91	**92**	−13	104	−9	93	−35	127	−37	127
75	0	82	−25	101	−17	96	−14	95	**93**	−3	70	−3	74	−2	73	−5	79
76	−7	74	−23	67	−26	81	−25	84	**94**	−8	93	−9	92	−12	104	−11	104
77	−21	107	−28	82	−35	98	−25	86	**95**	−10	90	−8	87	−9	91	−11	91
78	−27	127	−20	94	−24	102	−12	89	**96**	−30	127	−23	126	−31	127	−30	127
79	−31	127	−16	83	−23	97	−17	91	**97**	−1	74	5	54	3	55	0	65
80	−24	127	−22	110	−27	119	−31	127	**98**	−6	97	6	60	7	56	−2	79
81	−18	95	−21	91	−24	99	−14	76	**99**	−7	91	6	59	7	55	0	72
82	−27	127	−18	102	−21	110	−18	103	**100**	−20	127	6	69	8	61	−4	92
83	−21	114	−13	93	−18	102	−13	90	**101**	−4	56	−1	48	−3	53	−6	56
84	−30	127	−29	127	−36	127	−37	127	**102**	−5	82	0	68	0	68	3	68
85	−17	123	−7	92	0	80	11	80	**103**	−7	76	−4	69	−7	74	−8	71
86	−12	115	−5	89	−5	89	5	76	**104**	−22	125	−8	88	−9	88	−13	98
87	−16	122	−7	96	−7	94	2	84									

Table 9-19 – Values of variables m and n for ctxIdx from 105 to 165

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
105	−7	93	−2	85	−13	103	−4	86	**136**	−13	101	5	53	0	58	−5	75
106	−11	87	−6	78	−13	91	−12	88	**137**	−13	91	−2	61	−1	60	−8	80
107	−3	77	−1	75	−9	89	−5	82	**138**	−12	94	0	56	−3	61	−21	83
108	−5	71	−7	77	−14	92	−3	72	**139**	−10	88	0	56	−8	67	−21	64
109	−4	63	2	54	−8	76	−4	67	**140**	−16	84	−13	63	−25	84	−13	31
110	−4	68	5	50	−12	87	−8	72	**141**	−10	86	−5	60	−14	74	−25	64
111	−12	84	−3	68	−23	110	−16	89	**142**	−7	83	−1	62	−5	65	−29	94
112	−7	62	1	50	−24	105	−9	69	**143**	−13	87	4	57	5	52	9	75
113	−7	65	6	42	−10	78	−1	59	**144**	−19	94	−6	69	2	57	17	63
114	8	61	−4	81	−20	112	5	66	**145**	1	70	4	57	0	61	−8	74
115	5	56	1	63	−17	99	4	57	**146**	0	72	14	39	−9	69	−5	35
116	−2	66	−4	70	−78	127	−4	71	**147**	−5	74	4	51	−11	70	−2	27
117	1	64	0	67	−70	127	−2	71	**148**	18	59	13	68	18	55	13	91
118	0	61	2	57	−50	127	2	58	**149**	−8	102	3	64	−4	71	3	65
119	−2	78	−2	76	−46	127	−1	74	**150**	−15	100	1	61	0	58	−7	69
120	1	50	11	35	−4	66	−4	44	**151**	0	95	9	63	7	61	8	77
121	7	52	4	64	−5	78	−1	69	**152**	−4	75	7	50	9	41	−10	66
122	10	35	1	61	−4	71	0	62	**153**	2	72	16	39	18	25	3	62
123	0	44	11	35	−8	72	−7	51	**154**	−11	75	5	44	9	32	−3	68
124	11	38	18	25	2	59	−4	47	**155**	−3	71	4	52	5	43	−20	81
125	1	45	12	24	−1	55	−6	42	**156**	15	46	11	48	9	47	0	30
126	0	46	13	29	−7	70	−3	41	**157**	−13	69	−5	60	0	44	1	7
127	5	44	13	36	−6	75	−6	53	**158**	0	62	−1	59	0	51	−3	23
128	31	17	−10	93	−8	89	8	76	**159**	0	65	0	59	2	46	−21	74
129	1	51	−7	73	−34	119	−9	78	**160**	21	37	22	33	19	38	16	66
130	7	50	−2	73	−3	75	−11	83	**161**	−15	72	5	44	−4	66	−23	124
131	28	19	13	46	32	20	9	52	**162**	9	57	14	43	15	38	17	37
132	16	33	9	49	30	22	0	67	**163**	16	54	−1	78	12	42	44	−18
133	14	62	−7	100	−44	127	−5	90	**164**	0	62	0	60	9	34	50	−34
134	−13	108	9	53	0	54	1	67	**165**	12	72	9	69	0	89	−22	127
135	−15	100	2	53	−5	61	−15	72									

Table 9-20 – Values of variables m and n for ctxIdx from 166 to 226

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
166	24	0	11	28	4	45	4	39	**197**	26	−17	28	3	36	−28	28	−3
167	15	9	2	40	10	28	0	42	**198**	30	−25	28	4	38	−28	24	10
168	8	25	3	44	10	31	7	34	**199**	28	−20	32	0	38	−27	27	0
169	13	18	0	49	33	−11	11	29	**200**	33	−23	34	−1	34	−18	34	−14
170	15	9	0	46	52	−43	8	31	**201**	37	−27	30	6	35	−16	52	−44
171	13	19	2	44	18	15	6	37	**202**	33	−23	30	6	34	−14	39	−24
172	10	37	2	51	28	0	7	42	**203**	40	−28	32	9	32	−8	19	17
173	12	18	0	47	35	−22	3	40	**204**	38	−17	31	19	37	−6	31	25
174	6	29	4	39	38	−25	8	33	**205**	33	−11	26	27	35	0	36	29
175	20	33	2	62	34	0	13	43	**206**	40	−15	26	30	30	10	24	33
176	15	30	6	46	39	−18	13	36	**207**	41	−6	37	20	28	18	34	15
177	4	45	0	54	32	−12	4	47	**208**	38	1	28	34	26	25	30	20
178	1	58	3	54	102	−94	3	55	**209**	41	17	17	70	29	41	22	73
179	0	62	2	58	0	0	2	58	**210**	30	−6	1	67	0	75	20	34
180	7	61	4	63	56	−15	6	60	**211**	27	3	5	59	2	72	19	31
181	12	38	6	51	33	−4	8	44	**212**	26	22	9	67	8	77	27	44
182	11	45	6	57	29	10	11	44	**213**	37	−16	16	30	14	35	19	16
183	15	39	7	53	37	−5	14	42	**214**	35	−4	18	32	18	31	15	36
184	11	42	6	52	51	−29	7	48	**215**	38	−8	18	35	17	35	15	36
185	13	44	6	55	39	−9	4	56	**216**	38	−3	22	29	21	30	21	28
186	16	45	11	45	52	−34	4	52	**217**	37	3	24	31	17	45	25	21
187	12	41	14	36	69	−58	13	37	**218**	38	5	23	38	20	42	30	20
188	10	49	8	53	67	−63	9	49	**219**	42	0	18	43	18	45	31	12
189	30	34	−1	82	44	−5	19	58	**220**	35	16	20	41	27	26	27	16
190	18	42	7	55	32	7	10	48	**221**	39	22	11	63	16	54	24	42
191	10	55	−3	78	55	−29	12	45	**222**	14	48	9	59	7	66	0	93
192	17	51	15	46	32	1	0	69	**223**	27	37	9	64	16	56	14	56
193	17	46	22	31	0	0	20	33	**224**	21	60	−1	94	11	73	15	57
194	0	89	−1	84	27	36	8	63	**225**	12	68	−2	89	10	67	26	38
195	26	−19	25	7	33	−25	35	−18	**226**	2	97	−9	108	−10	116	−24	127
196	22	−17	30	−7	34	−30	33	−25									

Table 9-21 – Values of variables m and n for ctxIdx from 227 to 275

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
227	−3	71	−6	76	−23	112	−24	115	**252**	−12	73	−6	55	−16	72	−14	75
228	−6	42	−2	44	−15	71	−22	82	**253**	−8	76	0	58	−7	69	−10	79
229	−5	50	0	45	−7	61	−9	62	**254**	−7	80	0	64	−4	69	−9	83
230	−3	54	0	52	0	53	0	53	**255**	−9	88	−3	74	−5	74	−12	92
231	−2	62	−3	64	−5	66	0	59	**256**	−17	110	−10	90	−9	86	−18	108
232	0	58	−2	59	−11	77	−14	85	**257**	−11	97	0	70	2	66	−4	79
233	1	63	−4	70	−9	80	−13	89	**258**	−20	84	−4	29	−9	34	−22	69
234	−2	72	−4	75	−9	84	−13	94	**259**	−11	79	5	31	1	32	−16	75
235	−1	74	−8	82	−10	87	−11	92	**260**	−6	73	7	42	11	31	−2	58
236	−9	91	−17	102	−34	127	−29	127	**261**	−4	74	1	59	5	52	1	58
237	−5	67	−9	77	−21	101	−21	100	**262**	−13	86	−2	58	−2	55	−13	78
238	−5	27	3	24	−3	39	−14	57	**263**	−13	96	−3	72	−2	67	−9	83
239	−3	39	0	42	−5	53	−12	67	**264**	−11	97	−3	81	0	73	−4	81
240	−2	44	0	48	−7	61	−11	71	**265**	−19	117	−11	97	−8	89	−13	99
241	0	46	0	55	−11	75	−10	77	**266**	−8	78	0	58	3	52	−13	81
242	−16	64	−6	59	−15	77	−21	85	**267**	−5	33	8	5	7	4	−6	38
243	−8	68	−7	71	−17	91	−16	88	**268**	−4	48	10	14	10	8	−13	62
244	−10	78	−12	83	−25	107	−23	104	**269**	−2	53	14	18	17	8	−6	58
245	−6	77	−11	87	−25	111	−15	98	**270**	−3	62	13	27	16	19	−2	59
246	−10	86	−30	119	−28	122	−37	127	**271**	−13	71	2	40	3	37	−16	73
247	−12	92	1	58	−11	76	−10	82	**272**	−10	79	0	58	−1	61	−10	76
248	−15	55	−3	29	−10	44	−8	48	**273**	−12	86	−3	70	−5	73	−13	86
249	−10	60	−1	36	−10	52	−8	61	**274**	−13	90	−6	79	−1	70	−9	83
250	−6	62	1	38	−10	57	−8	66	**275**	−14	97	−8	85	−4	78	−10	87
251	−4	65	2	43	−9	58	−7	70									

Table 9-22 – Values of variables m and n for ctxIdx from 277 to 337

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
277	−6	93	−13	106	−21	126	−22	127	**308**	−16	96	−1	51	−16	77	−10	67
278	−6	84	−16	106	−23	124	−25	127	**309**	−7	88	7	49	−2	64	1	68
279	−8	79	−10	87	−20	110	−25	120	**310**	−8	85	8	52	2	61	0	77
280	0	66	−21	114	−26	126	−27	127	**311**	−7	85	9	41	−6	67	2	64
281	−1	71	−18	110	−25	124	−19	114	**312**	−9	85	6	47	−3	64	0	68
282	0	62	−14	98	−17	105	−23	117	**313**	−13	88	2	55	2	57	−5	78
283	−2	60	−22	110	−27	121	−25	118	**314**	4	66	13	41	−3	65	7	55
284	−2	59	−21	106	−27	117	−26	117	**315**	−3	77	10	44	−3	66	5	59
285	−5	75	−18	103	−17	102	−24	113	**316**	−3	76	6	50	0	62	2	65
286	−3	62	−21	107	−26	117	−28	118	**317**	−6	76	5	53	9	51	14	54
287	−4	58	−23	108	−27	116	−31	120	**318**	10	58	13	49	−1	66	15	44
288	−9	66	−26	112	−33	122	−37	124	**319**	−1	76	4	63	−2	71	5	60
289	−1	79	−10	96	−10	95	−10	94	**320**	−1	83	6	64	−2	75	2	70
290	0	71	−12	95	−14	100	−15	102	**321**	−7	99	−2	69	−1	70	−2	76
291	3	68	−5	91	−8	95	−10	99	**322**	−14	95	−2	59	−9	72	−18	86
292	10	44	−9	93	−17	111	−13	106	**323**	2	95	6	70	14	60	12	70
293	−7	62	−22	94	−28	114	−50	127	**324**	0	76	10	44	16	37	5	64
294	15	36	−5	86	−6	89	−5	92	**325**	−5	74	9	31	0	47	−12	70
295	14	40	9	67	−2	80	17	57	**326**	0	70	12	43	18	35	11	55
296	16	27	−4	80	−4	82	−5	86	**327**	−11	75	3	53	11	37	5	56
297	12	29	−10	85	−9	85	−13	94	**328**	1	68	14	34	12	41	0	69
298	1	44	−1	70	−8	81	−12	91	**329**	0	65	10	38	10	41	2	65
299	20	36	7	60	−1	72	−2	77	**330**	−14	73	−3	52	2	48	−6	74
300	18	32	9	58	5	64	0	71	**331**	3	62	13	40	12	41	5	54
301	5	42	5	61	1	67	−1	73	**332**	4	62	17	32	13	41	7	54
302	1	48	12	50	9	56	4	64	**333**	−1	68	7	44	0	59	−6	76
303	10	62	15	50	0	69	−7	81	**334**	−13	75	7	38	3	50	−11	82
304	17	46	18	49	1	69	5	64	**335**	11	55	13	50	19	40	−2	77
305	9	64	17	54	7	69	15	57	**336**	5	64	10	57	3	66	−2	77
306	−12	104	10	41	−7	69	1	67	**337**	12	70	26	43	18	50	25	42
307	−11	97	7	46	−6	67	0	68									

Table 9-23 – Values of variables m and n for ctxIdx from 338 to 398

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
338	15	6	14	11	19	−6	17	−13	**369**	32	−26	31	−4	40	−37	37	−17
339	6	19	11	14	18	−6	16	−9	**370**	37	−30	27	6	38	−30	32	1
340	7	16	9	11	14	0	17	−12	**371**	44	−32	34	8	46	−33	34	15
341	12	14	18	11	26	−12	27	−21	**372**	34	−18	30	10	42	−30	29	15
342	18	13	21	9	31	−16	37	−30	**373**	34	−15	24	22	40	−24	24	25
343	13	11	23	−2	33	−25	41	−40	**374**	40	−15	33	19	49	−29	34	22
344	13	15	32	−15	33	−22	42	−41	**375**	33	−7	22	32	38	−12	31	16
345	15	16	32	−15	37	−28	48	−47	**376**	35	−5	26	31	40	−10	35	18
346	12	23	34	−21	39	−30	39	−32	**377**	33	0	21	41	38	−3	31	28
347	13	23	39	−23	42	−30	46	−40	**378**	38	2	26	44	46	−5	33	41
348	15	20	42	−33	47	−42	52	−51	**379**	33	13	23	47	31	20	36	28
349	14	26	41	−31	45	−36	46	−41	**380**	23	35	16	65	29	30	27	47
350	14	44	46	−28	49	−34	52	−39	**381**	13	58	14	71	25	44	21	62
351	17	40	38	−12	41	−17	43	−19	**382**	29	−3	8	60	12	48	18	31
352	17	47	21	29	32	9	32	11	**383**	26	0	6	63	11	49	19	26
353	24	17	45	−24	69	−71	61	−55	**384**	22	30	17	65	26	45	36	24
354	21	21	53	−45	63	−63	56	−46	**385**	31	−7	21	24	22	22	24	23
355	25	22	48	−26	66	−64	62	−50	**386**	35	−15	23	20	23	22	27	16
356	31	27	65	−43	77	−74	81	−67	**387**	34	−3	26	23	27	21	24	30
357	22	29	43	−19	54	−39	45	−20	**388**	34	3	27	32	33	20	31	29
358	19	35	39	−10	52	−35	35	−2	**389**	36	−1	28	23	26	28	22	41
359	14	50	30	9	41	−10	28	15	**390**	34	5	28	24	30	24	22	42
360	10	57	18	26	36	0	34	1	**391**	32	11	23	40	27	34	16	60
361	7	63	20	27	40	−1	39	1	**392**	35	5	24	32	18	42	15	52
362	−2	77	0	57	30	14	30	17	**393**	34	12	28	29	25	39	14	60
363	−4	82	−14	82	28	26	20	38	**394**	39	11	23	42	18	50	3	78
364	−3	94	−5	75	23	37	18	45	**395**	30	29	19	57	12	70	−16	123
365	9	69	−19	97	12	55	15	54	**396**	34	26	22	53	21	54	21	53
366	−12	109	−35	125	11	65	0	79	**397**	29	39	22	61	14	71	22	56
367	36	−35	27	0	37	−33	36	−16	**398**	19	66	11	86	11	83	25	61
368	36	−34	28	0	39	−36	37	−14									

Table 9-24 – Values of variables m and n for ctxIdx from 402 to 459

ctxIdx	I slices		Value of cabac_init_idc						ctxIdx	I slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
402	−17	120	−4	79	−5	85	−3	78	**431**	−2	55	−12	56	−9	57	−12	59
403	−20	112	−7	71	−6	81	−8	74	**432**	0	61	−6	60	−6	63	−8	63
404	−18	114	−5	69	−10	77	−9	72	**433**	1	64	−5	62	−4	65	−9	67
405	−11	85	−9	70	−7	81	−10	72	**434**	0	68	−8	66	−4	67	−6	68
406	−15	92	−8	66	−17	80	−18	75	**435**	−9	92	−8	76	−7	82	−10	79
407	−14	89	−10	68	−18	73	−12	71	**436**	−14	106	−5	85	−3	81	−3	78
408	−26	71	−19	73	−4	74	−11	63	**437**	−13	97	−6	81	−3	76	−8	74
409	−15	81	−12	69	−10	83	−5	70	**438**	−15	90	−10	77	−7	72	−9	72
410	−14	80	−16	70	−9	71	−17	75	**439**	−12	90	−7	81	−6	78	−10	72
411	0	68	−15	67	−9	67	−14	72	**440**	−18	88	−17	80	−12	72	−18	75
412	−14	70	−20	62	−1	61	−16	67	**441**	−10	73	−18	73	−14	68	−12	71
413	−24	56	−19	70	−8	66	−8	53	**442**	−9	79	−4	74	−3	70	−11	63
414	−23	68	−16	66	−14	66	−14	59	**443**	−14	86	−10	83	−6	76	−5	70
415	−24	50	−22	65	0	59	−9	52	**444**	−10	73	−9	71	−5	66	−17	75
416	−11	74	−20	63	2	59	−11	68	**445**	−10	70	−9	67	−5	62	−14	72
417	23	−13	9	−2	17	−10	9	−2	**446**	−10	69	−1	61	0	57	−16	67
418	26	−13	26	−9	32	−13	30	−10	**447**	−5	66	−8	66	−4	61	−8	53
419	40	−15	33	−9	42	−9	31	−4	**448**	−9	64	−14	66	−9	60	−14	59
420	49	−14	39	−7	49	−5	33	−1	**449**	−5	58	0	59	1	54	−9	52
421	44	3	41	−2	53	0	33	7	**450**	2	59	2	59	2	58	−11	68
422	45	6	45	3	64	3	31	12	**451**	21	−10	21	−13	17	−10	9	−2
423	44	34	49	9	68	10	37	23	**452**	24	−11	33	−14	32	−13	30	−10
424	33	54	45	27	66	27	31	38	**453**	28	−8	39	−7	42	−9	31	−4
425	19	82	36	59	47	57	20	64	**454**	28	−1	46	−2	49	−5	33	−1
426	−3	75	−6	66	−5	71	−9	71	**455**	29	3	51	2	53	0	33	7
427	−1	23	−7	35	0	24	−7	37	**456**	29	9	60	6	64	3	31	12
428	1	34	−7	42	−1	36	−8	44	**457**	35	20	61	17	68	10	37	23
429	1	43	−8	45	−2	42	−11	49	**458**	29	36	55	34	66	27	31	38
430	0	54	−5	48	−2	52	−10	56	**459**	14	67	42	62	47	57	20	64

Table 9-25 – Values of variables m and n for ctxIdx from 460 to 483

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
460	−17	123	−7	92	0	80	11	80	**472**	−17	123	−7	92	0	80	11	80
461	−12	115	−5	89	−5	89	5	76	**473**	−12	115	−5	89	−5	89	5	76
462	−16	122	−7	96	−7	94	2	84	**474**	−16	122	−7	96	−7	94	2	84
463	−11	115	−13	108	−4	92	5	78	**475**	−11	115	−13	108	−4	92	5	78
464	−12	63	−3	46	0	39	−6	55	**476**	−12	63	−3	46	0	39	−6	55
465	−2	68	−1	65	0	65	4	61	**477**	−2	68	−1	65	0	65	4	61
466	−15	84	−1	57	−15	84	−14	83	**478**	−15	84	−1	57	−15	84	−14	83
467	−13	104	−9	93	−35	127	−37	127	**479**	−13	104	−9	93	−35	127	−37	127
468	−3	70	−3	74	−2	73	−5	79	**480**	−3	70	−3	74	−2	73	−5	79
469	−8	93	−9	92	−12	104	−11	104	**481**	−8	93	−9	92	−12	104	−11	104
470	−10	90	−8	87	−9	91	−11	91	**482**	−10	90	−8	87	−9	91	−11	91
471	−30	127	−23	126	−31	127	−30	127	**483**	−30	127	−23	126	−31	127	−30	127

Table 9-26 – Values of variables m and n for ctxIdx from 484 to 571

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
484	−7	93	−2	85	−13	103	−4	86	**528**	−7	93	−2	85	−13	103	−4	86
485	−11	87	−6	78	−13	91	−12	88	**529**	−11	87	−6	78	−13	91	−12	88
486	−3	77	−1	75	−9	89	−5	82	**530**	−3	77	−1	75	−9	89	−5	82
487	−5	71	−7	77	−14	92	−3	72	**531**	−5	71	−7	77	−14	92	−3	72
488	−4	63	2	54	−8	76	−4	67	**532**	−4	63	2	54	−8	76	−4	67
489	−4	68	5	50	−12	87	−8	72	**533**	−4	68	5	50	−12	87	−8	72
490	−12	84	−3	68	−23	110	−16	89	**534**	−12	84	−3	68	−23	110	−16	89
491	−7	62	1	50	−24	105	−9	69	**535**	−7	62	1	50	−24	105	−9	69
492	−7	65	6	42	−10	78	−1	59	**536**	−7	65	6	42	−10	78	−1	59
493	8	61	−4	81	−20	112	5	66	**537**	8	61	−4	81	−20	112	5	66
494	5	56	1	63	−17	99	4	57	**538**	5	56	1	63	−17	99	4	57
495	−2	66	−4	70	−78	127	−4	71	**539**	−2	66	−4	70	−78	127	−4	71
496	1	64	0	67	−70	127	−2	71	**540**	1	64	0	67	−70	127	−2	71
497	0	61	2	57	−50	127	2	58	**641**	0	61	2	57	−50	127	2	58
498	−2	78	−2	76	−46	127	−1	74	**542**	−2	78	−2	76	−46	127	−1	74

Table 9-26 – Values of variables m and n for ctxIdx from 484 to 571

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
499	1	50	11	35	−4	66	−4	44	**543**	1	50	11	35	−4	66	−4	44
500	7	52	4	64	−5	78	−1	69	**544**	7	52	4	64	−5	78	−1	69
501	10	35	1	61	−4	71	0	62	**545**	10	35	1	61	−4	71	0	62
502	0	44	11	35	−8	72	−7	51	**546**	0	44	11	35	−8	72	−7	51
503	11	38	18	25	2	59	−4	47	**547**	11	38	18	25	2	59	−4	47
504	1	45	12	24	−1	55	−6	42	**548**	1	45	12	24	−1	55	−6	42
505	0	46	13	29	−7	70	−3	41	**549**	0	46	13	29	−7	70	−3	41
506	5	44	13	36	−6	75	−6	53	**550**	5	44	13	36	−6	75	−6	53
507	31	17	−10	93	−8	89	8	76	**551**	31	17	−10	93	−8	89	8	76
508	1	51	−7	73	−34	119	−9	78	**552**	1	51	−7	73	−34	119	−9	78
509	7	50	−2	73	−3	75	−11	83	**553**	7	50	−2	73	−3	75	−11	83
510	28	19	13	46	32	20	9	52	**554**	28	19	13	46	32	20	9	52
511	16	33	9	49	30	22	0	67	**555**	16	33	9	49	30	22	0	67
512	14	62	−7	100	−44	127	−5	90	**556**	14	62	−7	100	−44	127	−5	90
513	−13	108	9	53	0	54	1	67	**557**	−13	108	9	53	0	54	1	67
514	−15	100	2	53	−5	61	−15	72	**558**	−15	100	2	53	−5	61	−15	72
515	−13	101	5	53	0	58	−5	75	**559**	−13	101	5	53	0	58	−5	75
516	−13	91	−2	61	−1	60	−8	80	**560**	−13	91	−2	61	−1	60	−8	80
517	−12	94	0	56	−3	61	−21	83	**561**	−12	94	0	56	−3	61	−21	83
518	−10	88	0	56	−8	67	−21	64	**562**	−10	88	0	56	−8	67	−21	64
519	−16	84	−13	63	−25	84	−13	31	**563**	−16	84	−13	63	−25	84	−13	31
520	−10	86	−5	60	−14	74	−25	64	**564**	−10	86	−5	60	−14	74	−25	64
521	−7	83	−1	62	−5	65	−29	94	**565**	−7	83	−1	62	−5	65	−29	94
522	−13	87	4	57	5	52	9	75	**566**	−13	87	4	57	5	52	9	75
523	−19	94	−6	69	2	57	17	63	**567**	−19	94	−6	69	2	57	17	63
524	1	70	4	57	0	61	−8	74	**568**	1	70	4	57	0	61	−8	74
525	0	72	14	39	−9	69	−5	35	**569**	0	72	14	39	−9	69	−5	35
526	−5	74	4	51	−11	70	−2	27	**570**	−5	74	4	51	−11	70	−2	27
527	18	59	13	68	18	55	13	91	**571**	18	59	13	68	18	55	13	91

Table 9-27 – Values of variables m and n for ctxIdx from 572 to 659

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
572	24	0	11	28	4	45	4	39	**616**	24	0	11	28	4	45	4	39
573	15	9	2	40	10	28	0	42	**617**	15	9	2	40	10	28	0	42
574	8	25	3	44	10	31	7	34	**618**	8	25	3	44	10	31	7	34
575	13	18	0	49	33	−11	11	29	**619**	13	18	0	49	33	−11	11	29
576	15	9	0	46	52	−43	8	31	**620**	15	9	0	46	52	−43	8	31
577	13	19	2	44	18	15	6	37	**621**	13	19	2	44	18	15	6	37
578	10	37	2	51	28	0	7	42	**622**	10	37	2	51	28	0	7	42
579	12	18	0	47	35	−22	3	40	**623**	12	18	0	47	35	−22	3	40
580	6	29	4	39	38	−25	8	33	**624**	6	29	4	39	38	−25	8	33
581	20	33	2	62	34	0	13	43	**625**	20	33	2	62	34	0	13	43
582	15	30	6	46	39	−18	13	36	**626**	15	30	6	46	39	−18	13	36
583	4	45	0	54	32	−12	4	47	**627**	4	45	0	54	32	−12	4	47
584	1	58	3	54	102	−94	3	55	**628**	1	58	3	54	102	−94	3	55
585	0	62	2	58	0	0	2	58	**629**	0	62	2	58	0	0	2	58
586	7	61	4	63	56	−15	6	60	**630**	7	61	4	63	56	−15	6	60
587	12	38	6	51	33	−4	8	44	**631**	12	38	6	51	33	−4	8	44
588	11	45	6	57	29	10	11	44	**632**	11	45	6	57	29	10	11	44
589	15	39	7	53	37	−5	14	42	**633**	15	39	7	53	37	−5	14	42
590	11	42	6	52	51	−29	7	48	**634**	11	42	6	52	51	−29	7	48
591	13	44	6	55	39	−9	4	56	**635**	13	44	6	55	39	−9	4	56
592	16	45	11	45	52	−34	4	52	**636**	16	45	11	45	52	−34	4	52
593	12	41	14	36	69	−58	13	37	**637**	12	41	14	36	69	−58	13	37
594	10	49	8	53	67	−63	9	49	**638**	10	49	8	53	67	−63	9	49
595	30	34	−1	82	44	−5	19	58	**639**	30	34	−1	82	44	−5	19	58
596	18	42	7	55	32	7	10	48	**640**	18	42	7	55	32	7	10	48
597	10	55	−3	78	55	−29	12	45	**641**	10	55	−3	78	55	−29	12	45
598	17	51	15	46	32	1	0	69	**642**	17	51	15	46	32	1	0	69
599	17	46	22	31	0	0	20	33	**643**	17	46	22	31	0	0	20	33
600	0	89	−1	84	27	36	8	63	**644**	0	89	−1	84	27	36	8	63
601	26	−19	25	7	33	−25	35	−18	**645**	26	−19	25	7	33	−25	35	−18
602	22	−17	30	−7	34	−30	33	−25	**646**	22	−17	30	−7	34	−30	33	−25

Table 9-27 – Values of variables m and n for ctxIdx from 572 to 659

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
603	26	−17	28	3	36	−28	28	−3	**647**	26	−17	28	3	36	−28	28	−3
604	30	−25	28	4	38	−28	24	10	**648**	30	−25	28	4	38	−28	24	10
605	28	−20	32	0	38	−27	27	0	**649**	28	−20	32	0	38	−27	27	0
606	33	−23	34	−1	34	−18	34	−14	**650**	33	−23	34	−1	34	−18	34	−14
607	37	−27	30	6	35	−16	52	−44	**651**	37	−27	30	6	35	−16	52	−44
608	33	−23	30	6	34	−14	39	−24	**652**	33	−23	30	6	34	−14	39	−24
609	40	−28	32	9	32	−8	19	17	**653**	40	−28	32	9	32	−8	19	17
610	38	−17	31	19	37	−6	31	25	**654**	38	−17	31	19	37	−6	31	25
611	33	−11	26	27	35	0	36	29	**655**	33	−11	26	27	35	0	36	29
612	40	−15	26	30	30	10	24	33	**656**	40	−15	26	30	30	10	24	33
613	41	−6	37	20	28	18	34	15	**657**	41	−6	37	20	28	18	34	15
614	38	1	28	34	26	25	30	20	**658**	38	1	28	34	26	25	30	20
615	41	17	17	70	29	41	22	73	**659**	41	17	17	70	29	41	22	73

Table 9-28 – Values of variables m and n for ctxIdx from 660 to 717

ctxIdx	I slices		Value of cabac_init_idc						ctxIdx	I slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
660	−17	120	−4	79	−5	85	−3	78	**689**	2	59	2	59	2	58	−11	68
661	−20	112	−7	71	−6	81	−8	74	**690**	23	−13	9	−2	17	−10	9	−2
662	−18	114	−5	69	−10	77	−9	72	**691**	26	−13	26	−9	32	−13	30	−10
663	−11	85	−9	70	−7	81	−10	72	**692**	40	−15	33	−9	42	−9	31	−4
664	−15	92	−8	66	−17	80	−18	75	**693**	49	−14	39	−7	49	−5	33	−1
665	−14	89	−10	68	−18	73	−12	71	**694**	44	3	41	−2	53	0	33	7
666	−26	71	−19	73	−4	74	−11	63	**695**	45	6	45	3	64	3	31	12
667	−15	81	−12	69	−10	83	−5	70	**696**	44	34	49	9	68	10	37	23
668	−14	80	−16	70	−9	71	−17	75	**697**	33	54	45	27	66	27	31	38
669	0	68	−15	67	−9	67	−14	72	**698**	19	82	36	59	47	57	20	64
670	−14	70	−20	62	−1	61	−16	67	**699**	21	−10	21	−13	17	−10	9	−2
671	−24	56	−19	70	−8	66	−8	53	**700**	24	−11	33	−14	32	−13	30	−10
672	−23	68	−16	66	−14	66	−14	59	**701**	28	−8	39	−7	42	−9	31	−4
673	−24	50	−22	65	0	59	−9	52	**702**	28	−1	46	−2	49	−5	33	−1
674	−11	74	−20	63	2	59	−11	68	**703**	29	3	51	2	53	0	33	7
675	−14	106	−5	85	−3	81	−3	78	**704**	29	9	60	6	64	3	31	12
676	−13	97	−6	81	−3	76	−8	74	**705**	35	20	61	17	68	10	37	23
677	−15	90	−10	77	−7	72	−9	72	**706**	29	36	55	34	66	27	31	38
678	−12	90	−7	81	−6	78	−10	72	**707**	14	67	42	62	47	57	20	64
679	−18	88	-17	80	−12	72	−18	75	**708**	−3	75	−6	66	−5	71	−9	71
680	−10	73	−18	73	−14	68	−12	71	**709**	−1	23	−7	35	0	24	−7	37
681	−9	79	−4	74	−3	70	−11	63	**710**	1	34	−7	42	−1	36	−8	44
682	−14	86	−10	83	−6	76	−5	70	**711**	1	43	−8	45	−2	42	−11	49
683	−10	73	−9	71	−5	66	−17	75	**712**	0	54	−5	48	−2	52	−10	56
684	−10	70	−9	67	−5	62	−14	72	**713**	−2	55	−12	56	−9	57	−12	59
685	−10	69	−1	61	0	57	−16	67	**714**	0	61	−6	60	−6	63	−8	63
686	−5	66	−8	66	−4	61	−8	53	**715**	1	64	−5	62	−4	65	−9	67
687	−9	64	−14	66	−9	60	−14	59	**716**	0	68	−8	66	−4	67	−6	68
688	−5	58	0	59	1	54	−9	52	**717**	−9	92	−8	76	−7	82	−10	79

Table 9-29 – Values of variables m and n for ctxIdx from 718 to 775

ctxIdx	I slices		Value of cabac_init_idc						ctxIdx	I slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
718	−17	120	−4	79	−5	85	−3	78	**747**	2	59	2	59	2	58	−11	68
719	−20	112	−7	71	−6	81	−8	74	**748**	23	−13	9	−2	17	−10	9	−2
720	−18	114	−5	69	−10	77	−9	72	**749**	26	−13	26	−9	32	−13	30	−10
721	−11	85	−9	70	−7	81	−10	72	**750**	40	−15	33	−9	42	−9	31	−4
722	−15	92	−8	66	−17	80	−18	75	**751**	49	−14	39	−7	49	−5	33	−1
723	−14	89	−10	68	−18	73	−12	71	**752**	44	3	41	−2	53	0	33	7
724	−26	71	−19	73	−4	74	−11	63	**753**	45	6	45	3	64	3	31	12
725	−15	81	−12	69	−10	83	−5	70	**754**	44	34	49	9	68	10	37	23
726	−14	80	−16	70	−9	71	−17	75	**755**	33	54	45	27	66	27	31	38
727	0	68	−15	67	−9	67	−14	72	**756**	19	82	36	59	47	57	20	64
728	−14	70	−20	62	−1	61	−16	67	**757**	21	−10	21	−13	17	−10	9	−2
729	−24	56	−19	70	−8	66	−8	53	**758**	24	−11	33	−14	32	−13	30	−10
730	−23	68	−16	66	−14	66	−14	59	**759**	28	−8	39	−7	42	−9	31	−4
731	−24	50	−22	65	0	59	−9	52	**760**	28	−1	46	−2	49	−5	33	−1
732	−11	74	−20	63	2	59	−11	68	**761**	29	3	51	2	53	0	33	7
733	−14	106	−5	85	−3	81	−3	78	**762**	29	9	60	6	64	3	31	12
734	−13	97	−6	81	−3	76	−8	74	**763**	35	20	61	17	68	10	37	23
735	−15	90	−10	77	−7	72	−9	72	**764**	29	36	55	34	66	27	31	38
736	−12	90	−7	81	−6	78	−10	72	**765**	14	67	42	62	47	57	20	64
737	−18	88	−17	80	−12	72	−18	75	**766**	−3	75	−6	66	−5	71	−9	71
738	−10	73	−18	73	−14	68	−12	71	**767**	−1	23	−7	35	0	24	−7	37
739	−9	79	−4	74	−3	70	−11	63	**768**	1	34	−7	42	−1	36	−8	44
740	−14	86	−10	83	−6	76	−5	70	**769**	1	43	−8	45	−2	42	−11	49
741	−10	73	−9	71	−5	66	−17	75	**770**	0	54	−5	48	−2	52	−10	56
742	−10	70	−9	67	−5	62	−14	72	**771**	−2	55	−12	56	−9	57	−12	59
743	−10	69	−1	61	0	57	−16	67	**772**	0	61	−6	60	−6	63	−8	63
744	−5	66	−8	66	−4	61	−8	53	**773**	1	64	−5	62	−4	65	−9	67
745	−9	64	−14	66	−9	60	−14	59	**774**	0	68	−8	66	−4	67	−6	68
746	−5	58	0	59	1	54	−9	52	**775**	−9	92	−8	76	−7	82	−10	79

Table 9-30 – Values of variables m and n for ctxIdx from 776 to 863

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
776	−6	93	−13	106	−21	126	−22	127	**820**	−6	93	−13	106	−21	126	−22	127
777	−6	84	−16	106	−23	124	−25	127	**821**	−6	84	−16	106	−23	124	−25	127
778	−8	79	−10	87	−20	110	−25	120	**822**	−8	79	−10	87	−20	110	−25	120
779	0	66	−21	114	−26	126	−27	127	**823**	0	66	−21	114	−26	126	−27	127
780	−1	71	−18	110	−25	124	−19	114	**824**	−1	71	−18	110	−25	124	−19	114
781	0	62	−14	98	−17	105	−23	117	**825**	0	62	−14	98	−17	105	−23	117
782	−2	60	−22	110	−27	121	−25	118	**826**	−2	60	−22	110	−27	121	−25	118
783	−2	59	−21	106	−27	117	−26	117	**827**	−2	59	−21	106	−27	117	−26	117
784	−5	75	−18	103	−17	102	−24	113	**828**	−5	75	−18	103	−17	102	−24	113
785	−3	62	−21	107	−26	117	−28	118	**829**	−3	62	−21	107	−26	117	−28	118
786	−4	58	−23	108	−27	116	−31	120	**830**	−4	58	−23	108	−27	116	−31	120
787	−9	66	−26	112	−33	122	−37	124	**831**	−9	66	−26	112	−33	122	−37	124
788	−1	79	−10	96	−10	95	−10	94	**832**	−1	79	−10	96	−10	95	−10	94
789	0	71	−12	95	−14	100	−15	102	**833**	0	71	−12	95	−14	100	−15	102
790	3	68	−5	91	−8	95	−10	99	**834**	3	68	−5	91	−8	95	−10	99
791	10	44	−9	93	−17	111	−13	106	**835**	10	44	−9	93	−17	111	−13	106
792	−7	62	−22	94	−28	114	−50	127	**836**	−7	62	−22	94	−28	114	−50	127
793	15	36	−5	86	−6	89	−5	92	**837**	15	36	−5	86	−6	89	−5	92
794	14	40	9	67	−2	80	17	57	**838**	14	40	9	67	−2	80	17	57
795	16	27	−4	80	−4	82	−5	86	**839**	16	27	−4	80	−4	82	−5	86
796	12	29	−10	85	−9	85	−13	94	**840**	12	29	−10	85	−9	85	−13	94
797	1	44	−1	70	−8	81	−12	91	**841**	1	44	−1	70	−8	81	−12	91
798	20	36	7	60	−1	72	−2	77	**842**	20	36	7	60	−1	72	−2	77
799	18	32	9	58	5	64	0	71	**843**	18	32	9	58	5	64	0	71
800	5	42	5	61	1	67	−1	73	**844**	5	42	5	61	1	67	−1	73
801	1	48	12	50	9	56	4	64	**845**	1	48	12	50	9	56	4	64
802	10	62	15	50	0	69	−7	81	**846**	10	62	15	50	0	69	−7	81
803	17	46	18	49	1	69	5	64	**847**	17	46	18	49	1	69	5	64
804	9	64	17	54	7	69	15	57	**848**	9	64	17	54	7	69	15	57
805	−12	104	10	41	−7	69	1	67	**849**	−12	104	10	41	−7	69	1	67
806	−11	97	7	46	−6	67	0	68	**850**	−11	97	7	46	−6	67	0	68

Table 9-30 – Values of variables m and n for ctxIdx from 776 to 863

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
807	−16	96	−1	51	−16	77	−10	67	**851**	−16	96	−1	51	−16	77	−10	67
808	−7	88	7	49	−2	64	1	68	**852**	−7	88	7	49	−2	64	1	68
809	−8	85	8	52	2	61	0	77	**853**	−8	85	8	52	2	61	0	77
810	−7	85	9	41	−6	67	2	64	**854**	−7	85	9	41	−6	67	2	64
811	−9	85	6	47	−3	64	0	68	**855**	−9	85	6	47	−3	64	0	68
812	−13	88	2	55	2	57	−5	78	**856**	−13	88	2	55	2	57	−5	78
813	4	66	13	41	−3	65	7	55	**857**	4	66	13	41	−3	65	7	55
814	−3	77	10	44	−3	66	5	59	**858**	−3	77	10	44	−3	66	5	59
815	−3	76	6	50	0	62	2	65	**859**	−3	76	6	50	0	62	2	65
816	−6	76	5	53	9	51	14	54	**860**	−6	76	5	53	9	51	14	54
817	10	58	13	49	−1	66	15	44	**861**	10	58	13	49	−1	66	15	44
818	−1	76	4	63	−2	71	5	60	**862**	−1	76	4	63	−2	71	5	60
819	−1	83	6	64	−2	75	2	70	**863**	−1	83	6	64	−2	75	2	70

Table 9-31 – Values of variables m and n for ctxIdx from 864 to 951

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
864	15	6	14	11	19	−6	17	−13	**908**	15	6	14	11	19	−6	17	−13
865	6	19	11	14	18	−6	16	−9	**909**	6	19	11	14	18	−6	16	−9
866	7	16	9	11	14	0	17	−12	**910**	7	16	9	11	14	0	17	−12
867	12	14	18	11	26	−12	27	−21	**911**	12	14	18	11	26	−12	27	−21
868	18	13	21	9	31	−16	37	−30	**912**	18	13	21	9	31	−16	37	−30
869	13	11	23	−2	33	−25	41	−40	**913**	13	11	23	−2	33	−25	41	−40
870	13	15	32	−15	33	−22	42	−41	**914**	13	15	32	−15	33	−22	42	−41
871	15	16	32	−15	37	−28	48	−47	**915**	15	16	32	−15	37	−28	48	−47
872	12	23	34	−21	39	−30	39	−32	**916**	12	23	34	−21	39	−30	39	−32
873	13	23	39	−23	42	−30	46	−40	**917**	13	23	39	−23	42	−30	46	−40
874	15	20	42	−33	47	−42	52	−51	**918**	15	20	42	−33	47	−42	52	−51
875	14	26	41	−31	45	−36	46	−41	**919**	14	26	41	−31	45	−36	46	−41
876	14	44	46	−28	49	−34	52	−39	**920**	14	44	46	−28	49	−34	52	−39
877	17	40	38	−12	41	−17	43	−19	**921**	17	40	38	−12	41	−17	43	−19
878	17	47	21	29	32	9	32	11	**922**	17	47	21	29	32	9	32	11
879	24	17	45	−24	69	−71	61	−55	**923**	24	17	45	−24	69	−71	61	−55
880	21	21	53	−45	63	−63	56	−46	**924**	21	21	53	−45	63	−63	56	−46
881	25	22	48	−26	66	−64	62	−50	**925**	25	22	48	−26	66	−64	62	−50
882	31	27	65	−43	77	−74	81	−67	**926**	31	27	65	−43	77	−74	81	−67
883	22	29	43	−19	54	−39	45	−20	**927**	22	29	43	−19	54	−39	45	−20
884	19	35	39	−10	52	−35	35	−2	**928**	19	35	39	−10	52	−35	35	−2
885	14	50	30	9	41	−10	28	15	**929**	14	50	30	9	41	−10	28	15
886	10	57	18	26	36	0	34	1	**930**	10	57	18	26	36	0	34	1
887	7	63	20	27	40	−1	39	1	**931**	7	63	20	27	40	−1	39	1
888	−2	77	0	57	30	14	30	17	**932**	−2	77	0	57	30	14	30	17
889	−4	82	−14	82	28	26	20	38	**933**	−4	82	−14	82	28	26	20	38
890	−3	94	−5	75	23	37	18	45	**934**	−3	94	−5	75	23	37	18	45
891	9	69	−19	97	12	55	15	54	**935**	9	69	−19	97	12	55	15	54
892	−12	109	−35	125	11	65	0	79	**936**	−12	109	−35	125	11	65	0	79
893	36	−35	27	0	37	−33	36	−16	**937**	36	−35	27	0	37	−33	36	−16
894	36	−34	28	0	39	−36	37	−14	**938**	36	−34	28	0	39	−36	37	−14

Table 9-31 – Values of variables m and n for ctxIdx from 864 to 951

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
895	32	−26	31	−4	40	−37	37	−17	**939**	32	−26	31	−4	40	−37	37	−17
896	37	−30	27	6	38	−30	32	1	**940**	37	−30	27	6	38	−30	32	1
897	44	−32	34	8	46	−33	34	15	**941**	44	−32	34	8	46	−33	34	15
898	34	−18	30	10	42	−30	29	15	**942**	34	−18	30	10	42	−30	29	15
899	34	−15	24	22	40	−24	24	25	**943**	34	−15	24	22	40	−24	24	25
900	40	−15	33	19	49	−29	34	22	**944**	40	−15	33	19	49	−29	34	22
901	33	−7	22	32	38	−12	31	16	**945**	33	−7	22	32	38	−12	31	16
902	35	−5	26	31	40	−10	35	18	**946**	35	−5	26	31	40	−10	35	18
903	33	0	21	41	38	−3	31	28	**947**	33	0	21	41	38	−3	31	28
904	38	2	26	44	46	−5	33	41	**948**	38	2	26	44	46	−5	33	41
905	33	13	23	47	31	20	36	28	**949**	33	13	23	47	31	20	36	28
906	23	35	16	65	29	30	27	47	**950**	23	35	16	65	29	30	27	47
907	13	58	14	71	25	44	21	62	**951**	13	58	14	71	25	44	21	62

Table 9-32 – Values of variables m and n for ctxIdx from 952 to 1011

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
952	−3	71	−6	76	−23	112	−24	115	**982**	−3	71	−6	76	−23	112	−24	115
953	−6	42	−2	44	−15	71	−22	82	**983**	−6	42	−2	44	−15	71	−22	82
954	−5	50	0	45	−7	61	−9	62	**984**	−5	50	0	45	−7	61	−9	62
955	−3	54	0	52	0	53	0	53	**985**	−3	54	0	52	0	53	0	53
956	−2	62	−3	64	−5	66	0	59	**986**	−2	62	−3	64	−5	66	0	59
957	0	58	−2	59	−11	77	−14	85	**987**	0	58	−2	59	−11	77	−14	85
958	1	63	−4	70	−9	80	−13	89	**988**	1	63	−4	70	−9	80	−13	89
959	−2	72	−4	75	−9	84	−13	94	**989**	−2	72	−4	75	−9	84	−13	94
960	−1	74	−8	82	−10	87	−11	92	**990**	−1	74	−8	82	−10	87	−11	92
961	−9	91	−17	102	−34	127	−29	127	**991**	−9	91	−17	102	−34	127	−29	127
962	−5	67	−9	77	−21	101	−21	100	**992**	−5	67	−9	77	−21	101	−21	100
963	−5	27	3	24	−3	39	−14	57	**993**	−5	27	3	24	−3	39	−14	57
964	−3	39	0	42	−5	53	−12	67	**994**	−3	39	0	42	−5	53	−12	67
965	−2	44	0	48	−7	61	−11	71	**995**	−2	44	0	48	−7	61	−11	71
966	0	46	0	55	−11	75	−10	77	**996**	0	46	0	55	−11	75	−10	77
967	−16	64	−6	59	−15	77	−21	85	**997**	−16	64	−6	59	−15	77	−21	85
968	−8	68	−7	71	−17	91	−16	88	**998**	−8	68	−7	71	−17	91	−16	88
969	−10	78	−12	83	−25	107	−23	104	**999**	−10	78	−12	83	−25	107	−23	104
970	−6	77	−11	87	−25	111	−15	98	**1000**	−6	77	−11	87	−25	111	−15	98
971	−10	86	−30	119	−28	122	−37	127	**1001**	−10	86	−30	119	−28	122	−37	127
972	−12	92	1	58	−11	76	−10	82	**1002**	−12	92	1	58	−11	76	−10	82
973	−15	55	−3	29	−10	44	−8	48	**1003**	−15	55	−3	29	−10	44	−8	48
974	−10	60	−1	36	−10	52	−8	61	**1004**	−10	60	−1	36	−10	52	−8	61
975	−6	62	1	38	−10	57	−8	66	**1005**	−6	62	1	38	−10	57	−8	66
976	−4	65	2	43	−9	58	−7	70	**1006**	−4	65	2	43	−9	58	−7	70
977	−12	73	−6	55	−16	72	−14	75	**1007**	−12	73	−6	55	−16	72	−14	75
978	−8	76	0	58	−7	69	−10	79	**1008**	−8	76	0	58	−7	69	−10	79
979	−7	80	0	64	−4	69	−9	83	**1009**	−7	80	0	64	−4	69	−9	83
980	−9	88	−3	74	−5	74	−12	92	**1010**	−9	88	−3	74	−5	74	−12	92
981	−17	110	−10	90	−9	86	−18	108	**1011**	−17	110	−10	90	−9	86	−18	108

Table 9-33 – Values of variables m and n for ctxIdx from 1012 to 1023

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	N	m	n
1012	−3	70	−3	74	−2	73	−5	79	**1018**	−10	90	−8	87	−9	91	−11	91
1013	−8	93	−9	92	−12	104	−11	104	**1019**	−30	127	−23	126	−31	127	−30	127
1014	−10	90	−8	87	−9	91	−11	91	**1020**	−3	70	−3	74	−2	73	−5	79
1015	−30	127	−23	126	−31	127	−30	127	**1021**	−8	93	−9	92	−12	104	−11	104
1016	−3	70	−3	74	−2	73	−5	79	**1022**	−10	90	−8	87	−9	91	−11	91
1017	−8	93	−9	92	−12	104	−11	104	**1023**	−30	127	−23	126	−31	127	−30	127

9.3.1.2 Initialisation process for the arithmetic decoding engine

This process is invoked before decoding the first macroblock of a slice or after the decoding of any pcm_alignment_zero_bit and all pcm_sample_luma and pcm_sample_chroma data for a macroblock of type I_PCM.

Outputs of this process are the initialised decoding engine registers codIRange and codIOffset both in 16 bit register precision.

The status of the arithmetic decoding engine is represented by the variables codIRange and codIOffset. In the initialisation procedure of the arithmetic decoding process, codIRange is set equal to 510 and codIOffset is set equal to the value returned from read_bits(9) interpreted as a 9 bit binary representation of an unsigned integer with most significant bit written first.

The bitstream shall not contain data that result in a value of codIOffset being equal to 510 or 511.

NOTE – The description of the arithmetic decoding engine in this Recommendation | International Standard utilizes 16 bit register precision. However, a minimum register precision of 9 bits is required for storing the values of the variables codIRange and codIOffset after invocation of the arithmetic decoding process (DecodeBin) as specified in subclause 9.3.3.2. The arithmetic decoding process for a binary decision (DecodeDecision) as specified in subclause 9.3.3.2.1 and the decoding process for a binary decision before termination (DecodeTerminate) as specified in subclause 9.3.3.2.4 require a minimum register precision of 9 bits for the variables codIRange and codIOffset. The bypass decoding process for binary decisions (DecodeBypass) as specified in subclause 9.3.3.2.3 requires a minimum register precision of 10 bits for the variable codIOffset and a minimum register precision of 9 bits for the variable codIRange.

9.3.2 Binarization process

Input to this process is a request for a syntax element.

Output of this process is the binarization of the syntax element, maxBinIdxCtx, ctxIdxOffset, and bypassFlag.

Table 9-34 specifies the type of binarization process, maxBinIdxCtx, and ctxIdxOffset associated with each syntax element.

The specification of the unary (U) binarization process, the truncated unary (TU) binarization process, the concatenated unary / k-th order Exp-Golomb (UEGk) binarization process, and the fixed-length (FL) binarization process are given in subclauses 9.3.2.1 to 9.3.2.4, respectively. Other binarizations are specified in subclauses 9.3.2.5 to 9.3.2.7.

Except for I slices, the binarizations for the syntax element mb_type as specified in subclause 9.3.2.5 consist of bin strings given by a concatenation of prefix and suffix bit strings. The UEGk binarization as specified in subclause 9.3.2.3, which is used for the binarization of the syntax elements mvd_lX (X = 0, 1) and coeff_abs_level_minus1, and the binarization of the coded_block_pattern also consist of a concatenation of prefix and suffix bit strings. For these binarization processes, the prefix and the suffix bit string are separately indexed using the binIdx variable as specified further in subclause 9.3.3. The two sets of prefix bit strings and suffix bit strings are referred to as the binarization prefix part and the binarization suffix part, respectively.

Associated with each binarization or binarization part of a syntax element is a specific value of the context index offset (ctxIdxOffset) variable and a specific value of the maxBinIdxCtx variable as given in Table 9-34. When two values for each of these variables are specified for one syntax element in Table 9-34, the value in the upper row is related to the prefix part while the value in the lower row is related to the suffix part of the binarization of the corresponding syntax element.

The use of the DecodeBypass process and the variable bypassFlag is derived as follows:

– If no value is assigned to ctxIdxOffset for the corresponding binarization or binarization part in Table 9-34 labelled as "na", all bins of the bit strings of the corresponding binarization or of the binarization prefix/suffix part are decoded by invoking the DecodeBypass process as specified in subclause 9.3.3.2.3. In such a case, bypassFlag is set equal to 1, where bypassFlag is used to indicate that for parsing the value of the bin from the bitstream the DecodeBypass process is applied.

– Otherwise, for each possible value of binIdx up to the specified value of maxBinIdxCtx given in Table 9-34, a specific value of the variable ctxIdx is further specified in subclause 9.3.3. bypassFlag is set equal to 0.

The possible values of the context index ctxIdx are in the range 0 to 1023, inclusive. The value assigned to ctxIdxOffset specifies the lower value of the range of ctxIdx assigned to the corresponding binarization or binarization part of a syntax element.

ctxIdx = ctxIdxOffset = 276 is assigned to the syntax element end_of_slice_flag and the bin of mb_type, which specifies the I_PCM macroblock type as further specified in subclause 9.3.3.1. For parsing the value of the corresponding bin from the bitstream, the arithmetic decoding process for decisions before termination (DecodeTerminate) as specified in subclause 9.3.3.2.4 is applied.

NOTE – The bins of mb_type in I slices and the bins of the suffix for mb_type in SI slices that correspond to the same value of binIdx share the same ctxIdx. The last bin of the prefix of mb_type and the first bin of the suffix of mb_type in P, SP, and B slices may share the same ctxIdx.

Table 9-34 – Syntax elements and associated types of binarization, maxBinIdxCtx, and ctxIdxOffset

Syntax element	Type of binarization	maxBinIdxCtx	ctxIdxOffset
mb_type (SI slices only)	prefix and suffix as specified in subclause 9.3.2.5	prefix: 0 suffix: 6	prefix: 0 suffix: 3
mb_type (I slices only)	as specified in subclause 9.3.2.5	6	3
mb_skip_flag (P, SP slices only)	FL, cMax=1	0	11
mb_type (P, SP slices only)	prefix and suffix as specified in subclause 9.3.2.5	prefix: 2 suffix: 5	prefix: 14 suffix: 17
sub_mb_type[ ] (P, SP slices only)	as specified in subclause 9.3.2.5	2	21
mb_skip_flag (B slices only)	FL, cMax=1	0	24
mb_type (B slices only)	prefix and suffix as specified in subclause 9.3.2.5	prefix: 3 suffix: 5	prefix: 27 suffix: 32
sub_mb_type[ ] (B slices only)	as specified in subclause 9.3.2.5	3	36
mvd_l0[ ][ ][0], mvd_l1[ ][ ][0]	prefix and suffix as given by UEG3 with signedValFlag=1, uCoff=9	prefix: 4 suffix: na	prefix: 40 suffix: na (uses DecodeBypass)
mvd_l0[ ][ ][1], mvd_l1[ ][ ][1]		prefix: 4 suffix: na	prefix: 47 suffix: na (uses DecodeBypass)
ref_idx_l0, ref_idx_l1	U	2	54
mb_qp_delta	as specified in subclause 9.3.2.7	2	60
intra_chroma_pred_mode	TU, cMax=3	1	64
prev_intra4x4_pred_mode_flag, prev_intra8x8_pred_mode_flag	FL, cMax=1	0	68
rem_intra4x4_pred_mode, rem_intra8x8_pred_mode	FL, cMax=7	0	69
mb_field_decoding_flag	FL, cMax=1	0	70

Table 9-34 – Syntax elements and associated types of binarization, maxBinIdxCtx, and ctxIdxOffset

Syntax element	Type of binarization	maxBinIdxCtx	ctxIdxOffset
coded_block_pattern	prefix and suffix as specified in subclause 9.3.2.6	prefix: 3 suffix: 1	prefix: 73 suffix: 77
coded_block_flag (blocks with ctxBlockCat < 5)	FL, cMax=1	0	85
significant_coeff_flag (frame coded blocks with ctxBlockCat < 5)	FL, cMax=1	0	105
last_significant_coeff_flag (frame coded blocks with ctxBlockCat < 5)	FL, cMax=1	0	166
coeff_abs_level_minus1 (blocks with ctxBlockCat < 5)	prefix and suffix as given by UEG0 with signedValFlag=0, uCoff=14	prefix: 1 suffix: na	prefix: 227 suffix: na, (uses DecodeBypass)
coeff_sign_flag	FL, cMax=1	0	na, (uses DecodeBypass)
end_of_slice_flag	FL, cMax=1	0	276
significant_coeff_flag (field coded blocks with ctxBlockCat < 5)	FL, cMax=1	0	277
last_significant_coeff_flag (field coded blocks with ctxBlockCat < 5)	FL, cMax=1	0	338
transform_size_8x8_flag	FL, cMax=1	0	399
significant_coeff_flag (frame coded blocks with ctxBlockCat = = 5)	FL, cMax=1	0	402
last_significant_coeff_flag (frame coded blocks with ctxBlockCat = = 5)	FL, cMax=1	0	417
coeff_abs_level_minus1 (blocks with ctxBlockCat = = 5)	prefix and suffix as given by UEG0 with signedValFlag=0, uCoff=14	prefix: 1 suffix: na	prefix: 426 suffix: na, (uses DecodeBypass)
significant_coeff_flag (field coded blocks with ctxBlockCat = = 5)	FL, cMax=1	0	436
last_significant_coeff_flag (field coded blocks with ctxBlockCat = = 5)	FL, cMax=1	0	451
coded_block_flag (5 < ctxBlockCat < 9)	FL, cMax=1	0	460
coded_block_flag (9 < ctxBlockCat < 13)	FL, cMax=1	0	472
coded_block_flag (ctxBlockCat = = 5, 9, or 13)	FL, cMax=1	0	1012
significant_coeff_flag (frame coded blocks with 5 < ctxBlockCat < 9)	FL, cMax=1	0	484
significant_coeff_flag (frame coded blocks with 9 < ctxBlockCat < 13)	FL, cMax=1	0	528
last_significant_coeff_flag (frame coded blocks with 5 < ctxBlockCat < 9)	FL, cMax=1	0	572
last_significant_coeff_flag	FL, cMax=1	0	616

Table 9-34 – Syntax elements and associated types of binarization, maxBinIdxCtx, and ctxIdxOffset

Syntax element	Type of binarization	maxBinIdxCtx	ctxIdxOffset
(frame coded blocks with 9 < ctxBlockCat < 13)			
coeff_abs_level_minus1 (blocks with 5 < ctxBlockCat < 9)	prefix and suffix as given by UEG0 with signedValFlag=0, uCoff=14	prefix: 1 suffix: na	prefix: 952 suffix: na, (uses DecodeBypass)
coeff_abs_level_minus1 (blocks with 9 < ctxBlockCat < 13)	prefix and suffix as given by UEG0 with signedValFlag=0, uCoff=14	prefix: 1 suffix: na	prefix: 982 suffix: na, (uses DecodeBypass)
significant_coeff_flag (field coded blocks with 5 < ctxBlockCat < 9)	FL, cMax=1	0	776
significant_coeff_flag (field coded blocks with 9 < ctxBlockCat < 13)	FL, cMax=1	0	820
last_significant_coeff_flag (field coded blocks with 5 < ctxBlockCat < 9)	FL, cMax=1	0	864
last_significant_coeff_flag (field coded blocks with 9 < ctxBlockCat < 13)	FL, cMax=1	0	908
significant_coeff_flag (frame coded blocks with ctxBlockCat = = 9)	FL, cMax=1	0	660
significant_coeff_flag (frame coded blocks with ctxBlockCat = = 13)	FL, cMax=1	0	718
last_significant_coeff_flag (frame coded blocks with ctxBlockCat = = 9)	FL, cMax=1	0	690
last_significant_coeff_flag (frame coded blocks with ctxBlockCat = = 13)	FL, cMax=1	0	748
coeff_abs_level_minus1 (blocks with ctxBlockCat = = 9)	prefix and suffix as given by UEG0 with signedValFlag=0, uCoff=14	prefix: 1 suffix: na	prefix: 708 suffix: na, (uses DecodeBypass)
coeff_abs_level_minus1 (blocks with ctxBlockCat = = 13)	prefix and suffix as given by UEG0 with signedValFlag=0, uCoff=14	prefix: 1 suffix: na	prefix: 766 suffix: na, (uses DecodeBypass)
significant_coeff_flag (field coded blocks with ctxBlockCat = = 9)	FL, cMax=1	0	675
significant_coeff_flag (field coded blocks with ctxBlockCat = = 13)	FL, cMax=1	0	733
last_significant_coeff_flag (field coded blocks with ctxBlockCat = = 9)	FL, cMax=1	0	699
last_significant_coeff_flag (field coded blocks with ctxBlockCat = = 13)	FL, cMax=1	0	757

9.3.2.1 Unary (U) binarization process

Input to this process is a request for a U binarization for a syntax element.

Output of this process is the U binarization of the syntax element.

The bin string of a syntax element having (unsigned integer) value synElVal is a bit string of length synElVal + 1 indexed by binIdx. The bins for binIdx less than synElVal are equal to 1. The bin with binIdx equal to synElVal is equal to 0.

Table 9-35 illustrates the bin strings of the unary binarization for a syntax element.

Table 9-35 – Bin string of the unary binarization (informative)

Value of syntax element	Bin string					
0 (I_NxN)	0					
1	1	0				
2	1	1	0			
3	1	1	1	0		
4	1	1	1	1	0	
5	1	1	1	1	1	0
…						
binIdx	0	1	2	3	4	5

9.3.2.2 Truncated unary (TU) binarization process

Input to this process is a request for a TU binarization for a syntax element and cMax.

Output of this process is the TU binarization of the syntax element.

For syntax element (unsigned integer) values less than cMax, the U binarization process as specified in subclause 9.3.2.1 is invoked. For the syntax element value equal to cMax the bin string is a bit string of length cMax with all bins being equal to 1.

NOTE – TU binarization is always invoked with a cMax value equal to the largest possible value of the syntax element being decoded.

9.3.2.3 Concatenated unary/ k-th order Exp-Golomb (UEGk) binarization process

Input to this process is a request for a UEGk binarization for a syntax element, signedValFlag and uCoff.

Output of this process is the UEGk binarization of the syntax element.

A UEGk bin string is a concatenation of a prefix bit string and a suffix bit string. The prefix of the binarization is specified by invoking the TU binarization process for the prefix part Min(uCoff, Abs(synElVal)) of a syntax element value synElVal as specified in subclause 9.3.2.2 with cMax = uCoff, where uCoff > 0.

The variable k for a UEGk bin string is dependent on the syntax element for which a UEGk binarization is requested. Table 9-34 specifies the associated types of binarization for syntax elements, including the value of k for syntax elements that use UEGk binarization.

NOTE 1 – For the syntax elements mvd_l0[ ][ ][ ] and mvd_l1[ ][ ][ ] a UEG3 binarization is used (k is equal to 3). For the syntax element coeff_abs_level_minus1 a UEG0 binarization is used (k is equal to 0).

The UEGk bin string is derived as follows:

– If one of the following is true, the bin string of a syntax element having value synElVal consists only of a prefix bit string:

  – signedValFlag is equal to 0 and the prefix bit string is not equal to the bit string of length uCoff with all bits equal to 1,

  – signedValFlag is equal to 1 and the prefix bit string is equal to the bit string that consists of a single bit with value equal to 0.

– Otherwise, the bin string of the UEGk suffix part of a syntax element value synElVal is specified by a process equivalent to the following pseudo-code with k being initialised to the value that is specified in Table 9-34 for the requested UEGk binarization process. At the beginning of the following pseudo-code, the bin string of a syntax element having value synElVal is set equal to the empty string. Each call of the function put(X), with X being equal to 0 or 1, adds the binary value X at the end of the bin string.

```
if( Abs( synElVal ) >= uCoff ) {
    sufS = Abs( synElVal ) − uCoff
    stopLoop = 0
    do {
        if( sufS >= ( 1 << k ) ) {
            put( 1 )
            sufS = sufS − ( 1<<k )
            k++
        } else {
            put( 0 )                                        (9-6)
            while( k− − )
                put( ( sufS >> k ) & 1 )
            stopLoop = 1
        }
    } while( !stopLoop )
}
if( signedValFlag && synElVal ! = 0)
    if( synElVal > 0 )
        put( 0 )
    else
        put( 1 )
```

NOTE 2 – The specification for the k-th order Exp-Golomb (EGk) code uses 1's and 0's in reverse meaning for the unary part of the Exp-Golomb code of 0-th order as specified in subclause 9.1.

9.3.2.4 Fixed-length (FL) binarization process

Input to this process is a request for a FL binarization for a syntax element and cMax.

Output of this process is the FL binarization of the syntax element.

FL binarization is constructed by using a fixedLength-bit unsigned integer bin string of the syntax element value, where fixedLength = Ceil(Log2(cMax + 1)). The indexing of bins for the FL binarization is such that the binIdx = 0 relates to the least significant bit with increasing values of binIdx towards the most significant bit.

9.3.2.5 Binarization process for macroblock type and sub-macroblock type

Input to this process is a request for a binarization for syntax elements mb_type or sub_mb_type[ ].

Output of this process is the binarization of the syntax element.

The binarization scheme for decoding of macroblock type in I slices is specified in Table 9-36.

For macroblock types in SI slices, the binarization consists of bin strings specified as a concatenation of a prefix and a suffix bit string as follows.

The prefix bit string consists of a single bit, which is specified by b_0 = ((mb_type = = SI) ? 0 : 1). For the syntax element value for which b_0 is equal to 0, the bin string only consists of the prefix bit string. For the syntax element value for which b_0 is equal to 1, the binarization is given by concatenating the prefix b_0 and the suffix bit string as specified in Table 9-36 for macroblock type in I slices indexed by subtracting 1 from the value of mb_type in SI slices.

Table 9-36 – Binarization for macroblock types in I slices

Value (name) of mb_type	Bin string						
0 (I_NxN)	0						
1 (I_16x16_0_0_0)	1	0	0	0	0	0	
2 (I_16x16_1_0_0)	1	0	0	0	0	1	
3 (I_16x16_2_0_0)	1	0	0	0	1	0	
4 (I_16x16_3_0_0)	1	0	0	0	1	1	
5 (I_16x16_0_1_0)	1	0	0	1	0	0	0
6 (I_16x16_1_1_0)	1	0	0	1	0	0	1
7 (I_16x16_2_1_0)	1	0	0	1	0	1	0
8 (I_16x16_3_1_0)	1	0	0	1	0	1	1
9 (I_16x16_0_2_0)	1	0	0	1	1	0	0
10 (I_16x16_1_2_0)	1	0	0	1	1	0	1
11 (I_16x16_2_2_0)	1	0	0	1	1	1	0
12 (I_16x16_3_2_0)	1	0	0	1	1	1	1
13 (I_16x16_0_0_1)	1	0	1	0	0	0	
14 (I_16x16_1_0_1)	1	0	1	0	0	1	
15 (I_16x16_2_0_1)	1	0	1	0	1	0	
16 (I_16x16_3_0_1)	1	0	1	0	1	1	
17 (I_16x16_0_1_1)	1	0	1	1	0	0	0
18 (I_16x16_1_1_1)	1	0	1	1	0	0	1
19 (I_16x16_2_1_1)	1	0	1	1	0	1	0
20 (I_16x16_3_1_1)	1	0	1	1	0	1	1
21 (I_16x16_0_2_1)	1	0	1	1	1	0	0
22 (I_16x16_1_2_1)	1	0	1	1	1	0	1
23 (I_16x16_2_2_1)	1	0	1	1	1	1	0
24 (I_16x16_3_2_1)	1	0	1	1	1	1	1
25 (I_PCM)	1	1					
binIdx	0	1	2	3	4	5	6

The binarization schemes for P macroblock types in P and SP slices and for B macroblocks in B slices are specified in Table 9-37.

The bin string for I macroblock types in P and SP slices corresponding to mb_type values 5 to 30 consists of a concatenation of a prefix, which consists of a single bit with value equal to 1 as specified in Table 9-37 and a suffix as specified in Table 9-36, indexed by subtracting 5 from the value of mb_type.

mb_type equal to 4 (P_8x8ref0) is not allowed.

For I macroblock types in B slices (mb_type values 23 to 48) the binarization consists of bin strings specified as a concatenation of a prefix bit string as specified in Table 9-37 and suffix bit strings as specified in Table 9-36, indexed by subtracting 23 from the value of mb_type.

Table 9-37 – Binarization for macroblock types in P, SP, and B slices

Slice type	Value (name) of mb_type	Bin string						
P, SP slice	0 (P_L0_16x16)	0	0	0				
	1 (P_L0_L0_16x8)	0	1	1				
	2 (P_L0_L0_8x16)	0	1	0				
	3 (P_8x8)	0	0	1				
	4 (P_8x8ref0)	na						
	5 to 30 (Intra, prefix only)	1						
B slice	0 (B_Direct_16x16)	0						
	1 (B_L0_16x16)	1	0	0				
	2 (B_L1_16x16)	1	0	1				
	3 (B_Bi_16x16)	1	1	0	0	0	0	
	4 (B_L0_L0_16x8)	1	1	0	0	0	1	
	5 (B_L0_L0_8x16)	1	1	0	0	1	0	
	6 (B_L1_L1_16x8)	1	1	0	0	1	1	
	7 (B_L1_L1_8x16)	1	1	0	1	0	0	
	8 (B_L0_L1_16x8)	1	1	0	1	0	1	
	9 (B_L0_L1_8x16)	1	1	0	1	1	0	
	10 (B_L1_L0_16x8)	1	1	0	1	1	1	
	11 (B_L1_L0_8x16)	1	1	1	1	1	0	
	12 (B_L0_Bi_16x8)	1	1	1	0	0	0	0
	13 (B_L0_Bi_8x16)	1	1	1	0	0	0	1
	14 (B_L1_Bi_16x8)	1	1	1	0	0	1	0
	15 (B_L1_Bi_8x16)	1	1	1	0	0	1	1
	16 (B_Bi_L0_16x8)	1	1	1	0	1	0	0
	17 (B_Bi_L0_8x16)	1	1	1	0	1	0	1
	18 (B_Bi_L1_16x8)	1	1	1	0	1	1	0
	19 (B_Bi_L1_8x16)	1	1	1	0	1	1	1
	20 (B_Bi_Bi_16x8)	1	1	1	1	0	0	0
	21 (B_Bi_Bi_8x16)	1	1	1	1	0	0	1
	22 (B_8x8)	1	1	1	1	1	1	
	23 to 48 (Intra, prefix only)	1	1	1	1	0	1	
binIdx		0	1	2	3	4	5	6

For P, SP, and B slices the specification of the binarization for sub_mb_type[ ] is given in Table 9-38.

Table 9-38 – Binarization for sub-macroblock types in P, SP, and B slices

Slice type	Value (name) of sub_mb_type[ ]	Bin string					
P, SP slice	0 (P_L0_8x8)	1					
	1 (P_L0_8x4)	0	0				
	2 (P_L0_4x8)	0	1	1			
	3 (P_L0_4x4)	0	1	0			
B slice	0 (B_Direct_8x8)	0					
	1 (B_L0_8x8)	1	0	0			
	2 (B_L1_8x8)	1	0	1			
	3 (B_Bi_8x8)	1	1	0	0	0	
	4 (B_L0_8x4)	1	1	0	0	1	
	5 (B_L0_4x8)	1	1	0	1	0	
	6 (B_L1_8x4)	1	1	0	1	1	
	7 (B_L1_4x8)	1	1	1	0	0	0
	8 (B_Bi_8x4)	1	1	1	0	0	1
	9 (B_Bi_4x8)	1	1	1	0	1	0
	10 (B_L0_4x4)	1	1	1	0	1	1
	11 (B_L1_4x4)	1	1	1	1	0	
	12 (B_Bi_4x4)	1	1	1	1	1	
binIdx		0	1	2	3	4	5

9.3.2.6 Binarization process for coded block pattern

Input to this process is a request for a binarization for the syntax element coded_block_pattern.

Output of this process is the binarization of the syntax element.

The binarization of coded_block_pattern consists of a prefix part and (when present) a suffix part. The prefix part of the binarization is given by the FL binarization of CodedBlockPatternLuma with cMax = 15. When ChromaArrayType is not equal to 0 or 3, the suffix part is present and consists of the TU binarization of CodedBlockPatternChroma with cMax = 2. The relationship between the value of the syntax element coded_block_pattern and the values of CodedBlockPatternLuma and CodedBlockPatternChroma is given as specified in subclause 7.4.5.

9.3.2.7 Binarization process for mb_qp_delta

Input to this process is a request for a binarization for the syntax element mb_qp_delta.

Output of this process is the binarization of the syntax element.

The bin string of mb_qp_delta is derived by the U binarization of the mapped value of the syntax element mb_qp_delta, where the assignment rule between the signed value of mb_qp_delta and its mapped value is given as specified in Table 9-3.

9.3.3 Decoding process flow

Input to this process is a binarization of the requested syntax element, maxBinIdxCtx, bypassFlag and ctxIdxOffset as specified in subclause 9.3.2.

Output of this process is the value of the syntax element.

This process specifies how each bit of a bit string is parsed for each syntax element.

After parsing each bit, the resulting bit string is compared to all bin strings of the binarization of the syntax element and the following applies:

– If the bit string is equal to one of the bin strings, the corresponding value of the syntax element is the output.

– Otherwise (the bit string is not equal to one of the bin strings), the next bit is parsed.

While parsing each bin, the variable binIdx is incremented by 1 starting with binIdx being set equal to 0 for the first bin.

When the binarization of the corresponding syntax element consists of a prefix and a suffix binarization part,, the variable binIdx is set equal to 0 for the first bin of each part of the bin string (prefix part or suffix part). In this case, after parsing the prefix bit string, the parsing process of the suffix bit string related to the binarizations specified in subclauses 9.3.2.3 and 9.3.2.5 is invoked depending on the resulting prefix bit string as specified in subclauses 9.3.2.3 and 9.3.2.5. Note that for the binarization of the syntax element coded_block_pattern, the suffix bit string is present regardless of the prefix bit string of length 4 as specified in subclause 9.3.2.6.

Depending on the variable bypassFlag, the following applies:

– If bypassFlag is equal to 1, the bypass decoding process as specified in subclause 9.3.3.2.3 is applied for parsing the value of the bins from the bitstream.

– Otherwise (bypassFlag is equal to 0), the parsing of each bin is specified by the following two ordered steps:

  1. Given binIdx, maxBinIdxCtx and ctxIdxOffset, ctxIdx is derived as specified in subclause 9.3.3.1.

  2. Given ctxIdx, the value of the bin from the bitstream as specified in subclause 9.3.3.2 is decoded.

9.3.3.1 Derivation process for ctxIdx

Inputs to this process are binIdx, maxBinIdxCtx and ctxIdxOffset.

Output of this process is ctxIdx.

Table 9-39 shows the assignment of ctxIdx increments (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.

The ctxIdx to be used with a specific binIdx is specified by first determining the ctxIdxOffset associated with the given bin string or part thereof. The ctxIdx is determined as follows:

– If the ctxIdxOffset is listed in Table 9-39, the ctxIdx for a binIdx is the sum of ctxIdxOffset and ctxIdxInc, which is found in Table 9-39. When more than one value is listed in Table 9-39 for a binIdx, the assignment process for ctxIdxInc for that binIdx is further specified in the subclauses given in parenthesis of the corresponding table entry.

– Otherwise (ctxIdxOffset is not listed in Table 9-39), the ctxIdx is specified to be the sum of the following terms: ctxIdxOffset and ctxIdxBlockCatOffset(ctxBlockCat) as specified in Table 9-40 and ctxIdxInc(ctxBlockCat). Subclause 9.3.3.1.3 specifies which ctxBlockCat is used. Subclause 9.3.3.1.1.9 specifies the assignment of ctxIdxInc(ctxBlockCat) for coded_block_flag, and subclause 9.3.3.1.3 specifies the assignment of ctxIdxInc(ctxBlockCat) for significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1.

All bins with binIdx greater than maxBinIdxCtx are parsed using the value of ctxIdx being assigned to binIdx equal to maxBinIdxCtx.

All entries in Table 9-39 labelled with "na" correspond to values of binIdx that do not occur for the corresponding ctxIdxOffset.

ctxIdx = 276 is assigned to the binIdx of mb_type indicating the I_PCM mode. For parsing the value of the corresponding bins from the bitstream, the arithmetic decoding process for decisions before termination as specified in subclause 9.3.3.2.4 is applied.

Table 9-39 – 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

ctxIdxOffset	binIdx						
	0	**1**	**2**	**3**	**4**	**5**	**>= 6**
0	0,1,2 (subclause 9.3.3.1.1.3)	na	na	na	na	na	na
3	0,1,2 (subclause 9.3.3.1.1.3)	ctxIdx=276	3	4	5,6 (subclause 9.3.3.1.2)	6,7 (subclause 9.3.3.1.2)	7
11	0,1,2 (subclause 9.3.3.1.1.1)	na	na	na	na	na	na
14	0	1	2,3 (subclause 9.3.3.1.2)	na	na	na	na
17	0	ctxIdx=276	1	2	2,3 (subclause 9.3.3.1.2)	3	3
21	0	1	2	na	na	na	na
24	0,1,2 (subclause 9.3.3.1.1.1)	na	na	na	na	na	na
27	0,1,2 (subclause 9.3.3.1.1.3)	3	4,5 (subclause 9.3.3.1.2)	5	5	5	5
32	0	ctxIdx=276	1	2	2,3 (subclause 9.3.3.1.2)	3	3
36	0	1	2,3 (subclause 9.3.3.1.2)	3	3	3	na
40	0,1,2 (subclause 9.3.3.1.1.7)	3	4	5	6	6	6
47	0,1,2 (subclause 9.3.3.1.1.7)	3	4	5	6	6	6
54	0,1,2,3 (subclause 9.3.3.1.1.6)	4	5	5	5	5	5
60	0,1 (subclause 9.3.3.1.1.5)	2	3	3	3	3	3
64	0,1,2 (subclause 9.3.3.1.1.8)	3	3	na	na	na	na
68	0	na	na	na	na	na	na
69	0	0	0	na	na	na	na
70	0,1,2 (subclause 9.3.3.1.1.2)	na	na	na	na	na	na
73	0,1,2,3 (subclause 9.3.3.1.1.4)	0,1,2,3 (subclause 9.3.3.1.1.4)	0,1,2,3 (subclause 9.3.3.1.1.4)	0,1,2,3 (subclause 9.3.3.1.1.4)	na	na	na
77	0,1,2,3 (subclause 9.3.3.1.1.4)	4,5,6,7 (subclause 9.3.3.1.1.4)	na	na	na	na	na
276	0	na	na	na	na	na	na
399	0,1,2 (subclause 9.3.3.1.1.10)	na	na	na	na	na	na

Table 9-40 shows the values of ctxIdxBlockCatOffset depending on ctxBlockCat for the syntax elements coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1. The specification of ctxBlockCat is given in Table 9-42.

Table 9-40 – Assignment of ctxIdxBlockCatOffset to ctxBlockCat for syntax elements coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1

Syntax element	ctxBlockCat (as specified in Table 9-42)													
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
coded_block_flag	0	4	8	12	16	0	0	4	8	4	0	4	8	8
significant_coeff_flag	0	15	29	44	47	0	0	15	29	0	0	15	29	0
last_significant_coeff_flag	0	15	29	44	47	0	0	15	29	0	0	15	29	0
coeff_abs_level_minus1	0	10	20	30	39	0	0	10	20	0	0	10	20	0

9.3.3.1.1 Assignment process of ctxIdxInc using neighbouring syntax elements

Subclause 9.3.3.1.1.1 specifies the derivation process of ctxIdxInc for the syntax element mb_skip_flag.

Subclause 9.3.3.1.1.2 specifies the derivation process of ctxIdxInc for the syntax element mb_field_decoding_flag.

Subclause 9.3.3.1.1.3 specifies the derivation process of ctxIdxInc for the syntax element mb_type.

Subclause 9.3.3.1.1.4 specifies the derivation process of ctxIdxInc for the syntax element coded_block_pattern.

Subclause 9.3.3.1.1.5 specifies the derivation process of ctxIdxInc for the syntax element mb_qp_delta.

Subclause 9.3.3.1.1.6 specifies the derivation process of ctxIdxInc for the syntax elements ref_idx_l0 and ref_idx_l1.

Subclause 9.3.3.1.1.7 specifies the derivation process of ctxIdxInc for the syntax elements mvd_l0 and mvd_l1.

Subclause 9.3.3.1.1.8 specifies the derivation process of ctxIdxInc for the syntax element intra_chroma_pred_mode.

Subclause 9.3.3.1.1.9 specifies the derivation process of ctxIdxInc for the syntax element coded_block_flag.

Subclause 9.3.3.1.1.10 specifies the derivation process of ctxIdxInc for the syntax element transform_size_8x8_flag.

9.3.3.1.1.1 Derivation process of ctxIdxInc for the syntax element mb_skip_flag

Output of this process is ctxIdxInc.

When MbaffFrameFlag is equal to 1 and mb_field_decoding_flag has not been decoded (yet) for the current macroblock pair with top macroblock address 2 * (CurrMbAddr / 2), the inference rule for the syntax element mb_field_decoding_flag as specified in subclause 7.4.4 is applied.

The derivation process for neighbouring macroblocks specified in subclause 6.4.11.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

Let the variable condTermFlagN (with N being either A or B) be derived as follows:

– If mbAddrN is not available or mb_skip_flag for the macroblock mbAddrN is equal to 1, condTermFlagN is set equal to 0.

– Otherwise (mbAddrN is available and mb_skip_flag for the macroblock mbAddrN is equal to 0), condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived by:

$$\text{ctxIdxInc} = \text{condTermFlagA} + \text{condTermFlagB} \qquad (9\text{-}7)$$

9.3.3.1.1.2 Derivation process of ctxIdxInc for the syntax element mb_field_decoding_flag

Output of this process is ctxIdxInc.

The derivation process for neighbouring macroblock addresses and their availability in MBAFF frames as specified in subclause 6.4.10 is invoked and the output is assigned to mbAddrA and mbAddrB.

When both macroblocks mbAddrN and mbAddrN + 1 have mb_type equal to P_Skip or B_Skip, the inference rule for the syntax element mb_field_decoding_flag as specified in subclause 7.4.4 is applied for the macroblock mbAddrN.

Let the variable condTermFlagN (with N being either A or B) be derived as follows:

– If any of the following conditions are true, condTermFlagN is set equal to 0:
  – mbAddrN is not available,
  – the macroblock mbAddrN is a frame macroblock.
– Otherwise, condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived by

$$\text{ctxIdxInc} = \text{condTermFlagA} + \text{condTermFlagB} \quad (9\text{-}8)$$

9.3.3.1.1.3 Derivation process of ctxIdxInc for the syntax element mb_type

Input to this process is ctxIdxOffset.

Output of this process is ctxIdxInc.

The derivation process for neighbouring macroblocks specified in subclause 6.4.11.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

Let the variable condTermFlagN (with N being either A or B) be derived as follows:

– If any of the following conditions are true, condTermFlagN is set equal to 0:
  – mbAddrN is not available,
  – ctxIdxOffset is equal to 0 and mb_type for the macroblock mbAddrN is equal to SI,
  – ctxIdxOffset is equal to 3 and mb_type for the macroblock mbAddrN is equal to I_NxN,
  – ctxIdxOffset is equal to 27 and mb_type for the macroblock mbAddrN is equal to B_Skip or B_Direct_16x16.
– Otherwise, condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived as

$$\text{ctxIdxInc} = \text{condTermFlagA} + \text{condTermFlagB} \quad (9\text{-}9)$$

9.3.3.1.1.4 Derivation process of ctxIdxInc for the syntax element coded_block_pattern

Inputs to this process are ctxIdxOffset and binIdx.

Output of this process is ctxIdxInc.

Depending on the value of the variable ctxIdxOffset, the following ordered steps are specified:

– If ctxIdxOffset is equal to 73, the following applies:
  1. The derivation process for neighbouring 8x8 luma blocks specified in subclause 6.4.11.2 is invoked with luma8x8BlkIdx = binIdx as input and the output is assigned to mbAddrA, mbAddrB, luma8x8BlkIdxA, and luma8x8BlkIdxB.
  2. Let the variable condTermFlagN (with N being either A or B) be derived as follows:
     – If any of the following conditions are true, condTermFlagN is set equal to 0:
       – mbAddrN is not available,
       – mb_type for the macroblock mbAddrN is equal to I_PCM,
       – the macroblock mbAddrN is not the current macroblock CurrMbAddr and the macroblock mbAddrN does not have mb_type equal to P_Skip or B_Skip, and ((CodedBlockPatternLuma >> luma8x8BlkIdxN) & 1) is not equal to 0 for the value of CodedBlockPatternLuma for the macroblock mbAddrN,

– the macroblock mbAddrN is the current macroblock CurrMbAddr and the prior decoded bin value b_k of coded_block_pattern with k = luma8x8BlkIdxN is not equal to 0.

– Otherwise, condTermFlagN is set equal to 1.

3. The variable ctxIdxInc is derived as

$$ctxIdxInc = condTermFlagA + 2 * condTermFlagB \tag{9-10}$$

– Otherwise (ctxIdxOffset is equal to 77), the following ordered steps are specified:

1. The derivation process for neighbouring macroblocks specified in subclause 6.4.11.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

2. Let the variable condTermFlagN (with N being either A or B) be derived as follows:

   – If mbAddrN is available and mb_type for the macroblock mbAddrN is equal to I_PCM, condTermFlagN is set equal to 1.

   – Otherwise, if any of the following conditions are true, condTermFlagN is set equal to 0:

     – mbAddrN is not available or the macroblock mbAddrN has mb_type equal to P_Skip or B_Skip,

     – binIdx is equal to 0 and CodedBlockPatternChroma for the macroblock mbAddrN is equal to 0,

     – binIdx is equal to 1 and CodedBlockPatternChroma for the macroblock mbAddrN is not equal to 2.

   – Otherwise, condTermFlagN is set equal to 1.

3. The variable ctxIdxInc is derived as

$$ctxIdxInc = condTermFlagA + 2 * condTermFlagB + ((binIdx == 1) ? 4 : 0) \tag{9-11}$$

NOTE – When a macroblock is coded in Intra_16x16 macroblock prediction mode, the values of CodedBlockPatternLuma and CodedBlockPatternChroma for the macroblock are derived from mb_type as specified in Table 7-11.

9.3.3.1.1.5 Derivation process of ctxIdxInc for the syntax element mb_qp_delta

Output of this process is ctxIdxInc.

Let prevMbAddr be the macroblock address of the macroblock that precedes the current macroblock in decoding order. When the current macroblock is the first macroblock of a slice, prevMbAddr is marked as not available.

Let the variable ctxIdxInc be derived as follows:

– If any of the following conditions are true, ctxIdxInc is set equal to 0:

  – prevMbAddr is not available or the macroblock prevMbAddr has mb_type equal to P_Skip or B_Skip,

  – mb_type of the macroblock prevMbAddr is equal to I_PCM,

  – The macroblock prevMbAddr is not coded in Intra_16x16 macroblock prediction mode and both CodedBlockPatternLuma and CodedBlockPatternChroma for the macroblock prevMbAddr are equal to 0,

  – mb_qp_delta for the macroblock prevMbAddr is equal to 0.

– Otherwise, ctxIdxInc is set equal to 1.

9.3.3.1.1.6 Derivation process of ctxIdxInc for the syntax elements ref_idx_l0 and ref_idx_l1

Input to this process is mbPartIdx.

Output of this process is ctxIdxInc.

The interpretation of ref_idx_lX and Pred_LX within this subclause is specified as follows:

– If this process is invoked for the derivation of ref_idx_l0, ref_idx_lX is interpreted as ref_idx_l0 and Pred_LX is interpreted as Pred_L0.

– Otherwise (this process is invoked for the derivation of ref_idx_l1), ref_idx_lX is interpreted as ref_idx_l1 and Pred_LX is interpreted as Pred_L1.

The derivation process for neighbouring partitions specified in subclause 6.4.11.7 is invoked with mbPartIdx, currSubMbType set equal to sub_mb_type[mbPartIdx], and subMbPartIdx = 0 as input and the output is assigned to mbAddrA\mbPartIdxA and mbAddrB\mbPartIdxB.

With ref_idx_lX[mbPartIdxN] (with N being either A or B) specifying the syntax element for the macroblock mbAddrN, let the variable refIdxZeroFlagN be derived as follows:

– If MbaffFrameFlag is equal to 1, the current macroblock is a frame macroblock, and the macroblock mbAddrN is a field macroblock,

$$\text{refIdxZeroFlagN} = ((\text{ref\_idx\_lX}[\text{mbPartIdxN}] > 1) ? 0 : 1) \quad (9\text{-}12)$$

– Otherwise,

$$\text{refIdxZeroFlagN} = ((\text{ref\_idx\_lX}[\text{mbPartIdxN}] > 0) ? 0 : 1) \quad (9\text{-}13)$$

Let the variable predModeEqualFlagN be specified as follows:

– If mb_type for the macroblock mbAddrN is equal to B_Direct_16x16 or B_Skip, predModeEqualFlagN is set equal to 0.

– Otherwise, if the macroblock mbAddrN has mb_type equal to P_8x8 or B_8x8, the following applies:

  – If SubMbPredMode(sub_mb_type[mbPartIdxN]) is not equal to Pred_LX and not equal to BiPred, predModeEqualFlagN is set equal to 0, where sub_mb_type specifies the syntax element list for the macroblock mbAddrN.

  – Otherwise, predModeEqualFlagN is set equal to 1.

– Otherwise, the following applies:

  – If MbPartPredMode(mb_type, mbPartIdxN) is not equal to Pred_LX and not equal to BiPred, predModeEqualFlagN is set equal to 0, where mb_type specifies the syntax element for the macroblock mbAddrN.

  – Otherwise, predModeEqualFlagN is set equal to 1.

Let the variable condTermFlagN (with N being either A or B) be derived as follows:

– If any of the following conditions are true, condTermFlagN is set equal to 0:

  – mbAddrN is not available,

  – the macroblock mbAddrN has mb_type equal to P_Skip or B_Skip,

  – the macroblock mbAddrN is coded in an Intra macroblock prediction mode,

  – predModeEqualFlagN is equal to 0,

  – refIdxZeroFlagN is equal to 1.

– Otherwise, condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived as

$$\text{ctxIdxInc} = \text{condTermFlagA} + 2 * \text{condTermFlagB} \quad (9\text{-}14)$$

9.3.3.1.1.7 Derivation process of ctxIdxInc for the syntax elements mvd_l0 and mvd_l1

Inputs to this process are mbPartIdx, subMbPartIdx, and ctxIdxOffset.

Output of this process is ctxIdxInc.

The interpretation of mvd_lX and Pred_LX within this subclause is specified as follows:

– If this process is invoked for the derivation of mvd_l0, mvd_lX is interpreted as mvd_l0 and Pred_LX is interpreted as Pred_L0.

– Otherwise (this process is invoked for the derivation of mvd_l1), mvd_lX is interpreted as mvd_l1 and Pred_LX is interpreted as Pred_L1.

The derivation process for neighbouring partitions specified in subclause 6.4.11.7 is invoked with mbPartIdx, currSubMbType set equal to sub_mb_type[mbPartIdx], and subMbPartIdx as input and the output is assigned to mbAddrA\mbPartIdxA\subMbPartIdxA and mbAddrB\mbPartIdxB\subMbPartIdxB.

Let the variable compIdx be derived as follows:

– If ctxIdxOffset is equal to 40, compIdx is set equal to 0.

– Otherwise (ctxIdxOffset is equal to 47), compIdx is set equal to 1.

Let the variable predModeEqualFlagN be specified as follows:

– If mb_type for the macroblock mbAddrN is equal to B_Direct_16x16 or B_Skip, predModeEqualFlagN is set equal to 0.

– Otherwise, if the macroblock mbAddrN has mb_type equal to P_8x8 or B_8x8, the following applies:

  – If SubMbPredMode(sub_mb_type[mbPartIdxN]) is not equal to Pred_LX and not equal to BiPred, predModeEqualFlagN is set equal to 0, where sub_mb_type specifies the syntax element list for the macroblock mbAddrN.

  – Otherwise, predModeEqualFlagN is set equal to 1.

– Otherwise, the following applies:

  – If MbPartPredMode(mb_type, mbPartIdxN) is not equal to Pred_LX and not equal to BiPred, predModeEqualFlagN is set equal to 0, where mb_type specifies the syntax element for the macroblock mbAddrN.

  – Otherwise, predModeEqualFlagN is set equal to 1.

Let the variable absMvdCompN (with N being either A or B) be derived as follows:

– If any of the following conditions are true, absMvdCompN is set equal to 0:

  – mbAddrN is not available,

  – the macroblock mbAddrN has mb_type equal to P_Skip or B_Skip,

  – the macroblock mbAddrN is coded in an Intra macroblock prediction mode,

  – predModeEqualFlagN is equal to 0.

– Otherwise, the following applies:

  – If compIdx is equal to 1, MbaffFrameFlag is equal to 1, the current macroblock is a frame macroblock, and the macroblock mbAddrN is a field macroblock,

$$\text{absMvdCompN} = \text{Abs}(\ \text{mvd\_lX}[\ \text{mbPartIdxN}\ ][\ \text{subMbPartIdxN}\ ][\ \text{compIdx}\ ]\ ) * 2 \qquad (9\text{-}15)$$

  – Otherwise, if compIdx is equal to 1, MbaffFrameFlag is equal to 1, the current macroblock is a field macroblock, and the macroblock mbAddrN is a frame macroblock,

$$\text{absMvdCompN} = \text{Abs}(\ \text{mvd\_lX}[\ \text{mbPartIdxN}\ ][\ \text{subMbPartIdxN}\ ][\ \text{compIdx}\ ]\ ) / 2 \qquad (9\text{-}16)$$

  – Otherwise,

$$\text{absMvdCompN} = \text{Abs}(\ \text{mvd\_lX}[\ \text{mbPartIdxN}\ ][\ \text{subMbPartIdxN}\ ][\ \text{compIdx}\ ]\ ) \qquad (9\text{-}17)$$

The variable ctxIdxInc is derived as follows:

– If (absMvdCompA + absMvdCompB) is less than 3, ctxIdxInc is set equal to 0.

– Otherwise, if (absMvdCompA + absMvdCompB) is greater than 32, ctxIdxInc is set equal to 2.

– Otherwise ((absMvdCompA + absMvdCompB) is in the range of 3 to 32, inclusive), ctxIdxInc is set equal to 1.

9.3.3.1.1.8 Derivation process of ctxIdxInc for the syntax element intra_chroma_pred_mode

Output of this process is ctxIdxInc.

The derivation process for neighbouring macroblocks specified in subclause 6.4.11.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

Let the variable condTermFlagN (with N being replaced by either A or B) be derived as follows:

– If any of the following conditions are true, condTermFlagN is set equal to 0:
  – mbAddrN is not available,
  – The macroblock mbAddrN is coded in an Inter macroblock prediction mode,
  – mb_type for the macroblock mbAddrN is equal to I_PCM,
  – intra_chroma_pred_mode for the macroblock mbAddrN is equal to 0.
– Otherwise, condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived by:

ctxIdxInc = condTermFlagA + condTermFlagB (9-18)

9.3.3.1.1.9 Derivation process of ctxIdxInc for the syntax element coded_block_flag

Input to this process is ctxBlockCat and additional input is specified as follows:

– If ctxBlockCat is equal to 0, 6, or 10, no additional input.
– Otherwise, if ctxBlockCat is equal to 1 or 2, luma4x4BlkIdx.
– Otherwise, if ctxBlockCat is equal to 3, the chroma component index iCbCr.
– Otherwise, if ctxBlockCat is equal to 4, chroma4x4BlkIdx and the chroma component index iCbCr.
– Otherwise, if ctxBlockCat is equal to 5, luma8x8BlkIdx.
– Otherwise, if ctxBlockCat is equal to 7 or 8, cb4x4BlkIdx.
– Otherwise, if ctxBlockCat is equal to 9, cb8x8BlkIdx.
– Otherwise, if ctxBlockCat is equal to 11 or 12, cr4x4BlkIdx.
– Otherwise (ctxBlockCat is equal to 13), cr8x8BlkIdx.

Output of this process is ctxIdxInc(ctxBlockCat).

Let the variable transBlockN (with N being either A or B) be derived as follows:

– If ctxBlockCat is equal to 0, 6, or 10, the following ordered steps are specified:
  1. The derivation process for neighbouring macroblocks specified in subclause 6.4.11.1 is invoked and the output is assigned to mbAddrN (with N being either A or B).
  2. The variable transBlockN is derived as follows:
     – If mbAddrN is available and the macroblock mbAddrN is coded in Intra_16x16 macroblock prediction mode, the following applies:
       – If ctxBlockCat is equal to 0, the luma DC block of macroblock mbAddrN is assigned to transBlockN.
       – Otherwise, if ctxBlockCat is equal to 6, the Cb DC block of macroblock mbAddrN is assigned to transBlockN.
       – Otherwise (ctxBlockCat is equal to 10), the Cr DC block of macroblock mbAddrN is assigned to transBlockN.
     – Otherwise, transBlockN is marked as not available.
– Otherwise, if ctxBlockCat is equal to 1 or 2, the following ordered steps are specified:
  1. The derivation process for neighbouring 4x4 luma blocks specified in subclause 6.4.11.4 is invoked with luma4x4BlkIdx as input and the output is assigned to mbAddrN, luma4x4BlkIdxN (with N being either A or B).
  2. The variable transBlockN is derived as follows:
     – If mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip, B_Skip, or I_PCM, ((CodedBlockPatternLuma >> (luma4x4BlkIdxN >>2)) & 1) is not equal to 0 for the

macroblock mbAddrN, and transform_size_8x8_flag is equal to 0 for the macroblock mbAddrN, the 4x4 luma block with index luma4x4BlkIdxN of macroblock mbAddrN is assigned to transBlockN.

  - Otherwise, if mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip or B_Skip, ((CodedBlockPatternLuma >> (luma4x4BlkIdxN >>2)) & 1) is not equal to 0 for the macroblock mbAddrN, and transform_size_8x8_flag is equal to 1 for the macroblock mbAddrN, the 8x8 luma block with index (luma4x4BlkIdxN >> 2) of macroblock mbAddrN is assigned to transBlockN.

  - Otherwise, transBlockN is marked as not available.

- Otherwise, if ctxBlockCat is equal to 3, the following ordered steps are specified:

  1. The derivation process for neighbouring macroblocks specified in subclause 6.4.11.1 is invoked and the output is assigned to mbAddrN (with N being either A or B).

  2. The variable transBlockN is derived as follows:

     - If mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip, B_Skip, or I_PCM, and CodedBlockPatternChroma is not equal to 0 for the macroblock mbAddrN, the chroma DC block of chroma component iCbCr of macroblock mbAddrN is assigned to transBlockN.

     - Otherwise, transBlockN is marked as not available.

- Otherwise, if ctxBlockCat is equal to 4, the following ordered steps are specified:

  1. The derivation process for neighbouring 4x4 chroma blocks specified in subclause 6.4.11.5 is invoked with chroma4x4BlkIdx as input and the output is assigned to mbAddrN, chroma4x4BlkIdxN (with N being either A or B).

  2. The variable transBlockN is derived as follows:

     - If mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip, B_Skip, or I_PCM, and CodedBlockPatternChroma is equal to 2 for the macroblock mbAddrN, the 4x4 chroma block with chroma4x4BlkIdxN of the chroma component iCbCr of macroblock mbAddrN is assigned to transBlockN.

     - Otherwise, transBlockN is marked as not available.

- Otherwise, if ctxBlockCat is equal to 5, the following ordered steps are specified:

  1. The derivation process for neighbouring 8x8 luma blocks specified in subclause 6.4.11.2 is invoked with luma8x8BlkIdx as input and the output is assigned to mbAddrN, luma8x8BlkIdxN (with N being either A or B).

  2. The variable transBlockN is derived as follows:

     - If mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip, B_Skip, or I_PCM, ((CodedBlockPatternLuma >>luma8x8BlkIdx) & 1) is not equal to 0 for the macroblock mbAddrN, and transform_size_8x8_flag is equal to 1 for the macroblock mbAddrN, the 8x8 luma block with index luma8x8BlkIdxN of macroblock mbAddrN is assigned to transBlockN.

     - Otherwise, transBlockN is marked as not available.

- Otherwise, if ctxBlockCat is equal to7 or 8, the following ordered steps are specified:

  1. The derivation process for neighbouring 4x4 Cb blocks specified in subclause 6.4.11.5 is invoked with cb4x4BlkIdx as input and the output is assigned to mbAddrN, cb4x4BlkIdxN (with N being either A or B).

  2. The variable transBlockN is derived as follows:

     - If mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip, B_Skip, or I_PCM, ((CodedBlockPatternLuma >> (cb4x4BlkIdxN >>2)) & 1) is not equal to 0 for the macroblock mbAddrN, and transform_size_8x8_flag is equal to 0 for the macroblock mbAddrN, the 4x4 Cb block with index cb4x4BlkIdxN of macroblock mbAddrN is assigned to transBlockN.

     - Otherwise, if mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip or B_Skip, ((CodedBlockPatternLuma >> (cb4x4BlkIdxN >>2)) & 1) is not equal to 0 for the macroblock mbAddrN, and transform_size_8x8_flag is equal to 1 for the macroblock mbAddrN, the 8x8 Cb block with index (cb4x4BlkIdxN >> 2) of macroblock mbAddrN is assigned to transBlockN.

     - Otherwise, transBlockN is marked as not available.

– Otherwise, if ctxBlockCat is equal to 9, the following ordered steps are specified:

1. The derivation process for neighbouring 8x8 Cb blocks specified in subclause 6.4.11.3 is invoked with cb8x8BlkIdx as input and the output is assigned to mbAddrN, cb8x8BlkIdxN (with N being either A or B).

2. The variable transBlockN is derived as follows:

   – If mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip, B_Skip, or I_PCM, ((CodedBlockPatternLuma >>cb8x8BlkIdx) & 1) is not equal to 0 for the macroblock mbAddrN, and transform_size_8x8_flag is equal to 1 for the macroblock mbAddrN, the 8x8 Cb block with index cb8x8BlkIdxN of macroblock mbAddrN is assigned to transBlockN.

   – Otherwise, transBlockN is marked as not available.

– Otherwise, if ctxBlockCat is equal to 11 or 12, the following ordered steps are specified:

1. The derivation process for neighbouring 4x4 Cr blocks specified in subclause 6.4.11.5 is invoked with cr4x4BlkIdx as input and the output is assigned to mbAddrN, cr4x4BlkIdxN (with N being either A or B).

2. The variable transBlockN is derived as follows:

   – If mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip, B_Skip, or I_PCM, ((CodedBlockPatternLuma >> (cr4x4BlkIdxN >>2)) & 1) is not equal to 0 for the macroblock mbAddrN, and transform_size_8x8_flag is equal to 0 for the macroblock mbAddrN, the 4x4 Cr block with index cr4x4BlkIdxN of macroblock mbAddrN is assigned to transBlockN.

   – Otherwise, if mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip or B_Skip, ((CodedBlockPatternLuma >> (cr4x4BlkIdxN >>2)) & 1) is not equal to 0 for the macroblock mbAddrN, and transform_size_8x8_flag is equal to 1 for the macroblock mbAddrN, the 8x8 Cr block with index (cr4x4BlkIdxN >> 2) of macroblock mbAddrN is assigned to transBlockN.

   – Otherwise, transBlockN is marked as not available.

– Otherwise (ctxBlockCat is equal to 13), the following ordered steps are specified:

1. The derivation process for neighbouring 8x8 Cr blocks specified in subclause 6.4.11.3 is invoked with cr8x8BlkIdx as input and the output is assigned to mbAddrN, cr8x8BlkIdxN (with N being either A or B).

2. The variable transBlockN is derived as follows:

   – If mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip, B_Skip, or I_PCM,, ((CodedBlockPatternLuma >>cr8x8BlkIdx) & 1) is not equal to 0 for the macroblock mbAddrN, and transform_size_8x8_flag is equal to 1 for the macroblock mbAddrN, the 8x8 Cr block with index cr8x8BlkIdxN of macroblock mbAddrN is assigned to transBlockN.

   – Otherwise, transBlockN is marked as not available.

Let the variable condTermFlagN (with N being either A or B) be derived as follows:

– If any of the following conditions are true, condTermFlagN is set equal to 0:

  – mbAddrN is not available and the current macroblock is coded in an Inter macroblock prediction mode,

  – mbAddrN is available and transBlockN is not available and mb_type for the macroblock mbAddrN is not equal to I_PCM,

  – The current macroblock is coded in an Intra macroblock prediction mode, constrained_intra_pred_flag is equal to 1, the macroblock mbAddrN is available and coded in an Inter macroblock prediction mode, and slice data partitioning is in use (nal_unit_type is in the range of 2 through 4, inclusive).

– Otherwise, if any of the following conditions are true, condTermFlagN is set equal to 1:

  – mbAddrN is not available and the current macroblock is coded in an Intra macroblock prediction mode,

  – mb_type for the macroblock mbAddrN is equal to I_PCM.

– Otherwise, condTermFlagN is set equal to the value of the coded_block_flag of the transform block transBlockN that was decoded for the macroblock mbAddrN.

The variable ctxIdxInc(ctxBlockCat) is derived by

$$\text{ctxIdxInc(ctxBlockCat)} = \text{condTermFlagA} + 2 * \text{condTermFlagB} \tag{9-19}$$

9.3.3.1.1.10 Derivation process of ctxIdxInc for the syntax element transform_size_8x8_flag

Output of this process is ctxIdxInc.

The derivation process for neighbouring macroblocks specified in subclause 6.4.11.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

Let the variable condTermFlagN (with N being either A or B) be derived as follows:

– If any of the following conditions are true, condTermFlagN is set equal to 0:
  – mbAddrN is not available,
  – transform_size_8x8_flag for the macroblock mbAddrN is equal to 0.
– Otherwise, condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived by

$$\text{ctxIdxInc} = \text{condTermFlagA} + \text{condTermFlagB} \quad (9\text{-}20)$$

9.3.3.1.2 Assignment process of ctxIdxInc using prior decoded bin values

Inputs to this process are ctxIdxOffset and binIdx.

Output of this process is ctxIdxInc.

Table 9-41 contains the specification of ctxIdxInc for the given values of ctxIdxOffset and binIdx.

For each value of ctxIdxOffset and binIdx, ctxIdxInc is derived by using some of the values of prior decoded bin values (b_0, b_1, b_2,…, b_k), where the value of the index k is less than the value of binIdx.

Table 9-41 – Specification of ctxIdxInc for specific values of ctxIdxOffset and binIdx

Value (name) of ctxIdxOffset	binIdx	ctxIdxInc
3	4	$(b_3\ !=\ 0)\ ?\ 5:\ 6$
	5	$(b_3\ !=\ 0)\ ?\ 6:\ 7$
14	2	$(b_1\ !=\ 1)\ ?\ 2:\ 3$
17	4	$(b_3\ !=\ 0)\ ?\ 2:\ 3$
27	2	$(b_1\ !=\ 0)\ ?\ 4:\ 5$
32	4	$(b_3\ !=\ 0)\ ?\ 2:\ 3$
36	2	$(b_1\ !=\ 0)\ ?\ 2:\ 3$

9.3.3.1.3 Assignment process of ctxIdxInc for syntax elements significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1

Inputs to this process are ctxIdxOffset and binIdx.

Output of this process is ctxIdxInc.

The assignment process of ctxIdxInc for syntax elements significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1 as well as for coded_block_flag depends on categories of different blocks denoted by the variable ctxBlockCat. The specification of these block categories is given in Table 9-42.

Table 9-42 – Specification of ctxBlockCat for the different blocks

Block description	maxNumCoeff	ctxBlockCat
block of luma DC transform coefficient levels (i.e., list Intra16x16DCLevel as described in subclause 7.4.5.3)	16	0
block of luma AC transform coefficient levels (i.e., list Intra16x16ACLevel[i] as described in subclause 7.4.5.3)	15	1
block of 16 luma transform coefficient levels (i.e., list LumaLevel4x4[i] as described in subclause 7.4.5.3)	16	2
block of chroma DC transform coefficient levels when ChromaArrayType is equal to 1 or 2 (i.e., list ChromaDCLevel as described in subclause 7.4.5.3)	4 * NumC8x8	3
block of chroma AC transform coefficient levels when ChromaArrayType is equal to 1 or 2 (i.e., list ChromaACLevel as described in subclause 7.4.5.3)	15	4
block of 64 luma transform coefficient levels (i.e., list LumaLevel8x8[i] as described in subclause 7.4.5.3)	64	5
block of Cb DC transform coefficient levels when ChromaArrayType is equal to 3 (i.e., list CbIntra16x16DCLevel as described in subclause 7.4.5.3)	16	6
block of Cb AC transform coefficient levels when ChromaArrayType is equal to 3 (i.e., list CbIntra16x16ACLevel[i] as described in subclause 7.4.5.3)	15	7
block of 16 Cb transform coefficient levels when ChromaArrayType is equal to 3 (i.e., list CbLevel4x4[i] as described in subclause 7.4.5.3)	16	8
block of 64 Cb transform coefficient levels when ChromaArrayType is equal to 3 (i.e., list CbLevel8x8[i] as described in subclause 7.4.5.3)	64	9
block of Cr DC transform coefficient levels when ChromaArrayType is equal to 3 (i.e., list CrIntra16x16DCLevel as described in subclause 7.4.5.3)	16	10
block of Cr AC transform coefficient levels when ChromaArrayType is equal to 3 (i.e., list CrIntra16x16ACLevel[i] as described in subclause 7.4.5.3)	15	11
block of 16 Cr transform coefficient levels when ChromaArrayType is equal to 3 (i.e., list CrLevel4x4[i] as described in subclause 7.4.5.3)	16	12
block of 64 Cr transform coefficient levels when ChromaArrayType is equal to 3 (i.e., list CrLevel8x8[i] as described in subclause 7.4.5.3)	64	13

Let the variable levelListIdx be set equal to the index of the list of transform coefficient levels as specified in subclause 7.4.5.3.

For the syntax elements significant_coeff_flag and last_significant_coeff_flag in blocks with ctxBlockCat not equal to 3, 5, 9, and 13, the variable ctxIdxInc is derived by

$$\text{ctxIdxInc} = \text{levelListIdx} \tag{9-21}$$

where levelListIdx ranges from 0 to maxNumCoeff − 2, inclusive.

For the syntax elements significant_coeff_flag and last_significant_coeff_flag in blocks with ctxBlockCat = = 3, the variable ctxIdxInc is derived by

$$\text{ctxIdxInc} = \text{Min}(\ \text{levelListIdx} / \text{NumC8x8},\ 2\ ) \tag{9-22}$$

where levelListIdx ranges from 0 to 4 * NumC8x8 − 2, inclusive.

For the syntax elements significant_coeff_flag and last_significant_coeff_flag in 8x8 luma, Cb, or Cr blocks with ctxBlockCat = = 5, 9, or 13, Table 9-43 contains the specification of ctxIdxInc for the given values of levelListIdx, where levelListIdx ranges from 0 to 62, inclusive.

Table 9-43 – Mapping of scanning position to ctxIdxInc for ctxBlockCat = = 5, 9, or 13

levelListIdx	ctxIdxInc for significant_coeff_flag (frame coded macroblocks)	ctxIdxInc for significant_coeff_flag (field coded macroblocks)	ctxIdxInc for last_significant_coeff_flag	levelListIdx	ctxIdxInc for significant_coeff_flag (frame coded macroblocks)	ctxIdxInc for significant_coeff_flag (field coded macroblocks)	ctxIdxInc for last_significant_coeff_flag
0	0	0	0	**32**	7	9	3
1	1	1	1	**33**	6	9	3
2	2	1	1	**34**	11	10	3
3	3	2	1	**35**	12	10	3
4	4	2	1	**36**	13	8	3
5	5	3	1	**37**	11	11	3
6	5	3	1	**38**	6	12	3
7	4	4	1	**39**	7	11	3
8	4	5	1	**40**	8	9	4
9	3	6	1	**41**	9	9	4
10	3	7	1	**42**	14	10	4
11	4	7	1	**43**	10	10	4
12	4	7	1	**44**	9	8	4
13	4	8	1	**45**	8	13	4
14	5	4	1	**46**	6	13	4
15	5	5	1	**47**	11	9	4
16	4	6	2	**48**	12	9	5
17	4	9	2	**49**	13	10	5
18	4	10	2	**50**	11	10	5
19	4	10	2	**51**	6	8	5
20	3	8	2	**52**	9	13	6
21	3	11	2	**53**	14	13	6
22	6	12	2	**54**	10	9	6
23	7	11	2	**55**	9	9	6
24	7	9	2	**56**	11	10	7
25	7	9	2	**57**	12	10	7
26	8	10	2	**58**	13	14	7
27	9	10	2	**59**	11	14	7

Table 9-43 – Mapping of scanning position to ctxIdxInc for ctxBlockCat = = 5, 9, or 13

levelListIdx	ctxIdxInc for significant_coeff_flag (frame coded macroblocks)	ctxIdxInc for significant_coeff_flag (field coded macroblocks)	ctxIdxInc for last_significant_coeff_flag	levelListIdx	ctxIdxInc for significant_coeff_flag (frame coded macroblocks)	ctxIdxInc for significant_coeff_flag (field coded macroblocks)	ctxIdxInc for last_significant_coeff_flag
28	10	8	2	**60**	14	14	8
29	9	11	2	**61**	10	14	8
30	8	12	2	**62**	12	14	8
31	7	11	2				

Let numDecodAbsLevelEq1 denote the accumulated number of decoded transform coefficient levels with absolute value equal to 1, and let numDecodAbsLevelGt1 denote the accumulated number of decoded transform coefficient levels with absolute value greater than 1. Both numbers are related to the same transform coefficient block, where the current decoding process takes place. Then, for decoding of coeff_abs_level_minus1, ctxIdxInc for coeff_abs_level_minus1 is specified depending on binIdx as follows:

– If binIdx is equal to 0, ctxIdxInc is derived by

ctxIdxInc = ((numDecodAbsLevelGt1 != 0) ? 0: Min(4, 1 + numDecodAbsLevelEq1)) (9-23)

– Otherwise (binIdx is greater than 0), ctxIdxInc is derived by

ctxIdxInc = 5 + Min(4 − ((ctxBlockCat = = 3) ? 1 : 0), numDecodAbsLevelGt1) (9-24)

9.3.3.2 Arithmetic decoding process

Inputs to this process are the bypassFlag, ctxIdx as derived in subclause 9.3.3.1, and the state variables codIRange and codIOffset of the arithmetic decoding engine.

Output of this process is the value of the bin.

Figure 9-2 illustrates the whole arithmetic decoding process for a single bin. For decoding the value of a bin, the context index ctxIdx is passed to the arithmetic decoding process DecodeBin(ctxIdx), which is specified as follows:

– If bypassFlag is equal to 1, DecodeBypass( ) as specified in subclause 9.3.3.2.3 is invoked.

– Otherwise, if bypassFlag is equal to 0 and ctxIdx is equal to 276, DecodeTerminate( ) as specified in subclause 9.3.2.4 is invoked.

– Otherwise (bypassFlag is equal to 0 and ctxIdx is not equal to 276), DecodeDecision( ) as specified in subclause 9.3.3.2.1 is applied.

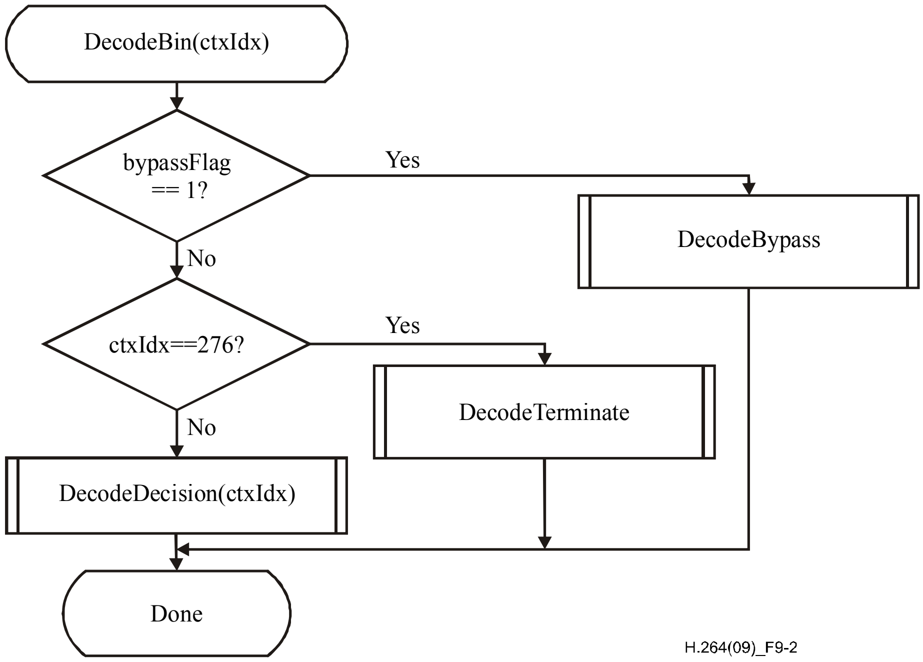


Figure 9-2 – Overview of the arithmetic decoding process for a single bin (informative)

NOTE – Arithmetic coding is based on the principle of recursive interval subdivision. Given a probability estimation p(0) and p(1) = 1 – p(0) of a binary decision (0, 1), an initially given code sub-interval with the range codIRange will be subdivided into two sub-intervals having range p(0) * codIRange and codIRange – p(0) * codIRange, respectively. Depending on the decision, which has been observed, the corresponding sub-interval will be chosen as the new code interval, and a binary code string pointing into that interval will represent the sequence of observed binary decisions. It is useful to distinguish between the most probable symbol (MPS) and the least probable symbol (LPS), so that binary decisions have to be identified as either MPS or LPS, rather than 0 or 1. Given this terminology, each context is specified by the probability p_{LPS} of the LPS and the value of MPS (valMPS), which is either 0 or 1.

The arithmetic core engine in this Recommendation | International Standard has three distinct properties:

– The probability estimation is performed by means of a finite-state machine with a table-based transition process between 64 different representative probability states { p_{LPS}(pStateIdx) | 0 <= pStateIdx < 64 } for the LPS probability p_{LPS}. The numbering of the states is arranged in such a way that the probability state with index pStateIdx = 0 corresponds to an LPS probability value of 0.5, with decreasing LPS probability towards higher state indices.

– The range codIRange representing the state of the coding engine is quantised to a small set $\{Q_1,\ldots,Q_4\}$ of pre-set quantisation values prior to the calculation of the new interval range. Storing a table containing all 64x4 pre-computed product values of Q_i * p_{LPS}(pStateIdx) allows a multiplication-free approximation of the product codIRange * p_{LPS}(pStateIdx).

– For syntax elements or parts thereof for which an approximately uniform probability distribution is assumed to be given a separate simplified encoding and decoding bypass process is used.

9.3.3.2.1 Arithmetic decoding process for a binary decision

Inputs to this process are ctxIdx, codIRange, and codIOffset.

Outputs of this process are the decoded value binVal, and the updated variables codIRange and codIOffset.

Figure 9-3 shows the flowchart for decoding a single decision (DecodeDecision):

1. The value of the variable codIRangeLPS is derived as follows:

   – Given the current value of codIRange, the variable qCodIRangeIdx is derived by

     qCodIRangeIdx =(codIRange >> 6) & 3 (9-25)

   – Given qCodIRangeIdx and pStateIdx associated with ctxIdx, the value of the variable rangeTabLPS as specified in Table 9-44 is assigned to codIRangeLPS:

     codIRangeLPS = rangeTabLPS[pStateIdx][qCodIRangeIdx] (9-26)

2. The variable codIRange is set equal to codIRange − codIRangeLPS and the following applies:

   – If codIOffset is greater than or equal to codIRange, the variable binVal is set equal to 1 − valMPS, codIOffset is decremented by codIRange, and codIRange is set equal to codIRangeLPS.

   – Otherwise, the variable binVal is set equal to valMPS.

Given the value of binVal, the state transition is performed as specified in subclause 9.3.3.2.1.1. Depending on the current value of codIRange, renormalization is performed as specified in subclause 9.3.3.2.2.

9.3.3.2.1.1 State transition process

Inputs to this process are the current pStateIdx, the decoded value binVal and valMPS values of the context variable associated with ctxIdx.

Outputs of this process are the updated pStateIdx and valMPS of the context variable associated with ctxIdx.

Depending on the decoded value binVal, the update of the two variables pStateIdx and valMPS associated with ctxIdx is derived as specified by the following pseudo-code:

```
if( binVal  = =  valMPS )
    pStateIdx = transIdxMPS( pStateIdx )
else {                                                    (9-27)
    if( pStateIdx  = =  0 )
        valMPS = 1 − valMPS
    pStateIdx = transIdxLPS( pStateIdx )
}
```

Table 9-45 specifies the transition rules transIdxMPS( ) and transIdxLPS( ) after decoding the value of valMPS and 1 − valMPS, respectively.

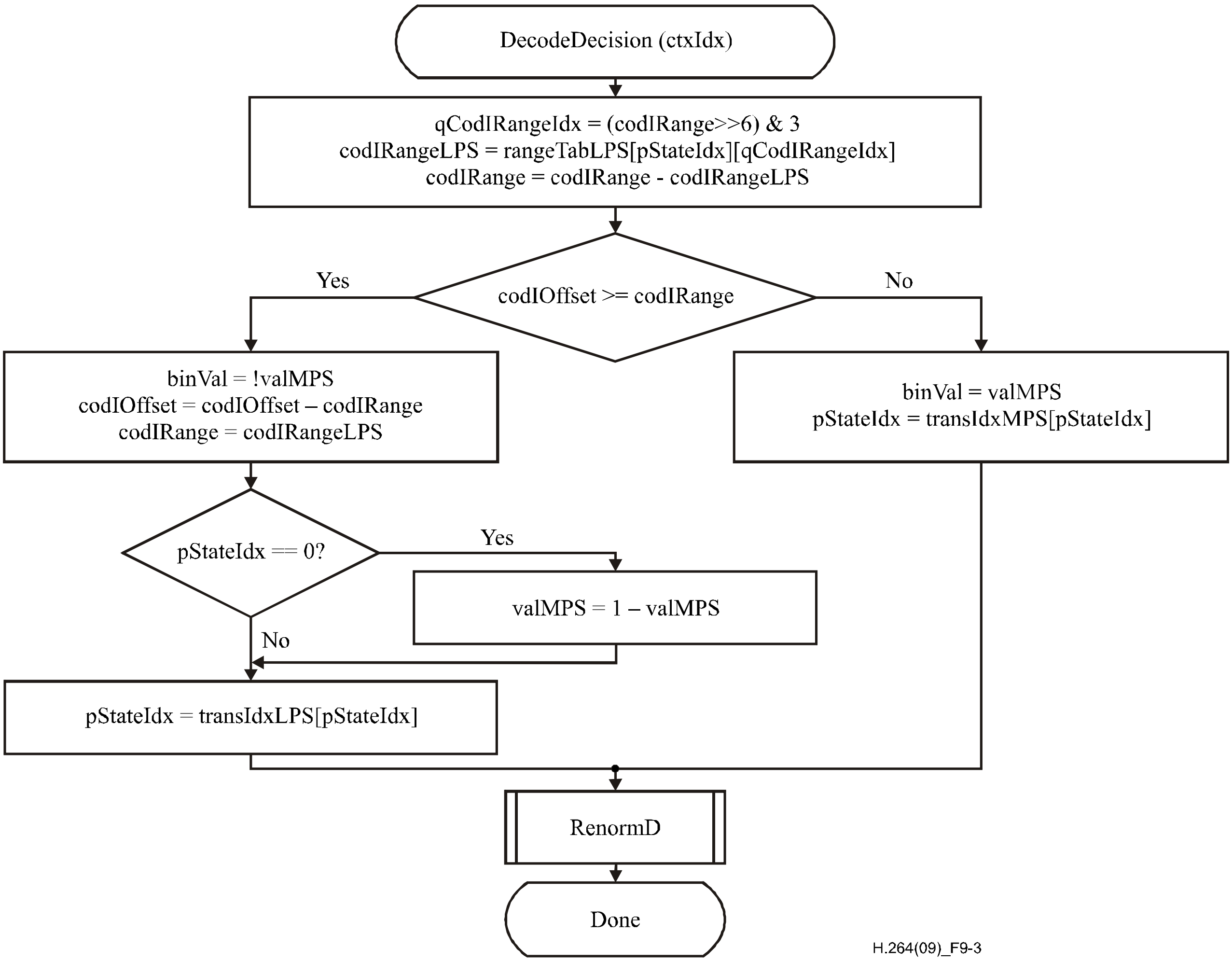


Figure 9-3 – Flowchart for decoding a decision

Table 9-44 – Specification of rangeTabLPS depending on pStateIdx and qCodIRangeIdx

pStateIdx	qCodIRangeIdx				pStateIdx	qCodIRangeIdx			
	0	1	2	3		0	1	2	3
0	128	176	208	240	32	27	33	39	45
1	128	167	197	227	33	26	31	37	43
2	128	158	187	216	34	24	30	35	41
3	123	150	178	205	35	23	28	33	39
4	116	142	169	195	36	22	27	32	37
5	111	135	160	185	37	21	26	30	35
6	105	128	152	175	38	20	24	29	33
7	100	122	144	166	39	19	23	27	31
8	95	116	137	158	40	18	22	26	30
9	90	110	130	150	41	17	21	25	28
10	85	104	123	142	42	16	20	23	27
11	81	99	117	135	43	15	19	22	25
12	77	94	111	128	44	14	18	21	24
13	73	89	105	122	45	14	17	20	23
14	69	85	100	116	46	13	16	19	22
15	66	80	95	110	47	12	15	18	21
16	62	76	90	104	48	12	14	17	20
17	59	72	86	99	49	11	14	16	19
18	56	69	81	94	50	11	13	15	18
19	53	65	77	89	51	10	12	15	17
20	51	62	73	85	52	10	12	14	16
21	48	59	69	80	53	9	11	13	15
22	46	56	66	76	54	9	11	12	14
23	43	53	63	72	55	8	10	12	14
24	41	50	59	69	56	8	9	11	13
25	39	48	56	65	57	7	9	11	12
26	37	45	54	62	58	7	9	10	12
27	35	43	51	59	59	7	8	10	11
28	33	41	48	56	60	6	8	9	11
29	32	39	46	53	61	6	7	9	10
30	30	37	43	50	62	6	7	8	9
31	29	35	41	48	63	2	2	2	2

Table 9-45 – State transition table

pStateIdx	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
transIdxLPS	0	0	1	2	2	4	4	5	6	7	8	9	9	11	11	12
transIdxMPS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
pStateIdx	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
transIdxLPS	13	13	15	15	16	16	18	18	19	19	21	21	22	22	23	24
transIdxMPS	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
pStateIdx	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
transIdxLPS	24	25	26	26	27	27	28	29	29	30	30	30	31	32	32	33
transIdxMPS	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
pStateIdx	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
transIdxLPS	33	33	34	34	35	35	35	36	36	36	37	37	37	38	38	63
transIdxMPS	49	50	51	52	53	54	55	56	57	58	59	60	61	62	62	63

9.3.3.2.2 Renormalization process in the arithmetic decoding engine

Inputs to this process are bits from slice data and the variables codIRange and codIOffset.

Outputs of this process are the updated variables codIRange and codIOffset.

A flowchart of the renormalization is shown in Figure 9-4. The current value of codIRange is first compared to 256 and further steps are specified as follows:

– If codIRange is greater than or equal to 256, no renormalization is needed and the RenormD process is finished;

– Otherwise (codIRange is less than 256), the renormalization loop is entered. Within this loop, the value of codIRange is doubled, i.e., left-shifted by 1 and a single bit is shifted into codIOffset by using read_bits(1).

The bitstream shall not contain data that result in a value of codIOffset being greater than or equal to codIRange upon completion of this process.

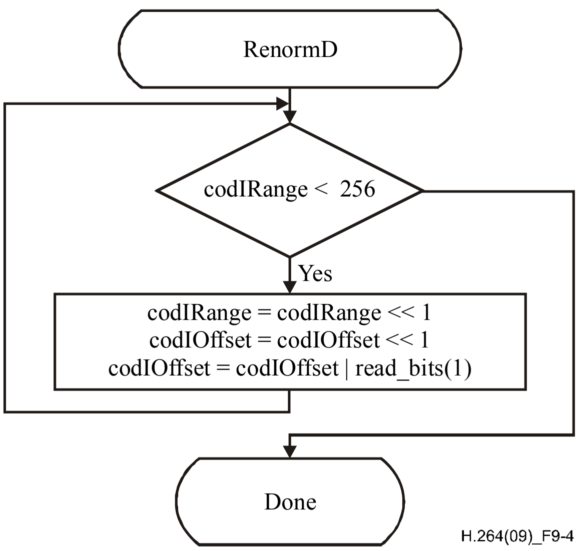


Figure 9-4 – Flowchart of renormalization

9.3.3.2.3 Bypass decoding process for binary decisions

Inputs to this process are bits from slice data and the variables codIRange and codIOffset.

Outputs of this process are the updated variable codIOffset and the decoded value binVal.

The bypass decoding process is invoked when bypassFlag is equal to 1. Figure 9-5 shows a flowchart of the corresponding process.

First, the value of codIOffset is doubled, i.e., left-shifted by 1 and a single bit is shifted into codIOffset by using read_bits(1). Then, the value of codIOffset is compared to the value of codIRange and further steps are specified as follows:

– If codIOffset is greater than or equal to codIRange, the variable binVal is set equal to 1 and codIOffset is decremented by codIRange.

– Otherwise (codIOffset is less than codIRange), the variable binVal is set equal to 0.

The bitstream shall not contain data that result in a value of codIOffset being greater than or equal to codIRange upon completion of this process.

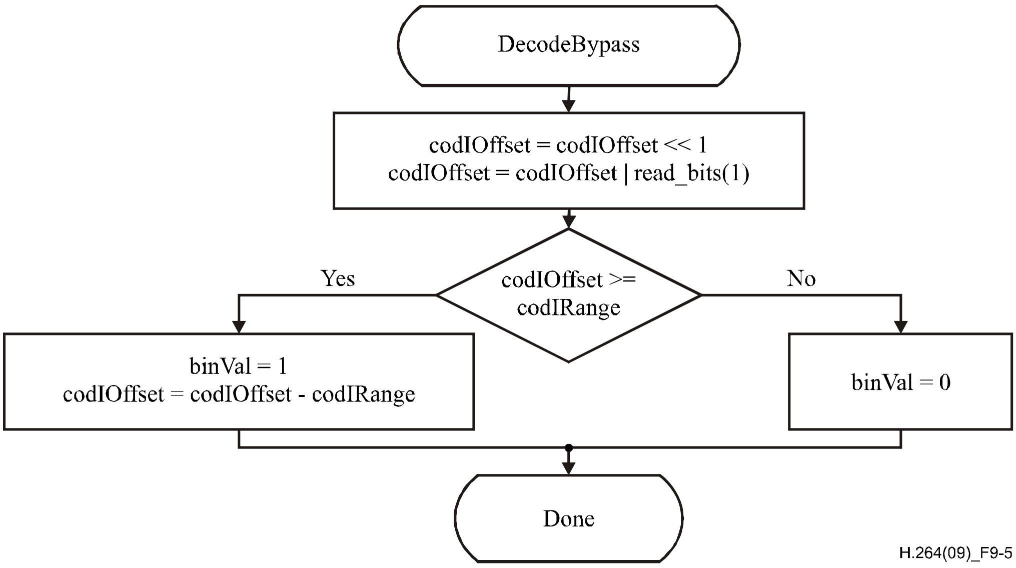


Figure 9-5 – Flowchart of bypass decoding process

9.3.3.2.4 Decoding process for binary decisions before termination

Inputs to this process are bits from slice data and the variables codIRange and codIOffset.

Outputs of this process are the updated variables codIRange and codIOffset, and the decoded value binVal.

This special decoding routine applies to decoding of end_of_slice_flag and of the bin indicating the I_PCM mode corresponding to ctxIdx equal to 276. Figure 9-6 shows the flowchart of the corresponding decoding process, which is specified as follows.

First, the value of codIRange is decremented by 2. Then, the value of codIOffset is compared to the value of codIRange and further steps are specified as follows:

– If codIOffset is greater than or equal to codIRange, the variable binVal is set equal to 1, no renormalization is carried out, and CABAC decoding is terminated. The last bit inserted in register codIOffset is equal to 1. When decoding end_of_slice_flag, this last bit inserted in register codIOffset is interpreted as rbsp_stop_one_bit.

– Otherwise (codIOffset is less than codIRange), the variable binVal is set equal to 0 and renormalization is performed as specified in subclause 9.3.3.2.2.

NOTE – This procedure may also be implemented using DecodeDecision(ctxIdx) with ctxIdx = 276. In the case where the decoded value is equal to 1, seven more bits would be read by DecodeDecision(ctxIdx) and a decoding process would have to adjust its bitstream pointer accordingly to properly decode following syntax elements.

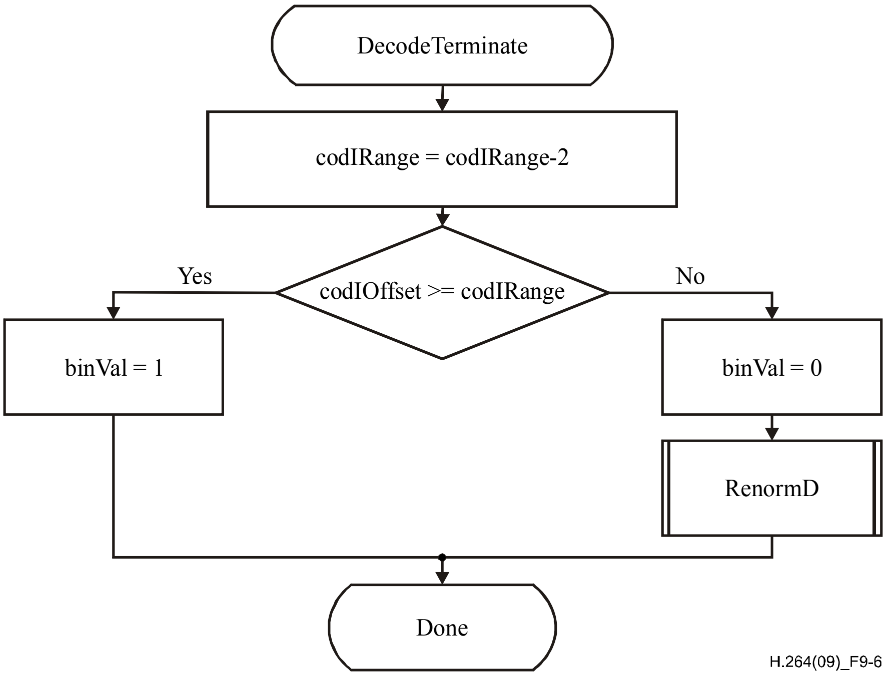


Figure 9-6 – Flowchart of decoding a decision before termination

9.3.4 Arithmetic encoding process (informative)

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

Inputs to this process are decisions that are to be encoded and written.

Outputs of this process are bits that are written to the RBSP.

This informative subclause describes an arithmetic encoding engine that matches the arithmetic decoding engine described in subclause 9.3.3.2. The encoding engine is essentially symmetric with the decoding engine, i.e., procedures are called in the same order. The following procedures are described in this section: InitEncoder, EncodeDecision, EncodeBypass, EncodeTerminate, which correspond to InitDecoder, DecodeDecision, DecodeBypass, and DecodeTerminate, respectively. The state of the arithmetic encoding engine is represented by a value of the variable codILow pointing to the lower end of a sub-interval and a value of the variable codIRange specifying the corresponding range of that sub-interval.

9.3.4.1 Initialisation process for the arithmetic encoding engine (informative)

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

This process is invoked before encoding the first macroblock of a slice, and after encoding any pcm_alignment_zero_bit and all pcm_sample_luma and pcm_sample_chroma data for a macroblock of type I_PCM.

Outputs of this process are the values codILow, codIRange, firstBitFlag, bitsOutstanding, and BinCountsInNALunits of the arithmetic encoding engine.

In the initialisation procedure of the encoder, codILow is set equal to 0, and codIRange is set equal to 510. Furthermore, firstBitFlag is set equal to 1 and the counter bitsOutstanding is set equal to 0.

Depending on whether the current slice is the first slice of a coded picture, the following applies:

– If the current slice is the first slice of a coded picture, the counter BinCountsInNALunits is set equal to 0.

– Otherwise (the current slice is not the first slice of a coded picture), the counter BinCountsInNALunits is not modified. The value of BinCountsInNALunits is the result of encoding all the slices of a coded picture that precede the current slice in decoding order. After initialising for the first slice of a coded picture as specified in this subclause, BinCountsInNALunits is incremented as specified in subclauses 9.3.4.2, 9.3.4.4, and 9.3.4.5.

NOTE – The minimum register precision required for storing the values of the variables codILow and codIRange after invocation of any of the arithmetic encoding processes specified in subclauses 9.3.4.2, 9.3.4.4, and 9.3.4.5 is 10 bits and 9 bits, respectively. The encoding process for a binary decision (EncodeDecision) as specified in subclause 9.3.4.2 and the encoding process for a binary decision before termination (EncodeTerminate) as specified in subclause 9.3.4.5 require a minimum register precision of

10 bits for the variable codILow and a minimum register precision of 9 bits for the variable codIRange. The bypass encoding process for binary decisions (EncodeBypass) as specified in subclause 9.3.4.4 requires a minimum register precision of 11 bits for the variable codILow and a minimum register precision of 9 bits for the variable codIRange. The precision required for the counters bitsOutstanding and BinCountsInNALunits should be sufficiently large to prevent overflow of the related registers. When maxBinCountInSlice denotes the maximum total number of binary decisions to encode in one slice and maxBinCountInPic denotes the maximum total number of binary decisions to encode a picture, the minimum register precision required for the variables bitsOutstanding and BinCountsInNALunits is given by Ceil(Log2(maxBinCountInSlice + 1)) and Ceil(Log2(maxBinCountInPic + 1)), respectively.

9.3.4.2 Encoding process for a binary decision (informative)

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

Inputs to this process are the context index ctxIdx, the value of binVal to be encoded, and the variables codIRange, codILow and BinCountsInNALunits.

Outputs of this process are the variables codIRange, codILow, and BinCountsInNALunits.

Figure 9-7 shows the flowchart for encoding a single decision. In a first step, the variable codIRangeLPS is derived as follows.

Given the current value of codIRange, codIRange is mapped to the index qCodIRangeIdx of a quantised value of codIRange by using Equation 9-25. The value of qCodIRangeIdx and the value of pStateIdx associated with ctxIdx are used to determine the value of the variable rangeTabLPS as specified in Table 9-44, which is assigned to codIRangeLPS. The value of codIRange − codIRangeLPS is assigned to codIRange.

In a second step, the value of binVal is compared to valMPS associated with ctxIdx. When binVal is different from valMPS, codIRange is added to codILow and codIRange is set equal to the value codIRangeLPS. Given the encoded decision, the state transition is performed as specified in subclause 9.3.3.2.1.1. Depending on the current value of codIRange, renormalization is performed as specified in subclause 9.3.4.3. Finally, the variable BinCountsInNALunits is incremented by 1.

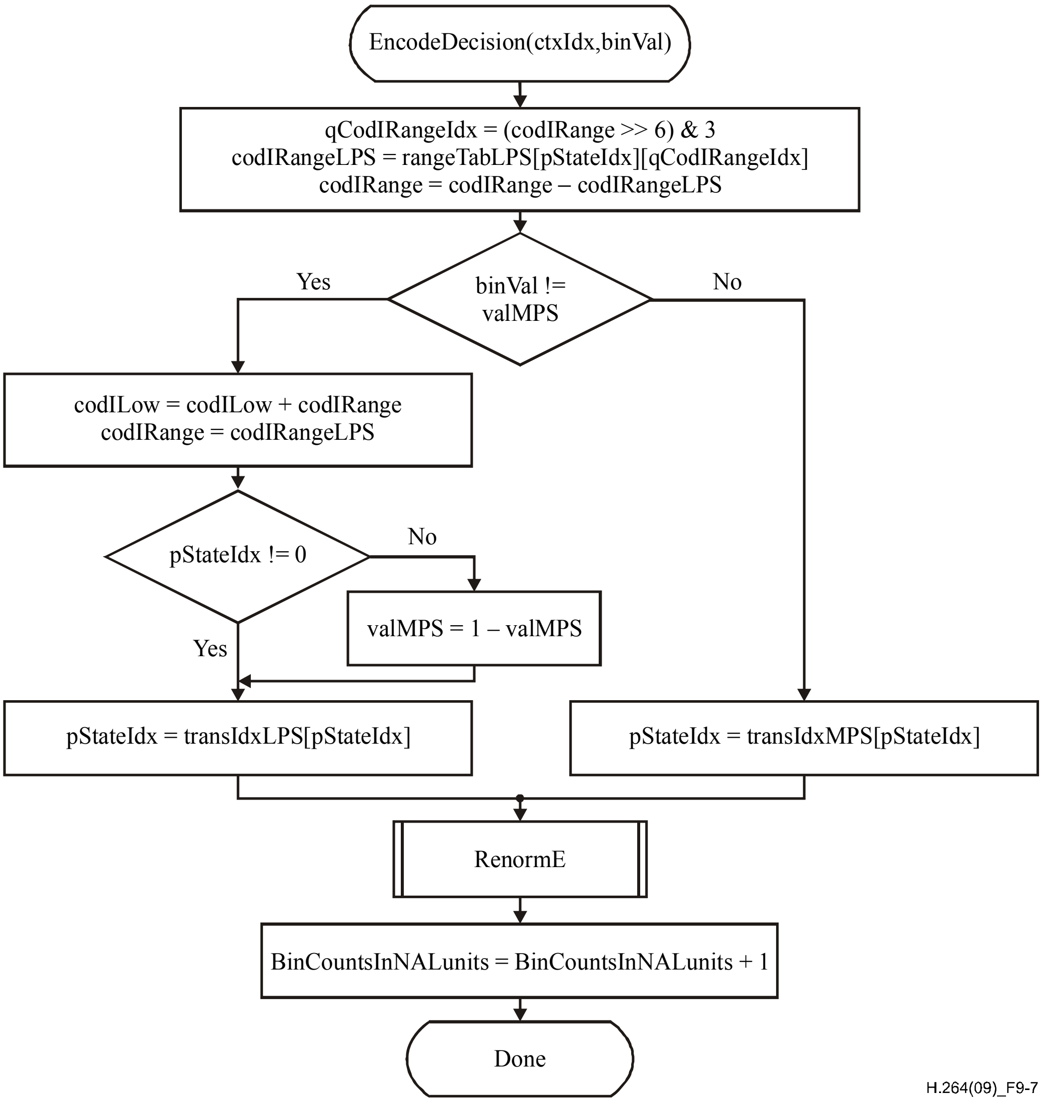


Figure 9-7 – Flowchart for encoding a decision

9.3.4.3 Renormalization process in the arithmetic encoding engine (informative)

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

Inputs to this process are the variables codIRange, codILow, firstBitFlag, and bitsOutstanding.

Outputs of this process are zero or more bits written to the RBSP and the updated variables codIRange, codILow, firstBitFlag, and bitsOutstanding.

Renormalization is illustrated in Figure 9-8.

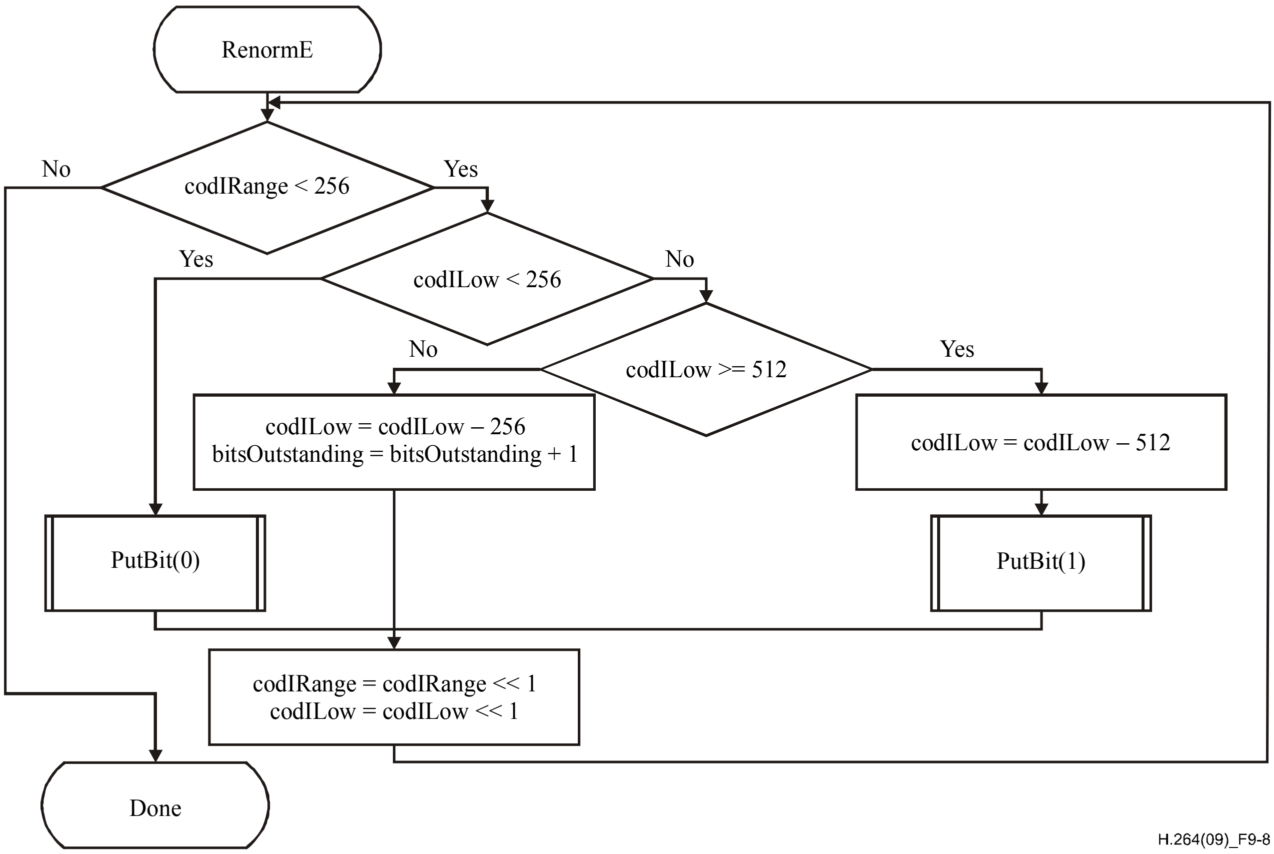


Figure 9-8 – Flowchart of renormalization in the encoder

The PutBit( ) procedure described in Figure 9-9 provides carry over control. It uses the function WriteBits(B, N) that writes N bits with value B to the bitstream and advances the bitstream pointer by N bit positions. This function assumes the existence of a bitstream pointer with an indication of the position of the next bit to be written to the bitstream by the encoding process.

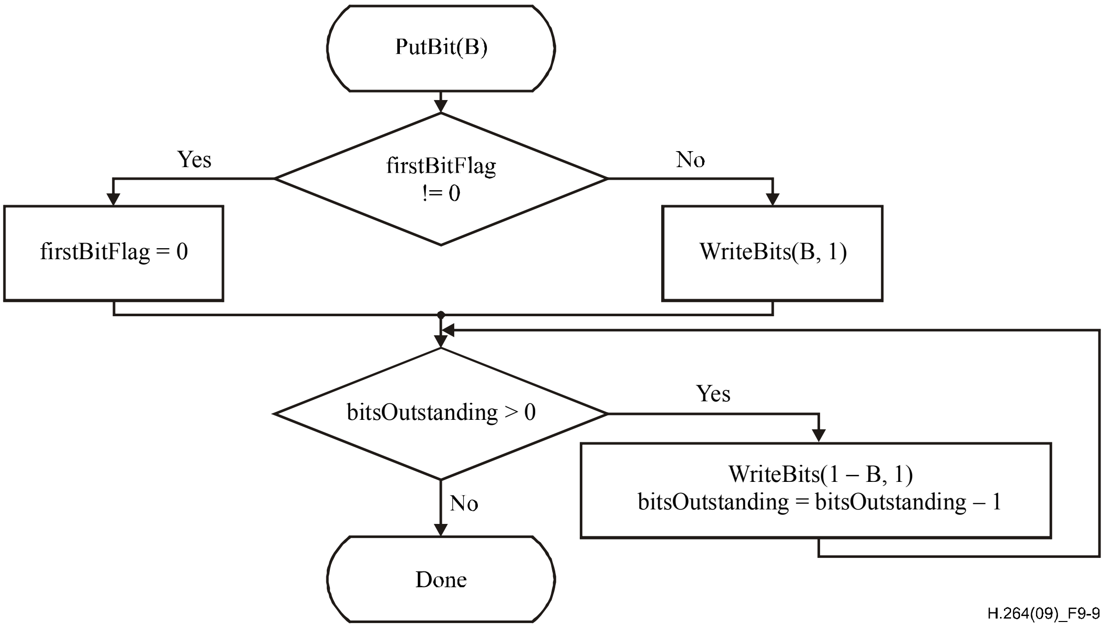


Figure 9-9 – Flowchart of PutBit(B)

9.3.4.4 Bypass encoding process for binary decisions (informative)

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

Inputs to this process are the variables binVal, codILow, codIRange, bitsOutstanding, and BinCountsInNALunits.

Output of this process is a bit written to the RBSP and the updated variables codILow, bitsOutstanding, and BinCountsInNALunits.

This encoding process applies to all binary decisions with bypassFlag equal to 1. Renormalization is included in the specification of this process as given in Figure 9-10.

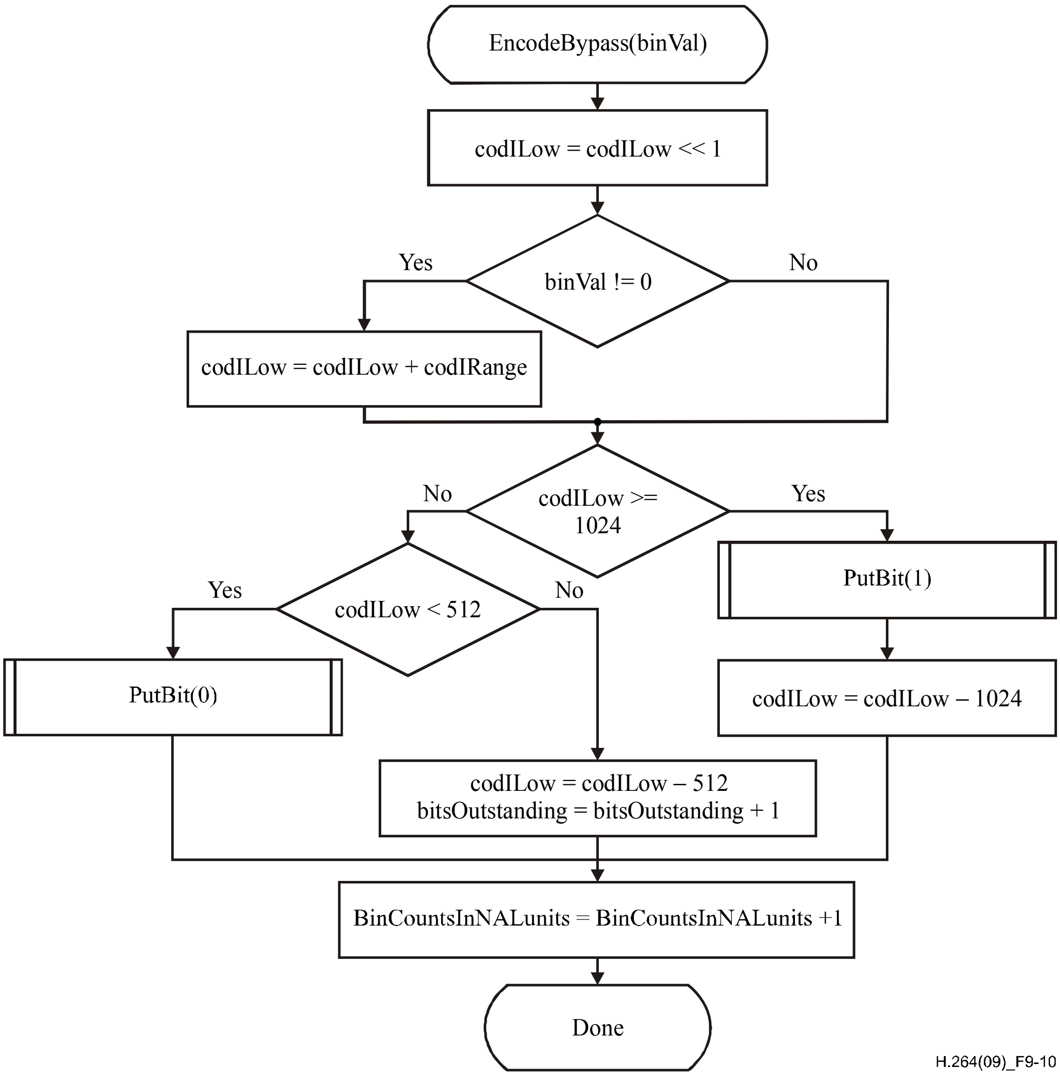


Figure 9-10 – Flowchart of encoding bypass

9.3.4.5 Encoding process for a binary decision before termination (informative)

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

Inputs to this process are the variables binVal, codIRange, codILow, bitsOutstanding, and BinCountsInNALunits.

Outputs of this process are zero or more bits written to the RBSP and the updated variables codILow, codIRange, bitsOutstanding, and BinCountsInNALunits.

This encoding routine shown in Figure 9-11 applies to encoding of the end_of_slice_flag and of the bin indicating the I_PCM mb_type both associated with ctxIdx equal to 276.

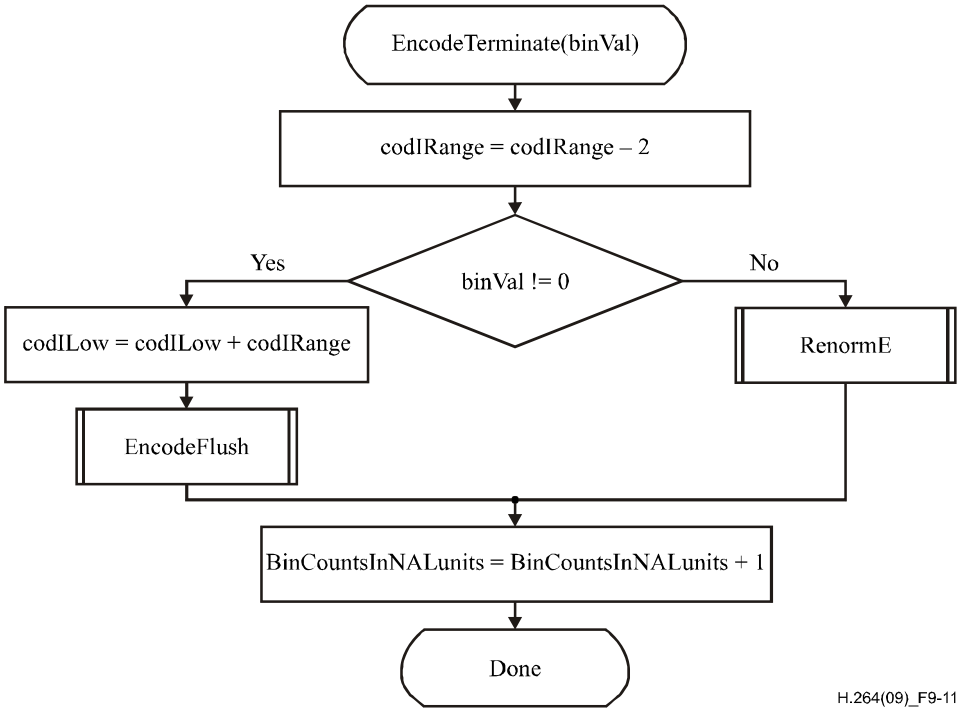


Figure 9-11 – Flowchart of encoding a decision before termination

When the value of binVal to encode is equal to 1, CABAC encoding is terminated and the flushing procedure shown in Figure 9-12 is applied. In this flushing procedure, the last bit written by WriteBits(B, N) is equal to 1. When encoding end_of_slice_flag, this last bit is interpreted as the rbsp_stop_one_bit.

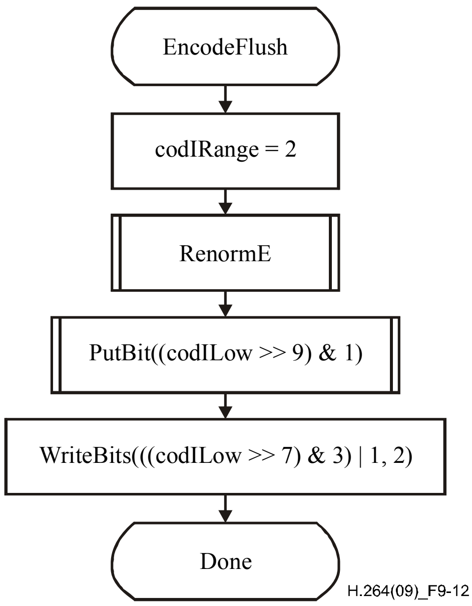


Figure 9-12 – Flowchart of flushing at termination

9.3.4.6 Byte stuffing process (informative)

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

This process is invoked after encoding the last macroblock of the last slice of a picture and after encapsulation.

Inputs to this process are the number of bytes NumBytesInVclNALunits of all VCL NAL units of a picture, the number of macroblocks PicSizeInMbs in the picture, and the number of binary symbols BinCountsInNALunits resulting from encoding the contents of all VCL NAL units of the picture.

NOTE – The value of BinCountsInNALunits is the result of encoding all slices of a coded picture. After initialising for the first slice of a coded picture as specified in subclause 9.3.4.1, BinCountsInNALunits is incremented as specified in subclauses 9.3.4.2, 9.3.4.4, and 9.3.4.5.

Outputs of this process are zero or more bytes appended to the NAL unit.

Let the variable k be set equal to Ceil((Ceil(3 * (32 * BinCountsInNALunits − RawMbBits * PicSizeInMbs) ÷ 1024) − NumBytesInVclNALunits) ÷ 3). Depending on the variable k the following applies:

– If k is less than or equal to 0, no cabac_zero_word is appended to the NAL unit.

– Otherwise (k is greater than 0), the 3-byte sequence 0x000003 is appended k times to the NAL unit after encapsulation, where the first two bytes 0x0000 represent a cabac_zero_word and the third byte 0x03 represents an emulation_prevention_three_byte.

Annex A

Profiles and levels

(This annex forms an integral part of this Recommendation | International Standard)

Profiles and levels specify restrictions on bitstreams and hence limits on the capabilities needed to decode the bitstreams. Profiles and levels may also be used to indicate interoperability points between individual decoder implementations.

NOTE 1 – This Recommendation | International Standard does not include individually selectable "options" at the decoder, as this would increase interoperability difficulties.

Each profile specifies a subset of algorithmic features and limits that shall be supported by all decoders conforming to that profile.

NOTE 2 – Encoders are not required to make use of any particular subset of features supported in a profile.

Each level specifies a set of limits on the values that may be taken by the syntax elements of this Recommendation | International Standard. The same set of level definitions is used with all profiles, but individual implementations may support a different level for each supported profile. For any given profile, levels generally correspond to decoder processing load and memory capability.

The profiles that are specified in subclause A.2 are also referred to as the profiles specified in Annex A.

A.1 Requirements on video decoder capability

Capabilities of video decoders conforming to this Recommendation | International Standard are specified in terms of the ability to decode video streams conforming to the constraints of profiles and levels specified in this annex. For each such profile, the level supported for that profile shall also be expressed.

Specific values are specified in this annex for the syntax elements profile_idc and level_idc. All other values of profile_idc and level_idc are reserved for future use by ITU-T | ISO/IEC.

NOTE – Decoders should not infer that when a reserved value of profile_idc or level_idc falls between the values specified in this Recommendation | International Standard that this indicates intermediate capabilities between the specified profiles or levels, as there are no restrictions on the method to be chosen by ITU-T | ISO/IEC for the use of such future reserved values.

A.2 Profiles

All constraints for picture parameter sets that are specified in subclauses A.2.1 to A.2.11 are constraints for picture parameter sets that are activated in the bitstream. All constraints for sequence parameter sets that are specified in subclauses A.2.1 to A.2.11 are constraints for sequence parameter sets that are activated in the bitstream.

A.2.1 Baseline profile

Bitstreams conforming to the Baseline profile shall obey the following constraints:

– Only I and P slice types may be present.
– NAL unit streams shall not contain nal_unit_type values in the range of 2 to 4, inclusive.
– Sequence parameter sets shall have frame_mbs_only_flag equal to 1.
– The syntax elements chroma_format_idc, bit_depth_luma_minus8, bit_depth_chroma_minus8, qpprime_y_zero_transform_bypass_flag, and seq_scaling_matrix_present_flag shall not be present in sequence parameter sets.
– Picture parameter sets shall have weighted_pred_flag and weighted_bipred_idc both equal to 0.
– Picture parameter sets shall have entropy_coding_mode_flag equal to 0.
– Picture parameter sets shall have num_slice_groups_minus1 in the range of 0 to 7, inclusive.
– The syntax elements transform_8x8_mode_flag, pic_scaling_matrix_present_flag, and second_chroma_qp_index_offset shall not be present in picture parameter sets.
– The syntax element level_prefix shall not be greater than 15 (when present).
– The syntax elements pcm_sample_luma[i], with i = 0..255, and pcm_sample_chroma[i], with i = 0..2 * MbWidthC * MbHeightC − 1, shall not be equal to 0 (when present).
– The level constraints specified for the Baseline profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the Baseline profile is indicated by profile_idc being equal to 66.

Decoders conforming to the Baseline profile at a specific level shall be capable of decoding all bitstreams in which profile_idc is equal to 66 or constraint_set0_flag is equal to 1 and in which level_idc and constraint_set3_flag represent a level less than or equal to the specified level.

A.2.1.1 Constrained Baseline profile

Bitstreams conforming to the Constrained Baseline profile shall obey all constraints specified in subclause A.2.1 for the Baseline profile and all constraints specified in subclause A.2.2 for the Main profile.

Conformance of a bitstream to the Constrained Baseline profile is indicated by profile_idc being equal to 66 with constraint_set1_flag being equal to 1.

NOTE – This specification of the Constrained Baseline profile is technically identical to specification of the use of the Baseline profile with constraint_set1_flag equal to 1. Thus, any existing specifications (in other documents that reference this Recommendation | International Standard) that have referred to the use of the Baseline profile with constraint_set1_flag equal to 1 should thus be interpreted as continuing in force as being technically identical to referring to the use of the Constrained Baseline profile (without any need for revision of these existing specifications to instead refer explicitly to the use of the Constrained Baseline profile).

Decoders conforming to the Constrained Baseline profile at a specific level shall be capable of decoding all bitstreams in which all of the following are true:

– profile_idc is equal to 66 or constraint_set0_flag is equal to 1,
– constraint_set1_flag is equal to 1,
– level_idc and constraint_set3_flag represent a level less than or equal to the specified level.

A.2.2 Main profile

Bitstreams conforming to the Main profile shall obey the following constraints:

– Only I, P, and B slice types may be present.
– NAL unit streams shall not contain nal_unit_type values in the range of 2 to 4, inclusive.
– Arbitrary slice order is not allowed.
– The syntax elements chroma_format_idc, bit_depth_luma_minus8, bit_depth_chroma_minus8, qpprime_y_zero_transform_bypass_flag, and seq_scaling_matrix_present_flag shall not be present in sequence parameter sets.
– Picture parameter sets shall have num_slice_groups_minus1 equal to 0 only.
– Picture parameter sets shall have redundant_pic_cnt_present_flag equal to 0 only.
– The syntax elements transform_8x8_mode_flag, pic_scaling_matrix_present_flag, and second_chroma_qp_index_offset shall not be present in picture parameter sets.

– The syntax element level_prefix shall not be greater than 15 (when present).

– The syntax elements pcm_sample_luma[i], with i = 0..255, and pcm_sample_chroma[i], with i = 0..2 * MbWidthC * MbHeightC − 1, shall not be equal to 0 (when present).

– The level constraints specified for the Main profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the Main profile is indicated by profile_idc being equal to 77.

Decoders conforming to the Main profile at a specified level shall be capable of decoding all bitstreams in which profile_idc is equal to 77 or constraint_set1_flag is equal to 1 and in which level_idc and constraint_set3_flag represent a level less than or equal to the specified level.

A.2.3 Extended profile

Bitstreams conforming to the Extended profile shall obey the following constraints:

– Sequence parameter sets shall have direct_8x8_inference_flag equal to 1.

– The syntax elements chroma_format_idc, bit_depth_luma_minus8, bit_depth_chroma_minus8, qpprime_y_zero_transform_bypass_flag, and seq_scaling_matrix_present_flag shall not be present in sequence parameter sets.

– Picture parameter sets shall have entropy_coding_mode_flag equal to 0.

– Picture parameter sets shall have num_slice_groups_minus1 in the range of 0 to 7, inclusive.

– The syntax elements transform_8x8_mode_flag, pic_scaling_matrix_present_flag, and second_chroma_qp_index_offset shall not be present in picture parameter sets.

– The syntax element level_prefix shall not be greater than 15 (when present).

– The syntax elements pcm_sample_luma[i], with i = 0..255, and pcm_sample_chroma[i], with i = 0..2 * MbWidthC * MbHeightC − 1, shall not be equal to 0 (when present).

– The level constraints specified for the Extended profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the Extended profile is indicated by profile_idc being equal to 88.

Decoders conforming to the Extended profile at a specified level shall be capable of decoding all bitstreams in which profile_idc is equal to 88 or constraint_set2_flag is equal to 1 and in which level_idc represents a level less than or equal to specified level.

Decoders conforming to the Extended profile at a specified level shall also be capable of decoding all bitstreams in which profile_idc is equal to 66 or constraint_set0_flag is equal to 1, in which level_idc and constraint_set3_flag represent a level less than or equal to the specified level.

A.2.4 High profile

Bitstreams conforming to the High profile shall obey the following constraints:

– Only I, P, and B slice types may be present.

– NAL unit streams shall not contain nal_unit_type values in the range of 2 to 4, inclusive.

– Arbitrary slice order is not allowed.

– Picture parameter sets shall have num_slice_groups_minus1 equal to 0 only.

– Picture parameter sets shall have redundant_pic_cnt_present_flag equal to 0 only.

– Sequence parameter sets shall have chroma_format_idc in the range of 0 to 1 inclusive.

– Sequence parameter sets shall have bit_depth_luma_minus8 equal to 0 only.

– Sequence parameter sets shall have bit_depth_chroma_minus8 equal to 0 only.

– Sequence parameter sets shall have qpprime_y_zero_transform_bypass_flag equal to 0 only.

– The level constraints specified for the High profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the High profile is indicated by profile_idc being equal to 100. Decoders conforming to the High profile at a specific level shall be capable of decoding all bitstreams in which either or both of the following conditions are true:

– (profile_idc is equal to 77 or constraint_set1_flag is equal to 1) and the combination of level_idc and constraint_set3_flag represent a level less than or equal to the specified level,

– profile_idc is equal to 100 and level_idc represents a level less than or equal to the specified level.

NOTE – The value 100 for profile_idc indicates that the bitstream conforms to the High profile as specified in this subclause. When profile_idc is equal to 100 and constraint_set3_flag is equal to 1, this indicates that the bitstream conforms to the High profile and additionally conforms to the constraints specified for the High 10 Intra profile in subclause A.2.8. For example, such a bitstream must have bit_depth_luma_minus8 equal to 0, have bit_depth_chroma_minus8 equal to 0, obey the MinCR, MaxBR and MaxCPB constraints of the High profile, contain only IDR pictures, have max_num_ref_frames equal to 0, have dpb_output_delay equal to 0, and obey the maximum slice size constraint of the High 10 Intra profile.

A.2.4.1 Progressive High profile

Bitstreams conforming to the Progressive High profile shall obey all constraints specified in A.2.4 for the High profile, and shall additionally obey the constraint that sequence parameter sets shall have frame_mbs_only_flag equal to 1.

Conformance of a bitstream to the Progressive High profile is indicated by profile_idc being equal to 100 with constraint_set4_flag being equal to 1.

Decoders conforming to the Progressive High profile at a specific level shall be capable of decoding all bitstreams in which one or more of the following conditions is true:

- (profile_idc is equal to 66 or constraint_set0_flag is equal to 1), constraint_set1_flag is equal to 1, and the combination of level_idc and constraint_set3_flag represents a level less than or equal to the specified level.
- profile_idc is equal to 77, constraint_set0_flag is equal to 1, and the combination of level_idc and constraint_set3_flag represents a level less than or equal to the specified level.
- profile_idc is equal to 77, constraint_set4_flag is equal to 1, and the combination of level_idc and constraint_set3_flag represents a level less than or equal to the specified level.
- profile_idc is equal to 88, constraint_set1_flag is equal to 1, constraint_set4_flag is equal to 1, and the combination of level_idc and constraint_set3_flag represents a level less than or equal to the specified level.
- profile_idc is equal to 100, constraint_set4_flag is equal to 1, and level_idc represents a level less than or equal to the specified level.

A.2.5 High 10 profile

Bitstreams conforming to the High 10 profile shall obey the following constraints:

- Only I, P, and B slice types may be present.
- NAL unit streams shall not contain nal_unit_type values in the range of 2 to 4, inclusive.
- Arbitrary slice order is not allowed.
- Picture parameter sets shall have num_slice_groups_minus1 equal to 0 only.
- Picture parameter sets shall have redundant_pic_cnt_present_flag equal to 0 only.
- Sequence parameter sets shall have chroma_format_idc in the range of 0 to 1 inclusive.
- Sequence parameter sets shall have bit_depth_luma_minus8 in the range of 0 to 2 inclusive.
- Sequence parameter sets shall have bit_depth_chroma_minus8 in the range of 0 to 2 inclusive.
- Sequence parameter sets shall have qpprime_y_zero_transform_bypass_flag equal to 0 only.
- The level constraints specified for the High 10 profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the High 10 profile is indicated by profile_idc being equal to 110. Decoders conforming to the High 10 profile at a specific level shall be capable of decoding all bitstreams in which either or both of the following conditions are true:

- (profile_idc is equal to 77 or constraint_set1_flag is equal to 1) and the combination of level_idc and constraint_set3_flag represent a level less than or equal to the specified level,
- profile_idc is equal to 100 or 110 and level_idc represents a level less than or equal to the specified level.

A.2.6 High 4:2:2 profile

Bitstreams conforming to the High 4:2:2 profile shall obey the following constraints:

- Only I, P, and B slice types may be present.
- NAL unit streams shall not contain nal_unit_type values in the range of 2 to 4, inclusive.
- Arbitrary slice order is not allowed.
- Picture parameter sets shall have num_slice_groups_minus1 equal to 0 only.
- Picture parameter sets shall have redundant_pic_cnt_present_flag equal to 0 only.
- Sequence parameter sets shall have chroma_format_idc in the range of 0 to 2 inclusive.

– Sequence parameter sets shall have bit_depth_luma_minus8 in the range of 0 to 2 inclusive.
– Sequence parameter sets shall have bit_depth_chroma_minus8 in the range of 0 to 2 inclusive.
– Sequence parameter sets shall have qpprime_y_zero_transform_bypass_flag equal to 0 only.
– The level constraints specified for the High 4:2:2 profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the High 4:2:2 profile is indicated by profile_idc being equal to 122. Decoders conforming to the High 4:2:2 profile at a specific level shall be capable of decoding all bitstreams in which either or both of the following conditions are true:

– (profile_idc is equal to 77 or constraint_set1_flag is equal to 1) and the combination of level_idc and constraint_set3_flag represent a level less than or equal to the specified level,
– profile_idc is equal to 100, 110, or 122 and level_idc represents a level less than or equal to the specified level.

A.2.7 High 4:4:4 Predictive profile

Bitstreams conforming to the High 4:4:4 Predictive profile shall obey the following constraints:

– Only I, P, B slice types may be present.
– NAL unit streams shall not contain nal_unit_type values in the range of 2 to 4, inclusive.
– Arbitrary slice order is not allowed.
– Picture parameter sets shall have num_slice_groups_minus1 equal to 0 only.
– Picture parameter sets shall have redundant_pic_cnt_present_flag equal to 0 only.
– Sequence parameter sets shall have bit_depth_luma_minus8 in the range of 0 to 6 inclusive.
– Sequence parameter sets shall have bit_depth_chroma_minus8 in the range of 0 to 6 inclusive.
– The level constraints specified for the High 4:4:4 Predictive profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the High 4:4:4 Predictive profile is indicated by profile_idc being equal to 244. Decoders conforming to the High 4:4:4 Predictive profile at a specific level shall be capable of decoding all bitstreams in which either or both of the following conditions are true:

– (profile_idc is equal to 77 or constraint_set1_flag is equal to 1) and the combination of level_idc and constraint_set3_flag represent a level less than or equal to the specified level,
– profile_idc is equal to 44, 100, 110, 122, or 244 and the value of level_idc represents a level less than or equal to the specified level.

A.2.8 High 10 Intra profile

Bitstreams conforming to the High 10 Intra profile shall obey the following constraints:

– All constraints specified in subclause A.2.5 for the High 10 profile shall be obeyed.
– All pictures shall be IDR pictures.
– Sequence parameter sets shall have max_num_ref_frames equal to 0.
– When vui_parameters_present_flag is equal to 1 and bitstream_restriction_flag is equal to 1, sequence parameter sets shall have max_num_reorder_frames equal to 0.
– When vui_parameters_present_flag is equal to 1 and bitstream_restriction_flag is equal to 1, sequence parameter sets shall have max_dec_frame_buffering equal to 0.
– Picture timing SEI messages, whether present in the bitstream (by non-VCL NAL units) or conveyed equivalently by other means not specified in this Recommendation | International Standard, shall have dpb_output_delay equal to 0.
– The level constraints specified for the High 10 Intra profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the High 10 Intra profile is indicated by constraint_set3_flag being equal to 1 with profile_idc equal to 110. Decoders conforming to the High 10 Intra profile at a specific level shall be capable of decoding all bitstreams in which all of the following conditions are true:

– profile_idc is equal to 100 or 110,
– constraint_set3_flag is equal to 1,
– level_idc represents a level less than or equal to the specified level.

NOTE 1 – The value 100 for profile_idc indicates that the bitstream conforms to the High profile as specified in subclause A.2.4. When profile_idc is equal to 100 and constraint_set3_flag is equal to 1, this indicates that the bitstream conforms to the High profile and additionally conforms to the constraints specified for the High 10 Intra profile in this subclause. For example, such a bitstream must have bit_depth_luma_minus8 equal to 0, have bit_depth_chroma_minus8 equal to 0, obey the MinCR, MaxBR and MaxCPB constraints of the High profile, contain only IDR pictures, have max_num_ref_frames equal to 0, have dpb_output_delay equal to 0, and obey the maximum slice size constraint of the High 10 Intra profile.

The operation of the deblocking filter process specified in subclause 8.7 is not required for decoder conformance to the High 10 Intra profile.

NOTE 2 – The deblocking filter process specified in subclause 8.7 or some similar post-processing filter should be performed, although this is not a requirement for decoder conformance to the High 10 Intra profile. The syntax elements sent by an encoder for control of the deblocking filter process specified in subclause 8.7 are considered only as advisory information for decoders conformance to the High 10 Intra profile. However, the application of the deblocking filter process specified in subclause 8.7 is required for decoder conformance to the High 10, High 4:2:2, and High 4:4:4 Predictive profiles when decoding bitstreams that conform to the High 10 Intra profile.

A.2.9 High 4:2:2 Intra profile

Bitstreams conforming to the High 4:2:2 Intra profile shall obey the following constraints:

– All constraints specified in subclause A.2.6 for the High 4:2:2 profile shall be obeyed.

– All pictures shall be IDR pictures.

– Sequence parameter sets shall have max_num_ref_frames equal to 0.

– When vui_parameters_present_flag is equal to 1 and bitstream_restriction_flag is equal to 1, sequence parameter sets shall have max_num_reorder_frames equal to 0.

– When vui_parameters_present_flag is equal to 1 and bitstream_restriction_flag is equal to 1, sequence parameter sets shall have max_dec_frame_buffering equal to 0.

– Picture timing SEI messages, whether present in the bitstream (by non-VCL NAL units) or conveyed equivalently by other means not specified in this Recommendation | International Standard, shall have dpb_output_delay equal to 0.

– The level constraints specified for the High 4:2:2 Intra profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the High 4:2:2 Intra profile is indicated by constraint_set3_flag being equal to 1 with profile_idc equal to 122. Decoders conforming to the High 4:2:2 Intra profile at a specific level shall be capable of decoding all bitstreams in which all of the following conditions are true:

– profile_idc is equal to 100, 110, or 122,

– constraint_set3_flag is equal to 1,

– level_idc represents a level less than or equal to the specified level.

The operation of the deblocking filter process specified in subclause 8.7 is not required for decoder conformance to the High 4:2:2 Intra profile.

NOTE – The deblocking filter process specified in subclause 8.7 or some similar post-processing filter should be performed, although this is not a requirement for decoder conformance to the High 4:2:2 Intra profile. The syntax elements sent by an encoder for control of the deblocking filter process specified in subclause 8.7 are considered only as advisory information for decoders conformance to the High 4:2:2 Intra profile. However, the application of the deblocking filter process specified in subclause 8.7 is required for decoder conformance to the High 4:2:2, and High 4:4:4 Predictive profiles when decoding bitstreams that conform to the High 4:2:2 Intra profile.

A.2.10 High 4:4:4 Intra profile

Bitstreams conforming to the High 4:4:4 Intra profile shall obey the following constraints:

– All constraints specified in subclause A.2.7 for the High 4:4:4 Predictive profile shall be obeyed.

– All pictures shall be IDR pictures.

– Sequence parameter sets shall have max_num_ref_frames equal to 0.

– When vui_parameters_present_flag is equal to 1 and bitstream_restriction_flag is equal to 1, sequence parameter sets shall have max_num_reorder_frames equal to 0.

– When vui_parameters_present_flag is equal to 1 and bitstream_restriction_flag is equal to 1, sequence parameter sets shall have max_dec_frame_buffering equal to 0.

– Picture timing SEI messages, whether present in the bitstream (by non-VCL NAL units) or conveyed equivalently by other means not specified in this Recommendation | International Standard, shall have dpb_output_delay equal to 0.

– The level constraints specified for the High 4:4:4 Intra profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the High 4:4:4 Intra profile is indicated by constraint_set3_flag being equal to 1 with profile_idc equal to 244. Decoders conforming to the High 4:4:4 Intra profile at a specific level shall be capable of decoding all bitstreams in which all of the following conditions are true:

– profile_idc is equal to 44, 100, 110, 122, or 244,

– constraint_set3_flag is equal to 1,

– level_idc represents a level less than or equal to the specified level.

The operation of the deblocking filter process specified in subclause 8.7 is not required for decoder conformance to the High 4:4:4 Intra profile.

NOTE – The deblocking filter process specified in subclause 8.7 or some similar post-processing filter should be performed, although this is not a requirement for decoder conformance to the High 4:4:4 Intra and CAVLC 4:4:4 Intra profiles. The syntax elements sent by an encoder for control of the deblocking filter process specified in subclause 8.7 are considered only as advisory information for decoders conformance to the High 4:4:4 Intra and CAVLC 4:4:4 Intra profiles. However, the application of the deblocking filter process specified in subclause 8.7 is required for decoder conformance to the High 4:4:4 Predictive profile when decoding bitstreams that conform to the High 4:4:4 Intra and CAVLC 4:4:4 Intra profiles.

A.2.11 CAVLC 4:4:4 Intra profile

Bitstreams conforming to the CAVLC 4:4:4 Intra profile shall obey the following constraints:

– All constraints specified in subclause A.2.10 for the High 4:4:4 Intra profile shall be obeyed.

– Picture parameter sets shall have entropy_coding_mode_flag equal to 0.

– The level constraints specified for the CAVLC 4:4:4 Intra profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the CAVLC 4:4:4 Intra profile is indicated by profile_idc being equal to 44. Decoders conforming to the CAVLC 4:4:4 Intra profile at a specific level shall be capable of decoding all bitstreams in which all of the following conditions are true:

– profile_idc is equal to 44,

– level_idc represents a level less than or equal to the specified level.

The operation of the deblocking filter process specified in subclause 8.7 is not required for decoder conformance to the CAVLC 4:4:4 Intra profile.

NOTE – The deblocking filter process specified in subclause 8.7 or some similar post-processing filter should be performed, although this is not a requirement for decoder conformance to the High 4:4:4 Intra and CAVLC 4:4:4 Intra profiles. The syntax elements sent by an encoder for control of the deblocking filter process specified in subclause 8.7 are considered only as advisory information for decoders conformance to the High 4:4:4 Intra and CAVLC 4:4:4 Intra profiles. However, the application of the deblocking filter process specified in subclause 8.7 is required for decoder conformance to the High 4:4:4 Predictive profile when decoding bitstreams that conform to the High 4:4:4 Intra and CAVLC 4:4:4 Intra profiles.

A.3 Levels

The following is specified for expressing the constraints in this annex.

– Let access unit n be the n-th access unit in decoding order with the first access unit being access unit 0.

– Let picture n be the primary coded picture or the corresponding decoded picture of access unit n.

Let the variable fR be derived as follows:

– If picture n is a frame, fR is set equal to 1 ÷ 172.

– Otherwise (picture n is a field), fR is set equal to 1 ÷ (172 * 2).

A.3.1 Level limits common to the Baseline, Constrained Baseline, Main, and Extended profiles

Bitstreams conforming to the Baseline, Constrained Baseline, Main, or Extended profiles at a specified level shall obey the following constraints:

a) The nominal removal time of access unit n with n > 0 from the CPB as specified in subclause C.1.2, satisfies the constraint that $t_{r,n}(n) - t_r(n - 1)$ is greater than or equal to Max(PicSizeInMbs ÷ MaxMBPS, fR), where MaxMBPS is the value specified in Table A-1 that applies to picture n − 1 and PicSizeInMbs is the number of macroblocks in picture n − 1.

b) The difference between consecutive output times of pictures from the DPB as specified in subclause C.2.2, satisfies the constraint that $\Delta t_{o,dpb}(n) >=$ Max(PicSizeInMbs ÷ MaxMBPS, fR), where MaxMBPS is the value specified in Table A-1 for picture n and PicSizeInMbs is the number of macroblocks of picture n, provided that picture n is a picture that is output and is not the last picture of the bitstream that is output.

c) The sum of the NumBytesInNALunit variables for access unit 0 is less than or equal to 384 *(Max(PicSizeInMbs, fR * MaxMBPS) + MaxMBPS * ($t_r(0) - t_{r,n}(0)$)) ÷ MinCR, where MaxMBPS and MinCR are the values specified in Table A-1 that apply to picture 0 and PicSizeInMbs is the number of macroblocks in picture 0.

d) The sum of the NumBytesInNALunit variables for access unit n with n > 0 is less than or equal to 384 * MaxMBPS * ($t_r(n) - t_r(n - 1)$) ÷ MinCR, where MaxMBPS and MinCR are the values specified in Table A-1 that apply to picture n.

e) PicWidthInMbs * FrameHeightInMbs <= MaxFS, where MaxFS is specified in Table A-1

f) PicWidthInMbs <= Sqrt(MaxFS * 8)

g) FrameHeightInMbs <= Sqrt(MaxFS * 8)

h) max_dec_frame_buffering <= MaxDpbFrames, where MaxDpbFrames is equal to Min(MaxDpbMbs / (PicWidthInMbs * FrameHeightInMbs), 16) and MaxDpbMbs is given in Table A-1.

i) For the VCL HRD parameters, BitRate[SchedSelIdx] <= 1000 * MaxBR and CpbSize[SchedSelIdx] <= 1000 * MaxCPB for at least one value of SchedSelIdx, where BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are given as follows:

   – If vcl_hrd_parameters_present_flag is equal to 1, BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are given by Equations E-37 and E-38, respectively, using the syntax elements of the hrd_parameters( ) syntax structure that immediately follows vcl_hrd_parameters_present_flag.

   – Otherwise (vcl_hrd_parameters_present_flag is equal to 0), BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are inferred as specified in subclause E.2.2 for VCL HRD parameters.

   MaxBR and MaxCPB are specified in Table A-1 in units of 1000 bits/s and 1000 bits, respectively. The bitstream shall satisfy these conditions for at least one value of SchedSelIdx in the range 0 to cpb_cnt_minus1, inclusive.

j) For the NAL HRD parameters, BitRate[SchedSelIdx] <= 1200 * MaxBR and CpbSize[SchedSelIdx] <= 1200 * MaxCPB for at least one value of SchedSelIdx, where BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are given as follows:

   – If nal_hrd_parameters_present_flag is equal to 1, BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are given by Equations E-37 and E-38, respectively, using the syntax elements of the hrd_parameters( ) syntax structure that immediately follows nal_hrd_parameters_present_flag.

   – Otherwise (nal_hrd_parameters_present_flag is equal to 0), BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are inferred as specified in subclause E.2.2 for NAL HRD parameters.

   MaxBR and MaxCPB are specified in Table A-1 in units of 1200 bits/s and 1200 bits, respectively. The bitstream shall satisfy these conditions for at least one value of SchedSelIdx in the range 0 to cpb_cnt_minus1.

k) The vertical motion vector component range for luma motion vectors does not exceed MaxVmvR in units of luma frame samples, where MaxVmvR is specified in Table A-1

   NOTE 1 – When chroma_format_idc is equal to 1 and the current macroblock is a field macroblock, the motion vector component range for chroma motion vectors may exceed MaxVmvR in units of luma frame samples, due to the method of deriving chroma motion vectors as specified in subclause 8.4.1.4.

l) The horizontal motion vector range does not exceed the range of −2048 to 2047.75, inclusive, in units of luma samples

m) Let setOf2Mb be the set of unsorted pairs of macroblocks that contains the unsorted pairs of macroblocks (mbA, mbB) of a coded video sequence for which any of the following conditions are true:

   – mbA and mbB are macroblocks that belong to the same slice and are consecutive in decoding order,

   – arbitrary slice order is not allowed, mbA is the last macroblock (in decoding order) of a slice, and mbB is the first macroblock (in decoding order) of the next slice in decoding order,

      NOTE 2 – The macroblocks mbA and mbB can belong to different pictures.

   – arbitrary slice order is allowed, mbA is the last macroblock (in decoding order) of a slice of a particular picture, and mbB is the first macroblock (in decoding order) of any other slice of the same picture,

   – arbitrary slice order is allowed, mbA is the last macroblock (in decoding order) of a slice of a particular picture, and mbB is the first macroblock (in decoding order) of any slice of the next picture in decoding order.

For each unsorted pair of macroblocks (mbA, mbB) of the set setOf2Mb, the total number of motion vectors (given by the sum of the number of motion vectors for macroblock mbA and the number of motion vectors for macroblock mbB) does not exceed MaxMvsPer2Mb, where MaxMvsPer2Mb is specified in Table A-1. The number of motion vectors for each macroblock is the value of the variable MvCnt after the completion of the intra or inter prediction process for the macroblock.

NOTE 3 – The constraint specifies that the total number of motion vectors for two consecutive macroblocks in decoding order must not exceed MaxMvsPer2Mb. When arbitrary slice order is allowed, it is specified that this constraint must also be obeyed when slices of a picture are reordered, e.g., during transmission.

n) The number of bits of macroblock_layer( ) data for any macroblock is not greater than 3200. Depending on entropy_coding_mode_flag, the bits of macroblock_layer( ) data are counted as follows:

   – If entropy_coding_mode_flag is equal to 0, the number of bits of macroblock_layer( ) data is given by the number of bits in the macroblock_layer( ) syntax structure for a macroblock.

   – Otherwise (entropy_coding_mode_flag is equal to 1), the number of bits of macroblock_layer( ) data for a macroblock is given by the number of times read_bits(1) is called in subclauses 9.3.3.2.2 and 9.3.3.2.3 when parsing the macroblock_layer( ) associated with the macroblock.

Table A-1 specifies the limits for each level. A definition of all levels identified in the "Level number" column of Table A-1 is specified for the Baseline, Constrained Baseline, Main, and Extended profiles. Each entry in Table A-1 indicates, for the level corresponding to the row of the table, the absence or value of a limit that is imposed by the variable corresponding to the column of the table, as follows:

– If the table entry is marked as "-", no limit is imposed by the value of the variable as a requirement of bitstream conformance to the profile at the specified level.

– Otherwise, the table entry specifies the value of the variable for the associated limit that is imposed as a requirement of bitstream conformance to the profile at the specified level.

For purposes of comparison of level capabilities, a level shall be considered to be a lower (higher) level than some other level if the level appears nearer to the top (bottom) row of Table A-1 than the other level.

In bitstreams conforming to the Baseline, Constrained Baseline, Main, or Extended profiles, the conformance of the bitstream to a specified level is indicated by the syntax elements level_idc and constraint_set3_flag as follows:

– If level_idc is equal to 11 and constraint_set3_flag is equal to 1, the indicated level is level 1b.

– Otherwise (level_idc is not equal to 11 or constraint_set3_flag is not equal to 1), level_idc is equal to a value of ten times the level number (of the indicated level) specified in Table A-1.

Table A-1 – Level limits

Level number	Max macroblock processing rate MaxMBPS (MB/s)	Max frame size MaxFS (MBs)	Max decoded picture buffer size MaxDpbMbs (MBs)	Max video bit rate MaxBR (1000 bits/s, 1200 bits/s, cpbBrVclFactor bits/s, or cpbBrNalFactor bits/s)	Max CPB size MaxCPB (1000 bits, 1200 bits, cpbBrVclFactor bits, or cpbBrNalFactor bits)	Vertical MV component range MaxVmvR (luma frame samples)	Min compression ratio MinCR	Max number of motion vectors per two consecutive MBs MaxMvsPer2Mb
1	1 485	99	396	64	175	[−64,+63.75]	2	-
1b	1 485	99	396	128	350	[−64,+63.75]	2	-
1.1	3 000	396	900	192	500	[−128,+127.75]	2	-
1.2	6 000	396	2 376	384	1 000	[−128,+127.75]	2	-
1.3	11 880	396	2 376	768	2 000	[−128,+127.75]	2	-
2	11 880	396	2 376	2 000	2 000	[−128,+127.75]	2	-
2.1	19 800	792	4 752	4 000	4 000	[−256,+255.75]	2	-
2.2	20 250	1 620	8 100	4 000	4 000	[−256,+255.75]	2	-
3	40 500	1 620	8 100	10 000	10 000	[−256,+255.75]	2	32
3.1	108 000	3 600	18 000	14 000	14 000	[−512,+511.75]	4	16
3.2	216 000	5 120	20 480	20 000	20 000	[−512,+511.75]	4	16
4	245 760	8 192	32 768	20 000	25 000	[−512,+511.75]	4	16
4.1	245 760	8 192	32 768	50 000	62 500	[−512,+511.75]	2	16
4.2	522 240	8 704	34 816	50 000	62 500	[−512,+511.75]	2	16
5	589 824	22 080	110 400	135 000	135 000	[−512,+511.75]	2	16
5.1	983 040	36 864	184 320	240 000	240 000	[−512,+511.75]	2	16
5.2	2 073 600	36 864	184 320	240 000	240 000	[−512,+511.75]	2	16

Levels with non-integer level numbers in Table A-1 are referred to as "intermediate levels".

NOTE 4 – All levels have the same status, but some applications may choose to use only the integer-numbered levels.

Informative subclause A.3.4 shows the effect of these limits on frame rates for several example picture formats.

A.3.2 Level limits common to the High, Progressive High, High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, and CAVLC 4:4:4 Intra profiles

Bitstreams conforming to the High, Progressive High, High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, or CAVLC 4:4:4 Intra profiles at a specified level shall obey the following constraints:

a) The nominal removal time of access unit n (with n > 0) from the CPB as specified in subclause C.1.2, satisfies the constraint that $t_{r,n}(n) - t_r(n - 1)$ is greater than or equal to Max(PicSizeInMbs ÷ MaxMBPS, fR), where MaxMBPS is the value specified in Table A-1 that applies to picture n − 1, and PicSizeInMbs is the number of macroblocks in picture n − 1.

b) The difference between consecutive output times of pictures from the DPB as specified in subclause C.2.2, satisfies the constraint that $\Delta t_{o,dpb}(n) >=$ Max(PicSizeInMbs ÷ MaxMBPS, fR), where MaxMBPS is the value specified in Table A-1 for picture n, and PicSizeInMbs is the number of macroblocks of picture n, provided that picture n is a picture that is output and is not the last picture of the bitstream that is output.

c) PicWidthInMbs * FrameHeightInMbs <= MaxFS, where MaxFS is specified in Table A-1

d) PicWidthInMbs <= Sqrt(MaxFS * 8)

e) FrameHeightInMbs <= Sqrt(MaxFS * 8)

f) max_dec_frame_buffering <= MaxDpbFrames, where MaxDpbFrames is equal to Min(MaxDpbMbs / (PicWidthInMbs * FrameHeightInMbs), 16) and MaxDpbMbs is specified in Table A-1.

g) The vertical motion vector component range does not exceed MaxVmvR in units of luma frame samples, where MaxVmvR is specified in Table A-1.

h) The horizontal motion vector range does not exceed the range of −2048 to 2047.75, inclusive, in units of luma samples.

i) Let setOf2Mb be the set of unsorted pairs of macroblocks that contains the unsorted pairs of macroblocks (mbA, mbB) of a coded video sequence for which any of the following conditions are true:

– mbA and mbB are macroblocks that belong to the same slice and are consecutive in decoding order,

– separate_colour_plane_flag is equal to 0, mbA is the last macroblock (in decoding order) of a slice, and mbB is the first macroblock (in decoding order) of the next slice in decoding order,

– separate_colour_plane_flag is equal to 1, mbA is the last macroblock (in decoding order) of a slice with a particular value of colour_plane_id, and mbB is the first macroblock (in decoding order) of the next slice with the same value of colour_plane_id in decoding order.

NOTE 1 – In the two above conditions, the macroblocks mbA and mbB can belong to different pictures.

For each unsorted pair of macroblocks (mbA, mbB) of the set setOf2Mb, the total number of motion vectors (given by the sum of the number of motion vectors for macroblock mbA and the number of motion vectors for macroblock mbB) does not exceed MaxMvsPer2Mb, where MaxMvsPer2Mb is specified in Table A-1. The number of motion vectors for each macroblock is the value of the variable MvCnt after the completion of the intra or inter prediction process for the macroblock.

NOTE 2 – When separate_colour_plane_flag is equal to 0, the constraint specifies that the total number of motion vectors for two consecutive macroblocks in decoding order must not exceed MaxMvsPer2Mb. When separate_colour_plane_flag is equal to 1, the constraint specifies that the total number of motion vectors for two consecutive macroblocks (in decoding order) with the same value of colour_plane_id must not exceed MaxMvsPer2Mb. For macroblocks that are consecutive in decoding order but are associated with a different value of colour_plane_id, no constraint for the total number of motion vectors is specified.

j) The number of bits of macroblock_layer( ) data for any macroblock is not greater than 128 + RawMbBits. Depending on entropy_coding_mode_flag, the bits of macroblock_layer( ) data are counted as follows:

– If entropy_coding_mode_flag is equal to 0, the number of bits of macroblock_layer( ) data is given by the number of bits in the macroblock_layer( ) syntax structure for a macroblock.

– Otherwise (entropy_coding_mode_flag is equal to 1), the number of bits of macroblock_layer( ) data for a macroblock is given by the number of times read_bits(1) is called in subclauses 9.3.3.2.2 and 9.3.3.2.3 when parsing the macroblock_layer( ) associated with the macroblock.

Table A-1 specifies the limits for each level. A definition of all levels identified in the "Level number" column of Table A-1 is specified for the High, Progressive High, High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, and CAVLC 4:4:4 Intra profiles. Each entry in Table A-1 indicates, for the level corresponding to the row of the table, the absence or value of a limit that is imposed by the variable corresponding to the column of the table, as follows:

– If the table entry is marked as "-", no limit is imposed by the value of the variable as a requirement of bitstream conformance to the profile at the specified level.

– Otherwise, the table entry specifies the value of the variable for the associated limit that is imposed as a requirement of bitstream conformance to the profile at the specified level.

The use of the MinCR parameter column of Table A-1 for the High, High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, and CAVLC 4:4:4 Intra profiles is specified in subclause A.3.3.

In bitstreams conforming to the High, High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, or CAVLC 4:4:4 Intra profiles, the conformance of the bitstream to a specified level is indicated by the syntax element level_idc as follows:

– If level_idc is equal to 9, the indicated level is level 1b.

– Otherwise (level_idc is not equal to 9), level_idc is equal to a value of ten times the level number (of the indicated level) specified in Table A-1.

A.3.3 Profile-specific level limits

a) In bitstreams conforming to the Main, High, Progressive High, High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, or CAVLC 4:4:4 Intra profiles, the removal time of access unit 0 shall satisfy the constraint that the number of slices in picture 0 is less than or equal to $(\text{Max}(\text{PicSizeInMbs}, \text{fR} * \text{MaxMBPS}) + \text{MaxMBPS} * (t_r(0) - t_{r,n}(0))) \div \text{SliceRate}$, where MaxMBPS and SliceRate are the values specified in Tables A-1 and A-4, respectively, that apply to picture 0 and PicSizeInMbs is the number of macroblocks in picture 0.

b) In bitstreams conforming to the Main, High, Progressive High, High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, or CAVLC 4:4:4 Intra profiles, the difference between consecutive removal times of access units n and n − 1 with n > 0 shall satisfy the constraint that the number of slices in picture n is less than or equal to $\text{MaxMBPS} * (t_r(n) - t_r(n - 1)) \div \text{SliceRate}$, where MaxMBPS and SliceRate are the values specified in Tables A-1 and A-4, respectively, that apply to picture n.

c) In bitstreams conforming to the Main, High, Progressive High, High 10, High 4:2:2, High 4:4:4 Predictive profiles, sequence parameter sets shall have direct_8x8_inference_flag equal to 1 for the levels specified in Table A-4.

NOTE 1 – direct_8x8_inference_flag is not relevant to the Baseline, Constrained Baseline, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, and CAVLC 4:4:4 Intra profiles as these profiles do not allow B slice types, and direct_8x8_inference_flag is equal to 1 for all levels of the Extended profile.

d) In bitstreams conforming to the Main, High, High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, CAVLC 4:4:4 Intra, or Extended profiles, sequence parameter sets shall have frame_mbs_only_flag equal to 1 for the levels specified in Table A-4 for the Main, High, High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, or CAVLC 4:4:4 Intra profiles and in Table A-5 for the Extended profile.

NOTE 2 – frame_mbs_only_flag is equal to 1 for all levels of the Baseline, Constrained Baseline, and Progressive High profiles (specified in subclauses A.2.1, A.2.1.1, and A.2.4.1, respectively).

e) In bitstreams conforming to the Main, High, Progressive High, High 10, High 4:2:2, High 4:4:4 Predictive, or Extended profiles, the value of sub_mb_type[mbPartIdx] with mbPartIdx = 0..3 in B macroblocks with mb_type equal to B_8x8 shall not be equal to B_Bi_8x4, B_Bi_4x8, or B_Bi_4x4 for the levels in which MinLumaBiPredSize is shown as 8x8 in Table A-4 for the Main, High, Progressive High, High 10, High 4:2:2, High 4:4:4 Predictive profiles and in Table A-5 for the Extended profile.

f) In bitstreams conforming to the Baseline, Constrained Baseline, or Extended profiles, $(\text{xInt}_{max} - \text{xInt}_{min} + 6) * (\text{yInt}_{max} - \text{yInt}_{min} + 6) <= \text{MaxSubMbRectSize}$ in macroblocks coded with mb_type equal to P_8x8, P_8x8ref0 or B_8x8 for all invocations of the process specified in subclause 8.4.2.2.1 used to generate the predicted luma sample array for a single reference picture list (reference picture list 0 or reference picture list 1) for each 8x8 sub-macroblock with the macroblock partition index mbPartIdx, where NumSubMbPart(sub_mb_type[mbPartIdx]) > 1, where MaxSubMbRectSize is specified in Table A-3 for the Baseline and Constrained Baseline profiles and in Table A-5 for the Extended profile and

– xInt_{min} is the minimum value of xInt_L among all luma sample predictions for the sub-macroblock
– xInt_{max} is the maximum value of xInt_L among all luma sample predictions for the sub-macroblock
– yInt_{min} is the minimum value of yInt_L among all luma sample predictions for the sub-macroblock
– yInt_{max} is the maximum value of yInt_L among all luma sample predictions for the sub-macroblock

g) In bitstreams conforming to the High, Progressive High, High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, or CAVLC 4:4:4 Intra profiles, for the VCL HRD parameters, BitRate[SchedSelIdx] <= cpbBrVclFactor * MaxBR and CpbSize[SchedSelIdx] <= cpbBrVclFactor * MaxCPB for at least one value of SchedSelIdx, where cpbBrVclFactor is specified in Table A-2 and BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are given as follows:

– If vcl_hrd_parameters_present_flag is equal to 1, BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are given by Equations E-37 and E-38, respectively, using the syntax elements of the hrd_parameters( ) syntax structure that immediately follows vcl_hrd_parameters_present_flag.
– Otherwise (vcl_hrd_parameters_present_flag is equal to 0), BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are inferred as specified in subclause E.2.2 for VCL HRD parameters.

MaxBR and MaxCPB are specified in Table A-1 in units of cpbBrVclFactor bits/s and cpbBrVclFactor bits, respectively. The bitstream shall satisfy these conditions for at least one value of SchedSelIdx in the range 0 to cpb_cnt_minus1, inclusive.

h) In bitstreams conforming to the High, Progressive High, High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, or CAVLC 4:4:4 Intra profiles, for the NAL HRD parameters, BitRate[SchedSelIdx] <= cpbBrNalFactor * MaxBR and CpbSize[SchedSelIdx] <= cpbBrNalFactor * MaxCPB for at least one value of SchedSelIdx, where cpbBrNalFactor is specified in Table A-2 and BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are given as follows:

   – If nal_hrd_parameters_present_flag is equal to 1, BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are given by Equations E-37 and E-38, respectively, using the syntax elements of the hrd_parameters( ) syntax structure that immediately follows nal_hrd_parameters_present_flag.

   – Otherwise (nal_hrd_parameters_present_flag is equal to 0), BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are inferred as specified in subclause E.2.2 for NAL HRD parameters.

   MaxBR and MaxCPB are specified in Table A-1 in units of cpbBrNalFactor bits/s and cpbBrNalFactor bits, respectively. The bitstream shall satisfy these conditions for at least one value of SchedSelIdx in the range 0 to cpb_cnt_minus1, inclusive.

i) In bitstreams conforming to the High or Progressive High profiles, the sum of the NumBytesInNALunit variables for access unit 0 is less than or equal to $384 * (\text{Max}(\text{PicSizeInMbs}, \text{fR} * \text{MaxMBPS}) + \text{MaxMBPS} * (t_r(0) - t_{r,n}(0))) \div \text{MinCR}$, where MaxMBPS and MinCR are the values specified in Table A-1 that apply to picture 0 and PicSizeInMbs is the number of macroblocks in picture 0.

   NOTE 3 – Such a limit involving MinCR is not imposed for bitstream conformance to the High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, and CAVLC 4:4:4 Intra profiles.

j) In bitstreams conforming to the High or Progressive High profiles, the sum of the NumBytesInNALunit variables for access unit n with n > 0 is less than or equal to $384 * \text{MaxMBPS} * (t_r(n) - t_r(n - 1)) \div \text{MinCR}$, where MaxMBPS and MinCR are the values specified in Table A-1 that apply to picture n.

   NOTE 4 – Such a limit involving MinCR is not imposed for bitstream conformance to the High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, and CAVLC 4:4:4 Intra profiles.

k) In bitstreams conforming to the High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, or CAVLC 4:4:4 Intra profiles, when PicSizeInMbs is greater than 1620, the number of macroblocks in any coded slice shall not exceed MaxFS / 4, where MaxFS is specified in Table A-1.

Table A-2 – Specification of cpbBrVclFactor and cpbBrNalFactor

Profile	cpbBrVclFactor	cpbBrNalFactor
High Progressive High	1 250	1 500
High 10 High 10 Intra	3 000	3 600
High 4:2:2 High 4:2:2 Intra	4 000	4 800
High 4:4:4 Predictive High 4:4:4 Intra CAVLC 4:4:4 Intra	4 000	4 800

A.3.3.1 Level limits of the Baseline and Constrained Baseline profile

Table A-3 specifies limits for each level that are specific to bitstreams conforming to the Baseline or Constrained Baseline profiles. Each entry in Table A-3 indicates, for the level corresponding to the row of the table, the absence or value of a limit that is imposed by the variable corresponding to the column of the table, as follows:

– If the table entry is marked as "-", no limit is imposed by the value of the variable as a requirement of bitstream conformance to the profile at the specified level.

– Otherwise, the table entry specifies the value of the variable for the associated limit that is imposed as a requirement of bitstream conformance to the profile at the specified level.

Table A-3 – Baseline and Constrained Baseline profile level limits

Level number	MaxSubMbRectSize
1	576
1b	576
1.1	576
1.2	576
1.3	576
2	576
2.1	576
2.2	576
3	576
3.1	-
3.2	-
4	-
4.1	-
4.2	-
5	-
5.1	-
5.2	-

A.3.3.2 Level limits of the Main, High, Progressive High, High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, and CAVLC 4:4:4 Intra profile

Table A-4 specifies limits for each level that are specific to bitstreams conforming to the Main, High, Progressive High, High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, or CAVLC 4:4:4 Intra profiles. Each entry in Table A-4 indicates, for the level corresponding to the row of the table, the absence or value of a limit that is imposed by the variable corresponding to the column of the table, as follows:

– If the table entry is marked as "-", no limit is imposed by the value of the variable as a requirement of bitstream conformance to the profile at the specified level.

– Otherwise, the table entry specifies the value of the variable for the associated limit that is imposed as a requirement of bitstream conformance to the profile at the specified level.

NOTE – The constraints for MinLumaBiPredSize and direct_8x8_inference_flag are not relevant to the High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, and CAVLC 4:4:4 Intra profiles, as these profiles do not support B slices.

Table A-4 – Main, High, Progressive High, High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, and CAVLC 4:4:4 Intra profile level limits

Level number	SliceRate	MinLumaBiPredSize	direct_8x8_inference_flag	frame_mbs_only_flag
1	-	-	-	1
1b	-	-	-	1
1.1	-	-	-	1
1.2	-	-	-	1
1.3	-	-	-	1
2	-	-	-	1
2.1	-	-	-	-
2.2	-	-	-	-
3	22	-	1	-
3.1	60	8x8	1	-
3.2	60	8x8	1	-
4	60	8x8	1	-
4.1	24	8x8	1	-
4.2	24	8x8	1	1
5	24	8x8	1	1
5.1	24	8x8	1	1
5.2	24	8x8	1	1

A.3.3.3 Level limits of the Extended profile

Table A-5 specifies limits for each level that are specific to bitstreams conforming to the Extended profile. Each entry in Table A-5 indicates, for the level corresponding to the row of the table, the absence or value of a limit that is imposed by the variable corresponding to the column of the table, as follows:

– If the table entry is marked as "-", no limit is imposed by the value of the variable as a requirement of bitstream conformance to the profile at the specified level.

– Otherwise, the table entry specifies the value of the variable for the associated limit that is imposed as a requirement of bitstream conformance to the profile at the specified level.

Table A-5 – Extended profile level limits

Level number	MaxSubMbRectSize	MinLumaBiPredSize	frame_mbs_only_flag
1	576	-	1
1b	576	-	1
1.1	576	-	1
1.2	576	-	1
1.3	576	-	1
2	576	-	1
2.1	576	-	-
2.2	576	-	-
3	576	-	-
3.1	-	8x8	-
3.2	-	8x8	-
4	-	8x8	-
4.1	-	8x8	-
4.2	-	8x8	1
5	-	8x8	1
5.1	-	8x8	1
5.2	-	8x8	1

A.3.4 Effect of level limits on frame rate (informative)

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

Table A-6 – Maximum frame rates (frames per second) for some example frame sizes

Level:					**1**	**1b**	**1.1**	**1.2**	**1.3**	**2**	**2.1**
Max frame size (macroblocks):					**99**	**99**	**396**	**396**	**396**	**396**	**792**
Max macroblocks/second:					**1 485**	**1 485**	**3 000**	**6 000**	**11 880**	**11 880**	**19 800**
Max frame size (samples):					25 344	25 344	101 376	101 376	101 376	101 376	202 752
Max samples/second:					380 160	380 160	768 000	1 536 000	3 041 280	3 041 280	5 068 800
Format	**Luma Width**	**Luma Height**	**MBs Total**	**Luma Samples**							
SQCIF	**128**	**96**	48	12 288	30.9	30.9	62.5	125.0	172.0	172.0	172.0
QCIF	**176**	**144**	99	25 344	15.0	15.0	30.3	60.6	120.0	120.0	172.0
QVGA	**320**	**240**	300	76 800	-	-	10.0	20.0	39.6	39.6	66.0
525 SIF	**352**	**240**	330	84 480	-	-	9.1	18.2	36.0	36.0	60.0
CIF	**352**	**288**	396	101 376	-	-	7.6	15.2	30.0	30.0	50.0
525 HHR	**352**	**480**	660	168 960	-	-	-	-	-	-	30.0
625 HHR	**352**	**576**	792	202 752	-	-	-	-	-	-	25.0
VGA	**640**	**480**	1 200	307 200	-	-	-	-	-	-	-
525 4SIF	**704**	**480**	1 320	337 920	-	-	-	-	-	-	-
525 SD	**720**	**480**	1 350	345 600	-	-	-	-	-	-	-
4CIF	**704**	**576**	1 584	405 504	-	-	-	-	-	-	-
625 SD	**720**	**576**	1 620	414 720	-	-	-	-	-	-	-
SVGA	**800**	**600**	1 900	486 400	-	-	-	-	-	-	-
XGA	**1024**	**768**	3 072	786 432	-	-	-	-	-	-	-
720p HD	**1280**	**720**	3 600	921 600	-	-	-	-	-	-	-
4VGA	**1280**	**960**	4 800	1 228 800	-	-	-	-	-	-	-
SXGA	**1280**	**1024**	5 120	1 310 720	-	-	-	-	-	-	-
525 16SIF	**1408**	**960**	5 280	1 351 680	-	-	-	-	-	-	-
16CIF	**1408**	**1152**	6 336	1 622 016	-	-	-	-	-	-	-
4SVGA	**1600**	**1200**	7 500	1 920 000	-	-	-	-	-	-	-
1080 HD	**1920**	**1088**	8 160	2 088 960	-	-	-	-	-	-	-
2Kx1K	**2048**	**1024**	8 192	2 097 152	-	-	-	-	-	-	-
2Kx1080	**2048**	**1088**	8 704	2 228 224	-	-	-	-	-	-	-
4XGA	**2048**	**1536**	12 288	3 145 728	-	-	-	-	-	-	-
16VGA	**2560**	**1920**	19 200	4 915 200	-	-	-	-	-	-	-
3616x1536 (2.35:1)	**3616**	**1536**	21 696	5 554 176	-	-	-	-	-	-	-
3672x1536 (2.39:1)	**3680**	**1536**	22 080	5 652 480	-	-	-	-	-	-	-
3840x2160	**3840**	**2160**	31 035	7 948 800	-	-	-	-	-	-	-
4Kx2K	**4096**	**2048**	32 768	8 388 608	-	-	-	-	-	-	-
4096x2160	**4096**	**2160**	34 560	8 847 360	-	-	-	-	-	-	-
4096x2304 (16:9)	**4096**	**2304**	36 864	9 437 184	-	-	-	-	-	-	-

Table A-6 (continued) – Maximum frame rates (frames per second) for some example frame sizes

Level:					2.2	3	3.1	3.2	4	4.1	4.2
Max frame size (macroblocks):					**1 620**	**1 620**	**3 600**	**5 120**	**8 192**	**8 192**	**8 704**
Max macroblocks/second:					**20 250**	**40 500**	**108 000**	**216 000**	**245 760**	**245 760**	**522 240**
Max frame size (samples):					414 720	414 720	921 600	1 310 720	2 097 152	2 097 152	2 228 224
Max samples/second:					5 184 000	10 368 000	27 648 000	55 296 000	62 914 560	62 914 560	133 693 440
Format	**Luma Width**	**Luma Height**	**MBs Total**	**Luma Samples**							
SQCIF	**128**	**96**	48	12 288	172.0	172.0	172.0	172.0	172.0	172.0	172.0
QCIF	**176**	**144**	99	25 344	172.0	172.0	172.0	172.0	172.0	172.0	172.0
QVGA	**320**	**240**	300	76 800	67.5	135.0	172.0	172.0	172.0	172.0	172.0
525 SIF	**352**	**240**	330	84 480	61.4	122.7	172.0	172.0	172.0	172.0	172.0
CIF	**352**	**288**	396	101 376	51.1	102.3	172.0	172.0	172.0	172.0	172.0
525 HHR	**352**	**480**	660	168 960	30.7	61.4	163.6	172.0	172.0	172.0	172.0
625 HHR	**352**	**576**	792	202 752	25.6	51.1	136.4	172.0	172.0	172.0	172.0
VGA	**640**	**480**	1 200	307 200	16.9	33.8	90.0	172.0	172.0	172.0	172.0
525 4SIF	**704**	**480**	1 320	337 920	15.3	30.7	81.8	163.6	172.0	172.0	172.0
525 SD	**720**	**480**	1 350	345 600	15.0	30.0	80.0	160.0	172.0	172.0	172.0
4CIF	**704**	**576**	1 584	405 504	12.8	25.6	68.2	136.4	155.2	155.2	172.0
625 SD	**720**	**576**	1 620	414 720	12.5	25.0	66.7	133.3	151.7	151.7	172.0
SVGA	**800**	**600**	1 900	486 400	-	-	56.8	113.7	129.3	129.3	172.0
XGA	**1024**	**768**	3 072	786 432	-	-	35.2	70.3	80.0	80.0	172.0
720p HD	**1280**	**720**	3 600	921 600	-	-	30.0	60.0	68.3	68.3	145.1
4VGA	**1280**	**960**	4 800	1 228 800	-	-	-	45.0	51.2	51.2	108.8
SXGA	**1280**	**1024**	5 120	1 310 720	-	-	-	42.2	48.0	48.0	102.0
525 16SIF	**1408**	**960**	5 280	1 351 680	-	-	-	-	46.5	46.5	98.9
16CIF	**1408**	**1152**	6 336	1 622 016	-	-	-	-	38.8	38.8	82.4
4SVGA	**1600**	**1200**	7 500	1 920 000	-	-	-	-	32.8	32.8	69.6
1080 HD	**1920**	**1088**	8 160	2 088 960	-	-	-	-	30.1	30.1	64.0
2Kx1K	**2048**	**1024**	8 192	2 097 152	-	-	-	-	30.0	30.0	63.8
2Kx1080	**2048**	**1088**	8 704	2 228 224	-	-	-	-	-	-	60.0
4XGA	**2048**	**1536**	12 288	3 145 728	-	-	-	-	-	-	-
16VGA	**2560**	**1920**	19 200	4 915 200	-	-	-	-	-	-	-
3616x1536 (2.35:1)	**3616**	**1536**	21 696	5 554 176	-	-	-	-	-	-	-
3672x1536 (2.39:1)	**3680**	**1536**	22 080	5 652 480	-	-	-	-	-	-	-
3840x2160	**3840**	**2160**	31 035	7 948 800	-	-	-	-	-	-	-
4Kx2K	**4096**	**2048**	32 768	8 388 608	-	-	-	-	-	-	-
4096x2160	**4096**	**2160**	34 560	8 847 360	-	-	-	-	-	-	-
4096x2304 (16:9)	**4096**	**2304**	36 864	9 437 184	-	-	-	-	-	-	-

Table A-6 (concluded) – Maximum frame rates (frames per second) for some example frame sizes

Level:					5	5.1	5.2
Max frame size (macroblocks):					22 080	36 864	36 864
Max macroblocks/second:					589 824	983 040	2 073 600
Max frame size (samples):					5 652 480	9 437 184	9 437 184
Max samples/second:					150 994 944	251 658 240	530 841 600
Format	Luma Width	Luma Height	MBs Total	Luma Samples			
SQCIF	128	96	48	12 288	172.0	172.0	172.0
QCIF	176	144	99	25 344	172.0	172.0	172.0
QVGA	320	240	300	76 800	172.0	172.0	172.0
525 SIF	352	240	330	84 480	172.0	172.0	172.0
CIF	352	288	396	101 376	172.0	172.0	172.0
525 HHR	352	480	660	168 960	172.0	172.0	172.0
625 HHR	352	576	792	202 752	172.0	172.0	172.0
VGA	640	480	1 200	307 200	172.0	172.0	172.0
525 4SIF	704	480	1 320	337 920	172.0	172.0	172.0
525 SD	720	480	1 350	345 600	172.0	172.0	172.0
4CIF	704	576	1 584	405 504	172.0	172.0	172.0
625 SD	720	576	1 620	414 720	172.0	172.0	172.0
SVGA	800	600	1 900	486 400	172.0	172.0	172.0
XGA	1024	768	3 072	786 432	172.0	172.0	172.0
720p HD	1280	720	3 600	921 600	163.8	172.0	172.0
4VGA	1280	960	4 800	1 228 800	122.9	172.0	172.0
SXGA	1280	1024	5 120	1 310 720	115.2	172.0	172.0
525 16SIF	1408	960	5 280	1 351 680	111.7	172.0	172.0
16CIF	1408	1152	6 336	1 622 016	93.1	155.2	172.0
4SVGA	1600	1200	7 500	1 920 000	78.6	131.1	172.0
1080 HD	1920	1088	8 160	2 088 960	72.3	120.5	172.0
2Kx1K	2048	1024	8 192	2 097 152	72.0	120.0	172.0
2Kx1080	2048	1088	8 704	2 228 224	67.8	112.9	172.0
4XGA	2048	1536	12 288	3 145 728	48.0	80.0	168.8
16VGA	2560	1920	19 200	4 915 200	30.7	51.2	108.0
3616x1536 (2.35:1)	3616	1536	21 696	5 554 176	27.2	45.3	95.6
3672x1536 (2.39:1)	3680	1536	22 080	5 652 480	26.7	44.5	93.9
3840x2160	3840	2160	31 035	7 948 800	-	31.7	66.8
4Kx2K	4096	2048	32 768	8 388 608	-	30.0	63.3
4096x2160	4096	2160	34 560	8 847 360	-	28.5	60.0
4096x2304 (16:9)	4096	2304	36 864	9 437 184	-	26.7	56.3

The following should be noted:

– This Recommendation | International Standard is a variable-frame-size specification. The specific frame sizes in Table A-6 are illustrative examples only.

– As used in Table A-6, "525" refers to typical use for environments using 525 analogue scan lines (of which approximately 480 lines contain the visible picture region), and "625" refers to environments using 625 analogue scan lines (of which approximately 576 lines contain the visible picture region).

– XGA is also known as (aka) XVGA, 4SVGA aka UXGA, 16XGA aka 4Kx3K, CIF aka 625 SIF, 625 HHR aka 2CIF aka half 625 D-1, aka half 625 ITU-R BT.601, 525 SD aka 525 D-1 aka 525 ITU-R BT.601, 625 SD aka 625 D-1 aka 625 ITU-R BT.601.

– Frame rates given are correct for progressive scan modes. The frame rates are also correct for interlaced video coding for the cases of frame height divisible by 32.

A.3.5 Effect of level limits on maximum DPB size in units of frames (informative)

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

Table A-7 – Maximum DPB size (frames) for some example frame sizes

Level:				1	1b	1.1	1.2	1.3	2	2.1	2.2
Max frame size (macroblocks):				99	99	396	396	396	396	792	1 620
Max DPB size (macroblocks):				396	396	900	2 376	2 376	2 376	4 752	8 100
Format	Luma Width	Luma Height	MBs Total								
SQCIF	128	96	48	8	8	16	16	16	16	16	16
QCIF	176	144	99	4	4	9	16	16	16	16	16
QVGA	320	240	300	-	-	3	7	7	7	15	16
525 SIF	352	240	330	-	-	2	7	7	7	14	16
CIF	352	288	396	-	-	2	6	6	6	12	16
525 HHR	352	480	660	-	-	-	-	-	-	7	12
625 HHR	352	576	792	-	-	-	-	-	-	6	10
VGA	640	480	1 200	-	-	-	-	-	-	-	6
525 4SIF	704	480	1 320	-	-	-	-	-	-	-	6
525 SD	720	480	1 350	-	-	-	-	-	-	-	6
4CIF	704	576	1 584	-	-	-	-	-	-	-	5
625 SD	720	576	1 620	-	-	-	-	-	-	-	5
SVGA	800	600	1 900	-	-	-	-	-	-	-	-
XGA	1024	768	3 072	-	-	-	-	-	-	-	-
720p HD	1280	720	3 600	-	-	-	-	-	-	-	-
4VGA	1280	960	4 800	-	-	-	-	-	-	-	-
SXGA	1280	1024	5 120	-	-	-	-	-	-	-	-
525 16SIF	1408	960	5 280	-	-	-	-	-	-	-	-
16CIF	1408	1152	6 336	-	-	-	-	-	-	-	-
4SVGA	1600	1200	7 500	-	-	-	-	-	-	-	-
1080 HD	1920	1088	8 160	-	-	-	-	-	-	-	-
2Kx1K	2048	1024	8 192	-	-	-	-	-	-	-	-
2Kx1080	2048	1088	8 704	-	-	-	-	-	-	-	-
4XGA	2048	1536	12 288	-	-	-	-	-	-	-	-
16VGA	2560	1920	19 200	-	-	-	-	-	-	-	-
3616x1536 (2.35:1)	3616	1536	21 696	-	-	-	-	-	-	-	-
3672x1536 (2.39:1)	3680	1536	22 080	-	-	-	-	-	-	-	-
3840x2160	3840	2160	31 035	-	-	-	-	-	-	-	-
4Kx2K	4096	2048	32 768	-	-	-	-	-	-	-	-
4096x2160	4096	2160	34 560	-	-	-	-	-	-	-	-
4096x2304 (16:9)	4096	2304	36 864	-	-	-	-	-	-	-	-

Table A-7 (continued) – Maximum DPB size (frames) for some example frame sizes

Level:				3	3.1	3.2	4	4.1	4.2	5	5.1	5.2
Max frame size (macroblocks):				1 620	3 600	5 120	8 192	8 192	8 704	22 080	36 864	36 864
Max DPB size (macroblocks):				8 100	18 000	20 480	32 768	32 768	34 816	110 400	184 320	184 320
Format	Luma Width	Luma Height	MBs Total									
SQCIF	128	96	48	16	16	16	16	16	16	16	16	16
QCIF	176	144	99	16	16	16	16	16	16	16	16	16
QVGA	320	240	300	16	16	16	16	16	16	16	16	16
525 SIF	352	240	330	16	16	16	16	16	16	16	16	16
CIF	352	288	396	16	16	16	16	16	16	16	16	16
525 HHR	352	480	660	12	16	16	16	16	16	16	16	16
625 HHR	352	576	792	10	16	16	16	16	16	16	16	16
VGA	640	480	1 200	6	15	16	16	16	16	16	16	16
525 4SIF	704	480	1 320	6	13	15	16	16	16	16	16	16
525 SD	720	480	1 350	6	13	15	16	16	16	16	16	16
4CIF	704	576	1 584	5	11	12	16	16	16	16	16	16
625 SD	720	576	1 620	5	11	12	16	16	16	16	16	16
SVGA	800	600	1 900	-	9	10	16	16	16	16	16	16
XGA	1024	768	3 072	-	5	6	10	10	11	16	16	16
720p HD	1280	720	3 600	-	5	5	9	9	9	16	16	16
4VGA	1280	960	4 800	-	-	4	6	6	7	16	16	16
SXGA	1280	1024	5 120	-	-	4	6	6	6	16	16	16
525 16SIF	1408	960	5 280	-	-	-	6	6	6	16	16	16
16CIF	1408	1152	6 336	-	-	-	5	5	5	16	16	16
4SVGA	1600	1200	7 500	-	-	-	4	4	4	14	16	16
1080 HD	1920	1088	8 160	-	-	-	4	4	4	13	16	16
2Kx1K	2048	1024	8 192	-	-	-	4	4	4	13	16	16
2Kx1080	2048	1088	8 704	-	-	-	-	-	4	12	16	16
4XGA	2048	1536	12 288	-	-	-	-	-	-	8	15	15
16VGA	2560	1920	19 200	-	-	-	-	-	-	5	9	9
3616x1536 (2.35:1)	3616	1536	21 696	-	-	-	-	-	-	5	8	8
3672x1536 (2.39:1)	3680	1536	22 080	-	-	-	-	-	-	5	8	8
3840x2160	3840	2160	31 035	-	-	-	-	-	-	-	5	5
4Kx2K	4096	2048	32 768	-	-	-	-	-	-	-	5	5
4096x2160	4096	2160	34 560	-	-	-	-	-	-	-	5	5
4096x2304 (16:9)	4096	2304	36 864	-	-	-	-	-	-	-	5	5

The following should be noted:

– As used in Table A-7, "525" refers to typical use for environments using 525 analogue scan lines (of which approximately 480 lines contain the visible picture region), and "625" refers to environments using 625 analogue scan lines (of which approximately 576 lines contain the visible picture region).

– XGA is also known as (aka) XVGA, 4SVGA aka UXGA, 16XGA aka 4Kx3K, CIF aka 625 SIF, 625 HHR aka 2CIF aka half 625 D-1, aka half 625 ITU-R BT.601, 525 SD aka 525 D-1 aka 525 ITU-R BT.601, 625 SD aka 625 D-1 aka 625 ITU-R BT.601.

Annex B

Byte stream format

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies syntax and semantics of a byte stream format specified for use by applications that deliver some or all of the NAL unit stream as an ordered stream of bytes or bits within which the locations of NAL unit boundaries need to be identifiable from patterns in the data, such as Rec. ITU-T H.222.0 | ISO/IEC 13818-1 systems or Rec. ITU-T H.320 systems. For bit-oriented delivery, the bit order for the byte stream format is specified to start with the MSB of the first byte, proceed to the LSB of the first byte, followed by the MSB of the second byte, etc.

The byte stream format consists of a sequence of byte stream NAL unit syntax structures. Each byte stream NAL unit syntax structure contains one start code prefix followed by one nal_unit(NumBytesInNALunit) syntax structure. It may (and under some circumstances, it shall) also contain an additional zero_byte syntax element. It may also contain one or more additional trailing_zero_8bits syntax elements. When it is the first byte stream NAL unit in the bitstream, it may also contain one or more additional leading_zero_8bits syntax elements.

B.1 Byte stream NAL unit syntax and semantics

B.1.1 Byte stream NAL unit syntax

byte_stream_nal_unit(NumBytesInNALunit) {	**C**	**Descriptor**
while(next_bits(24) != 0x000001 && next_bits(32) != 0x00000001)		
leading_zero_8bits /* equal to 0x00 */		f(8)
if(next_bits(24) != 0x000001)		
zero_byte /* equal to 0x00 */		f(8)
start_code_prefix_one_3bytes /* equal to 0x000001 */		f(24)
nal_unit(NumBytesInNALunit)		
while(more_data_in_byte_stream( ) && next_bits(24) != 0x000001 && next_bits(32) != 0x00000001)		
trailing_zero_8bits /* equal to 0x00 */		f(8)
}		

B.1.2 Byte stream NAL unit semantics

The order of byte stream NAL units in the byte stream shall follow the decoding order of the NAL units contained in the byte stream NAL units (see subclause 7.4.1.2). The content of each byte stream NAL unit is associated with the same access unit as the NAL unit contained in the byte stream NAL unit (see subclause 7.4.1.2.3).

leading_zero_8bits is a byte equal to 0x00.

NOTE – The leading_zero_8bits syntax element can only be present in the first byte stream NAL unit of the bitstream, because (as shown in the syntax diagram of subclause B.1.1) any bytes equal to 0x00 that follow a NAL unit syntax structure and precede the four-byte sequence 0x00000001 (which is to be interpreted as a zero_byte followed by a start_code_prefix_one_3bytes) will be considered to be trailing_zero_8bits syntax elements that are part of the preceding byte stream NAL unit.

zero_byte is a single byte equal to 0x00.

When any of the following conditions are true, the zero_byte syntax element shall be present:

– the nal_unit_type within the nal_unit( ) is equal to 7 (sequence parameter set) or 8 (picture parameter set),

– the byte stream NAL unit syntax structure contains the first NAL unit of an access unit in decoding order, as specified in subclause 7.4.1.2.3.

start_code_prefix_one_3bytes is a fixed-value sequence of 3 bytes equal to 0x000001. This syntax element is called a start code prefix.

trailing_zero_8bits is a byte equal to 0x00.

B.2 Byte stream NAL unit decoding process

Input to this process consists of an ordered stream of bytes consisting of a sequence of byte stream NAL unit syntax structures.

Output of this process consists of a sequence of NAL unit syntax structures.

At the beginning of the decoding process, the decoder initialises its current position in the byte stream to the beginning of the byte stream. It then extracts and discards each leading_zero_8bits syntax element (if present), moving the current position in the byte stream forward one byte at a time, until the current position in the byte stream is such that the next four bytes in the bitstream form the four-byte sequence 0x00000001.

The decoder then performs the following step-wise process repeatedly to extract and decode each NAL unit syntax structure in the byte stream until the end of the byte stream has been encountered (as determined by unspecified means) and the last NAL unit in the byte stream has been decoded:

1. When the next four bytes in the bitstream form the four-byte sequence 0x00000001, the next byte in the byte stream (which is a zero_byte syntax element) is extracted and discarded and the current position in the byte stream is set equal to the position of the byte following this discarded byte.
2. The next three-byte sequence in the byte stream (which is a start_code_prefix_one_3bytes) is extracted and discarded and the current position in the byte stream is set equal to the position of the byte following this three-byte sequence.
3. NumBytesInNALunit is set equal to the number of bytes starting with the byte at the current position in the byte stream up to and including the last byte that precedes the location of any of the following:
   - A subsequent byte-aligned three-byte sequence equal to 0x000000,
   - A subsequent byte-aligned three-byte sequence equal to 0x000001,
   - The end of the byte stream, as determined by unspecified means.
4. NumBytesInNALunit bytes are removed from the bitstream and the current position in the byte stream is advanced by NumBytesInNALunit bytes. This sequence of bytes is nal_unit(NumBytesInNALunit) and is decoded using the NAL unit decoding process.
5. When the current position in the byte stream is not at the end of the byte stream (as determined by unspecified means) and the next bytes in the byte stream do not start with a three-byte sequence equal to 0x000001 and the next bytes in the byte stream do not start with a four byte sequence equal to 0x00000001, the decoder extracts and discards each trailing_zero_8bits syntax element, moving the current position in the byte stream forward one byte at a time, until the current position in the byte stream is such that the next bytes in the byte stream form the four-byte sequence 0x00000001 or the end of the byte stream has been encountered (as determined by unspecified means).

B.3 Decoder byte-alignment recovery (informative)

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

Many applications provide data to a decoder in a manner that is inherently byte aligned, and thus have no need for the bit-oriented byte alignment detection procedure described in this subclause.

A decoder is said to have byte-alignment with a bitstream when the decoder is able to determine whether or not the positions of data in the bitstream are byte-aligned. When a decoder does not have byte alignment with the encoder's byte stream, the decoder may examine the incoming bitstream for the binary pattern '00000000 00000000 00000000 00000001' (31 consecutive bits equal to 0 followed by a bit equal to 1). The bit immediately following this pattern is the first bit of an aligned byte following a start code prefix. Upon detecting this pattern, the decoder will be byte aligned with the encoder and positioned at the start of a NAL unit in the byte stream.

Once byte aligned with the encoder, the decoder can examine the incoming byte stream for subsequent three-byte sequences 0x000001 and 0x000003.

When the three-byte sequence 0x000001 is detected, this is a start code prefix.

When the three-byte sequence 0x000003 is detected, the third byte (0x03) is an emulation_prevention_three_byte to be discarded as specified in subclause 7.4.1.

When an error in the bitstream syntax is detected (e.g., a non-zero value of the forbidden_zero_bit or one of the three-byte or four-byte sequences that are prohibited in subclause 7.4.1), the decoder may consider the detected condition as an indication that byte alignment may have been lost and may discard all bitstream data until the detection of byte alignment at a later position in the bitstream as described in this subclause.

Annex C

Hypothetical reference decoder

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies the hypothetical reference decoder (HRD) and its use to check bitstream and decoder conformance.

Two types of bitstreams are subject to HRD conformance checking for this Recommendation | International Standard. The first such type of bitstream, called Type I bitstream, is a NAL unit stream containing only the VCL NAL units and filler data NAL units for all access units in the bitstream. The second type of bitstream, called a Type II bitstream, contains, in addition to the VCL NAL units and filler data NAL units for all access units in the bitstream, at least one of the following:

– additional non-VCL NAL units other than filler data NAL units,

– all leading_zero_8bits, zero_byte, start_code_prefix_one_3bytes, and trailing_zero_8bits syntax elements that form a byte stream from the NAL unit stream (as specified in Annex B).

Figure C-1 shows the types of bitstream conformance points checked by the HRD.

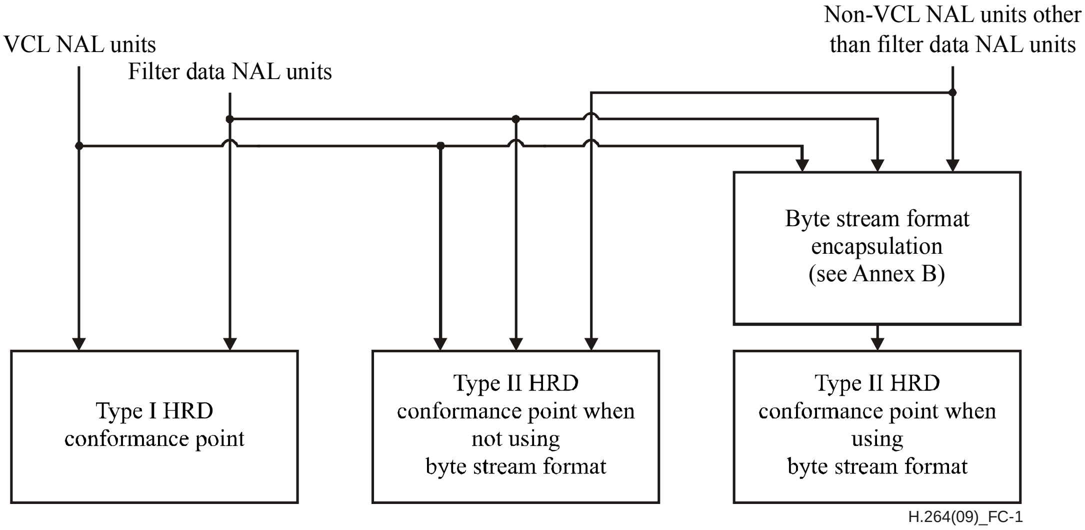


Figure C-1 – Structure of byte streams and NAL unit streams for HRD conformance checks

The syntax elements of non-VCL NAL units (or their default values for some of the syntax elements), required for the HRD, are specified in the semantic subclauses of clause 7, Annexes D and E, and subclauses G.7, G.13, G.14, H.7, H.13, and H.14.

Two types of HRD parameter sets (NAL HRD parameters and VCL HRD parameters) are used. The HRD parameter sets are signalled as follows:

– When the coded video sequence conforms to one or more of the profiles specified in Annex A and the decoding process specified in clauses 2-9 is applied, the HRD parameter sets are signalled through video usability information as specified in subclauses E.1 and E.2, which is part of the sequence parameter set syntax structure.

– When the coded video sequence conforms to one or more of the profiles specified in Annex G and the decoding process specified in Annex G is applied, the HRD parameter sets are signalled through the SVC video usability information extension as specified in subclauses G.14.1 and G.14.2, which is part of the subset sequence parameter set syntax structure.

NOTE 1 – For coded video sequences that conform to both, one or more of the profiles specified in Annex A and one or more of the profiles specified in Annex G, the signalling of the applicable HRD parameter sets is depending on whether the decoding process specified in clauses 2-9 or the decoding process specified in Annex G is applied.

– When the coded video sequence conforms to one or more of the profiles specified in Annex H and the decoding process specified in Annex H is applied, the HRD parameter sets are signalled through the MVC video usability information extension as specified in subclauses H.14.1 and H.14.2, which is part of the subset sequence parameter set syntax structure.

NOTE 2 – For coded video sequences that conform to both, one or more of the profiles specified in Annex A and one or more of the profiles specified in Annex H, the signalling of the applicable HRD parameter sets is depending on whether the decoding process specified in clauses 2-9 or the decoding process specified in Annex H is applied.

All sequence parameter sets and picture parameter sets referred to in the VCL NAL units, and corresponding buffering period and picture timing SEI messages shall be conveyed to the HRD, in a timely manner, either in the bitstream (by non-VCL NAL units), or by other means not specified in this Recommendation | International Standard.

In Annexes C, D, and E and subclauses G.12, G.13, G.14, H.12, H.13, and H.14, the specification for "presence" of non-VCL NAL units is also satisfied when those NAL units (or just some of them) are conveyed to decoders (or to the HRD) by other means not specified by this Recommendation | International Standard. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

NOTE 3 – As an example, synchronization of a non-VCL NAL unit, conveyed by means other than presence in the bitstream, with the NAL units that are present in the bitstream, can be achieved by indicating two points in the bitstream, between which the non-VCL NAL unit would have been present in the bitstream, had the encoder decided to convey it in the bitstream.

When the content of a non-VCL NAL unit is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the non-VCL NAL unit is not required to use the same syntax specified in this annex.

NOTE 4 – When HRD information is contained within the bitstream, it is possible to verify the conformance of a bitstream to the requirements of this subclause based solely on information contained in the bitstream. When the HRD information is not present in the bitstream, as is the case for all "stand-alone" Type I bitstreams, conformance can only be verified when the HRD data is supplied by some other means not specified in this Recommendation | International Standard.

The HRD contains a coded picture buffer (CPB), an instantaneous decoding process, a decoded picture buffer (DPB), and output cropping as shown in Figure C-2.

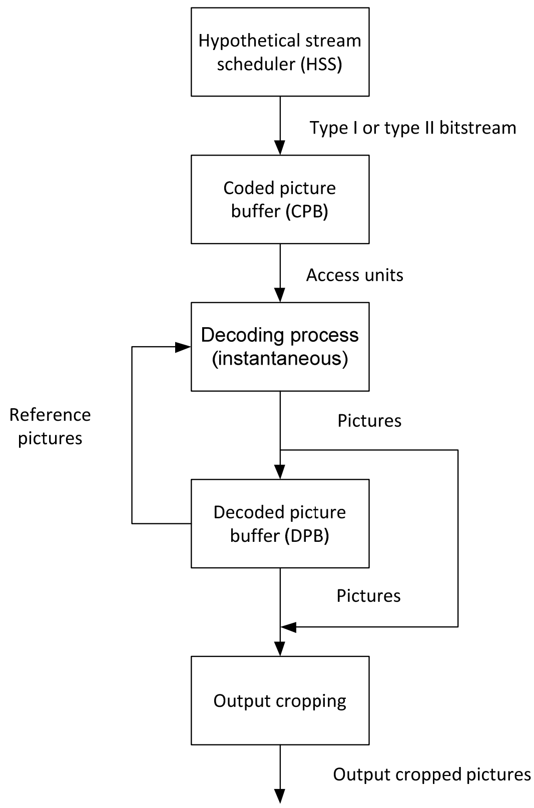


Figure C-2 – HRD buffer model

The CPB size (number of bits) is CpbSize[SchedSelIdx]. The DPB size (number of frame buffers) is Max(1, max_dec_frame_buffering). When the coded video sequence conforms to one or more of the profiles specified in Annex H and the decoding process specified in Annex H is applied, the DPB size is specified in units of view components.

The HRD operates as follows. Data associated with access units that flow into the CPB according to a specified arrival schedule are delivered by the HSS. The data associated with each access unit are removed and decoded instantaneously by the instantaneous decoding process at CPB removal times. Each decoded picture is placed in the DPB at its CPB removal time unless it is output at its CPB removal time and is a non-reference picture. When a picture is placed in the DPB it is removed from the DPB at the later of the DPB output time or the time that it is marked as "unused for reference".

For each picture in the bitstream, the variable OutputFlag for the decoded picture and, when applicable, the reference base picture is set as follows:

– If the coded video sequence containing the picture conforms to one or more of the profiles specified in Annex A and the decoding process specified in clauses 2-9 is applied, OutputFlag is set equal to 1.
– Otherwise, if the coded video sequence containing the picture conforms to one or more of the profiles specified in Annex G and the decoding process specified in Annex G is applied, the following applies:
  – For a reference base picture, OutputFlag is set equal to 0.
  – For a decoded picture, OutputFlag is set equal to the value of the output_flag syntax element of the target layer representation.
– Otherwise (the coded video sequence containing the picture conforms to one or more of the profiles specified in Annex H and the decoding process specified in Annex H is applied), the following applies:
  – For the decoded view components of the target output views, OutputFlag is set equal to 1.
  – For the decoded view components of other views, OutputFlag is set equal to 0.

The operation of the CPB is specified in subclause C.1. The instantaneous decoder operation is specified in clauses 2-9 (for coded video sequences conforming to one or more of the profiles specified in Annex A) and in Annex G (for coded video sequences conforming to one or more of the profiles specified in Annex G) and in Annex H (for coded video sequences conforming to one or more of the profiles specified in Annex H). The operation of the DPB is specified in subclause C.2. The output cropping is specified in subclause C.2.2.

NOTE 5 – Coded video sequences that conform to both, one or more of the profiles specified in Annex A and one or more of the profiles specified in Annex G, can be decoded either by the decoding process specified in clauses 2-9 or by the decoding process specified in Annex G. The decoding result and the HRD operation may be depending on which of the decoding processes is applied.

NOTE 6 – Coded video sequences that conform to both, one or more of the profiles specified in Annex A and one or more of the profiles specified in Annex H, can be decoded either by the decoding process specified in clauses 2-9 or by the decoding process specified in Annex H. The decoding result and the HRD operation may be depending on which of the decoding processes is applied.

HSS and HRD information concerning the number of enumerated delivery schedules and their associated bit rates and buffer sizes is specified in subclauses E.1.1, E.1.2, E.2.1, E.2.2, G.14.1, G.14.2, H.14.1, and H.14.2. The HRD is initialised as specified by the buffering period SEI message as specified in subclauses D.1.1 and D.2.1. The removal timing of access units from the CPB and output timing from the DPB are specified in the picture timing SEI message as specified in subclauses D.1.2 and D.2.2. All timing information relating to a specific access unit shall arrive prior to the CPB removal time of the access unit.

When the coded video sequence conforms to one or more of the profiles specified in Annex G and the decoding process specified in Annex G is applied, the following is specified:

(a) When an access unit contains one or more buffering period SEI messages that are included in scalable nesting SEI messages and are associated with values of DQId in the range of ((DQIdMax >> 4) << 4) to (((DQIdMax >> 4) << 4) + 15), inclusive, the last of these buffering period SEI messages in decoding order is the buffering period SEI message that initialises the HRD. Let hrdDQId be the largest value of 16 * sei_dependency_id[i] + sei_quality_id[i] that is associated with the scalable nesting SEI message containing the buffering period SEI message that initialises the HRD, let hrdDId and hrdQId be equal to hrdDQId >> 4 and hrdDQId & 15, respectively, and let hrdTId be the value of sei_temporal_id that is associated with the scalable nesting SEI message containing the buffering period SEI message that initialises the HRD.

(b) The picture timing SEI messages that specify the removal timing of access units from the CPB and output timing from the DPB are the picture timing SEI messages that are included in scalable nesting SEI messages associated with values of sei_dependency_id[i], sei_quality_id[i], and sei_temporal_id equal to hrdDId, hrdQId, and hrdTId, respectively.

(c) The HRD parameter sets that are used for conformance checking are the HRD parameter sets, included in the SVC video usability information extension of the active SVC sequence parameter set, that are associated with values of vui_ext_dependency_id[i], vui_ext_quality_id[i], and vui_ext_temporal_id[i] equal to hrdDId, hrdQId, and hrdTId, respectively. For the specification in this annex, num_units_in_tick, time_scale, fixed_frame_rate_flag, nal_hrd_parameters_present_flag, vcl_hrd_parameters_present_flag, low_delay_hrd_flag, and pic_struct_present_flag are substituted with the values of vui_ext_num_units_in_tick[i], vui_ext_time_scale[i], vui_ext_fixed_frame_rate_flag[i], vui_ext_nal_hrd_parameters_present_flag[i], vui_ext_vcl_hrd_parameters_present_flag[i], vui_ext_low_delay_hrd_flag[i], and vui_ext_pic_struct_present_flag[i], respectively, with i being the value for which vui_ext_dependency_id[i], vui_ext_quality_id[i], and vui_ext_temporal_id[i] are equal to hrdDId, hrdQId, and hrdTId, respectively.

When the coded video sequence conforms to one or more of the profiles specified in Annex H and the decoding process specified in Annex H is applied, the following is specified:

(a) When an access unit contains one or more buffering period SEI messages that are included in MVC scalable nesting SEI messages, the buffering period SEI message that is associated with the operation point being decoded is the buffering period SEI message that initialises the HRD. Let hrdVId[i] be equal to sei_op_view_id[i] for all i in the range of 0 to num_view_components_op_minus1, inclusive, and let hrdTId be the value of sei_op_temporal_id, that are associated with the MVC scalable nesting SEI message containing the buffering period SEI message that initialises the HRD.

(b) The picture timing SEI messages that specify the removal timing of access units from the CPB and output timing from the DPB are the picture timing SEI messages that are included in MVC scalable nesting SEI messages associated with values of sei_op_view_id[i] equal to hrdVId[i] for all i in the range of 0 to num_view_components_op_minus1, inclusive, and sei_temporal_id equal to hrdTId.

(c) The HRD parameter sets that are used for conformance checking are the HRD parameter sets, included in the MVC video usability information extension of the active MVC sequence parameter set, that are associated with values of vui_mvc_view_id[i][j] for all j in the range of 0 to vui_mvc_num_target_output_views_minus1[i], inclusive, equal to hrdVId[j], and the value of vui_mvc_temporal_id[i] equal to hrdTId. For the specification in this annex, num_units_in_tick, time_scale, fixed_frame_rate_flag, nal_hrd_parameters_present_flag, vcl_hrd_parameters_present_flag, low_delay_hrd_flag, and pic_struct_present_flag are substituted with the values of vui_mvc_num_units_in_tick[i], vui_mvc_time_scale[i], vui_mvc_fixed_frame_rate_flag[i], vui_mvc_nal_hrd_parameters_present_flag[i], vui_mvc_vcl_hrd_parameters_present_flag[i], vui_mvc_low_delay_hrd_flag[i], and vui_mvc_pic_struct_present_flag[i], respectively, with i being the value for which vui_mvc_view_id[i] is equal to hrdVId[j] for all j in the range of 0 to vui_mvc_num_traget_output_views_minus1[i], inclusive, and vui_mvc_temporal_id[i] equal to hrdTId.

The HRD is used to check conformance of bitstreams and decoders as specified in subclauses C.3 and C.4, respectively.

NOTE 7 – While conformance is guaranteed under the assumption that all frame-rates and clocks used to generate the bitstream match exactly the values signalled in the bitstream, in a real system each of these may vary from the signalled or specified value.

All the arithmetic in this annex is done with real values, so that no rounding errors can propagate. For example, the number of bits in a CPB just prior to or after removal of an access unit is not necessarily an integer.

The variable t_c is derived as follows and is called a clock tick:

$$t_c = \text{num\_units\_in\_tick} \div \text{time\_scale} \quad \text{(C-1)}$$

The following is specified for expressing the constraints in this annex:

– Let access unit n be the n-th access unit in decoding order with the first access unit being access unit 0.

– Let picture n be the primary coded picture or the decoded primary picture of access unit n.

C.1 Operation of coded picture buffer (CPB)

The specifications in this subclause apply independently to each set of CPB parameters that is present and to both the Type I and Type II conformance points shown in Figure C-1.

C.1.1 Timing of bitstream arrival

The HRD may be initialised at any one of the buffering period SEI messages. Prior to initialisation, the CPB is empty.

NOTE – After initialisation, the HRD is not initialised again by subsequent buffering period SEI messages.

Each access unit is referred to as access unit n, where the number n identifies the particular access unit. The access unit that is associated with the buffering period SEI message that initialises the CPB is referred to as access unit 0. The value of n is incremented by 1 for each subsequent access unit in decoding order.

The time at which the first bit of access unit n begins to enter the CPB is referred to as the initial arrival time $t_{ai}(n)$.

The initial arrival time of access units is derived as follows:

– If the access unit is access unit 0, $t_{ai}(0) = 0$,

– Otherwise (the access unit is access unit n with n > 0), the following applies:

  – If cbr_flag[SchedSelIdx] is equal to 1, the initial arrival time for access unit n, is equal to the final arrival time (which is derived below) of access unit n − 1, i.e.,

$$t_{ai}(n) = t_{af}(n - 1) \quad \text{(C-2)}$$

  – Otherwise (cbr_flag[SchedSelIdx] is equal to 0), the initial arrival time for access unit n is derived by

$$t_{ai}(n) = \mathrm{Max}(t_{af}(n - 1), t_{ai,earliest}(n)) \quad \text{(C-3)}$$

  where $t_{ai,earliest}(n)$ is derived as follows:

    – If access unit n is not the first access unit of a subsequent buffering period, $t_{ai,earliest}(n)$ is derived as

$$t_{ai,earliest}(n) = t_{r,n}(n) - (\text{initial\_cpb\_removal\_delay[SchedSelIdx]} + \text{initial\_cpb\_removal\_delay\_offset[SchedSelIdx]}) \div 90000 \quad \text{(C-4)}$$

    with $t_{r,n}(n)$ being the nominal removal time of access unit n from the CPB as specified in subclause C.1.2 and initial_cpb_removal_delay[SchedSelIdx] and initial_cpb_removal_delay_offset[SchedSelIdx] being specified in the previous buffering period SEI message.

    – Otherwise (access unit n is the first access unit of a subsequent buffering period), $t_{ai,earliest}(n)$ is derived as

$$t_{ai,earliest}(n) = t_{r,n}(n) - (\text{initial\_cpb\_removal\_delay[SchedSelIdx]} \div 90000) \quad \text{(C-5)}$$

    with initial_cpb_removal_delay[SchedSelIdx] being specified in the buffering period SEI message associated with access unit n.

The final arrival time for access unit n is derived by

$$t_{af}(n) = t_{ai}(n) + b(n) \div \text{BitRate[SchedSelIdx]} \quad \text{(C-6)}$$

where b(n) is the size in bits of access unit n, counting the bits of the VCL NAL units and the filler data NAL units for the Type I conformance point or all bits of the Type II bitstream for the Type II conformance point, where the Type I and Type II conformance points are as shown in Figure C-1.

The values of SchedSelIdx, BitRate[SchedSelIdx], and CpbSize[SchedSelIdx] are constrained as follows:

– If the content of the active sequence parameter sets for access unit n and access unit n − 1 differ, the HSS selects a value SchedSelIdx1 of SchedSelIdx from among the values of SchedSelIdx provided in the active sequence parameter set for access unit n that results in a BitRate[SchedSelIdx1] or CpbSize[SchedSelIdx1] for access unit n. The value of BitRate[SchedSelIdx1] or CpbSize[SchedSelIdx1] may differ from the value of BitRate[SchedSelIdx0] or CpbSize[SchedSelIdx0] for the value SchedSelIdx0 of SchedSelIdx that was in use for access unit n − 1.

– Otherwise, the HSS continues to operate with the previous values of SchedSelIdx, BitRate[SchedSelIdx] and CpbSize[SchedSelIdx].

When the HSS selects values of BitRate[SchedSelIdx] or CpbSize[SchedSelIdx] that differ from those of the previous access unit, the following applies:

– the variable BitRate[SchedSelIdx] comes into effect at time $t_{ai}(n)$

– the variable CpbSize[SchedSelIdx] comes into effect as follows:

  – If the new value of CpbSize[SchedSelIdx] exceeds the old CPB size, it comes into effect at time $t_{ai}(n)$,

– Otherwise, the new value of CpbSize[SchedSelIdx] comes into effect at the time $t_r(n)$.

C.1.2 Timing of coded picture removal

When an access unit n is the access unit with n equal to 0 (the access unit that initialises the HRD), the nominal removal time of the access unit from the CPB is specified by

$$t_{r,n}(0) = \text{initial\_cpb\_removal\_delay}[\text{SchedSelIdx}] \div 90000 \quad \text{(C-7)}$$

When an access unit n is the first access unit of a buffering period that does not initialise the HRD, the nominal removal time of the access unit from the CPB is specified by

$$t_{r,n}(n) = t_{r,n}(n_b) + t_c * \text{cpb\_removal\_delay}(n) \quad \text{(C-8)}$$

where $t_{r,n}(n_b)$ is the nominal removal time of the first access unit of the previous buffering period and cpb_removal_delay(n) is the value of cpb_removal_delay specified in the picture timing SEI message associated with access unit n.

The nominal removal time $t_{r,n}(n)$ of an access unit n that is not the first access unit of a buffering period is given by

$$t_{r,n}(n) = t_{r,n}(n_b) + t_c * \text{cpb\_removal\_delay}(n) \quad \text{(C-9)}$$

where $t_{r,n}(n_b)$ is the nominal removal time of the first access unit of the current buffering period and cpb_removal_delay(n) is the value of cpb_removal_delay specified in the picture timing SEI message associated with access unit n.

The removal time of access unit n is specified as follows:

– If low_delay_hrd_flag is equal to 0 or $t_{r,n}(n) >= t_{af}(n)$, the removal time of access unit n is specified by

$$t_r(n) = t_{r,n}(n) \quad \text{(C-10)}$$

– Otherwise (low_delay_hrd_flag is equal to 1 and $t_{r,n}(n) < t_{af}(n)$), the removal time of access unit n is specified by

$$t_r(n) = t_{r,n}(n) + t_c * \text{Ceil}((t_{af}(n) - t_{r,n}(n)) \div t_c) \quad \text{(C-11)}$$

NOTE – The latter case indicates that the size of access unit n, b(n), is so large that it prevents removal at the nominal removal time.

When an access unit n is the first access unit of a buffering period, n_b is set equal to n at the removal time $t_r(n)$ of the access unit n.

C.2 Operation of the decoded picture buffer (DPB)

The decoded picture buffer contains frame buffers. When a coded video sequence conforming to one or more of the profiles specified in Annex A is decoded by applying the decoding process specified in clauses 2-9, each of the frame buffers may contain a decoded frame, a decoded complementary field pair or a single (non-paired) decoded field that is marked as "used for reference" (reference pictures) or is held for future output (reordered or delayed pictures). When a coded video sequence conforming to one or more of the profiles specified in Annex G is decoded by applying the decoding process specified in Annex G, each frame buffer may contain a decoded frame, a decoded complementary field pair, a single (non-paired) decoded field, a decoded reference base frame, a decoded reference base complementary field pair or a single (non-paired) decoded reference base field that is marked as "used for reference" (reference pictures) or is held for future output (reordered or delayed pictures). When a coded video sequence conforming to one or more of the profiles specified in Annex H is decoded by applying the decoding process specified in Annex H, each of the frame buffers may contain a decoded frame view component, a decoded complementary field view component pair, or a single (non-paired) decoded field view component that is marked as "used for reference" (reference pictures) or is held for future output (reordered or delayed pictures) or is held as reference for inter-view prediction (inter-view only reference components).

Prior to initialisation, the DPB is empty (the DPB fullness is set to zero). The following steps specified in this subclause all happen instantaneously at $t_r(n)$ and in the order listed. When the decoding process specified in Annex H is applied, the view components of the current primary coded picture are processed by applying the ordered steps to each view component in increasing order of the associated view order index VOIdx.

1. The process of decoding gaps in frame_num and storing "non-existing" frames as specified in subclause C.2.1 is invoked.

2. The picture decoding and output process as specified in subclause C.2.2 is invoked.
3. The process of removing pictures from the DPB before possible insertion of the current picture as specified in subclause C.2.3 is invoked.
4. The process of marking and storing the current decoded picture as specified in subclause C.2.4 is invoked.

NOTE – When the decoding process specified in Annex G is applied, the DPB is only operated for decoded pictures and reference base pictures associated with decoded pictures. The DPB is not operated for layer pictures with dependency_id less than DependencyIdMax (and associated reference base pictures). All decoded pictures and associated reference base pictures are decoded pictures and associated reference base pictures for dependency_id equal to DependencyIdMax, which represent the results of the decoding process specified in subclause G.8.

C.2.1 Decoding of gaps in frame_num and storage of "non-existing" frames

When the decoding process specified in Annex H is applied, the process specified in this subclause is invoked for a particular view with view order index VOIdx, with "picture" being replaced by "view component", "frame" being replaced by "frame view component", and "field" being replaced by "field view component". During the invocation of the process for a particular view, only view components of the particular view are considered and view components of other views are not marked as "unused for reference" or removed from the DPB. The DPB fullness represents the total number of non-empty frame buffers, including frame buffers that contain view components of other views.

When applicable, gaps in frame_num are detected by the decoding process and the generated frames are marked and inserted into the DPB as specified below.

Gaps in frame_num are detected by the decoding process and the generated frames are marked as specified in subclauses 8.2.5.2 and G.8.2.5.

After the marking of each generated frame, each picture m marked by the "sliding window" process as "unused for reference" is removed from the DPB when it is also marked as "non-existing" or its DPB output time is less than or equal to the CPB removal time of the current picture n; i.e., $t_{o,dpb}(m) <= t_r(n)$, or it has OutputFlag equal to 0. When a frame or the last field in a frame buffer is removed from the DPB, the DPB fullness is decremented by one. The "non-existing" generated frame is inserted into the DPB and the DPB fullness is incremented by one.

C.2.2 Picture decoding and output

When the decoding process specified in Annex H is applied, the process specified in this subclause is invoked for a particular view with view order index VOIdx.

The decoding of the current picture or view component (when applying the decoding process specified in Annex H) and the derivation of the DPB output time (if applicable) is specified as follows:

– If the decoding process specified in clause 8 or Annex G is applied, the following applies:
  – The current primary coded picture n is decoded.
  – When picture n has OutputFlag equal to 1, its DPB output time $t_{o,dpb}(n)$ is derived by

$$t_{o,dpb}(n) = t_r(n) + t_c * \text{dpb\_output\_delay}(n) \tag{C-12}$$

    where dpb_output_delay(n) is the value of dpb_output_delay specified in the picture timing SEI message associated with access unit n.

– Otherwise (the decoding process specified in Annex H is applied), the following applies:
  – The view component with view order index VOIdx of the current primary coded picture n is decoded.
  – When VOIdx is equal to VOIdxMin and any of the view components of picture n has OutputFlag equal to 1, the DPB output time $t_{o,dpb}(n)$ for picture n is derived by Equation C-12, where dpb_output_delay(n) is the value of dpb_output_delay specified in the picture timing SEI message associated with access unit n.

The output of the current picture or view component (when applying the decoding process specified in Annex H) is specified as follows:

– If OutputFlag is equal to 1 and $t_{o,dpb}(n) = t_r(n)$, the current picture or view component is output.

  NOTE 1 – When the current picture or view component has nal_ref_idc greater than 0 (when using the decoding process specified in Annex G, nal_ref_idc is the syntax element of the target layer representation), it will be stored in the DPB.

– Otherwise, if OutputFlag is equal to 0, the current picture or view component is not output, but it may be stored in the DPB as specified in subclause C.2.4.

– Otherwise (OutputFlag is equal to 1 and $t_{o,dpb}(n) > t_r(n)$), the current picture or view component is output later and will be stored in the DPB (as specified in subclause C.2.4) and is output at time $t_{o,dpb}(n)$ unless indicated not to be output by the decoding or inference of no_output_of_prior_pics_flag equal to 1 at a time that precedes $t_{o,dpb}(n)$.

NOTE 2 – When the coded video sequence conforms to a profile specified in Annex H and the decoding process specified in Annex H is used, the view components of all the target output views of a picture are output at the same time instant and in increasing order of the view order index VOIdx.

When output, the picture or view component shall be cropped, using the cropping rectangle specified in the active sequence parameter set for the picture or view component.

When the decoding process specified in clause 8 or Annex G is applied, the current picture n is a picture that is output and is not the last picture of the bitstream that is output, the value of $\Delta t_{o,dpb}(n)$ is derived by

$$\Delta t_{o,dpb}(n) = t_{o,dpb}(n_n) - t_{o,dpb}(n) \qquad \text{(C-13)}$$

where n_n indicates the picture that follows after picture n in output order and has OutputFlag equal to 1.

When the decoding process specified in Annex H is applied, the current picture n is a picture that contains at least one view component that is output and the current picture is not the last picture of the bitstream that contains at least one view component that is output and VOIdx is equal to VOIdxMin, the value of $\Delta t_{o,dpb}(n)$ is derived by Equation C-13, where n_n indicates the picture that follows after picture n in output order and contains at least one view component with OutputFlag equal to 1.

The decoded picture or view component is temporarily stored (not in the DPB).

C.2.3 Removal of pictures from the DPB before possible insertion of the current picture

When the decoding process specified in Annex H is applied, the process specified in this subclause is invoked for a particular view with view order index VOIdx, with "picture" being replaced by "view component", "frame" being replaced by "frame view component", and "field" being replaced by "field view component".

When the decoding process specified in Annex H is applied, the following process is specified for removing inter-view only reference components of the current access unit from the DPB. By this process, view components of the current view with view order index VOIdx are not removed from the DPB, but inter-view only reference components of other views may be removed. The removal of inter-view only reference components is specified as follows:

– If the view order index VOIdx of the current view is equal to VOIdxMax, all inter-view only reference components m for which any of the following conditions are true are removed from the DPB:
  – OutputFlag is equal to 0,
  – the DPB output time $t_{o,dpb}(m)$ of the picture containing the view component m is less than or equal to the CPB removal time $t_r(n)$ of the current picture.
– Otherwise (the view order index VOIdx of the current view is less than VOIdxMax), all inter-view only reference components m for which both of the following conditions are true are removed from the DPB:
  – OutputFlag is equal to 0 or the DPB output time $t_{o,dpb}(m)$ of the picture containing the view component m is less than or equal to the CPB removal time $t_r(n)$ of the current picture,
  – one of the following conditions is true:
    – the current view component is a view component of an anchor picture and the view_id of the inter-view only reference component m is not equal to any value of anchor_ref_lX[k][j], with X being equal to 0 or 1, k being any integer value greater than the view order index VOIdx of the current view, and j being any integer value in the range of 0 to Max(0, num_anchor_refs_lX[k] − 1), inclusive,
    – the current view component is not a view component of an anchor picture and the view_id of the inter-view only reference component m is not equal to any value of non_anchor_ref_lX[k][j], with X being equal to 0 or 1, k being any integer value greater than the view order index VOIdx of the current view, and j being any integer value in the range of 0 to Max(0, num_non_anchor_refs_lX[k] − 1), inclusive.

When the decoding process specified in Annex H is applied, for the following processes specified in this subclause, only view components of the particular view for which this subclause is invoked are considered, and view components of other views are not marked as "unused for reference" or removed from the DPB. The DPB fullness represents the total number of non-empty frame buffers, including frame buffers that contain view components of other views.

The removal of pictures from the DPB before possible insertion of the current picture proceeds as follows:

– If the decoded picture is an IDR picture the following applies:

1. All reference pictures in the DPB are marked as "unused for reference" as specified in subclause 8.2.5.1 when a coded video sequence conforming to one or more of the profiles specified in Annex A is decoded by applying the decoding process specified in clauses 2-9, or as specified in subclause G.8.2.4 when a coded video sequence conforming to one or more of the profiles specified in Annex G is decoded by applying the decoding process specified in Annex G, or as specified in subclause H.8.3 when a coded video sequence conforming to one or more of the profiles specified in Annex H is decoded by applying the decoding process specified in Annex H.

2. When the IDR picture is not the first IDR picture decoded and the value of PicWidthInMbs or FrameHeightInMbs or max_dec_frame_buffering derived from the active sequence parameter set is different from the value of PicWidthInMbs or FrameHeightInMbs or max_dec_frame_buffering derived from the sequence parameter set that was active for the preceding picture, respectively, no_output_of_prior_pics_flag is inferred to be equal to 1 by the HRD, regardless of the actual value of no_output_of_prior_pics_flag.

   NOTE – Decoder implementations should try to handle frame or DPB size changes more gracefully than the HRD in regard to changes in PicWidthInMbs or FrameHeightInMbs.

3. When no_output_of_prior_pics_flag is equal to 1 or is inferred to be equal to 1, all frame buffers in the DPB are emptied without output of the pictures they contain, and DPB fullness is set to 0.

– Otherwise (the decoded picture is not an IDR picture), the following applies:

  – If the slice header of the current picture includes memory_management_control_operation equal to 5, all reference pictures in the DPB are marked as "unused for reference".

  – Otherwise (the slice header of the current picture does not include memory_management_control_operation equal to 5), the decoded reference picture marking process specified in subclause 8.2.5 is invoked when a coded video sequence conforming to one or more of the profiles specified in Annex A is decoded by applying the decoding process specified in clauses 2-9, or the decoded reference picture marking process specified in subclause G.8.2.4 is invoked when a coded video sequence conforming to one or more of the profiles specified in Annex G is decoded by applying the decoding process specified in Annex G, or the decoded reference picture marking process specified in subclause H.8.3 is invoked when a coded video sequence conforming to one or more of the profiles specified in Annex H is decoded by applying the decoding process specified in Annex H.

All pictures m in the DPB, for which all of the following conditions are true, are removed from the DPB:

– picture m is marked as "unused for reference" or picture m is a non-reference picture. When a picture is a reference frame, it is considered to be marked as "unused for reference" only when both of its fields have been marked as "unused for reference",

– picture m is marked as "non-existing" or it has OutputFlag equal to 0 or its DPB output time $t_{o,dpb}(m)$ is less than or equal to the CPB removal time $t_r(n)$ of the current picture n.

When a frame or the last field in a frame buffer is removed from the DPB, the DPB fullness is decremented by one.

C.2.4 Current decoded picture marking and storage

When the decoding process specified in Annex H is applied, the process specified in this subclause is invoked for a particular view with view order index VOIdx, with "picture" being replaced by "view component", "frame" being replaced by "frame view component", and "field" being replaced by "field view component". In subclause C.2.4.2, the DPB output time $t_{o,dpb}(n)$ and the CPB removal time $t_r(n)$ of a view component are the DPB output time and the CPB removal time of the picture n containing the view component.

The marking and storage of the current decoded picture is specified as follows:

– If the current picture is a reference picture, the marking and storage process for reference pictures as specified in subclause C.2.4.1 is invoked.

– Otherwise (the current picture is a non-reference picture), the storage process for non-reference pictures as specified in subclause C.2.4.2 is invoked.

C.2.4.1 Marking and storage of a reference picture into the DPB

The current picture is stored in the DPB as follows:

– If the current decoded picture is a second field (in decoding order) of a complementary reference field pair, and the first field of the pair is still in the DPB, the current decoded picture is stored in the same frame buffer as the first field of the pair.

– Otherwise, the current decoded picture is stored in an empty frame buffer, and the DPB fullness is incremented by one.

When the coded video sequence conforms to one or more of the profiles specified in Annex G and the decoding process specified in Annex G is applied and the current picture has store_ref_base_pic_flag equal to 1 (i.e., the current picture is associated with a reference base picture), the associated reference base picture is stored in the DPB as follows:

– If the reference base picture is a second field (in decoding order) of a complementary reference base field pair, and the first field of the pair is still in the DPB, the reference base picture is stored in the same frame buffer as the first field of the pair.

– Otherwise, the reference base picture is stored in an empty frame buffer, and the DPB fullness is incremented by one.

C.2.4.2 Storage of a non-reference picture into the DPB

The variable storePicFlag is derived as follows:

– If any of the following conditions are true, storePicFlag is set equal to 1:

  – the current picture n has OutputFlag equal to 1 and $t_{o,dpb}(n) > t_r(n)$,

  – the decoding process specified in Annex H is used and the current view component has a view order index VOIdx less than VOIdxMax and inter_view_flag equal to 1.

– Otherwise, storePicFlag is set equal to 0.

When storePicFlag is equal to 1, the current picture is stored in the DPB as follows:

– If the current decoded picture is a second field (in decoding order) of a complementary non-reference field pair, and the first field of the pair is still in the DPB, the current decoded picture is stored in the same frame buffer as the first field of the pair.

– Otherwise, the current decoded picture is stored in an empty frame buffer, and the DPB fullness is incremented by one.

C.3 Bitstream conformance

A bitstream of coded data conforming to this Recommendation | International Standard fulfils the following requirements.

The bitstream is constructed according to the syntax, semantics, and constraints specified in this Recommendation | International Standard outside of this annex.

The bitstream is tested by the HRD as specified below:

For Type I bitstreams, the number of tests carried out is equal to cpb_cnt_minus1 + 1 where cpb_cnt_minus1 is either the syntax element of hrd_parameters( ) following the vcl_hrd_parameters_present_flag or is determined by the application by other means not specified in this Recommendation | International Standard. One test is carried out for each bit rate and CPB size combination specified by hrd_parameters( ) following the vcl_hrd_parameters_present_flag. Each of these tests is conducted at the Type I conformance point shown in Figure C-1.

For Type II bitstreams there are two sets of tests. The number of tests of the first set is equal to cpb_cnt_minus1 + 1 where cpb_cnt_minus1 is either the syntax element of hrd_parameters( ) following the vcl_hrd_parameters_present_flag or is determined by the application by other means not specified in this Recommendation | International Standard. One test is carried out for each bit rate and CPB size combination. Each of these tests is conducted at the Type I conformance point shown in Figure C-1. For these tests, only VCL and filler data NAL units are counted for the input bit rate and CPB storage.

The number of tests of the second set, for Type II bitstreams, is equal to cpb_cnt_minus1 + 1 where cpb_cnt_minus1 is either the syntax element of hrd_parameters( ) following the nal_hrd_parameters_present_flag or is determined by the application by other means not specified in this Recommendation | International Standard. One test is carried out for each bit rate and CPB size combination specified by hrd_parameters( ) following the nal_hrd_parameters_present_flag. Each of these tests is conducted at the Type II conformance point shown in Figure C-1. For these tests, all NAL units (of a Type II NAL unit stream) or all bytes (of a byte stream) are counted for the input bit rate and CPB storage.

NOTE 1 – NAL HRD parameters established by a value of SchedSelIdx for the Type II conformance point shown in Figure C-1 are sufficient to also establish VCL HRD conformance for the Type I conformance point shown in Figure C-1 for the same values of initial_cpb_removal_delay[SchedSelIdx], BitRate[SchedSelIdx], and CpbSize[SchedSelIdx] for the VBR case (cbr_flag[SchedSelIdx] equal to 0). This is because the data flow into the Type I conformance point is a subset of the data flow into the Type II conformance point and because, for the VBR case, the CPB is allowed to become empty and stay empty until the time a next picture is scheduled to begin to arrive. For example, when a coded video sequence conforming to one or more of the profiles specified in Annex A is decoded by applying the decoding process specified in clauses 2-9, when NAL HRD parameters are provided for the Type II conformance point that not only fall within the bounds set for NAL HRD parameters for profile conformance in item j) of subclause A.3.1 or item h) of subclause A.3.3 (depending on the profile in use) but also fall within the bounds set for VCL HRD parameters for profile conformance in item i) of subclause A.3.1 or item g) of subclause A.3.3 (depending on the profile in use), conformance of the VCL HRD for the Type I conformance point is also assured to fall within the bounds of item i) of subclause A.3.1.

For conforming bitstreams, all of the following conditions shall be fulfilled for each of the tests:

1. For each access unit n, with n>0, associated with a buffering period SEI message, with $\Delta t_{g,90}(n)$ specified by

$$\Delta t_{g,90}(n) = 90000 * (t_{r,n}(n) - t_{af}(n - 1)) \qquad \text{(C-14)}$$

   the value of initial_cpb_removal_delay[SchedSelIdx] shall be constrained as follows:

   – If cbr_flag[SchedSelIdx] is equal to 0,

$$\text{initial\_cpb\_removal\_delay[SchedSelIdx]} <= \text{Ceil}(\Delta t_{g,90}(n)) \qquad \text{(C-15)}$$

   – Otherwise (cbr_flag[SchedSelIdx] is equal to 1),

$$\text{Floor}(\Delta t_{g,90}(n)) <= \text{initial\_cpb\_removal\_delay[SchedSelIdx]} <= \text{Ceil}(\Delta t_{g,90}(n)) \qquad \text{(C-16)}$$

   NOTE 2 – The exact number of bits in the CPB at the removal time of each picture may depend on which buffering period SEI message is selected to initialise the HRD. Encoders must take this into account to ensure that all specified constraints must be obeyed regardless of which buffering period SEI message is selected to initialise the HRD, as the HRD may be initialised at any one of the buffering period SEI messages.

2. A CPB overflow is specified as the condition in which the total number of bits in the CPB is larger than the CPB size. The CPB shall never overflow.

3. A CPB underflow is specified as the condition in which $t_{r,n}(n)$ is less than $t_{af}(n)$. When low_delay_hrd_flag is equal to 0, the CPB shall never underflow.

4. The nominal removal times of pictures from the CPB (starting from the second picture in decoding order), shall satisfy the constraints on $t_{r,n}(n)$ and $t_r(n)$ expressed in subclauses A.3.1 through A.3.3 for the profile and level specified in the bitstream when a coded video sequence conforming to one or more of the profiles specified in Annex A is decoded by applying the decoding process specified in clauses 2-9, and they shall satisfy the constraints on $t_{r,n}(n)$ and $t_r(n)$ expressed in subclauses G.10.2.1 and G.10.2.2 for profile and level specified in the bitstream when a coded video sequence conforming to one or more of the profiles specified in Annex G is decoded by applying the decoding process specified in Annex G, and they shall satisfy the constraints on $t_{r,n}(n)$ and $t_r(n)$ expressed in subclause H.10.2 for the profile and level specified in the bitstream when a coded video sequence conforming to one or more of the profiles specified in Annex H is decoded by applying the decoding process specified in Annex H.

5. Immediately after any decoded picture is added to the DPB, the fullness of the DPB shall be less than or equal to the DPB size as constrained by Annexes A, D, and E and subclauses G.10, G.13, G.14, H.10, H.13, and H.14 for the profile and level specified in the bitstream.

6. All reference pictures shall be present in the DPB when needed for prediction. Each picture shall be present in the DPB at its DPB output time unless it is not stored in the DPB at all, or is removed from the DPB before its output time by one of the processes specified in subclause C.2.

7. The value of $\Delta_{to,dpb}(n)$ as given by Equation C-13, which is the difference between the output time of a picture and that of the first picture following it in output order and having OutputFlag equal to 1, shall satisfy the constraint expressed in subclause A.3.1 for the profile and level specified in the bitstream when a coded video sequence conforming to one or more of the profiles specified in Annex A is decoded by applying the decoding process specified in clauses 2-9, and it shall satisfy the constraint expressed in subclause G.10.2.1 for profile and level specified in the bitstream when a coded video sequence conforming to one or more of the profiles specified in Annex G is decoded by applying the decoding process specified in Annex G, and it shall satisfy the constraints expressed in subclause H.10.2 for the profile and level specified in the bitstream when a coded video sequence conforming to one or more of the profiles specified in Annex H is decoded by applying the decoding process specified in Annex H.

C.4 Decoder conformance

A decoder conforming to this Recommendation | International Standard fulfils the following requirements.

A decoder claiming conformance to a specific profile and level shall be able to decode successfully all conforming bitstreams specified for decoder conformance in subclause C.3, provided that all sequence parameter sets and picture parameter sets referred to in the VCL NAL units, and appropriate buffering period and picture timing SEI messages are conveyed to the decoder, in a timely manner, either in the bitstream (by non-VCL NAL units), or by external means not specified by this Recommendation | International Standard.

There are two types of conformance that can be claimed by a decoder: output timing conformance and output order conformance.

To check conformance of a decoder, test bitstreams conforming to the claimed profile and level, as specified in subclause C.3 are delivered by a hypothetical stream scheduler (HSS) both to the HRD and to the decoder under test (DUT). All pictures output by the HRD shall also be output by the DUT and, for each picture output by the HRD, the values of all samples that are output by the DUT for the corresponding picture shall be equal to the values of the samples output by the HRD.

For output timing decoder conformance, the HSS operates as described above, with delivery schedules selected only from the subset of values of SchedSelIdx for which the bit rate and CPB size are restricted as specified in Annex A, Annex G, and Annex H for the specified profile and level, or with "interpolated" delivery schedules as specified below for which the bit rate and CPB size are restricted as specified in Annex A, Annex G, and Annex H. The same delivery schedule is used for both the HRD and DUT.

When the HRD parameters and the buffering period SEI messages are present with cpb_cnt_minus1 greater than 0, the decoder shall be capable of decoding the bitstream as delivered from the HSS operating using an "interpolated" delivery schedule specified as having peak bit rate r, CPB size c(r), and initial CPB removal delay (f(r) ÷ r) as follows:

$$\alpha = (r - \text{BitRate}[\ \text{SchedSelIdx} - 1\ ]) \div (\text{BitRate}[\ \text{SchedSelIdx}\ ] - \text{BitRate}[\ \text{SchedSelIdx} - 1\ ]), \qquad \text{(C-17)}$$

$$c(r) = \alpha * \text{CpbSize}[\ \text{SchedSelIdx}\ ] + (1 - \alpha) * \text{CpbSize}[\ \text{SchedSelIdx}-1\ ], \qquad \text{(C-18)}$$

$$\begin{aligned} f(r) = {} & \alpha * \text{initial\_cpb\_removal\_delay}[\ \text{SchedSelIdx}\ ] * \text{BitRate}[\ \text{SchedSelIdx}\ ] + \\ & (1 - \alpha) * \text{initial\_cpb\_removal\_delay}[\ \text{SchedSelIdx} - 1\ ] * \text{BitRate}[\ \text{SchedSelIdx} - 1\ ] \end{aligned} \qquad \text{(C-19)}$$

for any SchedSelIdx > 0 and r such that BitRate[SchedSelIdx − 1] <= r <= BitRate[SchedSelIdx] such that r and c(r) are within the limits as specified in Annex A, Annex G, and Annex H for the maximum bit rate and buffer size for the specified profile and level.

NOTE 1 – initial_cpb_removal_delay[SchedSelIdx] can be different from one buffering period to another and have to be re-calculated.

For output timing decoder conformance, an HRD as described above is used and the timing (relative to the delivery time of the first bit) of picture output is the same for both HRD and the DUT up to a fixed delay.

For output order decoder conformance, the HSS delivers the bitstream to the DUT "by demand" from the DUT, meaning that the HSS delivers bits (in decoding order) only when the DUT requires more bits to proceed with its processing.

NOTE 2 – This means that for this test, the coded picture buffer of the DUT could be as small as the size of the largest access unit.

A modified HRD as described below is used, and the HSS delivers the bitstream to the HRD by one of the schedules specified in the bitstream such that the bit rate and CPB size are restricted as specified in Annex A, Annex G, and Annex H. The order of pictures output shall be the same for both HRD and the DUT.

For output order decoder conformance, the HRD CPB size is equal to CpbSize[SchedSelIdx] for the selected schedule and the DPB size is equal to MaxDpbFrames. Removal time from the CPB for the HRD is equal to final bit arrival time and decoding is immediate. The operation of the DPB of this HRD is specified in subclause C.4.1.

C.4.1 Operation of the output order DPB

The decoded picture buffer contains frame buffers. When a coded video sequence conforming to one or more of the profiles specified in Annex A is decoded by applying the decoding process specified in clauses 2-9, each of the frame buffers may contain a decoded frame, a decoded complementary field pair or a single (non-paired) decoded field that is marked as "used for reference" or is held for future output (reordered pictures). When a coded video sequence conforming to one or more of the profiles specified in Annex G is decoded by applying the decoding process specified in Annex G, each frame buffer may contain a decoded frame, a decoded complementary field pair, a single (non-paired) decoded field, a decoded reference base frame, a decoded reference base complementary field pair or a single (non-paired) decoded reference base field that is marked as "used for reference" (reference pictures) or is held for future output (reordered or delayed pictures). When a coded video sequence conforming to one or more of the profiles specified in Annex H is decoded by applying the decoding process specified in Annex H, each of the frame buffers may contain a decoded frame view component, a decoded complementary field view component pair, or a single (non-paired) decoded field view component that is marked as "used for reference" (reference pictures) or is held for future output (reordered or delayed pictures) or is held for inter-view prediction (inter-view only reference components).

At HRD initialisation, the DPB fullness, measured in non-empty frame buffers, is set equal to 0. The following steps all happen instantaneously when an access unit is removed from the CPB, and in the order listed. When the decoding process specified in Annex H is applied, the view components of the current primary coded picture are processed by applying the ordered steps to each view component in increasing order of the associated view order index VOIdx.

1. The process of decoding gaps in frame_num and storing "non-existing" frames as specified in subclause C.4.2 is invoked.
2. The picture decoding and output process as specified in subclause C.4.3 is invoked.
3. The process of removing pictures from the DPB before possible insertion of the current picture as specified in subclause C.4.4 is invoked.
4. The process of marking and storing the current decoded picture as specified in subclause C.4.5 is invoked.

NOTE – When the decoding process specified in Annex G is applied, the DPB is only operated for decoded pictures and reference base pictures associated with decoded pictures. The DPB is not operated for layer pictures with dependency_id less than DependencyIdMax (and associated reference base pictures). All decoded pictures and associated reference base pictures are decoded pictures and associated reference base pictures for dependency_id equal to DependencyIdMax, which represent the results of the decoding process specified in subclause G.8.

C.4.2 Decoding of gaps in frame_num and storage of "non-existing" pictures

When the decoding process specified in Annex H is applied, the process specified in this subclause is invoked for a particular view with view order index VOIdx, with "picture" being replaced by "view component", "frame" being replaced by "frame view component", and "field" being replaced by "field view component". During the invocation of the process for a particular view, only view components of the particular view are considered and view components of other views are not marked as "unused for reference" or removed from the DPB. The DPB fullness represents the total number of non-empty frame buffers, including frame buffers that contain view components of other views.

When applicable, gaps in frame_num are detected by the decoding process and the necessary number of "non-existing" frames are inferred in the order specified by the generation of values of UnusedShortTermFrameNum in Equation 7-23 and are marked as specified in subclauses 8.2.5.2 and G.8.2.5. Frame buffers containing a frame or a complementary field pair or a non-paired field which are marked as "not needed for output" and "unused for reference" are emptied (without output), and the DPB fullness is decremented by the number of frame buffers emptied. Each "non-existing" frame is stored in the DPB as follows:

– When there is no empty frame buffer (i.e., DPB fullness is equal to DPB size), the "bumping" process specified in subclause C.4.5.3 is invoked repeatedly until there is an empty frame buffer in which to store the "non-existing" frame.
– The "non-existing" frame is stored in an empty frame buffer and is marked as "not needed for output", and the DPB fullness is incremented by one.

C.4.3 Picture decoding

When the decoding process specified in Annex H is applied, the process specified in this subclause is invoked for a particular view with view order index VOIdx.

The decoding of the current picture or view component (when applying the decoding process specified in Annex H) is specified as follows:

– If the decoding process specified in clause 8 or Annex G is applied, the current primary coded picture n is decoded and is temporarily stored (not in the DPB).
– Otherwise (the decoding process specified in Annex H is applied), the view component with view order index VOIdx of the current primary coded picture n is decoded and is temporarily stored (not in the DPB).

C.4.4 Removal of pictures from the DPB before possible insertion of the current picture

When the decoding process specified in Annex H is applied, the process specified in this subclause is invoked for a particular view with view order index VOIdx, with "picture" being replaced by "view component", "frame" being replaced by "frame view component", and "field" being replaced by "field view component".

When the decoding process specified in Annex H is applied, the following process is specified for emptying frame buffers containing inter-view only reference components of the current access unit. By this process, frame buffers that contain view components of the current view with view order index VOIdx are not emptied, but frame buffers that contain inter-view only reference components of other views may be emptied. The process is specified as follows:

– If the view order index VOIdx of the current view is equal to VOIdxMax, all frame buffers containing a frame or a complementary field pair or a non-paired field which are marked as "not needed for output" and "unused for reference" are emptied (without output), and the DPB fullness is decremented by the number of frame buffers emptied.

  NOTE 1 – At this stage of the process, all frame buffers that contain a frame or a complementary field pair or a non-paired field marked as "not needed for output" and "unused for reference" are frame buffers that contain an inter-view only reference component (of the current access unit and a view with view order index less than VOIdx) with OutputFlag equal to 0.

– Otherwise (the view order index VOIdx of the current view is less than VOIdxMax), frame buffers containing a frame or a complementary field pair or a non-paired field for which both of the following conditions are true are emptied (without output), and the DPB fullness is decremented by the number of frame buffers emptied:

  – the frame or complementary field pair or non-paired field is marked as "not needed for output" and "unused for reference",

    NOTE 2 – At this stage of the process, all frame buffers that contain a frame or a complementary field pair or a non-paired field marked as "not needed for output" and "unused for reference" are frame buffers that contain an inter-view only reference component (of the current access unit and a view with view order index less than VOIdx) with OutputFlag equal to 0.

  – one of the following conditions is true:

    – the current view component is a view component of an anchor picture and the view_id of the frame or complementary field pair or non-paired field is not equal to any value of anchor_ref_lX[k][j], with X being equal to 0 or 1, k being any integer value greater than the view order index VOIdx of the current view, and j being any integer value in the range of 0 to Max(0, num_anchor_refs_lX[k] − 1), inclusive,

    – the current view component is not a view component of an anchor picture and the view_id of the frame or complementary field pair or non-paired field is not equal to any value of non_anchor_ref_lX[k][j], with X being equal to 0 or 1, k being any integer value greater than the view order index VOIdx of the current view, and j being any integer value in the range of 0 to Max(0, num_non_anchor_refs_lX[k] − 1), inclusive.

When the decoding process specified in Annex H is applied, for the following processes specified in this subclause, only view components of the particular view for which this subclause is invoked are considered, and frame buffers containing view components of other views are not emptied. The DPB fullness represents the total number of non-empty frame buffers, including frame buffers that contain view components of other views.

The removal of pictures from the DPB before possible insertion of the current picture proceeds as follows:

– If the decoded picture is an IDR picture the following applies:

  1. All reference pictures in the DPB are marked as "unused for reference" as specified in subclause 8.2.5 when a coded video sequence conforming to one or more of the profiles specified in Annex A is decoded by applying the decoding process specified in clauses 2-9, or as specified in subclause G.8.2.4 when a coded video sequence conforming to one or more of the profiles specified in Annex G is decoded by applying the decoding process specified in Annex G, or as specified in subclause H.8.3 when a coded video sequence conforming to one or more of the profiles specified in Annex H is decoded by applying the decoding process specified in Annex H.

  2. When the IDR picture is not the first IDR picture decoded and the value of PicWidthInMbs or FrameHeightInMbs or max_dec_frame_buffering derived from the active sequence parameter set is different from the value of PicWidthInMbs or FrameHeightInMbs or max_dec_frame_buffering derived from the sequence parameter set that was active for the preceding picture, respectively, no_output_of_prior_pics_flag is inferred to be equal to 1 by the HRD, regardless of the actual value of no_output_of_prior_pics_flag.

     NOTE 3 – Decoder implementations should try to handle changes in the value of PicWidthInMbs or FrameHeightInMbs or max_dec_frame_buffering more gracefully than the HRD.

  3. When no_output_of_prior_pics_flag is equal to 1 or is inferred to be equal to 1, all frame buffers in the DPB are emptied without output of the pictures they contain, and DPB fullness is set to 0.

– Otherwise (the decoded picture is not an IDR picture), the decoded reference picture marking process is invoked as specified in subclause 8.2.5 when a coded video sequence conforming to one or more of the profiles specified in Annex A is decoded by applying the decoding process specified in clauses 2-9, or as specified in subclause G.8.2.4 when a coded video sequence conforming to one or more of the profiles specified in Annex G is decoded by applying the decoding process specified in Annex G, or as specified in subclause H.8.3 when a coded video sequence conforming to one or more of the profiles specified in Annex H is decoded by applying the decoding process specified in Annex H. Frame buffers containing a frame or a complementary field pair or a non-paired field which are marked as "not needed for output" and "unused for reference" are emptied (without output), and the DPB fullness is decremented by the number of frame buffers emptied.

When the current picture has a memory_management_control_operation equal to 5 or is an IDR picture for which no_output_of_prior_pics_flag is not equal to 1 and is not inferred to be equal to 1, the following two steps are performed.

1. Frame buffers containing a frame or a complementary field pair or a non-paired field which are marked as "not needed for output" and "unused for reference" are emptied (without output), and the DPB fullness is decremented by the number of frame buffers emptied.
2. All non-empty frame buffers in the DPB are emptied by repeatedly invoking the "bumping" process specified in subclause C.4.5.3, and the DPB fullness is set to 0.

C.4.5 Current decoded picture marking and storage

When the decoding process specified in Annex H is applied, the process specified in this subclause is invoked for a particular view with view order index VOIdx, with "picture" being replaced by "view component", "frame" being replaced by "frame view component", and "field" being replaced by "field view component". During the invocation of the process for a particular view, only view components of the particular view are considered and frame buffers containing view components of other views are not emptied. The DPB fullness represents the total number of non-empty frame buffers, including frame buffers that contain view components of other views.

The marking and storage of the current decoded picture is specified as follows:

– If the current picture is a reference picture, the storage and marking process for decoded reference pictures as specified in subclause C.4.5.1 is invoked.

– Otherwise (the current picture is a non-reference picture), the storage and marking process for decoded non-reference pictures as specified in subclause C.4.5.2 is invoked.

C.4.5.1 Storage and marking of a reference decoded picture into the DPB

The current picture is stored in the DPB as follows:

– If the current decoded picture is the second field (in decoding order) of a complementary reference field pair, and the first field of the pair is still in the DPB, the current picture is stored in the same frame buffer as the first field of the pair and the following applies:
  – If the current decoded picture has OutputFlag equal to 1, it is marked as "needed for output".
  – Otherwise (the current decoded picture has OutputFlag equal to 0), it is marked as "not needed for output".

– Otherwise, the following operations are performed:
  1. When there is no empty frame buffer (i.e., DPB fullness is equal to DPB size), the "bumping" process specified in subclause C.4.5.3 is invoked repeatedly until there is an empty frame buffer in which to store the current decoded picture.
  2. The current decoded picture is stored in an empty frame buffer, the DPB fullness is incremented by one, and the following applies:
     – If the current decoded picture has OutputFlag equal to 1, it is marked as "needed for output".
     – Otherwise (the current decoded picture has OutputFlag equal to 0), it is marked as "not needed for output".

When the coded video sequence conforms to one or more of the profiles specified in Annex G and the decoding process specified in Annex G is applied and the current picture has store_ref_base_pic_flag equal to 1 (i.e., the current picture is associated with a reference base picture), the associated reference base picture is stored in the DPB as follows:

– If the reference base picture is a second field (in decoding order) of a complementary reference base field pair, and the first field of the pair is still in the DPB, the reference base picture is stored in the same frame buffer as the first field of the pair and marked as "not needed for output".

– Otherwise, the following operations are performed:
  1. When there is no empty frame buffer (i.e., DPB fullness is equal to DPB size), the "bumping" process specified in subclause C.4.5.3 is invoked repeatedly until there is an empty frame buffer in which to store the reference base picture.

2. The reference base picture is stored in an empty frame buffer and marked as "not needed for output" and the DPB fullness is incremented by one.

C.4.5.2 Storage and marking of a non-reference decoded picture into the DPB

The current picture is associated with a variable StoreInterViewOnlyRefFlag, which is derived as follows:

– If the decoding process specified in Annex H is applied, the current view component has a view order index VOIdx less than VOIdxMax and inter_view_flag equal to 1, StoreInterViewOnlyRefFlag is set equal to 1.

– Otherwise, StoreInterViewOnlyRefFlag is set equal to 0.

The current picture is stored in the DPB or output as follows:

– If the current decoded picture is the second field (in decoding order) of a complementary non-reference field pair and the first field of the pair is still in the DPB, the current picture is stored in the same frame buffer as the first field of the pair and the following applies:
  – If the current decoded picture has OutputFlag equal to 1, it is marked as "needed for output".
  – Otherwise (the current decoded picture has OutputFlag equal to 0), it is marked as "not needed for output".

– Otherwise, if the current picture has OutputFlag equal to 0 and StoreInterViewOnlyRefFlag equal to 0, the DPB is not modified and the current picture is not output.

– Otherwise, if the current picture has StoreInterViewOnlyRefFlag equal to 1, the following operations are performed:
  1. When there is no empty frame buffer (i.e., DPB fullness is equal to DPB size), the "bumping" process specified in subclause C.4.5.3 is invoked repeatedly until there is an empty frame buffer in which to store the current decoded picture.
  2. The current decoded picture is stored in an empty frame buffer, the DPB fullness is incremented by one, and the following applies:
     – If the current decoded picture has OutputFlag equal to 1, it is marked as "needed for output".
     – Otherwise (the current decoded picture has OutputFlag equal to 0), it is marked as "not needed for output".

– Otherwise, the following operations are performed repeatedly until the current decoded picture has been cropped and output or has been stored in the DPB:
  – If there is no empty frame buffer (i.e., DPB fullness is equal to DPB size), the following applies:
     – If the current picture does not have a lower value of PicOrderCnt( ) than all pictures in the DPB that are marked as "needed for output", the "bumping" process described in subclause C.4.5.3 is performed.
     – Otherwise (the current picture has a lower value of PicOrderCnt( ) than all pictures in the DPB that are marked as "needed for output"), the current picture is cropped, using the cropping rectangle specified in the active sequence parameter set for the picture and the cropped picture is output.
  – Otherwise (there is an empty frame buffer, i.e., DPB fullness is less than DPB size), the current decoded picture is stored in an empty frame buffer and is marked as "needed for output", and the DPB fullness is incremented by one.

C.4.5.3 "Bumping" process

When the decoding process specified in Annex H is applied, the process specified in this subclause is invoked for a particular view with view order index VOIdx, with "picture" being replaced by "view component", "frame" being replaced by "frame view component", and "field" being replaced by "field view component". During the invocation of the process for a particular view, only view components of the particular view are considered and frame buffers containing view components of other views are not emptied. The DPB fullness represents the total number of non-empty frame buffers, including frame buffers that contain view components of other views.

The "bumping" process is invoked in the following cases.

– There is no empty frame buffer (i.e., DPB fullness is equal to DPB size) and an empty frame buffer is needed for storage of an inferred "non-existing" frame, as specified in subclause C.4.2.

– The current picture is an IDR picture and no_output_of_prior_pics_flag is not equal to 1 and is not inferred to be equal to 1, as specified in subclause C.4.4.

– The current picture has memory_management_control_operation equal to 5, as specified in subclause C.4.4.

– There is no empty frame buffer (i.e., DPB fullness is equal to DPB size) and an empty frame buffer is needed for storage of a decoded (non-IDR) reference picture or a reference base picture, as specified in subclause C.4.5.1.

– There is no empty frame buffer (i.e., DPB fullness is equal to DPB size) and the current picture is a non-reference picture that is not the second field of a complementary non-reference field pair and the current picture has OutputFlag equal to 1 and there are pictures in the DPB that are marked as "needed for output" that precede the current non-reference picture in output order, as specified in subclause C.4.5.2, so an empty buffer is needed for storage of the current picture.

– There is no empty frame buffer (i.e., DPB fullness is equal to DPB size) and the current picture is a non-reference picture that is not the second field of a complementary non-reference field pair and the current picture has StoreInterViewOnlyRefFlag equal to 1, as specified in subclause C.4.5.2, so an empty buffer is needed for storage of the current picture.

The "bumping" process consists of the following ordered steps:

1. The picture or complementary reference field pair that is considered first for output is selected as follows:

   a. The frame buffer is selected that contains the picture having the smallest value of PicOrderCnt( ) of all pictures in the DPB marked as "needed for output".

   b. Depending on the frame buffer, the following applies:

      – If this frame buffer contains a complementary non-reference field pair with both fields marked as "needed for output" and both fields have the same PicOrderCnt( ), the first of these two fields in decoding order is considered first for output.

      – Otherwise, if this frame buffer contains a complementary reference field pair with both fields marked as "needed for output" and both fields have the same PicOrderCnt( ), the entire complementary reference field pair is considered first for output.

        NOTE – When the two fields of a complementary reference field pair have the same value of PicOrderCnt( ), this "bumping" process will output these pictures together, although the two fields have different output times from a decoder that satisfies output timing conformance criteria (as specified in subclause C.2.2).

      – Otherwise, the picture in this frame buffer that has the smallest value of PicOrderCnt( ) is considered first for output.

2. Depending on whether a single picture or a complementary reference field pair is considered for output, the following applies:

   – If a single picture is considered first for output, this picture is cropped, using the cropping rectangle specified in the active sequence parameter set for the picture, the cropped picture is output, and the picture is marked as "not needed for output".

   – Otherwise (a complementary reference field pair is considered first for output), the two fields of the complementary reference field pair are both cropped, using the cropping rectangle specified in the active sequence parameter set for the pictures, the two fields of the complementary reference field pair are output together, and both fields of the complementary reference field pair are marked as "not needed for output".

3. The frame buffer that included the picture or complementary reference field pair that was cropped and output is checked, and when any of the following conditions are true, the frame buffer is emptied and the DPB fullness is decremented by 1:

   – The frame buffer contains a non-reference non-paired field.

   – The frame buffer contains a non-reference frame.

   – The frame buffer contains a complementary non-reference field pair with both fields marked as "not needed for output".

   – The frame buffer contains a non-paired reference field marked as "unused for reference".

   – The frame buffer contains a reference frame with both fields marked as "unused for reference".

   – The frame buffer contains a complementary reference field pair with both fields marked as "unused for reference" and "not needed for output".

Annex D

Supplemental enhancement information

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies syntax and semantics for SEI message payloads.

SEI messages assist in processes related to decoding, display or other purposes. However, SEI messages are not required for constructing the luma or chroma samples by the decoding process. Conforming decoders are not required to process this information for output order conformance to this Recommendation | International Standard (see Annex C for the specification of conformance). Some SEI message information is required to check bitstream conformance and for output timing decoder conformance.

In Annex D, specification for presence of SEI messages are also satisfied when those messages (or some subset of them) are conveyed to decoders (or to the HRD) by other means not specified by this Recommendation | International Standard. When present in the bitstream, SEI messages shall obey the syntax and semantics specified in subclauses 7.3.2.3 and 7.4.2.3 and this annex. When the content of an SEI message is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the SEI message is not required to use the same syntax specified in this annex. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

D.1 SEI payload syntax

sei_payload(payloadType, payloadSize) {	C	Descriptor
if(payloadType = = 0)		
buffering_period(payloadSize)	5	
else if(payloadType = = 1)		
pic_timing(payloadSize)	5	
else if(payloadType = = 2)		
pan_scan_rect(payloadSize)	5	
else if(payloadType = = 3)		
filler_payload(payloadSize)	5	
else if(payloadType = = 4)		
user_data_registered_itu_t_t35(payloadSize)	5	
else if(payloadType = = 5)		
user_data_unregistered(payloadSize)	5	
else if(payloadType = = 6)		
recovery_point(payloadSize)	5	
else if(payloadType = = 7)		
dec_ref_pic_marking_repetition(payloadSize)	5	
else if(payloadType = = 8)		
spare_pic(payloadSize)	5	
else if(payloadType = = 9)		
scene_info(payloadSize)	5	
else if(payloadType = = 10)		
sub_seq_info(payloadSize)	5	
else if(payloadType = = 11)		
sub_seq_layer_characteristics(payloadSize)	5	
else if(payloadType = = 12)		
sub_seq_characteristics(payloadSize)	5	

else if(payloadType = = 13)		
full_frame_freeze(payloadSize)	5	
else if(payloadType = = 14)		
full_frame_freeze_release(payloadSize)	5	
else if(payloadType = = 15)		
full_frame_snapshot(payloadSize)	5	
else if(payloadType = = 16)		
progressive_refinement_segment_start(payloadSize)	5	
else if(payloadType = = 17)		
progressive_refinement_segment_end(payloadSize)	5	
else if(payloadType = = 18)		
motion_constrained_slice_group_set(payloadSize)	5	
else if(payloadType = = 19)		
film_grain_characteristics(payloadSize)	5	
else if(payloadType = = 20)		
deblocking_filter_display_preference(payloadSize)	5	
else if(payloadType = = 21)		
stereo_video_info(payloadSize)	5	
else if(payloadType = = 22)		
post_filter_hint(payloadSize)	5	
else if(payloadType = = 23)		
tone_mapping_info(payloadSize)	5	
else if(payloadType = = 24)		
scalability_info(payloadSize) /* specified in Annex G */	5	
else if(payloadType = = 25)		
sub_pic_scalable_layer(payloadSize) /* specified in Annex G */	5	
else if(payloadType = = 26)		
non_required_layer_rep(payloadSize) /* specified in Annex G */	5	
else if(payloadType = = 27)		
priority_layer_info(payloadSize) /* specified in Annex G */	5	
else if(payloadType = = 28)		
layers_not_present(payloadSize) /* specified in Annex G */	5	
else if(payloadType = = 29)		
layer_dependency_change(payloadSize) /* specified in Annex G */	5	
else if(payloadType = = 30)		
scalable_nesting(payloadSize) /* specified in Annex G */	5	
else if(payloadType = = 31)		
base_layer_temporal_hrd(payloadSize) /* specified in Annex G */	5	
else if(payloadType = = 32)		
quality_layer_integrity_check(payloadSize) /* specified in Annex G */	5	
else if(payloadType = = 33)		
redundant_pic_property(payloadSize) /* specified in Annex G */	5	
else if(payloadType = = 34)		
tl0_dep_rep_index(payloadSize) /* specified in Annex G */	5	
else if(payloadType = = 35)		
tl_switching_point(payloadSize) /* specified in Annex G */	5	
else if(payloadType = = 36)		
parallel_decoding_info(payloadSize) /* specified in Annex H */	5	
else if(payloadType = = 37)		

mvc_scalable_nesting(payloadSize) /* specified in Annex H */	5	
else if(payloadType = = 38)		
view_scalability_info(payloadSize) /* specified in Annex H */	5	
else if(payloadType = = 39)		
multiview_scene_info(payloadSize) /* specified in Annex H */	5	
else if(payloadType = = 40)		
multiview_acquisition_info(payloadSize) /* specified in Annex H */	5	
else if(payloadType = = 41)		
non_required_view_component(payloadSize) /* specified in Annex H */	5	
else if(payloadType = = 42)		
view_dependency_change(payloadSize) /* specified in Annex H */	5	
else if(payloadType = = 43)		
operation_points_not_present(payloadSize) /* specified in Annex H */	5	
else if(payloadType = = 44)		
base_view_temporal_hrd(payloadSize) /* specified in Annex H */	5	
else if(payloadType = = 45)		
frame_packing_arrangement(payloadSize)	5	
else		
reserved_sei_message(payloadSize)	5	
if(!byte_aligned( )) {		
bit_equal_to_one /* equal to 1 */	5	f(1)
while(!byte_aligned( ))		
bit_equal_to_zero /* equal to 0 */	5	f(1)
}		
}		

D.1.1 Buffering period SEI message syntax

buffering_period(payloadSize) {	**C**	**Descriptor**
seq_parameter_set_id	5	ue(v)
if(NalHrdBpPresentFlag)		
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
initial_cpb_removal_delay[SchedSelIdx **]**	5	u(v)
initial_cpb_removal_delay_offset[SchedSelIdx **]**	5	u(v)
}		
if(VclHrdBpPresentFlag)		
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
initial_cpb_removal_delay[SchedSelIdx **]**	5	u(v)
initial_cpb_removal_delay_offset[SchedSelIdx **]**	5	u(v)
}		
}		

D.1.2 Picture timing SEI message syntax

pic_timing(payloadSize) {	**C**	**Descriptor**
if(CpbDpbDelaysPresentFlag) {		

cpb_removal_delay	5	u(v)
dpb_output_delay	5	u(v)
}		
if(pic_struct_present_flag) {		
pic_struct	5	u(4)
for(i = 0; i < NumClockTS ; i++) {		
clock_timestamp_flag[i **]**	5	u(1)
if(clock_timestamp_flag[i]) {		
ct_type	5	u(2)
nuit_field_based_flag	5	u(1)
counting_type	5	u(5)
full_timestamp_flag	5	u(1)
discontinuity_flag	5	u(1)
cnt_dropped_flag	5	u(1)
n_frames	5	u(8)
if(full_timestamp_flag) {		
seconds_value /* 0..59 */	5	u(6)
minutes_value /* 0..59 */	5	u(6)
hours_value /* 0..23 */	5	u(5)
} else {		
seconds_flag	5	u(1)
if(seconds_flag) {		
seconds_value /* range 0..59 */	5	u(6)
minutes_flag	5	u(1)
if(minutes_flag) {		
minutes_value /* 0..59 */	5	u(6)
hours_flag	5	u(1)
if(hours_flag)		
hours_value /* 0..23 */	5	u(5)
}		
}		
}		
if(time_offset_length > 0)		
time_offset	5	i(v)
}		
}		
}		
}		

D.1.3 Pan-scan rectangle SEI message syntax

pan_scan_rect(payloadSize) {	C	Descriptor
pan_scan_rect_id	5	ue(v)
pan_scan_rect_cancel_flag	5	u(1)
if(!pan_scan_rect_cancel_flag) {		
pan_scan_cnt_minus1	5	ue(v)
for(i = 0; i <= pan_scan_cnt_minus1; i++) {		
pan_scan_rect_left_offset[i **]**	5	se(v)
pan_scan_rect_right_offset[i **]**	5	se(v)
pan_scan_rect_top_offset[i **]**	5	se(v)
pan_scan_rect_bottom_offset[i **]**	5	se(v)
}		
pan_scan_rect_repetition_period	5	ue(v)
}		
}		

D.1.4 Filler payload SEI message syntax

filler_payload(payloadSize) {	C	Descriptor
for(k = 0; k < payloadSize; k++)		
ff_byte /* equal to 0xFF */	5	f(8)
}		

D.1.5 User data registered by Rec. ITU-T T.35 SEI message syntax

user_data_registered_itu_t_t35(payloadSize) {	C	Descriptor
itu_t_t35_country_code	5	b(8)
if(itu_t_t35_country_code != 0xFF)		
i = 1		
else {		
itu_t_t35_country_code_extension_byte	5	b(8)
i = 2		
}		
do {		
itu_t_t35_payload_byte	5	b(8)
i++		
} while(i < payloadSize)		
}		

D.1.6 User data unregistered SEI message syntax

user_data_unregistered(payloadSize) {	C	Descriptor
uuid_iso_iec_11578	**5**	u(128)
for(i = 16; i < payloadSize; i++)		
user_data_payload_byte	5	b(8)
}		

D.1.7 Recovery point SEI message syntax

recovery_point(payloadSize) {	C	Descriptor
recovery_frame_cnt	5	ue(v)
exact_match_flag	5	u(1)
broken_link_flag	5	u(1)
changing_slice_group_idc	5	u(2)
}		

D.1.8 Decoded reference picture marking repetition SEI message syntax

dec_ref_pic_marking_repetition(payloadSize) {	C	Descriptor
original_idr_flag	5	u(1)
original_frame_num	5	ue(v)
if(!frame_mbs_only_flag) {		
original_field_pic_flag	5	u(1)
if(original_field_pic_flag)		
original_bottom_field_flag	5	u(1)
}		
dec_ref_pic_marking( )	5	
}		

D.1.9 Spare picture SEI message syntax

spare_pic(payloadSize) {	**C**	**Descriptor**
target_frame_num	5	ue(v)
spare_field_flag	5	u(1)
if(spare_field_flag)		
target_bottom_field_flag	5	u(1)
num_spare_pics_minus1	5	ue(v)
for(i = 0; i < num_spare_pics_minus1 + 1; i++) {		
delta_spare_frame_num[i **]**	5	ue(v)
if(spare_field_flag)		
spare_bottom_field_flag[i **]**	5	u(1)
spare_area_idc[i **]**	5	ue(v)
if(spare_area_idc[i] = = 1)		
for(j = 0; j < PicSizeInMapUnits; j++)		
spare_unit_flag[i **][** j **]**	5	u(1)
else if(spare_area_idc[i] = = 2) {		
mapUnitCnt = 0		
for(j=0; mapUnitCnt < PicSizeInMapUnits; j++) {		
zero_run_length[i **][** j **]**	5	ue(v)
mapUnitCnt += zero_run_length[i][j] + 1		
}		
}		
}		
}		

D.1.10 Scene information SEI message syntax

scene_info(payloadSize) {	**C**	**Descriptor**
scene_info_present_flag	5	u(1)
if(scene_info_present_flag) {		
scene_id	5	ue(v)
scene_transition_type	5	ue(v)
if(scene_transition_type > 3)		
second_scene_id	5	ue(v)
}		
}		

D.1.11 Sub-sequence information SEI message syntax

sub_seq_info(payloadSize) {	C	**Descriptor**
sub_seq_layer_num	5	ue(v)
sub_seq_id	5	ue(v)
first_ref_pic_flag	5	u(1)
leading_non_ref_pic_flag	5	u(1)
last_pic_flag	5	u(1)
sub_seq_frame_num_flag	5	u(1)
if(sub_seq_frame_num_flag)		
sub_seq_frame_num	5	ue(v)
}		

D.1.12 Sub-sequence layer characteristics SEI message syntax

sub_seq_layer_characteristics(payloadSize) {	C	**Descriptor**
num_sub_seq_layers_minus1	5	ue(v)
for(layer = 0; layer <= num_sub_seq_layers_minus1; layer++) {		
accurate_statistics_flag	5	u(1)
average_bit_rate	5	u(16)
average_frame_rate	5	u(16)
}		
}		

D.1.13 Sub-sequence characteristics SEI message syntax

sub_seq_characteristics(payloadSize) {	C	**Descriptor**
sub_seq_layer_num	5	ue(v)
sub_seq_id	5	ue(v)
duration_flag	5	u(1)
if(duration_flag)		
sub_seq_duration	5	u(32)
average_rate_flag	5	u(1)
if(average_rate_flag) {		
accurate_statistics_flag	5	u(1)
average_bit_rate	5	u(16)
average_frame_rate	5	u(16)
}		
num_referenced_subseqs	5	ue(v)
for(n = 0; n < num_referenced_subseqs; n++) {		
ref_sub_seq_layer_num	5	ue(v)
ref_sub_seq_id	5	ue(v)
ref_sub_seq_direction	5	u(1)
}		
}		

D.1.14 Full-frame freeze SEI message syntax

full_frame_freeze(payloadSize) {	C	Descriptor
full_frame_freeze_repetition_period	5	ue(v)
}		

D.1.15 Full-frame freeze release SEI message syntax

full_frame_freeze_release(payloadSize) {	C	Descriptor
}		

D.1.16 Full-frame snapshot SEI message syntax

full_frame_snapshot(payloadSize) {	C	Descriptor
snapshot_id	5	ue(v)
}		

D.1.17 Progressive refinement segment start SEI message syntax

progressive_refinement_segment_start(payloadSize) {	C	Descriptor
progressive_refinement_id	5	ue(v)
num_refinement_steps_minus1	5	ue(v)
}		

D.1.18 Progressive refinement segment end SEI message syntax

progressive_refinement_segment_end(payloadSize) {	C	Descriptor
progressive_refinement_id	5	ue(v)
}		

D.1.19 Motion-constrained slice group set SEI message syntax

motion_constrained_slice_group_set(payloadSize) {	C	Descriptor
num_slice_groups_in_set_minus1	5	ue(v)
if(num_slice_groups_minus1 > 0)		
for(i = 0; i <= num_slice_groups_in_set_minus1; i++)		
slice_group_id[i **]**	5	u(v)
exact_sample_value_match_flag	5	u(1)
pan_scan_rect_flag	5	u(1)
if(pan_scan_rect_flag)		
pan_scan_rect_id	5	ue(v)
}		

D.1.20 Film grain characteristics SEI message syntax

film_grain_characteristics(payloadSize) {	C	Descriptor
film_grain_characteristics_cancel_flag	5	u(1)
if(!film_grain_characteristics_cancel_flag) {		
model_id	5	u(2)
separate_colour_description_present_flag	5	u(1)
if(separate_colour_description_present_flag) {		
film_grain_bit_depth_luma_minus8	5	u(3)
film_grain_bit_depth_chroma_minus8	5	u(3)
film_grain_full_range_flag	5	u(1)
film_grain_colour_primaries	5	u(8)
film_grain_transfer_characteristics	5	u(8)
film_grain_matrix_coefficients	5	u(8)
}		
blending_mode_id	5	u(2)
log2_scale_factor	5	u(4)
for(c = 0; c < 3; c++)		
comp_model_present_flag[c]	5	u(1)
for(c = 0; c < 3; c++)		
if(comp_model_present_flag[c]) {		
num_intensity_intervals_minus1[c **]**	5	u(8)
num_model_values_minus1[c **]**	5	u(3)
for(i = 0; i <= num_intensity_intervals_minus1[c]; i++) {		
intensity_interval_lower_bound[c **][** i **]**	5	u(8)
intensity_interval_upper_bound[c **][** i **]**	5	u(8)
for(j = 0; j <= num_model_values_minus1[c]; j++)		
comp_model_value[c **][** i **][** j **]**	5	se(v)
}		
}		
film_grain_characteristics_repetition_period	5	ue(v)
}		
}		

D.1.21 Deblocking filter display preference SEI message syntax

deblocking_filter_display_preference(payloadSize) {	C	Descriptor
deblocking_display_preference_cancel_flag	5	u(1)
if(!deblocking_display_preference_cancel_flag) {		
display_prior_to_deblocking_preferred_flag	5	u(1)
dec_frame_buffering_constraint_flag	5	u(1)
deblocking_display_preference_repetition_period	5	ue(v)
}		
}		

D.1.22 Stereo video information SEI message syntax

stereo_video_info(payloadSize) {	C	Descriptor
field_views_flag	5	u(1)
if(field_views_flag)		
top_field_is_left_view_flag	5	u(1)
else {		
current_frame_is_left_view_flag	5	u(1)
next_frame_is_second_view_flag	5	u(1)
}		
left_view_self_contained_flag	5	u(1)
right_view_self_contained_flag	5	u(1)
}		

D.1.23 Post-filter hint SEI message syntax

post_filter_hint(payloadSize) {	C	Descriptor
filter_hint_size_y	5	ue(v)
filter_hint_size_x	5	ue(v)
filter_hint_type	5	u(2)
for(colour_component = 0; colour_component < 3; colour_component ++)		
for(cy = 0; cy < filter_hint_size_y; cy ++)		
for(cx = 0; cx < filter_hint_size_x; cx ++)		
filter_hint[colour_component **][** cy **][** cx **]**	5	se(v)
additional_extension_flag	5	u(1)
}		

D.1.24 Tone mapping information SEI message syntax

tone_mapping_info(payloadSize) {	**C**	**Descriptor**
tone_map_id	5	ue(v)
tone_map_cancel_flag	5	u(1)
if(!tone_map_cancel_flag) {		
tone_map_repetition_period	5	ue(v)
coded_data_bit_depth	5	u(8)
target_bit_depth	5	u(8)
model_id	5	ue(v)
if(model_id = = 0) {		
min_value	5	u(32)
max_value	5	u(32)
}		
if(model_id = = 1) {		
sigmoid_midpoint	5	u(32)
sigmoid_width	5	u(32)
}		
if(model_id = = 2)		
for(i = 0; i < (1 << target_bit_depth); i++)		
start_of_coded_interval[i]	5	u(v)
if(model_id = = 3) {		
num_pivots	5	u(16)
for(i=0; i < num_pivots; i++) {		
coded_pivot_value[i]	5	u(v)
target_pivot_value[i]	5	u(v)
}		
}		
}		
}		

D.1.25 Frame packing arrangement SEI message syntax

frame_packing_arrangement(payloadSize) {	C	Descriptor
frame_packing_arrangement_id	5	ue(v)
frame_packing_arrangement_cancel_flag	5	u(1)
if(!frame_packing_arrangement_cancel_flag) {		
frame_packing_arrangement_type	5	u(7)
quincunx_sampling_flag	5	u(1)
content_interpretation_type	5	u(6)
spatial_flipping_flag	5	u(1)
frame0_flipped_flag	5	u(1)
field_views_flag	5	u(1)
current_frame_is_frame0_flag	5	u(1)
frame0_self_contained_flag	5	u(1)
frame1_self_contained_flag	5	u(1)
if(!quincunx_sampling_flag && frame_packing_arrangement_type != 5) {		
frame0_grid_position_x	5	u(4)
frame0_grid_position_y	5	u(4)
frame1_grid_position_x	5	u(4)
frame1_grid_position_y	5	u(4)
}		
frame_packing_arrangement_reserved_byte	5	u(8)
frame_packing_arrangement_repetition_period	5	ue(v)
}		
frame_packing_arrangement_extension_flag	5	u(1)
}		

D.1.26 Reserved SEI message syntax

reserved_sei_message(payloadSize) {	C	Descriptor
for(i = 0; i < payloadSize; i++)		
reserved_sei_message_payload_byte	5	b(8)
}		

D.2 SEI payload semantics

D.2.1 Buffering period SEI message semantics

The presence of the buffering period SEI message in the bitstream is specified as follows:

– If NalHrdBpPresentFlag is equal to 1 or VclHrdBpPresentFlag is equal to 1, one buffering period SEI message can be present in any access unit of the bitstream, and one buffering period SEI message shall be present in every IDR access unit and every access unit associated with a recovery point SEI message.

– Otherwise (NalHrdBpPresentFlag is equal to 0 and VclHrdBpPresentFlag is equal to 0), no buffering period SEI messages shall be present in any access unit of the bitstream.

NOTE 1 – For some applications, the frequent presence of a buffering period SEI message may be desirable.

A buffering period is specified as the set of access units between two instances of the buffering period SEI message in decoding order.

seq_parameter_set_id specifies the sequence parameter set for the current coded video sequence. The value of seq_parameter_set_id shall be equal to the value of seq_parameter_set_id in the picture parameter set referenced by the primary coded picture associated with the buffering period SEI message. The value of seq_parameter_set_id shall be in the range of 0 to 31, inclusive.

NOTE 2 – When the sequence parameter set identified by seq_parameter_set_id is not already active, the buffering SEI message will activate the identified sequence parameter set for the current coded video sequence as specified in subclause 7.4.1.2.1.

initial_cpb_removal_delay[SchedSelIdx] specifies the delay for the SchedSelIdx-th CPB between the time of arrival in the CPB of the first bit of the coded data associated with the access unit associated with the buffering period SEI message and the time of removal from the CPB of the coded data associated with the same access unit, for the first buffering period after HRD initialisation. The syntax element has a length in bits given by initial_cpb_removal_delay_length_minus1 + 1. It is in units of a 90 kHz clock. initial_cpb_removal_delay[SchedSelIdx] shall not be equal to 0 and shall not exceed 90000 * (CpbSize[SchedSelIdx] ÷ BitRate[SchedSelIdx]), the time-equivalent of the CPB size in 90 kHz clock units.

initial_cpb_removal_delay_offset[SchedSelIdx] is used for the SchedSelIdx-th CPB in combination with the cpb_removal_delay to specify the initial delivery time of coded access units to the CPB. initial_cpb_removal_delay_offset[SchedSelIdx] is in units of a 90 kHz clock. The initial_cpb_removal_delay_offset[SchedSelIdx] syntax element is a fixed length code having a length in bits given by initial_cpb_removal_delay_length_minus1 + 1. This syntax element is not used by decoders and is needed only for the delivery scheduler (HSS) specified in Annex C.

Over the entire coded video sequence, the sum of initial_cpb_removal_delay[SchedSelIdx] and initial_cpb_removal_delay_offset[SchedSelIdx] shall be constant for each value of SchedSelIdx.

D.2.2 Picture timing SEI message semantics

NOTE 1 – The syntax of the picture timing SEI message is dependent on the content of the sequence parameter set that is active for the primary coded picture associated with the picture timing SEI message. However, unless the picture timing SEI message of an IDR access unit is preceded by a buffering period SEI message within the same access unit, the activation of the associated sequence parameter set (and, for IDR pictures that are not the first picture in the bitstream, the determination that the primary coded picture is an IDR picture) does not occur until the decoding of the first coded slice NAL unit of the primary coded picture. Since the coded slice NAL unit of the primary coded picture follows the picture timing SEI message in NAL unit order, there may be cases in which it is necessary for a decoder to store the RBSP containing the picture timing SEI message until determining the parameters of the sequence parameter that will be active for the primary coded picture, and then perform the parsing of the picture timing SEI message.

The presence of the picture timing SEI message in the bitstream is specified as follows:

– If CpbDpbDelaysPresentFlag is equal to 1 or pic_struct_present_flag is equal to 1, one picture timing SEI message shall be present in every access unit of the coded video sequence.

– Otherwise (CpbDpbDelaysPresentFlag is equal to 0 and pic_struct_present_flag is equal to 0), no picture timing SEI messages shall be present in any access unit of the coded video sequence.

cpb_removal_delay specifies how many clock ticks (see subclause E.2.1) to wait after removal from the CPB of the access unit associated with the most recent buffering period SEI message in a preceding access unit before removing from the buffer the access unit data associated with the picture timing SEI message. This value is also used to calculate an earliest possible time of arrival of access unit data into the CPB for the HSS, as specified in Annex C. The syntax element is a fixed length code having a length in bits given by cpb_removal_delay_length_minus1 + 1. The cpb_removal_delay is the remainder of a modulo $2^{(\text{cpb\_removal\_delay\_length\_minus1} + 1)}$ counter.

NOTE 2 – The value of cpb_removal_delay_length_minus1 that determines the length (in bits) of the syntax element cpb_removal_delay is the value of cpb_removal_delay_length_minus1 coded in the sequence parameter set that is active for the primary coded picture associated with the picture timing SEI message, although cpb_removal_delay specifies a number of clock ticks relative to the removal time of the preceding access unit containing a buffering period SEI message, which may be an access unit of a different coded video sequence.

dpb_output_delay is used to compute the DPB output time of the picture. It specifies how many clock ticks to wait after removal of an access unit from the CPB before the decoded picture can be output from the DPB (see subclause C.2).

NOTE 3 – A picture is not removed from the DPB at its output time when it is still marked as "used for short-term reference" or "used for long-term reference".

NOTE 4 – Only one dpb_output_delay is specified for a decoded picture.

The length of the syntax element dpb_output_delay is given in bits by dpb_output_delay_length_minus1 + 1. When max_dec_frame_buffering is equal to 0, dpb_output_delay shall be equal to 0.

The output time derived from the dpb_output_delay of any picture that is output from an output timing conforming decoder as specified in subclause C.2 shall precede the output time derived from the dpb_output_delay of all pictures in any subsequent coded video sequence in decoding order.

The output time derived from the dpb_output_delay of the second field, in decoding order, of a complementary non-reference field pair shall exceed the output time derived from the dpb_output_delay of the first field of the same complementary non-reference field pair.

The picture output order established by the values of this syntax element shall be the same order as established by the values of PicOrderCnt( ) as specified in subclauses C.4.1 to C.4.5, except that when the two fields of a complementary reference field pair have the same value of PicOrderCnt( ), the two fields have different output times.

For pictures that are not output by the "bumping" process of subclause C.4.5 because they precede, in decoding order, an IDR picture with no_output_of_prior_pics_flag equal to 1 or inferred to be equal to 1, the output times derived from dpb_output_delay shall be increasing with increasing value of PicOrderCnt( ) relative to all pictures within the same coded video sequence subsequent to any picture having a memory_management_control_operation equal to 5.

pic_struct indicates whether a picture should be displayed as a frame or one or more fields, according to Table D-1. Frame doubling (pic_struct equal to 7) indicates that the frame should be displayed two times consecutively, and frame tripling (pic_struct equal to 8) indicates that the frame should be displayed three times consecutively.

NOTE 5 – Frame doubling can facilitate the display, for example, of 25p video on a 50p display and 29.97p video on a 59.94p display. Using frame doubling and frame tripling in combination on every other frame can facilitate the display of 23.98p video on a 59.94p display.

When pic_struct is present (pic_struct_present_flag is equal to 1), the constraints specified in the third column of Table D-1 shall be obeyed.

NOTE 6 – When pic_struct_present_flag is equal to 0, then in many cases default values may be inferred. In the absence of other indications of the intended display type of a picture, the decoder should infer the value of pic_struct as follows:

– If field_pic_flag is equal to 1, pic_struct should be inferred to be equal to (1 + bottom_field_flag).
– Otherwise, if TopFieldOrderCnt is equal to BottomFieldOrderCnt, pic_struct should be inferred to be equal to 0.
– Otherwise, if TopFieldOrderCnt is less than BottomFieldOrderCnt, pic_struct should be inferred to be equal to 3.
– Otherwise (field_pic_flag is equal to 0 and TopFieldOrderCnt is greater than BottomFieldOrderCnt), pic_struct should be inferred to be equal to 4.

pic_struct is only a hint as to how the decoded video should be displayed on an assumed display type (e.g., interlaced or progressive) at an assumed display rate. When another display type or display rate is used by the decoder, then pic_struct does not indicate the display method, but may aid in processing the decoded video for the alternative display. When it is desired for pic_struct to have an effective value in the range of 5 to 8, inclusive, pic_struct_present_flag should be equal to 1, as the above inference rule will not produce these values.

Table D-1 – Interpretation of pic_struct

Value	Indicated display of picture	Restrictions	NumClockTS
0	(progressive) frame	field_pic_flag shall be 0, TopFieldOrderCnt shall be equal to BottomFieldOrderCnt	1
1	top field	field_pic_flag shall be 1, bottom_field_flag shall be 0	1
2	bottom field	field_pic_flag shall be 1, bottom_field_flag shall be 1	1
3	top field, bottom field, in that order	field_pic_flag shall be 0, TopFieldOrderCnt shall be less than or equal to BottomFieldOrderCnt	2
4	bottom field, top field, in that order	field_pic_flag shall be 0, BottomFieldOrderCnt shall be less than or equal to TopFieldOrderCnt	2
5	top field, bottom field, top field repeated, in that order	field_pic_flag shall be 0, TopFieldOrderCnt shall be less than or equal to BottomFieldOrderCnt	3
6	bottom field, top field, bottom field repeated, in that order	field_pic_flag shall be 0, BottomFieldOrderCnt shall be less than or equal to TopFieldOrderCnt	3
7	frame doubling	field_pic_flag shall be 0, fixed_frame_rate_flag shall be 1, TopFieldOrderCnt shall be equal to BottomFieldOrderCnt	2
8	frame tripling	field_pic_flag shall be 0, fixed_frame_rate_flag shall be 1, TopFieldOrderCnt shall be equal to BottomFieldOrderCnt	3
9..15	reserved		

When fixed_frame_rate_flag is equal to 1, it is a requirement of bitstream conformance that the constraints specified as follows shall be obeyed throughout the operation of the following process, which is operated in output order.

1. Prior to output of the first picture of the bitstream (in output order) and prior to the output of the first picture (in output order) of each subsequent coded video sequence for which the content of the active sequence parameter set differs from that of the previously-active sequence parameter set, the variable lastFieldBottom is set equal to "not determined".

2. After the output of each picture, the value of lastFieldBottom is checked and set as follows, using the values of field_pic_flag, bottom_field_flag, pic_struct, TopFieldOrderCnt and BottomFieldOrderCnt (when applicable) for the picture that was output.
   - If field_pic_flag is equal to 1, it is a requirement of bitstream conformance that the value of lastFieldBottom shall not be equal to bottom_field_flag. The value of lastFieldBottom is then set equal to bottom_field_flag.
   - Otherwise (field_pic_flag is equal to 0), the following applies:
     - If pic_struct is present and is equal to 3 or 5, it is a requirement of bitstream conformance that the value of lastFieldBottom shall not be equal to 0. The value of lastFieldBottom is then set equal to $1 - ((\text{pic\_struct} - 1) >> 2)$.
     - Otherwise, if pic_struct is present and is equal to 4 or 6, it is a requirement of bitstream conformance that the value of lastFieldBottom shall not be equal to 1. The value of lastFieldBottom is then set equal to $((\text{pic\_struct} - 1) >> 2)$.

– Otherwise, if TopFieldOrderCnt is less than BottomFieldOrderCnt, it is a requirement of bitstream conformance that the value of lastFieldBottom shall not be equal to 0. The value of lastFieldBottom is then set equal to 1.

– Otherwise, if TopFieldOrderCnt is greater than BottomFieldOrderCnt, it is a requirement of bitstream conformance that the value of lastFieldBottom shall not be equal to 1. The value of lastFieldBottom is then set equal to 0.

– Otherwise (TopFieldOrderCnt is equal to BottomFieldOrderCnt and pic_struct is not present or is not in the range of 3 to 6, inclusive), lastFieldBottom may have any value, and its value is not changed.

NumClockTS is determined by pic_struct as specified in Table D-1. There are up to NumClockTS sets of clock timestamp information for a picture, as specified by clock_timestamp_flag[i] for each set. The sets of clock timestamp information apply to the field(s) or the frame(s) associated with the picture by pic_struct.

The contents of the clock timestamp syntax elements indicate a time of origin, capture, or alternative ideal display. This indicated time is computed as

$$\begin{aligned}\text{clockTimestamp} = (\,(\,\text{hH} * 60 + \text{mM}\,) * 60 + \text{sS}\,) * \text{time\_scale} +\\ \text{nFrames} * (\,\text{num\_units\_in\_tick} * (\,1 + \text{nuit\_field\_based\_flag}\,)\,) + \text{tOffset},\end{aligned} \quad \text{(D-1)}$$

in units of clock ticks of a clock with clock frequency equal to time_scale Hz, relative to some unspecified point in time for which clockTimestamp is equal to 0. Output order and DPB output timing are not affected by the value of clockTimestamp. When two or more frames with pic_struct equal to 0 are consecutive in output order and have equal values of clockTimestamp, the indication is that the frames represent the same content and that the last such frame in output order is the preferred representation.

NOTE 7 – clockTimestamp time indications may aid display on devices with refresh rates other than those well-matched to DPB output times.

clock_timestamp_flag[i **]** equal to 1 indicates that a number of clock timestamp syntax elements are present and follow immediately. clock_timestamp_flag[i] equal to 0 indicates that the associated clock timestamp syntax elements are not present. When NumClockTS is greater than 1 and clock_timestamp_flag[i] is equal to 1 for more than one value of i, the value of clockTimestamp shall be non-decreasing with increasing value of i.

ct_type indicates the scan type (interlaced or progressive) of the source material as specified in Table D-2.

Two fields of a coded frame may have different values of ct_type.

When clockTimestamp is equal for two fields of opposite parity that are consecutive in output order, both with ct_type equal to 0 (progressive) or ct_type equal to 2 (unknown), the two fields are indicated to have come from the same original progressive frame. Two consecutive fields in output order shall have different values of clockTimestamp when the value of ct_type for either field is 1 (interlaced).

Table D-2 – Mapping of ct_type to source picture scan

Value	Original picture scan
0	progressive
1	interlaced
2	unknown
3	reserved

nuit_field_based_flag is used in calculating clockTimestamp, as specified in Equation D-1.

counting_type specifies the method of dropping values of the n_frames as specified in Table D-3.

Table D-3 – Definition of counting_type values

Value	Interpretation
0	no dropping of n_frames count values and no use of time_offset
1	no dropping of n_frames count values
2	dropping of individual zero values of n_frames count
3	dropping of individual MaxFPS − 1 values of n_frames count
4	dropping of the two lowest (value 0 and 1) n_frames counts when seconds_value is equal to 0 and minutes_value is not an integer multiple of 10
5	dropping of unspecified individual n_frames count values
6	dropping of unspecified numbers of unspecified n_frames count values
7..31	reserved

full_timestamp_flag equal to 1 specifies that the n_frames syntax element is followed by seconds_value, minutes_value, and hours_value. full_timestamp_flag equal to 0 specifies that the n_frames syntax element is followed by seconds_flag.

discontinuity_flag equal to 0 indicates that the difference between the current value of clockTimestamp and the value of clockTimestamp computed from the previous clock timestamp in output order can be interpreted as the time difference between the times of origin or capture of the associated frames or fields. discontinuity_flag equal to 1 indicates that the difference between the current value of clockTimestamp and the value of clockTimestamp computed from the previous clock timestamp in output order should not be interpreted as the time difference between the times of origin or capture of the associated frames or fields. When discontinuity_flag is equal to 0, the value of clockTimestamp shall be greater than or equal to all values of clockTimestamp present for the preceding picture in DPB output order.

cnt_dropped_flag specifies the skipping of one or more values of n_frames using the counting method specified by counting_type.

n_frames specifies the value of nFrames used to compute clockTimestamp. n_frames shall be less than

$$\text{MaxFPS} = \text{Ceil}(\ \text{time\_scale} \div (\ 2 * \text{num\_units\_in\_tick}\ )\ ) \qquad \text{(D-2)}$$

NOTE 8 – n_frames is a frame-based counter. For field-specific timing indications, time_offset should be used to indicate a distinct clockTimestamp for each field.

When counting_type is equal to 2 and cnt_dropped_flag is equal to 1, n_frames shall be equal to 1 and the value of n_frames for the previous picture in output order shall not be equal to 0 unless discontinuity_flag is equal to 1.

NOTE 9 – When counting_type is equal to 2, the need for increasingly large magnitudes of tOffset in Equation D-1 when using fixed non-integer frame rates (e.g., 12.5 frames per second with time_scale equal to 50 and num_units_in_tick equal to 2 and nuit_field_based_flag equal to 0) can be avoided by occasionally skipping over the value n_frames equal to 0 when counting (e.g., counting n_frames from 0 to 12, then incrementing seconds_value and counting n_frames from 1 to 12, then incrementing seconds_value and counting n_frames from 0 to 12, etc.).

When counting_type is equal to 3 and cnt_dropped_flag is equal to 1, n_frames shall be equal to 0 and the value of n_frames for the previous picture in output order shall not be equal to MaxFPS − 1 unless discontinuity_flag is equal to 1.

NOTE 10 – When counting_type is equal to 3, the need for increasingly large magnitudes of tOffset in Equation D-1 when using fixed non-integer frame rates (e.g., 12.5 frames per second with time_scale equal to 50 and num_units_in_tick equal to 2 and nuit_field_based_flag equal to 0) can be avoided by occasionally skipping over the value n_frames equal to MaxFPS − 1 when counting (e.g., counting n_frames from 0 to 12, then incrementing seconds_value and counting n_frames from 0 to 11, then incrementing seconds_value and counting n_frames from 0 to 12, etc.).

When counting_type is equal to 4 and cnt_dropped_flag is equal to 1, n_frames shall be equal to 2 and the specified value of sS shall be zero and the specified value of mM shall not be an integer multiple of ten and n_frames for the previous picture in output order shall not be equal to 0 or 1 unless discontinuity_flag is equal to 1.

NOTE 11 – When counting_type is equal to 4, the need for increasingly large magnitudes of tOffset in Equation D-1 when using fixed non-integer frame rates (e.g., 30000÷1001 frames per second with time_scale equal to 60000 and num_units_in_tick equal to 1 001 and nuit_field_based_flag equal to 1) can be reduced by occasionally skipping over the values of n_frames equal to 0 and 1 when counting (e.g., counting n_frames from 0 to 29, then incrementing seconds_value and counting n_frames from 0 to 29, etc., until the seconds_value is zero and minutes_value is not an integer multiple of ten, then counting n_frames from 2 to 29, then incrementing seconds_value and counting n_frames from 0 to 29, etc.). This counting method is well known in industry and is often referred to as "NTSC drop-frame" counting.

When counting_type is equal to 5 or 6 and cnt_dropped_flag is equal to 1, n_frames shall not be equal to 1 plus the value of n_frames for the previous picture in output order modulo MaxFPS unless discontinuity_flag is equal to 1.

NOTE 12 – When counting_type is equal to 5 or 6, the need for increasingly large magnitudes of tOffset in Equation D-1 when using fixed non-integer frame rates can be avoided by occasionally skipping over some values of n_frames when counting. The specific values of n_frames that are skipped are not specified when counting_type is equal to 5 or 6.

seconds_flag equal to 1 specifies that seconds_value and minutes_flag are present when full_timestamp_flag is equal to 0. seconds_flag equal to 0 specifies that seconds_value and minutes_flag are not present.

seconds_value specifies the value of sS used to compute clockTimestamp. The value of seconds_value shall be in the range of 0 to 59, inclusive. When seconds_value is not present, the previous seconds_value in decoding order shall be used as sS to compute clockTimestamp.

minutes_flag equal to 1 specifies that minutes_value and hours_flag are present when full_timestamp_flag is equal to 0 and seconds_flag is equal to 1. minutes_flag equal to 0 specifies that minutes_value and hours_flag are not present.

minutes_value specifies the value of mM used to compute clockTimestamp. The value of minutes_value shall be in the range of 0 to 59, inclusive. When minutes_value is not present, the previous minutes_value in decoding order shall be used as mM to compute clockTimestamp.

hours_flag equal to 1 specifies that hours_value is present when full_timestamp_flag is equal to 0 and seconds_flag is equal to 1 and minutes_flag is equal to 1.

hours_value specifies the value of hH used to compute clockTimestamp. The value of hours_value shall be in the range of 0 to 23, inclusive. When hours_value is not present, the previous hours_value in decoding order shall be used as hH to compute clockTimestamp.

time_offset specifies the value of tOffset used to compute clockTimestamp. The number of bits used to represent time_offset shall be equal to time_offset_length. When time_offset is not present, the value 0 shall be used as tOffset to compute clockTimestamp.

D.2.3 Pan-scan rectangle SEI message semantics

The pan-scan rectangle SEI message syntax elements specify the coordinates of a rectangle relative to the cropping rectangle of the sequence parameter set. Each coordinate of this rectangle is specified in units of one-sixteenth sample spacing relative to the luma sampling grid.

pan_scan_rect_id contains an identifying number that may be used to identify the purpose of the pan-scan rectangle (for example, to identify the rectangle as the area to be shown on a particular display device or as the area that contains a particular actor in the scene). The value of pan_scan_rect_id shall be in the range of 0 to $2^{32} - 2$, inclusive.

Values of pan_scan_rect_id from 0 to 255 and from 512 to $2^{31} - 1$ may be used as determined by the application. Values of pan_scan_rect_id from 256 to 511 and from 2^{31} to $2^{32} - 2$ are reserved for future use by ITU-T | ISO/IEC. Decoders encountering a value of pan_scan_rect_id in the range of 256 to 511 or in the range of 2^{31} to $2^{32} - 2$ shall ignore (remove from the bitstream and discard) it.

pan_scan_rect_cancel_flag equal to 1 indicates that the SEI message cancels the persistence of any previous pan-scan rectangle SEI message in output order. pan_scan_rect_cancel_flag equal to 0 indicates that pan-scan rectangle information follows.

pan_scan_cnt_minus1 specifies the number of pan-scan rectangles that are present in the SEI message. pan_scan_cnt_minus1 shall be in the range of 0 to 2, inclusive. pan_scan_cnt_minus1 equal to 0 indicates that a single pan-scan rectangle is present that applies to all fields of the decoded picture. pan_scan_cnt_minus1 shall be equal to 0 when the current picture is a field. pan_scan_cnt_minus1 equal to 1 indicates that two pan-scan rectangles are present, the first of which applies to the first field of the picture in output order and the second of which applies to the second field of the picture in output order. pan_scan_cnt_minus1 equal to 2 indicates that three pan-scan rectangles are present, the first of which applies to the first field of the picture in output order, the second of which applies to the second field of the picture in output order, and the third of which applies to a repetition of the first field as a third field in output order.

pan_scan_rect_left_offset[i **]**, **pan_scan_rect_right_offset[** i **]**, **pan_scan_rect_top_offset[** i **]**, and **pan_scan_rect_bottom_offset[** i **]**, specify, as signed integer quantities in units of one-sixteenth sample spacing relative to the luma sampling grid, the location of the pan-scan rectangle. The values of each of these four syntax elements shall be in the range of $-2^{31} + 1$ to $2^{31} - 1$, inclusive.

The pan-scan rectangle is specified, in units of one-sixteenth sample spacing relative to a luma frame sampling grid, as the region with frame horizontal coordinates from 16*CropUnitX * frame_crop_left_offset + pan_scan_rect_left_offset[i] to 16 * (16 * PicWidthInMbs − CropUnitX * frame_crop_right_offset) + pan_scan_rect_right_offset[i] − 1 and with vertical coordinates from 16 *CropUnitY * frame_crop_top_offset + pan_scan_rect_top_offset[i] to 16 * (16 * PicHeightInMbs − CropUnitY * frame_crop_bottom_offset) + pan_scan_rect_bottom_offset[i] − 1, inclusive. The value of 16 * CropUnitX * frame_crop_left_offset + pan_scan_rect_left_offset[i] shall be less than or equal to 16 * (16 * PicWidthInMbs − CropUnitX * frame_crop_right_offset) + pan_scan_rect_right_offset[i] − 1; and the value of 16 * CropUnitY * frame_crop_top_offset + pan_scan_rect_top_offset[i] shall be less than or equal to 16 * (16 * PicHeightInMbs − CropUnitY * frame_crop_bottom_offset) + pan_scan_rect_bottom_offset[i] − 1.

When the pan-scan rectangular area includes samples outside of the cropping rectangle, the region outside of the cropping rectangle may be filled with synthesized content (such as black video content or neutral grey video content) for display.

pan_scan_rect_repetition_period specifies the persistence of the pan-scan rectangle SEI message and may specify a picture order count interval within which another pan-scan rectangle SEI message with the same value of pan_scan_rect_id or the end of the coded video sequence shall be present in the bitstream. The value of pan_scan_rect_repetition_period shall be in the range of 0 to 16 384, inclusive. When pan_scan_cnt_minus1 is greater than 0, pan_scan_rect_repetition_period shall not be greater than 1.

pan_scan_rect_repetition_period equal to 0 specifies that the pan-scan rectangle information applies to the current decoded picture only.

pan_scan_rect_repetition_period equal to 1 specifies that the pan-scan rectangle information persists in output order until any of the following conditions are true:

– A new coded video sequence begins.
– A picture in an access unit containing a pan-scan rectangle SEI message with the same value of pan_scan_rect_id is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic).

pan_scan_rect_repetition_period equal to 0 or equal to 1 indicates that another pan-scan rectangle SEI message with the same value of pan_scan_rect_id may or may not be present.

pan_scan_rect_repetition_period greater than 1 specifies that the pan-scan rectangle information persists until any of the following conditions are true:

– A new coded video sequence begins.
– A picture in an access unit containing a pan-scan rectangle SEI message with the same value of pan_scan_rect_id is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + pan_scan_rect_repetition_period.

pan_scan_rect_repetition_period greater than 1 indicates that another pan-scan rectangle SEI message with the same value of pan_scan_rect_id shall be present for a picture in an access unit that is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + pan_scan_rect_repetition_period; unless the bitstream ends or a new coded video sequence begins without output of such a picture.

D.2.4 Filler payload SEI message semantics

This message contains a series of payloadSize bytes of value 0xFF, which can be discarded.

ff_byte shall be a byte having the value 0xFF.

D.2.5 User data registered by Rec. ITU-T T.35 SEI message semantics

This message contains user data registered as specified by Rec. ITU-T T.35, the contents of which are not specified by this Recommendation | International Standard.

itu_t_t35_country_code shall be a byte having a value specified as a country code by Rec. ITU-T T.35 Annex A.

itu_t_t35_country_code_extension_byte shall be a byte having a value specified as a country code by Rec. ITU-T T.35 Annex B.

itu_t_t35_payload_byte shall be a byte containing data registered as specified by Rec. ITU-T T.35.

The ITU-T T.35 terminal provider code and terminal provider oriented code shall be contained in the first one or more bytes of the itu_t_t35_payload_byte, in the format specified by the Administration that issued the terminal provider code. Any remaining itu_t_t35_payload_byte data shall be data having syntax and semantics as specified by the entity identified by the ITU-T T.35 country code and terminal provider code.

D.2.6 User data unregistered SEI message semantics

This message contains unregistered user data identified by a UUID, the contents of which are not specified by this Recommendation | International Standard.

uuid_iso_iec_11578 shall have a value specified as a UUID according to the procedures of ISO/IEC 11578:1996 Annex A.

user_data_payload_byte shall be a byte containing data having syntax and semantics as specified by the UUID generator.

D.2.7 Recovery point SEI message semantics

The recovery point SEI message assists a decoder in determining when the decoding process will produce acceptable pictures for display after the decoder initiates random access or after the encoder indicates a broken link in the coded video sequence. When the decoding process is started with the access unit in decoding order associated with the recovery point SEI message, all decoded pictures at or subsequent to the recovery point in output order specified in this SEI message are indicated to be correct or approximately correct in content. Decoded pictures produced by random access at or before the picture associated with the recovery point SEI message need not be correct in content until the indicated recovery point, and the operation of the decoding process starting at the picture associated with the recovery point SEI message may contain references to pictures not available in the decoded picture buffer.

In addition, by use of the broken_link_flag, the recovery point SEI message can indicate to the decoder the location of some pictures in the bitstream that can result in serious visual artefacts when displayed, even when the decoding process was begun at the location of a previous IDR access unit in decoding order.

NOTE 1 – The broken_link_flag can be used by encoders to indicate the location of a point after which the decoding process for the decoding of some pictures may cause references to pictures that, though available for use in the decoding process, are not the pictures that were used for reference when the bitstream was originally encoded (e.g., due to a splicing operation performed during the generation of the bitstream).

The recovery point is specified as a count in units of frame_num increments subsequent to the frame_num of the current access unit at the position of the SEI message.

NOTE 2 – When HRD information is present in the bitstream, a buffering period SEI message should be associated with the access unit associated with the recovery point SEI message in order to establish initialisation of the HRD buffer model after a random access.

Any picture parameter set RBSP that is referred to by a picture associated with a recovery point SEI message or by any picture following such a picture in decoding order shall be available to the decoding process prior to its activation, regardless of whether or not the decoding process is started at the beginning of the bitstream or with the access unit, in decoding order, that is associated with the recovery point SEI message.

Any sequence parameter set RBSP that is referred to by a picture associated with a recovery point SEI message or by any picture following such a picture in decoding order shall be available to the decoding process prior to its activation, regardless of whether or not the decoding process is started at the beginning of the bitstream or with the access unit, in decoding order, that is associated with the recovery point SEI message.

recovery_frame_cnt specifies the recovery point of output pictures in output order. All decoded pictures in output order are indicated to be correct or approximately correct in content starting at the output order position of the reference picture having the frame_num equal to the frame_num of the VCL NAL units for the current access unit incremented by recovery_frame_cnt in modulo MaxFrameNum arithmetic. recovery_frame_cnt shall be in the range of 0 to MaxFrameNum − 1, inclusive.

exact_match_flag indicates whether decoded pictures at and subsequent to the specified recovery point in output order derived by starting the decoding process at the access unit associated with the recovery point SEI message shall be an exact match to the pictures that would be produced by starting the decoding process at the location of a previous IDR access unit in the NAL unit stream. The value 0 indicates that the match need not be exact and the value 1 indicates that the match shall be exact.

When decoding starts from the location of the recovery point SEI message, all references to not available reference pictures shall be inferred as references to pictures containing only macroblocks coded using Intra macroblock prediction modes and having sample values given by Y samples equal to $(1 << (\text{BitDepth}_Y - 1))$, Cb samples equal to $(1 << (\text{BitDepth}_C - 1))$, and Cr samples equal to $(1 << (\text{BitDepth}_C - 1))$ (mid-level grey) for purposes of determining the conformance of the value of exact_match_flag.

NOTE 3 – When performing random access, decoders should infer all references to not available reference pictures as references to pictures containing only intra macroblocks and having sample values given by Y equal to $(1 << (\text{BitDepth}_Y - 1))$, Cb equal to $(1 << (\text{BitDepth}_C - 1))$, and Cr equal to $(1 << (\text{BitDepth}_C - 1))$ (mid-level grey), regardless of the value of exact_match_flag.

When exact_match_flag is equal to 0, the quality of the approximation at the recovery point is chosen by the encoding process and is not specified by this Recommendation | International Standard.

NOTE 4 – Under some circumstances, the decoding process of pictures depends on the difference DiffPicOrderCnt(picA, picB) between the PicOrderCnt( ) values for two pictures picA and picB. However, no particular values of TopFieldOrderCnt and BottomFieldOrderCnt (as applicable) are specified to be assigned to the reference pictures that are not available due to the initiation of random access at the location of a picture associated with a recovery point SEI message. Also, no particular value has been specified for initialization (for random access purposes) of the related variables prevPicOrderCntMsb, prevPicOrderCntLsb, prevFraneNumOffset, and prevFrameNum. Thus, any values for these variables may be assigned that could hypothetically have resulted from operation of the decoding process starting with a hypothetical preceding IDR picture in decoding order, although such values may not be the same as the values that would have been obtained if the decoding process had started with the actual preceding IDR picture in the bitstream. When performing random access at a picture associated with a recovery point SEI message, it is suggested that decoders should derive the picture order count variables TopFieldOrderCnt and BottomFieldOrderCnt according to the following method:

– A bit range greater than 32 bits should be allocated for the variables TopFieldOrderCnt and BottomFieldOrderCnt for each current picture to be decoded, as well as for the intermediate variables used for deriving these variables as specified in subclause 8.2.1. (Due to the lack of assurance of correspondence of the values used for initialization of the related variables when random access is performed to the values that would be obtained if the decoding process had begun with the preceding IDR picture in decoding order, the calculations involving these variables in the decoding process of subsequent pictures may result in violation of the 32 bit range.)

– Any value within in the range of -2^{31} to $2^{31} - 1$, inclusive, may be assigned to the values of the variables TopFieldOrderCnt and BottomFieldOrderCnt of the reference pictures that are not available due to the random access operation. For example, the value 0 may be assigned to these variables.

– For the derivation of the picture order count variables for the picture at which random access is performed, prevPicOrderCntMsb may be set equal to any integer multiple of MaxPicOrderCntLsb in the range of -2^{31} to $2^{31} - 1$, inclusive, prevPicOrderCntLsb may be set equal to any value in the range of 0 to MaxPicOrderCntLsb − 1, inclusive, prevFrameNumOffset may be set equal to any integer multiple of MaxFrameNum in the range of 0 to $2^{31} - 1$, inclusive, and prevFrameNum may be set equal to any value in the range of 0 to MaxFrameNum − 1, inclusive. For example, the value 0 may be assigned to all of the variables prevPicOrderCntMsb, prevPicOrderCntLsb, prevFrameNumOffset, and prevFrameNum.

When exact_match_flag is equal to 1, it is a requirement of bitstream conformance that the values of the samples in the decoded pictures at or subsequent to the recovery point in output order shall be independent of the values that a decoder assigns to the variables prevPicOrderCntMsb, prevPicOrderCntLsb, prevFrameNumOffset, and prevFrameNum used in subclause 8.2.1 for deriving the picture order count variables for the initialization of the decoding process at the picture associated with the recovery point SEI message, and of the values that are assigned to the TopFieldOrderCnt and BottomFieldOrderCnt variables of the reference pictures that are not available due to the random access operation.

broken_link_flag indicates the presence or absence of a broken link in the NAL unit stream at the location of the recovery point SEI message and is assigned further semantics as follows:

– If broken_link_flag is equal to 1, pictures produced by starting the decoding process at the location of a previous IDR access unit may contain undesirable visual artefacts to the extent that decoded pictures at and subsequent to the access unit associated with the recovery point SEI message in decoding order should not be displayed until the specified recovery point in output order.

– Otherwise (broken_link_flag is equal to 0), no indication is given regarding any potential presence of visual artefacts.

Regardless of the value of the broken_link_flag, pictures subsequent to the specified recovery point in output order are specified to be correct or approximately correct in content.

NOTE 5 – When a sub-sequence information SEI message is present in conjunction with a recovery point SEI message in which broken_link_flag is equal to 1 and when sub_seq_layer_num is equal to 0, sub_seq_id should be different from the latest sub_seq_id for sub_seq_layer_num equal to 0 that was decoded prior to the location of the recovery point SEI message. When broken_link_flag is equal to 0, the sub_seq_id in sub-sequence layer 0 should remain unchanged.

changing_slice_group_idc equal to 0 indicates that decoded pictures are correct or approximately correct in content at and subsequent to the recovery point in output order when all macroblocks of the primary coded pictures are decoded within the changing slice group period, i.e., the period between the access unit associated with the recovery point SEI message (inclusive) and the specified recovery point (inclusive) in decoding order. changing_slice_group_idc shall be equal to 0 when num_slice_groups_minus1 is equal to 0 in any primary coded picture within the changing slice group period.

When changing_slice_group_idc is equal to 1 or 2, num_slice_groups_minus1 shall be equal to 1 and the macroblock-to-slice-group map type 3, 4, or 5 shall be applied in each primary coded picture in the changing slice group period.

> changing_slice_group_idc equal to 1 indicates that within the changing slice group period no sample values outside the decoded macroblocks covered by slice group 0 are used for inter prediction of any macroblock within slice group 0. In addition, changing_slice_group_idc equal to 1 indicates that when all macroblocks in slice group 0 within the changing slice group period are decoded, decoded pictures will be correct or approximately correct in content at and subsequent to the specified recovery point in output order regardless of whether any macroblock in slice group 1 within the changing slice group period is decoded.

changing_slice_group_idc equal to 2 indicates that within the changing slice group period no sample values outside the decoded macroblocks covered by slice group 1 are used for inter prediction of any macroblock within slice group 1. In addition, changing_slice_group_idc equal to 2 indicates that when all macroblocks in slice group 1 within the changing slice group period are decoded, decoded pictures will be correct or approximately correct in content at and subsequent to the specified recovery point in output order regardless of whether any macroblock in slice group 0 within the changing slice group period is decoded.

changing_slice_group_idc shall be in the range of 0 to 2, inclusive.

D.2.8 Decoded reference picture marking repetition SEI message semantics

The decoded reference picture marking repetition SEI message is used to repeat the decoded reference picture marking syntax structure that was located in the slice headers of an earlier picture in the same coded video sequence in decoding order.

original_idr_flag shall be equal to 1 when the decoded reference picture marking syntax structure occurred originally in an IDR picture. original_idr_flag shall be equal to 0 when the repeated decoded reference picture marking syntax structure did not occur in an IDR picture originally.

original_frame_num shall be equal to the frame_num of the picture where the repeated decoded reference picture marking syntax structure originally occurred. The picture indicated by original_frame_num is the previous coded picture having the specified value of frame_num. The value of original_frame_num used to refer to a picture having a memory_management_control_operation equal to 5 shall be 0.

original_field_pic_flag shall be equal to the field_pic_flag of the picture where the repeated decoded reference picture marking syntax structure originally occurred.

original_bottom_field_flag shall be equal to the bottom_field_flag of the picture where the repeated decoded reference picture marking syntax structure originally occurred.

dec_ref_pic_marking( ) shall contain a copy of the decoded reference picture marking syntax structure of the picture that has a value of frame_num equal to original_frame_num. The IdrPicFlag used in the specification of the repeated dec_ref_pic_marking( ) syntax structure shall be the IdrPicFlag of the slice header(s) of the picture that has a value of frame_num equal to original_frame_num (i.e., IdrPicFlag as used in subclause 7.3.3.3 shall be considered equal to original_idr_flag).

D.2.9 Spare picture SEI message semantics

This SEI message indicates that certain slice group map units, called spare slice group map units, in one or more decoded reference pictures resemble the co-located slice group map units in a specified decoded picture called the target picture. A spare slice group map unit may be used to replace a co-located, incorrectly decoded slice group map unit, in the target picture. A decoded picture containing spare slice group map units is called a spare picture.

A spare picture SEI message shall not be present in an IDR access unit. The value of the PicSizeInMapUnits variable for the target picture (as specified later in this subclause) shall be equal to the value of the PicSizeInMapUnits variable for the sequence parameter set that is active when processing the spare picture SEI message.

For all spare pictures identified in a spare picture SEI message, the value of frame_mbs_only_flag shall be equal to the value of frame_mbs_only_flag of the target picture in the same SEI message. The spare pictures in the SEI message are constrained as follows:

– If the target picture is a decoded field, all spare pictures identified in the same SEI message shall be decoded fields.
– Otherwise (the target picture is a decoded frame), all spare pictures identified in the same SEI message shall be decoded frames.

For all spare pictures identified in a spare picture SEI message, the values of pic_width_in_mbs_minus1 and pic_height_in_map_units_minus1 shall be equal to the values of pic_width_in_mbs_minus1 and pic_height_in_map_units_minus1, respectively, of the target picture in the same SEI message. The picture associated (as specified in subclause 7.4.1.2.3) with this SEI message shall appear after the target picture, in decoding order.

target_frame_num indicates the frame_num of the target picture.

spare_field_flag equal to 0 indicates that the target picture and the spare pictures are decoded frames. spare_field_flag equal to 1 indicates that the target picture and the spare pictures are decoded fields.

target_bottom_field_flag equal to 0 indicates that the target picture is a top field. target_bottom_field_flag equal to 1 indicates that the target picture is a bottom field.

A target picture is a decoded reference picture for which the corresponding primary coded picture precedes the current picture, in decoding order, and in which the values of frame_num, field_pic_flag (when present) and bottom_field_flag (when present) are equal to target_frame_num, spare_field_flag and target_bottom_field_flag, respectively.

num_spare_pics_minus1 indicates the number of spare pictures for the specified target picture. The number of spare pictures is equal to num_spare_pics_minus1 + 1. The value of num_spare_pics_minus1 shall be in the range of 0 to 15, inclusive.

delta_spare_frame_num[i] is used to identify the spare picture that contains the i-th set of spare slice group map units, hereafter called the i-th spare picture, as specified below. The value of delta_spare_frame_num[i] shall be in the range of 0 to MaxFrameNum − 2 + spare_field_flag, inclusive.

The frame_num of the i-th spare picture, spareFrameNum[i], is derived as follows for all values of i from 0 to num_spare_pics_minus1, inclusive:

```
candidateSpareFrameNum = target_frame_num − 1 + spare_field_flag
for ( i = 0; i <= num_spare_pics_minus1; i++ ) {
    if( candidateSpareFrameNum < 0 )
        candidateSpareFrameNum = MaxFrameNum − 1
    spareFrameNum[ i ] = candidateSpareFrameNum − delta_spare_frame_num[ i ]          (D-3)
    if( spareFrameNum[ i ] < 0 )
        spareFrameNum[ i ] = MaxFrameNum + spareFrameNum[ i ]
    candidateSpareFrameNum = spareFrameNum[ i ] − 1 + spare_field_flag
}
```

spare_bottom_field_flag[i] equal to 0 indicates that the i-th spare picture is a top field. spare_bottom_field_flag[i] equal to 1 indicates that the i-th spare picture is a bottom field.

The 0-th spare picture is a decoded reference picture for which the corresponding primary coded picture precedes the target picture, in decoding order, and in which the values of frame_num, field_pic_flag (when present) and bottom_field_flag (when present) are equal to spareFrameNum[0], spare_field_flag and spare_bottom_field_flag[0], respectively. The i-th spare picture is a decoded reference picture for which the corresponding primary coded picture precedes the (i − 1)-th spare picture, in decoding order, and in which the values of frame_num, field_pic_flag (when present) and bottom_field_flag (when present) are equal to spareFrameNum[i], spare_field_flag and spare_bottom_field_flag[i], respectively.

spare_area_idc[i] indicates the method used to identify the spare slice group map units in the i-th spare picture. spare_area_idc[i] shall be in the range of 0 to 2, inclusive. spare_area_idc[i] equal to 0 indicates that all slice group map units in the i-th spare picture are spare units. spare_area_idc[i] equal to 1 indicates that the value of the syntax element spare_unit_flag[i][j] is used to identify the spare slice group map units. spare_area_idc[i] equal to 2 indicates that the zero_run_length[i][j] syntax element is used to derive the values of spareUnitFlagInBoxOutOrder[i][j], as described below.

spare_unit_flag[i][j] equal to 0 indicates that the j-th slice group map unit in raster scan order in the i-th spare picture is a spare unit. spare_unit_flag[i][j] equal to 1 indicates that the j-th slice group map unit in raster scan order in the i-th spare picture is not a spare unit.

zero_run_length[i][j] is used to derive the values of spareUnitFlagInBoxOutOrder[i][j] when spare_area_idc[i] is equal to 2. In this case, the spare slice group map units identified in spareUnitFlagInBoxOutOrder[i][j] appear in counter-clockwise box-out order, as specified in subclause 8.2.2.4, for each spare picture. spareUnitFlagInBoxOutOrder[i][j] equal to 0 indicates that the j-th slice group map unit in counter-clockwise box-out order in the i-th spare picture is a spare unit. spareUnitFlagInBoxOutOrder[i][j] equal to 1 indicates that the j-th slice group map unit in counter-clockwise box-out order in the i-th spare picture is not a spare unit.

When spare_area_idc[0] is equal to 2, spareUnitFlagInBoxOutOrder[0][j] is derived as specified by the following pseudo-code:

```
for( j = 0, loop = 0; j < PicSizeInMapUnits; loop++ ) {
   for( k = 0; k < zero_run_length[ 0 ][ loop ]; k++ )
      spareUnitFlagInBoxOutOrder[ 0 ][ j++ ] = 0                    (D-4)
   spareUnitFlagInBoxOutOrder[ 0 ][ j++ ] = 1
}
```

When spare_area_idc[i] is equal to 2 and the value of i is greater than 0, spareUnitFlagInBoxOutOrder[i][j] is derived as specified by the following pseudo-code:

```
for( j = 0, loop = 0; j < PicSizeInMapUnits; loop++ ) {
   for( k = 0; k < zero_run_length[ i ][ loop ]; k++ )
      spareUnitFlagInBoxOutOrder[ i ][ j ] = spareUnitFlagInBoxOutOrder[ i − 1 ][ j++ ]      (D-5)
   spareUnitFlagInBoxOutOrder[ i ][ j ] = !spareUnitFlagInBoxOutOrder[ i − 1 ][ j++ ]
}
```

D.2.10 Scene information SEI message semantics

A scene and a scene transition are herein defined as a set of consecutive pictures in output order.

NOTE 1 – Decoded pictures within one scene generally have similar content. The scene information SEI message is used to label pictures with scene identifiers and to indicate scene changes. The message specifies how the source pictures for the labelled pictures were created. The decoder may use the information to select an appropriate algorithm to conceal transmission errors. For example, a specific algorithm may be used to conceal transmission errors that occurred in pictures belonging to a gradual scene transition. Furthermore, the scene information SEI message may be used in a manner determined by the application, such as for indexing the scenes of a coded sequence.

A scene information SEI message labels all pictures, in decoding order, from the primary coded picture to which the SEI message is associated (inclusive), as specified in subclause 7.4.1.2.3, to the primary coded picture to which the next scene information SEI message (if present) in decoding order is associated (exclusive) or (otherwise) to the last access unit in the bitstream (inclusive). These pictures are herein referred to as the target pictures.

scene_info_present_flag equal to 0 indicates that the scene or scene transition to which the target pictures belong is unspecified. scene_info_present_flag equal to 1 indicates that the target pictures belong to the same scene or scene transition.

scene_id identifies the scene to which the target pictures belong. When the value of scene_transition_type of the target pictures is less than 4, and the previous picture in output order is marked with a value of scene_transition_type less than 4, and the value of scene_id is the same as the value of scene_id of the previous picture in output order, this indicates that the source scene for the target pictures and the source scene for the previous picture (in output order) are considered by the encoder to have been the same scene. When the value of scene_transition_type of the target pictures is greater than 3, and the previous picture in output order is marked with a value of scene_transition_type less than 4, and the value of scene_id is the same as the value of scene_id of the previous picture in output order, this indicates that one of the source scenes for the target pictures and the source scene for the previous picture (in output order) are considered by the encoder to have been the same scene. When the value of scene_id is not equal to the value of scene_id of the previous picture in output order, this indicates that the target pictures and the previous picture (in output order) are considered by the encoder to have been from different source scenes.

The value of scene_id shall be in the range of 0 to $2^{32} - 2$, inclusive.

Values of scene_id in the range of 0 to 255, inclusive, and in the range of 512 to $2^{31} - 1$, inclusive, may be used as determined by the application. Values of scene_id in the range of 256 to 511, inclusive, and in the range of 2^{31} to $2^{32} - 2$, inclusive, are reserved for future use by ITU-T | ISO/IEC. Decoders encountering a value of scene_id in the range of 256 to 511, inclusive, or in the range of 2^{31} to $2^{32} - 2$, inclusive, shall ignore (remove from the bitstream and discard) it.

scene_transition_type specifies in which type of a scene transition (if any) the target pictures are involved. The valid values of scene_transition_type are specified in Table D-4.

Table D-4 – scene_transition_type values

Value	Description
0	No transition
1	Fade to black
2	Fade from black
3	Unspecified transition from or to constant colour
4	Dissolve
5	Wipe
6	Unspecified mixture of two scenes

When scene_transition_type is greater than 3, the target pictures include contents both from the scene labelled by its scene_id and the next scene, in output order, which is labelled by second_scene_id (see below). The term "the current scene" is used to indicate the scene labelled by scene_id. The term "the next scene" is used to indicate the scene labelled by second_scene_id. It is not required for any following picture, in output order, to be labelled with scene_id equal to second_scene_id of the current SEI message.

Scene transition types are specified as follows.

"No transition" specifies that the target pictures are not involved in a gradual scene transition.

NOTE 2 – When two consecutive pictures in output order have scene_transition_type equal to 0 and different values of scene_id, a scene cut occurred between the two pictures.

"Fade to black" indicates that the target pictures are part of a sequence of pictures, in output order, involved in a fade to black scene transition, i.e., the luma samples of the scene gradually approach zero and the chroma samples of the scene gradually approach 128.

NOTE 3 – When two pictures are labelled to belong to the same scene transition and their scene_transition_type is "Fade to black", the later one, in output order, is darker than the previous one.

"Fade from black" indicates that the target pictures are part of a sequence of pictures, in output order, involved in a fade from black scene transition, i.e., the luma samples of the scene gradually diverge from zero and the chroma samples of the scene may gradually diverge from 128.

NOTE 4 – When two pictures are labelled to belong to the same scene transition and their scene_transition_type is "Fade from black", the later one in output order is lighter than the previous one.

"Dissolve" indicates that the sample values of each target picture (before encoding) were generated by calculating a sum of co-located weighted sample values of a picture from the current scene and a picture from the next scene. The weight of the current scene gradually decreases from full level to zero level, whereas the weight of the next scene gradually increases from zero level to full level. When two pictures are labelled to belong to the same scene transition and their scene_transition_type is "Dissolve", the weight of the current scene for the later one, in output order, is less than the weight of the current scene for the previous one, and the weight of the next scene for the later one, in output order, is greater than the weight of the next scene for the previous one.

"Wipe" indicates that some of the sample values of each target picture (before encoding) were generated by copying co-located sample values of a picture in the current scene and the remaining sample values of each target picture (before encoding) were generated by copying co-located sample values of a picture in the next scene. When two pictures are labelled to belong to the same scene transition and their scene_transition_type is "Wipe", the number of samples copied from the next scene to the later picture in output order is greater than the number of samples copied from the next scene to the previous picture.

second_scene_id identifies the next scene in the gradual scene transition in which the target pictures are involved. The value of second_scene_id shall not be equal to the value of scene_id. The value of second_scene_id shall not be equal to the value of scene_id in the previous picture in output order. When the next picture in output order is marked with a value of scene_transition_type less than 4, and the value of second_scene_id is the same as the value of scene_id of the next picture in output order, this indicates that the encoder considers one of the source scenes for the target pictures and the source scene for the next picture (in output order) to have been the same scene. When the value of second_scene_id is not equal to the value of scene_id or second_scene_id (if present) of the next picture in output order, this indicates that the encoder considers the target pictures and the next picture (in output order) to have been from different source scenes.

When the value of scene_id of a picture is equal to the value of scene_id of the following picture in output order and the value of scene_transition_type in both of these pictures is less than 4, this indicates that the encoder considers the two pictures to have been from the same source scene. When the values of scene_id, scene_transition_type and second_scene_id (if present) of a picture are equal to the values of scene_id, scene_transition_type and second_scene_id (respectively) of the following picture in output order and the value of scene_transition_type is greater than 0, this indicates that the encoder considers the two pictures to have been from the same source gradual scene transition.

The value of second_scene_id shall be in the range of 0 to $2^{32} - 2$, inclusive.

Values of second_scene_id in the range of 0 to 255, inclusive, and in the range of 512 to $2^{31} - 1$, inclusive, may be used as determined by the application. Values of second_scene_id in the range of 256 to 511, inclusive, and in the range of 2^{31} to $2^{32} - 2$, inclusive, are reserved for future use by ITU-T | ISO/IEC. Decoders encountering a value of second_scene_id in the range of 256 to 511, inclusive, or in the range of 2^{31} to $2^{32} - 2$, inclusive, shall ignore (remove from the bitstream and discard) it.

D.2.11 Sub-sequence information SEI message semantics

The sub-sequence information SEI message is used to indicate the position of a picture in data dependency hierarchy that consists of sub-sequence layers and sub-sequences.

A sub-sequence layer contains a subset of the coded pictures in a sequence. Sub-sequence layers are numbered with non-negative integers. A layer having a larger layer number is a higher layer than a layer having a smaller layer number. The layers are ordered hierarchically based on their dependency on each other so that any picture in a layer shall not be predicted from any picture on any higher layer.

NOTE 1 – In other words, any picture in layer 0 must not be predicted from any picture in layer 1 or above, pictures in layer 1 may be predicted from layer 0, pictures in layer 2 may be predicted from layers 0 and 1, etc.

NOTE 2 – The subjective quality is expected to increase along with the number of decoded layers.

A sub-sequence is a set of coded pictures within a sub-sequence layer. A picture shall reside in one sub-sequence layer and in one sub-sequence only. Any picture in a sub-sequence shall not be predicted from any picture in another sub-sequence in the same or in a higher sub-sequence layer. A sub-sequence in layer 0 can be decoded independently of any picture that does not belong to the sub-sequence.

The sub-sequence information SEI message concerns the current access unit. The primary coded picture in the access unit is herein referred to as the current picture.

The sub-sequence information SEI message shall not be present unless gaps_in_frame_num_value_allowed_flag in the sequence parameter set referenced by the picture associated with the sub-sequence SEI message is equal to 1.

sub_seq_layer_num specifies the sub-sequence layer number of the current picture. When sub_seq_layer_num is greater than 0, memory management control operations shall not be used in any slice header of the current picture. When the current picture resides in a sub-sequence for which the first picture in decoding order is an IDR picture, the value of sub_seq_layer_num shall be equal to 0. For a non-paired reference field, the value of sub_seq_layer_num shall be equal to 0. sub_seq_layer_num shall be in the range of 0 to 255, inclusive.

sub_seq_id identifies the sub-sequence within a layer. When the current picture resides in a sub-sequence for which the first picture in decoding order is an IDR picture, the value of sub_seq_id shall be the same as the value of idr_pic_id of the IDR picture. sub_seq_id shall be in the range of 0 to 65535, inclusive.

first_ref_pic_flag equal to 1 specifies that the current picture is the first reference picture of the sub-sequence in decoding order. When the current picture is not the first picture of the sub-sequence in decoding order, the first_ref_pic_flag shall be equal to 0.

leading_non_ref_pic_flag equal to 1 specifies that the current picture is a non-reference picture preceding any reference picture in decoding order within the sub-sequence or that the sub-sequence contains no reference pictures. When the current picture is a reference picture or the current picture is a non-reference picture succeeding at least one reference picture in decoding order within the sub-sequence, the leading_non_ref_pic_flag shall be equal to 0.

last_pic_flag equal to 1 indicates that the current picture is the last picture of the sub-sequence (in decoding order), including all reference and non-reference pictures of the sub-sequence. When the current picture is not the last picture of the sub-sequence (in decoding order), last_pic_flag shall be equal to 0.

The current picture is assigned to a sub-sequence as follows:

– If one or more of the following conditions is true, the current picture is the first picture of a sub-sequence in decoding order:
  – no earlier picture in decoding order is labelled with the same values of sub_seq_id and sub_seq_layer_num as the current picture,

– the value of leading_non_ref_pic_flag is equal to 1 and the value of leading_non_ref_pic_flag is equal to 0 in the previous picture in decoding order having the same values of sub_seq_id and sub_seq_layer_num as the current picture,

– the value of first_ref_pic_flag is equal to 1 and the value of leading_non_ref_pic_flag is equal to 0 in the previous picture in decoding order having the same values of sub_seq_id and sub_seq_layer_num as the current picture,

– the value of last_pic_flag is equal to 1 in the previous picture in decoding order having the same values of sub_seq_id and sub_seq_layer_num as the current picture.

– Otherwise, the current picture belongs to the same sub-sequence as the previous picture in decoding order having the same values of sub_seq_id and sub_seq_layer_num as the current picture.

sub_seq_frame_num_flag equal to 0 specifies that sub_seq_frame_num is not present. sub_seq_frame_num_flag equal to 1 specifies that sub_seq_frame_num is present.

sub_seq_frame_num shall be equal to 0 for the first reference picture of the sub-sequence and for any non-reference picture preceding the first reference picture of the sub-sequence in decoding order. sub_seq_frame_num is further constrained as follows:

– If the current picture is not the second field of a complementary field pair, sub_seq_frame_num shall be incremented by 1, in modulo MaxFrameNum operation, relative to the previous reference picture, in decoding order, that belongs to the sub-sequence.

– Otherwise (the current picture is the second field of a complementary field pair), the value of sub_seq_frame_num shall be the same as the value of sub_seq_frame_num for the first field of the complementary field pair.

sub_seq_frame_num shall be in the range of 0 to MaxFrameNum − 1, inclusive.

When the current picture is an IDR picture, it shall start a new sub-sequence in sub-sequence layer 0. Thus, the sub_seq_layer_num shall be 0, the sub_seq_id shall be different from the previous sub-sequence in sub-sequence layer 0, first_ref_pic_flag shall be 1, and leading_non_ref_pic_flag shall be equal to 0.

When the sub-sequence information SEI message is present for both coded fields of a complementary field pair, the values of sub_seq_layer_num, sub_seq_id, leading_non_ref_pic_flag and sub_seq_frame_num, when present, shall be the same for both of these pictures.

When the sub-sequence information SEI message is present only for one coded field of a complementary field pair, the values of sub_seq_layer_num, sub_seq_id, leading_non_ref_pic_flag and sub_seq_frame_num, when present, are also applicable to the other coded field of the complementary field pair.

D.2.12 Sub-sequence layer characteristics SEI message semantics

The sub-sequence layer characteristics SEI message specifies the characteristics of sub-sequence layers.

num_sub_seq_layers_minus1 plus 1 specifies the number of sub-sequence layers in the sequence. num_sub_seq_layers_minus1 shall be in the range of 0 to 255, inclusive.

A pair of average_bit_rate and average_frame_rate characterizes each sub-sequence layer. The first pair of average_bit_rate and average_frame_rate specifies the characteristics of sub-sequence layer 0. When present, the second pair specifies the characteristics of sub-sequence layers 0 and 1 jointly. Each pair in decoding order specifies the characteristics for a range of sub-sequence layers from layer number 0 to the layer number specified by the layer loop counter. The values are in effect from the point they are decoded until an update of the values is decoded.

accurate_statistics_flag equal to 1 indicates that the values of average_bit_rate and average_frame_rate are rounded from statistically correct values. accurate_statistics_flag equal to 0 indicates that the average_bit_rate and the average_frame_rate are estimates and may deviate somewhat from the correct values.

When accurate_statistics_flag is equal to 0, the quality of the approximation used in the computation of the values of average_bit_rate and the average_frame_rate is chosen by the encoding process and is not specified by this Recommendation | International Standard.

average_bit_rate indicates the average bit rate in units of 1000 bits per second. All NAL units in the range of sub-sequence layers specified above are taken into account in the calculation. The average bit rate is derived according to the access unit removal time specified in Annex C of the Recommendation | International Standard. In the following, bTotal is the number of bits in all NAL units succeeding a sub-sequence layer characteristics SEI message (including the bits of the NAL units of the current access unit) and preceding the next access unit (in decoding order) including a sub-sequence layer characteristics SEI message (if present) or the end of the stream (otherwise). t_1 is the removal time (in seconds) of the current access unit, and t_2 is the removal time (in seconds) of the latest access unit (in decoding order) before the next sub-sequence layer characteristics SEI message (if present) or the end of the stream (otherwise).

When accurate_statistics_flag is equal to 1, the following conditions shall be fulfilled as follows:

– If t_1 is not equal to t_2, the following condition shall be true:

$$average\_bit\_rate == Round(bTotal \div ((t_2 - t_1) * 1000)) \quad \text{(D-6)}$$

– Otherwise (t_1 is equal to t_2), the following condition shall be true:

$$average\_bit\_rate == 0 \quad \text{(D-7)}$$

average_frame_rate indicates the average frame rate in units of frames/(256 seconds). All NAL units in the range of sub-sequence layers specified above are taken into account in the calculation. In the following, fTotal is the number of frames, complementary field pairs and non-paired fields between the current picture (inclusive) and the next sub-sequence layer characteristics SEI message (if present) or the end of the stream (otherwise). t_1 is the removal time (in seconds) of the current access unit, and t_2 is the removal time (in seconds) of the latest access unit (in decoding order) before the next sub-sequence layer characteristics SEI message (if present) or the end of the stream (otherwise).

When accurate_statistics_flag is equal to 1, the following conditions shall be fulfilled as follows:

– If t_1 is not equal to t_2, the following condition shall be true:

$$average\_frame\_rate == Round(fTotal * 256 \div (t_2 - t_1)) \quad \text{(D-8)}$$

– Otherwise (t_1 is equal to t_2), the following condition shall be true:

$$average\_frame\_rate == 0 \quad \text{(D-9)}$$

D.2.13 Sub-sequence characteristics SEI message semantics

The sub-sequence characteristics SEI message indicates the characteristics of a sub-sequence. It also indicates inter prediction dependencies between sub-sequences. This message shall be contained in the first access unit in decoding order of the sub-sequence to which the sub-sequence characteristics SEI message applies. This sub-sequence is herein called the target sub-sequence.

sub_seq_layer_num identifies the sub-sequence layer number of the target sub-sequence. sub_seq_layer_num shall be in the range of 0 to 255, inclusive.

sub_seq_id identifies the target sub-sequence. sub_seq_id shall be in the range of 0 to 65535, inclusive.

duration_flag equal to 0 indicates that the duration of the target sub-sequence is not specified.

sub_seq_duration specifies the duration of the target sub-sequence in clock ticks of a 90-kHz clock.

average_rate_flag equal to 0 indicates that the average bit rate and the average frame rate of the target sub-sequence are unspecified.

accurate_statistics_flag indicates how reliable the values of average_bit_rate and average_frame_rate are. accurate_statistics_flag equal to 1, indicates that the average_bit_rate and the average_frame_rate are rounded from statistically correct values. accurate_statistics_flag equal to 0 indicates that the average_bit_rate and the average_frame_rate are estimates and may deviate from the statistically correct values.

average_bit_rate indicates the average bit rate in (1000 bits)/second of the target sub-sequence. All NAL units of the target sub-sequence are taken into account in the calculation. The average bit rate is derived according to the access unit removal time specified in subclause C.1.2. In the following, nB is the number of bits in all NAL units in the sub-sequence. t_1 is the removal time (in seconds) of the first access unit of the sub-sequence (in decoding order), and t_2 is the removal time (in seconds) of the last access unit of the sub-sequence (in decoding order).

When accurate_statistics_flag is equal to 1, the following conditions shall be fulfilled as follows:

– If t_1 is not equal to t_2, the following condition shall be true:

$$average\_bit\_rate == Round(nB \div ((t_2 - t_1) * 1000)) \quad \text{(D-10)}$$

– Otherwise (t_1 is equal to t_2), the following condition shall be true:

$$average\_bit\_rate == 0 \quad \text{(D-11)}$$

average_frame_rate indicates the average frame rate in units of frames/(256 seconds) of the target sub-sequence. All NAL units of the target sub-sequence are taken into account in the calculation. The average frame rate is derived according to the access unit removal time specified in subclause C.1.2. In the following, fC is the number of frames, complementary field pairs and non-paired fields in the sub-sequence. t_1 is the removal time (in seconds) of the first access unit of the sub-sequence (in decoding order), and t_2 is the removal time (in seconds) of the last access unit of the sub-sequence (in decoding order).

When accurate_statistics_flag is equal to 1, the following conditions shall be fulfilled as follows:

– If t_1 is not equal to t_2, the following condition shall be true:

$$\text{average\_frame\_rate} == \text{Round}(\, fC * 256 \div (\, t_2 - t_1\,)\,) \qquad \text{(D-12)}$$

– Otherwise (t_1 is equal to t_2), the following condition shall be true:

$$\text{average\_frame\_rate} == 0 \qquad \text{(D-13)}$$

num_referenced_subseqs specifies the number of sub-sequences that contain pictures that are used as reference pictures for inter prediction in the pictures of the target sub-sequence. num_referenced_subseqs shall be in the range of 0 to 255, inclusive.

ref_sub_seq_layer_num, ref_sub_seq_id, and **ref_sub_seq_direction** identify the sub-sequence that contains pictures that are used as reference pictures for inter prediction in the pictures of the target sub-sequence. Depending on ref_sub_seq_direction, the following applies:

– If ref_sub_seq_direction is equal to 0, a set of candidate sub-sequences consists of the sub-sequences which have a value of sub_seq_id equal to ref_sub_seq_id, which reside in the sub-sequence layer having sub_seq_layer_num equal to ref_sub_seq_layer_num, and for which the first picture in decoding order precedes the first picture of the target sub-sequence in decoding order.

– Otherwise (ref_sub_seq_direction is equal to 1), a set of candidate sub-sequences consists of the sub-sequences which have a value of sub_seq_id equal to ref_sub_seq_id, which reside in the sub-sequence layer having sub_seq_layer_num equal to ref_sub_seq_layer_num, and for which the first picture in decoding order succeeds the first picture of the target sub-sequence in decoding order.

The sub-sequence used as a reference for the target sub-sequence is the sub-sequence among the set of candidate sub-sequences for which the first picture is the closest to the first picture of the target sub-sequence in decoding order.

D.2.14 Full-frame freeze SEI message semantics

The full-frame freeze SEI message indicates that the current picture and any subsequent pictures in output order that meet specified conditions should not affect the content of the display. No more than one full-frame freeze SEI message shall be present in any access unit.

full_frame_freeze_repetition_period specifies the persistence of the full-frame freeze SEI message and may specify a picture order count interval within which another full-frame freeze SEI message or a full-frame freeze release SEI message or the end of the coded video sequence shall be present in the bitstream. The value of full_frame_freeze_repetition_period shall be in the range of 0 to 16 384, inclusive.

full_frame_freeze_repetition_period equal to 0 specifies that the full-frame freeze SEI message applies to the current decoded picture only.

full_frame_freeze_repetition_period equal to 1 specifies that the full-frame freeze SEI message persists in output order until any of the following conditions are true:

– A new coded video sequence begins.
– A picture in an access unit containing a full-frame freeze SEI message or a full-frame freeze release SEI message is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic).

full_frame_freeze_repetition_period greater than 1 specifies that the full-frame freeze SEI message persists until any of the following conditions are true:.

– A new coded video sequence begins.
– A picture in an access unit containing a full-frame freeze SEI message or a full-frame freeze release SEI message is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + full_frame_freeze_repetition_period.

full_frame_freeze_repetition_period greater than 1 indicates that another full-frame freeze SEI message or a full-frame freeze release SEI message shall be present for a picture in an access unit that is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + full_frame_freeze_repetition_period; unless the bitstream ends or a new coded video sequence begins without output of such a picture.

D.2.15 Full-frame freeze release SEI message semantics

The full-frame freeze release SEI message cancels the effect of any full-frame freeze SEI message sent with pictures that precede the current picture in output order. The full-frame freeze release SEI message indicates that the current picture and subsequent pictures in output order should affect the contents of the display.

No more than one full-frame freeze release SEI message shall be present in any access unit. A full-frame freeze release SEI message shall not be present in an access unit containing a full-frame freeze SEI message. When a full-frame freeze SEI message is present in an access unit containing a field of a complementary field pair in which the values of PicOrderCnt(CurrPic) for the two fields of the complementary field pair are equal to each other, a full-frame freeze release SEI message shall not be present in either of the two access units.

D.2.16 Full-frame snapshot SEI message semantics

The full-frame snapshot SEI message indicates that the current frame is labelled for use as determined by the application as a still-image snapshot of the video content.

snapshot_id specifies a snapshot identification number. snapshot_id shall be in the range of 0 to $2^{32} - 2$, inclusive.

Values of snapshot_id in the range of 0 to 255, inclusive, and in the range of 512 to $2^{31} - 1$, inclusive, may be used as determined by the application. Values of snapshot_id in the range of 256 to 511, inclusive, and in the range of 2^{31} to $2^{32} - 2$, inclusive, are reserved for future use by ITU-T | ISO/IEC. Decoders encountering a value of snapshot_id in the range of 256 to 511, inclusive, or in the range of 2^{31} to $2^{32} - 2$, inclusive, shall ignore (remove from the bitstream and discard) it.

D.2.17 Progressive refinement segment start SEI message semantics

The progressive refinement segment start SEI message specifies the beginning of a set of consecutive coded pictures that is labelled as the current picture followed by a sequence of one or more pictures of refinement of the quality of the current picture, rather than as a representation of a continually moving scene.

The tagged set of consecutive coded pictures shall continue until one of the following conditions is true. When a condition below becomes true, the next slice to be decoded does not belong to the tagged set of consecutive coded pictures:

– The next slice to be decoded belongs to an IDR picture.
– num_refinement_steps_minus1 is greater than 0 and the frame_num of the next slice to be decoded is (currFrameNum + num_refinement_steps_minus1 + 1) % MaxFrameNum, where currFrameNum is the value of frame_num of the picture in the access unit containing the SEI message.
– num_refinement_steps_minus1 is 0 and a progressive refinement segment end SEI message with the same progressive_refinement_id as the one in this SEI message is decoded.

The decoding order of pictures within the tagged set of consecutive pictures should be the same as their output order.

progressive_refinement_id specifies an identification number for the progressive refinement operation. progressive_refinement_id shall be in the range of 0 to $2^{32} - 2$, inclusive.

Values of progressive_refinement_id in the range of 0 to 255, inclusive, and in the range of 512 to $2^{31} - 1$, inclusive, may be used as determined by the application. Values of progressive_refinement_id in the range of 256 to 511, inclusive, and in the range of 2^{31} to $2^{32} - 2$, inclusive, are reserved for future use by ITU-T | ISO/IEC. Decoders encountering a value of progressive_refinement_id in the range of 256 to 511, inclusive, or in the range of 2^{31} to $2^{32} - 2$, inclusive, shall ignore (remove from the bitstream and discard) it.

num_refinement_steps_minus1 specifies the number of reference frames in the tagged set of consecutive coded pictures as follows:

– If num_refinement_steps_minus1 is equal to 0, the number of reference frames in the tagged set of consecutive coded pictures is unknown.
– Otherwise, the number of reference frames in the tagged set of consecutive coded pictures is equal to num_refinement_steps_minus1 + 1.

num_refinement_steps_minus1 shall be in the range of 0 to MaxFrameNum − 1, inclusive.

D.2.18 Progressive refinement segment end SEI message semantics

The progressive refinement segment end SEI message specifies the end of a set of consecutive coded pictures that has been labelled by use of a progressive refinement segment start SEI message as an initial picture followed by a sequence of one or more pictures of the refinement of the quality of the initial picture, and ending with the current picture.

progressive_refinement_id specifies an identification number for the progressive refinement operation. progressive_refinement_id shall be in the range of 0 to $2^{32} - 2$, inclusive.

The progressive refinement segment end SEI message specifies the end of any progressive refinement segment previously started using a progressive refinement segment start SEI message with the same value of progressive_refinement_id.

Values of progressive_refinement_id in the range of 0 to 255, inclusive, and in the range of 512 to $2^{31} - 1$, inclusive, may be used as determined by the application. Values of progressive_refinement_id in the range of 256 to 511, inclusive, and in the range of 2^{31} to $2^{32} - 2$, inclusive, are reserved for future use by ITU-T | ISO/IEC. Decoders encountering a value of progressive_refinement_id in the range of 256 to 511, inclusive, or in the range of 2^{31} to $2^{32} - 2$, inclusive, shall ignore (remove from the bitstream and discard) it.

D.2.19 Motion-constrained slice group set SEI message semantics

NOTE 1 – The syntax of the motion-constrained slice group set SEI message is dependent on the content of the picture parameter set that is active for the primary coded picture associated with the motion-constrained slice group set SEI message. However, the activation of the associated picture parameter set does not occur until the decoding of the first coded slice NAL unit of the primary coded picture. Since the coded slice NAL units of the primary coded picture follow the motion-constrained slice group set SEI message in NAL unit order, it may be necessary for a decoder to store the RBSP containing the motion-constrained slice group set SEI message until determining the parameters of the picture parameter set that will be active for the primary coded picture, and then perform the parsing of the motion-constrained slice group set SEI message.

This SEI message indicates that inter prediction over slice group boundaries is constrained as specified below. When present, the message shall only appear where it is associated, as specified in subclause 7.4.1.2.3, with an IDR access unit.

The target picture set for this SEI message contains all consecutive primary coded pictures in decoding order starting with the associated primary coded IDR picture (inclusive) and ending with the following primary coded IDR picture (exclusive) or with the very last primary coded picture in the bitstream (inclusive) in decoding order when there is no following primary coded IDR picture. The slice group set is a collection of one or more slice groups, identified by the slice_group_id[i] syntax element. When separate_colour_plane_flag is equal to 1, the term "primary coded pictures" represents the parts of the corresponding primary coded pictures that correspond to the NAL units having the same colour_plane_id.

This SEI message indicates that, for each picture in the target picture set, the inter prediction process is constrained as follows: No sample value outside the slice group set, and no sample value at a fractional sample position that is derived using one or more sample values outside the slice group set is used for inter prediction of any sample within the slice group set.

num_slice_groups_in_set_minus1 + 1 specifies the number of slice groups in the slice group set. The allowed range of num_slice_groups_in_set_minus1 is 0 to num_slice_groups_minus1, inclusive. The allowed range of num_slice_groups_minus1 is specified in Annex A and subclauses G.10 and H.10.

slice_group_id[i] with i = 0.. num_slice_groups_in_set_minus1 identifies the slice group(s) contained within the slice group set. The allowed range for slice_group_id[i] is from 0 to num_slice_groups_minus1, inclusive. The length of the slice_group_id[i] syntax element is Ceil(Log2(num_slice_groups_minus1 + 1)) bits.

When num_slice_groups_minus1 is equal to 0 (i.e., num_slice_groups_in_set_minus1 is equal to 0 and slice_group_id[0] is not present), the value of slice_group_id[0] shall be inferred to be equal to 0.

exact_sample_value_match_flag equal to 0 indicates that, within the target picture set, when the macroblocks that do not belong to the slice group set are not decoded, the value of each sample in the slice group set need not be exactly the same as the value of the same sample when all the macroblocks are decoded. exact_sample_value_match_flag equal to 1 indicates that, within the target picture set, when the macroblocks that do not belong to the slice group set are not decoded, the value of each sample in the slice group set shall be exactly the same as the value of the same sample when all the macroblocks in the target picture set are decoded.

NOTE 2 – When disable_deblocking_filter_idc is equal to 1 or 2 in all slices in the target picture set, exact_sample_value_match_flag should be 1.

pan_scan_rect_flag equal to 0 specifies that pan_scan_rect_id is not present. pan_scan_rect_flag equal to 1 specifies that pan_scan_rect_id is present.

pan_scan_rect_id indicates that the specified slice group set covers at least the pan-scan rectangle identified by pan_scan_rect_id within the target picture set.

NOTE 3 – Multiple motion_constrained_slice_group_set SEI messages may be associated with the same IDR picture. Consequently, more than one slice group set may be active within a target picture set.

NOTE 4 – The size, shape, and location of the slice groups in the slice group set may change within the target picture set.

D.2.20 Film grain characteristics SEI message semantics

This SEI message provides the decoder with a parameterised model for film grain synthesis. For example, an encoder may use the film grain characteristics SEI message to characterise film grain that was present in the original source video material and was removed by pre-processing filtering techniques. Synthesis of simulated film grain on the decoded images for the display process is optional and does not affect the decoding process specified in this Recommendation | International Standard. If synthesis of simulated film grain on the decoded images for the display process is performed, there is no requirement that the method by which the synthesis is performed be the same as the parameterised model for the film grain as provided in the film grain characteristics SEI message.

NOTE 1 – The display process is not specified in this Recommendation | International Standard.

NOTE 2 – The SMPTE specification "SMPTE RDD 5-2006. Film Grain Technology – Specifications for H.264/MPEG-4 AVC Bitstreams." specifies a film grain simulator based on the information provided in the film grain characteristics SEI message.

film_grain_characteristics_cancel_flag equal to 1 indicates that the SEI message cancels the persistence of any previous film grain characteristics SEI message in output order. film_grain_characteristics_cancel_flag equal to 0 indicates that film grain modelling information follows.

model_id identifies the film grain simulation model as specified in Table D-5. The value of model_id shall be in the range of 0 to 1, inclusive. The values of 2 and 3 for model_id are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore film grain characteristic SEI messages with model_id equal to 2 or 3.

Table D-5 – model_id values

Value	Description
0	frequency filtering
1	auto-regression
2	reserved
3	reserved

separate_colour_description_present_flag equal to 1 indicates that a distinct colour space description for the film grain characteristics specified in the SEI message is present in the film grain characteristics SEI message syntax. separate_colour_description_present_flag equal to 0 indicates that the colour description for the film grain characteristics specified in the SEI message is the same as for the coded video sequence as specified in subclause E.2.1.

NOTE 3 – When separate_colour_description_present_flag is equal to 1, the colour space specified for the film grain characteristics specified in the SEI message may differ from the colour space specified for the coded video as specified in subclause E.2.1.

film_grain_bit_depth_luma_minus8 plus 8 specifies the bit depth used for the luma component of the film grain characteristics specified in the SEI message. When film_grain_bit_depth_luma_minus8 is not present in the film grain characteristics SEI message, the value of film_grain_bit_depth_luma_minus8 shall be inferred to be equal to bit_depth_luma_minus8.

The value of filmGrainBitDepth[0] is derived as

$$\text{filmGrainBitDepth[0]} = \text{film\_grain\_bit\_depth\_luma\_minus8} + 8 \qquad \text{(D-14)}$$

film_grain_bit_depth_chroma_minus8 plus 8 specifies the bit depth used for the Cb and Cr components of the film grain characteristics specified in the SEI message. When film_grain_bit_depth_chroma_minus8 is not present in the film grain characteristics SEI message, the value of film_grain_bit_depth_chroma_minus8 shall be inferred to be equal to bit_depth_chroma_minus8.

The value of filmGrainBitDepth[c] for c = 1 and 2 is derived as

$$\text{filmGrainBitDepth[c]} = \text{film\_grain\_bit\_depth\_chroma\_minus8} + 8 \quad \text{with c = 1, 2} \qquad \text{(D-15)}$$

film_grain_full_range_flag has the same semantics as specified in subclause E.2.1 for the video_full_range_flag syntax element, except as follows:

– film_grain_full_range_flag specifies the colour space of the film grain characteristics specified in the SEI message, rather than the colour space used for the coded video sequence.

– When film_grain_full_range_flag is not present in the film grain characteristics SEI message, the value of film_grain_full_range_flag shall be inferred to be equal to video_full_range_flag.

film_grain_colour_primaries has the same semantics as specified in subclause E.2.1 for the colour_primaries syntax element, except as follows:

– film_grain_colour_primaries specifies the colour space of the film grain characteristics specified in the SEI message, rather than the colour space used for the coded video sequence.
– When film_grain_colour_primaries is not present in the film grain characteristics SEI message, the value of film_grain_colour_primaries shall be inferred to be equal to colour_primaries.

film_grain_transfer_characteristics has the same semantics as specified in subclause E.2.1 for the transfer_characteristics syntax element, except as follows:

– film_grain_transfer_characteristics specifies the colour space of the film grain characteristics specified in the SEI message, rather than the colour space used for the coded video sequence.
– When film_grain_transfer_characteristics is not present in the film grain characteristics SEI message, the value of film_grain_transfer_characteristics shall be inferred to be equal to transfer_characteristics.

film_grain_matrix_coefficients has the same semantics as specified in subclause E.2.1 for the matrix_coefficients syntax element, except as follows:

– film_grain_matrix_coefficients specifies the colour space of the film grain characteristics specified in the SEI message, rather than the colour space used for the coded video sequence.
– When film_grain_matrix_coefficients is not present in the film grain characteristics SEI message, the value of film_grain_matrix_coefficients shall be inferred to be equal to matrix_coefficients.
– The values allowed for film_grain_matrix_coefficients are not constrained by the value of chroma_format_idc.

The chroma_format_idc of the film grain characteristics specified in the film grain characteristics SEI message shall be inferred to be equal to 3 (4:4:4).

NOTE 4 – Because the use of a specific method is not required for performing film grain generation function used by the display process, a decoder may, if desired, down-convert the model information for chroma in order to simulate film grain for other chroma formats (4:2:0 or 4:2:2) rather than up-converting the decoded video (using a method not specified by this Recommendation | International Standard) before performing film grain generation.

blending_mode_id identifies the blending mode used to blend the simulated film grain with the decoded images as specified in Table D-6. blending_mode_id shall be in the range of 0 to 1, inclusive.

Table D-6 – blending_mode_id values

Value	Description
0	additive
1	multiplicative
2	reserved
3	reserved

Depending on blending_mode_id, the blending mode is specified as follows:

– If blending_mode_id is equal to 0 the blending mode is additive as specified by

$$I_{grain}[\ x, y, c\ ] = \text{Clip3}(\ 0, (\ 1 << \text{filmGrainBitDepth}[\ c\ ]\ ) - 1, I_{decoded}[\ x, y, c\ ] + G[\ x, y, c\ ]\ ) \qquad \text{(D-16)}$$

– Otherwise (blending_mode_id is equal to 1), the blending mode is multiplicative as specified by

$$\begin{aligned} I_{grain}[\ x, y, c\ ] = \text{Clip3}(\ 0, (\ 1 << \text{filmGrainBitDepth}[\ c\ ]\ ) - 1, I_{decoded}[\ x, y, c\ ] \\ + \text{Round}(\ (\ I_{decoded}[\ x, y, c\ ] * G[\ x, y, c\ ]\ ) \div (\ (\ 1 << \text{bitDepth}[\ c\ ]\ ) - 1\ )\ )\ ) \end{aligned} \qquad \text{(D-17)}$$

where $I_{decoded}[\ x, y, c\ ]$ represents the sample value at coordinates x, y of the colour component c of the decoded image $I_{decoded}$, G[x, y, c] is the simulated film grain value at the same position and colour component, filmGrainBitDepth[c] is the number of bits used for each sample in a fixed-length unsigned binary representation of the array $I_{grain}[\ x, y, c\ ]$, and bitDepth[c] is specified by

$$\text{bitDepth}[\ c\ ] = \begin{cases} \text{BitDepth}_Y & ; \quad c = 0 \\ \text{BitDepth}_C & ; \quad c = 1, 2 \end{cases} \qquad \text{(D-18)}$$

log2_scale_factor specifies a scale factor used in the film grain characterization equations.

comp_model_present_flag[c] equal to 0 indicates that film grain is not modelled on the c-th colour component, where c equal to 0 refers to the luma component, c equal to 1 refers to the Cb component, and c equal to 2 refers to the Cr component. comp_model_present_flag[c] equal to 1 indicates that syntax elements specifying modelling of film grain on colour component c are present in the SEI message.

num_intensity_intervals_minus1[c] plus 1 specifies the number of intensity intervals for which a specific set of model values has been estimated.

NOTE 5 – The intensity intervals may overlap in order to simulate multi-generational film grain.

num_model_values_minus1[c] plus 1 specifies the number of model values present for each intensity interval in which the film grain has been modelled. The value of num_model_values_minus1[c] shall be in the range of 0 to 5, inclusive.

intensity_interval_lower_bound[c][i] specifies the lower bound of the interval i of intensity levels for which the set of model values applies.

intensity_interval_upper_bound[c][i] specifies the upper bound of the interval i of intensity levels for which the set of model values applies.

Depending on model_id, the selection of the sets of model values is specified as follows:

– If model_id is equal to 0, the average value of each block b of 8x8 samples in $I_{decoded}$, referred as b_{avg}, is used to select the sets of model values with index s[j] that apply to all the samples in the block:

```
for( i = 0, j = 0; i <= num_intensity_intervals_minus1[ c ]; i++ )
   if( b_avg >= intensity_interval_lower_bound[ c ][ i ] && b_avg <= intensity_interval_upper_bound[ c ][ i ] ) {
      s[ j ] = i                                                                    (D-19)
      j++
   }
```

– Otherwise (model_id is equal to 1), the sets of model values used to generate the film grain are selected for each sample value in $I_{decoded}$ as follows:

```
for( i = 0, j = 0; i <= num_intensity_intervals_minus1[ c ]; i++ )
   if( I_decoded[ x, y, c ] >= intensity_interval_lower_bound[ c ][ i ] &&
         I_decoded[ x, y, c ] <= intensity_interval_upper_bound[ c ][ i ] ) {      (D-20)
      s[ j ] = i
      j++
   }
```

Samples that do not fall into any of the defined intervals are not modified by the grain generation function. Samples that fall into more than one interval will originate multi-generation grain. Multi-generation grain results from adding the grain computed independently for each intensity interval.

comp_model_value[c][i][j] represents each one of the model values present for the colour component c and the intensity interval i. The set of model values has different meaning depending on the value of model_id. The value of comp_model_value[c][i][j] shall be constrained as follows, and may be additionally constrained as specified elsewhere in this subclause.

– If model_id is equal to 0, comp_model_value[c][i][j] shall be in the range of 0 to $2^{\text{filmGrainBitDepth[c]}} - 1$, inclusive.

– Otherwise (model_id is equal to 1), comp_model_value[c][i][j] shall be in the range of $-2^{(\text{ filmGrainBitDepth[c]} - 1)}$ to $2^{(\text{ filmGrainBitDepth[c]} - 1)} - 1$, inclusive.

Depending on model_id, the synthesis of the film grain is modelled as follows:

– If model_id is equal to 0, a frequency filtering model enables simulating the original film grain for c = 0..2, x = 0..PicWidthInSamples$_L$, and y = 0..PicHeightInSamples$_L$ as specified by:

$$G[\ x, y, c\ ] = (\ \text{comp\_model\_value}[\ c\ ][\ s\ ][\ 0\ ] * Q[\ c\ ][\ x, y\ ] + \text{comp\_model\_value}[\ c\ ][\ s\ ][\ 5\ ] * G[\ x, y, c - 1\ ]\ ) >> \text{log2\_scale\_factor} \quad \text{(D-21)}$$

where Q[c] is a two-dimensional random process generated by filtering 16x16 blocks gaussRv with random-value elements gaussRv_{ij} generated with a normalized Gaussian distribution (independent and identically distributed Gaussian random variable samples with zero mean and unity variance) and where the value of an element G[x, y, c −1] used in the right-hand side of the equation is inferred to be equal to 0 when c − 1 is less than 0.

NOTE 6 – A normalized Gaussian random value can be generated from two independent, uniformly distributed random values over the interval from 0 to 1 (and not equal to 0), denoted as uRv_0 and uRv_1, using the Box-Muller transformation specified by

$$gaussRv_{ij} = \sqrt{-2 * Ln(uRv_0)} * Cos(2 * \pi * uRv_1) \tag{D-22}$$

where Ln(x) is the natural logarithm of x (the base-e logarithm, where e is natural logarithm base constant 2.718 281 828...), Cos(x) is the trigonometric cosine function operating on an argument x in units of radians, and π is Archimedes' constant 3.141 592 653....

The band-pass filtering of blocks gaussRv may be performed in the discrete cosine transform (DCT) domain as follows:

```
for( y = 0; y < 16; y++ )
    for( x = 0; x < 16; x++ )
        if( ( x < comp_model_value[ c ][ s ][ 3 ]  &&  y < comp_model_value[ c ][ s ][ 4 ] )  | |      (D-23)
          x > comp_model_value[ c ][ s ][ 1 ]  | |  y > comp_model_value[ c ][ s ][ 2 ] )
            gaussRv[ x, y ] = 0
filteredRv = IDCT16x16( gaussRv )
```

where IDCT16x16(z) refers to a unitary inverse discrete cosine transformation (IDCT) operating on a 16x16 matrix argument z as specified by

$$IDCT16x16(z) = r * z * r^T \tag{D-24}$$

where the superscript T indicates a matrix transposition and r is the 16x16 matrix with elements r_{ij} specified by

$$r_{ij} = \frac{((i == 0) ? 1 : \sqrt{2})}{4} Cos\left(\frac{i * (2 * j + 1) * \pi}{32} \right) \tag{D-25}$$

where Cos(x) is the trigonometric cosine function operating on an argument x in units of radians and π is Archimedes' constant 3.141 592 653.

Q[c] is formed by the frequency-filtered blocks filteredRv.

NOTE 7 – Coded model values are based on blocks of 16x16, but a decoder implementation may use other block sizes. For example, decoders implementing the IDCT on 8x8 blocks, should down-convert by a factor of two the set of coded model values comp_model_value[c][s][i] for i equal to 1..4.

NOTE 8 – To reduce the degree of visible blocks that can result from mosaicing the frequency-filtered blocks filteredRv, decoders may apply a low-pass filter to the boundaries between frequency-filtered blocks.

– Otherwise (model_id is equal to 1), an auto-regression model enables simulating the original film grain for c = 0..2, x = 0..$PicWidthInSamples_L$, and y = 0..$PicHeightInSamples_L$ as specified by

```
G[ x, y, c ] = ( comp_model_value[ c ][ s ][ 0 ] * n[ x, y, c ] +
    comp_model_value[ c ][ s ][ 1 ] * ( G[ x − 1, y, c ] + ( ( comp_model_value[ c ][ s ][ 4 ] * G[ x, y − 1, c ] )
    >>
        log2_scale_factor ) ) +
    comp_model_value[ c ][ s ][ 3 ] * ( ( ( comp_model_value[ c ][ s ][ 4 ] * G[ x − 1, y − 1, c ] ) >>
        log2_scale_factor ) + G[ x+1, y − 1, c ] ) +
    comp_model_value[ c ][ s ][ 5 ] * ( G[ x − 2, y, c ] +
        ( ( comp_model_value[ c ][ s ][ 4 ] * comp_model_value[ c ][ s ][ 4 ] * G[ x, y − 2, c ] ) >>
            ( 2 * log2_scale_factor ) ) ) +
    comp_model_value[ c ][ s ][ 2 ] * G[ x, y, c − 1 ] ) >> log2_scale_factor                            (D-26)
```

where n[x, y, c] is a random value with normalized Gaussian distribution (independent and identically distributed Gaussian random variable samples with zero mean and unity variance for each value of x, y, and c) and where the value of an element G[x, y, c] used in the right-hand side of the equation is inferred to be equal to 0 when any of the following conditions are true:

– x is less than 0,
– y is less than 0,
– x is greater than or equal to $PicWidthInSamples_L$,
– c is less than 0.

comp_model_value[c][i][0] provides the first model value for the model as specified by model_id. comp_model_value[c][i][0] corresponds to the standard deviation of the Gaussian noise term in the generation functions specified in Equations D-21 through D-26.

comp_model_value[c][i][1] provides the second model value for the model as specified by model_id. When model_id is equal to 0, comp_model_value[c][i][1] shall be greater than or equal to 0 and less than 16.

When not present in the film grain characteristics SEI message, comp_model_value[c][i][1] shall be inferred as follows:

– If model_id is equal to 0, comp_model_value[c][i][1] shall be inferred to be equal to 8.

– Otherwise (model_id is equal to 1), comp_model_value[c][i][1] shall be inferred to be equal to 0.

comp_model_value[c][i][1] is interpreted as follows:

– If model_id is equal to 0, comp_model_value[c][i][1] indicates the horizontal high cut frequency to be used to filter the DCT of a block of 16x16 random values.

– Otherwise (model_id is equal to 1), comp_model_value[c][i][1] indicates the first order spatial correlation for neighbouring samples (x − 1, y) and (x, y − 1).

comp_model_value[c][i][2] provides the third model value for the model as specified by model_id. When model_id is equal to 0, comp_model_value[c][i][2] shall be greater than or equal to 0 and less than 16.

When not present in the film grain characteristics SEI message, comp_model_value[c][i][2] shall be inferred as follows:

– If model_id is equal to 0, comp_model_value[c][i][2] shall be inferred to be equal to comp_model_value[c][i][1]

– Otherwise (model_id is equal to 1), comp_model_value[c][i][2] shall be inferred to be equal to 0.

comp_model_value[c][i][2] is interpreted as follows:

– If model_id is equal to 0, comp_model_value[c][i][2] indicates the vertical high cut frequency to be used to filter the DCT of a block of 16x16 random values.

– Otherwise (model_id is equal to 1), comp_model_value[c][i][2] indicates the colour correlation between consecutive colour components.

comp_model_value[c][i][3] provides the fourth model value for the model as specified by model_id. When model_id is equal to 0, comp_model_value[c][i][3] shall be greater than or equal to 0 and less than or equal to comp_model_value[c][i][1].

When not present in the film grain characteristics SEI message, comp_model_value[c][i][3] shall be inferred to be equal to 0.

comp_model_value[c][i][3] is interpreted as follows:

– If model_id is equal to 0, comp_model_value[c][i][3] indicates the horizontal low cut frequency to be used to filter the DCT of a block of 16x16 random values.

– Otherwise (model_id is equal to 1), comp_model_value[c][i][3] indicates the first order spatial correlation for neighbouring samples (x − 1, y − 1) and (x + 1, y − 1).

comp_model_value[c][i][4] provides the fifth model value for the model as specified by model_id. When model_id is equal to 0, comp_model_value[c][i][4] shall be greater than or equal to 0 and less than or equal to comp_model_value[c][i][2].

When not present in the film grain characteristics SEI message, comp_model_value[c][i][4] shall be inferred to be equal to model_id.

comp_model_value[c][i][4] is interpreted as follows:

– If model_id is equal to 0, comp_model_value[c][i][4] indicates the vertical low cut frequency to be used to filter the DCT of a block of 16x16 random values.

– Otherwise (model_id is equal to 1), comp_model_value[c][i][4] indicates the aspect ratio of the modelled grain.

comp_model_value[c][i][5] provides the sixth model value for the model as specified by model_id.

When not present in the film grain characteristics SEI message, comp_model_value[c][i][5] shall be inferred to be equal to 0.

comp_model_value[c][i][5] is interpreted as follows:

– If model_id is equal to 0, comp_model_value[c][i][5] indicates the colour correlation between consecutive colour components.

– Otherwise (model_id is equal to 1), comp_model_value[c][i][5] indicates the second order spatial correlation for neighbouring samples (x, y − 2) and (x − 2, y).

film_grain_characteristics_repetition_period specifies the persistence of the film grain characteristics SEI message and may specify a picture order count interval within which another film grain characteristics SEI message or the end of the coded video sequence shall be present in the bitstream. The value of film_grain_characteristics_repetition_period shall be in the range 0 to 16 384, inclusive.

film_grain_characteristics_repetition_period equal to 0 specifies that the film grain characteristics SEI message applies to the current decoded picture only.

film_grain_characteristics_repetition_period equal to 1 specifies that the film grain characteristics SEI message persists in output order until any of the following conditions are true:

– A new coded video sequence begins.

– A picture in an access unit containing a film grain characteristics SEI message is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic).

film_grain_characteristics_repetition_period greater than 1 specifies that the film grain characteristics SEI message persists until any of the following conditions are true:

– A new coded video sequence begins.

– A picture in an access unit containing a film grain characteristics SEI message is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + film_grain_characteristics_repetition_period.

film_grain_characteristics_repetition_period greater than 1 indicates that another film grain characteristics SEI message shall be present for a picture in an access unit that is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + film_grain_characteristics_repetition_period; unless the bitstream ends or a new coded video sequence begins without output of such a picture.

D.2.21 Deblocking filter display preference SEI message semantics

This SEI message provides the decoder with an indication of whether the display of the cropped result of the deblocking filter process specified in subclause 8.7 or of the cropped result of the picture construction process prior to the deblocking filter process specified in subclause 8.5.14 is preferred by the encoder for the display of each decoded picture that is output.

NOTE 1 – The display process is not specified in this Recommendation | International Standard. The means by which an encoder determines what to indicate as its preference expressed in a deblocking filter display preference SEI message is also not specified in this Recommendation | International Standard, and the expression of an expressed preference in a deblocking filter display preference SEI message does not impose any requirement on the display process.

deblocking_display_preference_cancel_flag equal to 1 indicates that the SEI message cancels the persistence of any previous deblocking filter display preference SEI message in output order. deblocking_display_preference_cancel_flag equal to 0 indicates that a display_prior_to_deblocking_preferred_flag and deblocking_display_preference_repetition_period follow.

NOTE 2 – In the absence of the deblocking filter display preference SEI message, or after the receipt of a deblocking filter display preference SEI message in which deblocking_display_preference_cancel_flag is equal to 1, the decoder should infer that the display of the cropped result of the deblocking filter process specified in subclause 8.7 is preferred over the display of the cropped result of the picture construction process prior to the deblocking filter process specified in subclause 8.5.14 for the display of each decoded picture that is output.

display_prior_to_deblocking_preferred_flag equal to 1 indicates that the encoder preference is for the display process (which is not specified in this Recommendation | International Standard) to display the cropped result of the picture construction process prior to the deblocking filter process specified in subclause 8.5.14 rather than the cropped result of the deblocking filter process specified in subclause 8.7 for each picture that is cropped and output as specified in Annex C. display_prior_to_deblocking_preferred_flag equal to 0 indicates that the encoder preference is for the display process (which is not specified in this Recommendation | International Standard) to display the cropped result of the deblocking filter process specified in subclause 8.7 rather than the cropped result of the picture construction process prior to the deblocking filter process specified in subclause 8.5.14 for each picture that is cropped and output as specified in Annex C.

NOTE 3 – The presence or absence of the deblocking filter display preference SEI message and the value of display_prior_to_deblocking_preferred_flag does not affect the requirements of the decoding process specified in this

Recommendation | International Standard. Rather, it only provides an indication of when, in addition to fulfilling the requirements of this Recommendation | International Standard for the decoding process, enhanced visual quality might be obtained by performing the display process (which is not specified in this Recommendation | International Standard) in an alternative fashion. Encoders that use the deblocking filter display preference SEI message should be designed with an awareness that unless the encoder restricts its use of the DPB capacity specified in Annex A and subclauses G.10 and H.10 for the profile and level in use, some decoders may not have sufficient memory capacity for the storage of the result of the picture construction process prior to the deblocking filter process specified in subclause 8.5.14 in addition to the storage of the result of the deblocking filter process specified in subclause 8.7 when reordering and delaying pictures for display, and such decoders would therefore not be able to benefit from the preference indication. By restricting its use of the DPB capacity, an encoder can be able to use at least half of the DPB capacity specified in Annex A and subclauses G.10 and H.10 while allowing the decoder to use the remaining capacity for storage of unfiltered pictures that have been indicated as preferable for display until the output time arrives for those pictures.

dec_frame_buffering_constraint_flag equal to 1 indicates that the use of the frame buffering capacity of the HRD decoded picture buffer (DPB) as specified by max_dec_frame_buffering has been constrained such that the coded video sequence will not require a decoded picture buffer with more than Max(1, max_dec_frame_buffering) frame buffers to enable the output of the decoded filtered or unfiltered pictures, as indicated by the deblocking filter display preference SEI messages, at the output times specified by the dpb_output_delay of the picture timing SEI messages. dec_frame_buffering_constraint_flag equal to 0 indicates that the use of the frame buffering capacity in the HRD may or may not be constrained in the manner that would be indicated by dec_frame_buffering_constraint_flag equal to 1.

For purposes of determining the constraint imposed when dec_frame_buffering_constraint_flag is equal to 1, the quantity of frame buffering capacity used at any given point in time by each frame buffer of the DPB that contains a picture shall be derived as follows:

– If both of the following criteria are satisfied for the frame buffer, the frame buffer is considered to use two frame buffers of capacity for its storage.
  – The frame buffer contains a frame or one or more fields that is marked as "used for reference", and
  – The frame buffer contains a picture for which both of the following criteria are fulfilled:
    – The HRD output time of the picture is greater than the given point in time.
    – It has been indicated in a deblocking filter display preference SEI message that the encoder preference for the picture is for the display process to display the cropped result of the picture construction process prior to the deblocking filter process specified in subclause 8.5.14 rather than the cropped result of the deblocking filter process specified in subclause 8.7.
– Otherwise, the frame buffer is considered to use one frame buffer of DPB capacity for its storage.

When dec_frame_buffering_constraint_flag is equal to 1, the frame buffering capacity used by all of the frame buffers in the DPB that contain pictures, as derived in this manner, shall not be greater than Max(1, max_dec_frame_buffering) during the operation of the HRD for the coded video sequence.

The value of dec_frame_buffering_constraint_flag shall be the same in all deblocking filter display preference SEI messages of the coded video sequence.

deblocking_display_preference_repetition_period specifies the persistence of the deblocking filter display preference SEI message and may specify a picture order count interval within which another deblocking filter display preference message or the end of the coded video sequence shall be present in the bitstream. The value of deblocking_display_preference_repetition_period shall be in the range 0 to 16 384, inclusive.

deblocking_display_preference_repetition_period equal to 0 specifies that the deblocking filter display preference SEI message applies to the current decoded picture only.

deblocking_display_preference_repetition_period equal to 1 specifies that the deblocking filter display preference SEI message persists in output order until any of the following conditions are true:

– A new coded video sequence begins.
– A picture in an access unit containing a deblocking filter display preference SEI message is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic).

deblocking_display_preference_repetition_period greater than 1 specifies that the deblocking filter display preference SEI message persists until any of the following conditions are true:

– A new coded video sequence begins.
– A picture in an access unit containing a deblocking filter display preference SEI message is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + deblocking_display_preference_repetition_period.

deblocking_display_preference_repetition_period greater than 1 indicates that another deblocking filter display preference SEI message shall be present for a picture in an access unit that is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + deblocking_display_preference_repetition_period; unless the bitstream ends or a new coded video sequence begins without output of such a picture.

D.2.22 Stereo video information SEI message semantics

NOTE 1 – The stereo video information SEI message is included in this Specification primarily for historical reasons. It is now suggested to use the frame packing arrangement SEI message rather than the stereo video information SEI message to signal stereo video information.

This SEI message provides the decoder with an indication that the entire coded video sequence consists of pairs of pictures forming stereo-view content.

The stereo video information SEI message shall not be present in any access unit of a coded video sequence unless a stereo video information SEI message is present in the first access unit of the coded video sequence.

field_views_flag equal to 1 indicates that all pictures in the current coded video sequence are fields and all fields of a particular parity are considered a left view and all fields of the opposite parity are considered a right view for stereo-view content. field_views_flag equal to 0 indicates that all pictures in the current coded video sequence are frames and alternating frames in output order represent a view of a stereo view. The value of field_views_flag shall be the same in all stereo video information SEI messages within a coded video sequence.

When the stereo video information SEI message is present and field_views_flag is equal to 1, the left view and right view of a stereo video pair shall be coded as a complementary field pair, the display time of the first field of the field pair in output order should be delayed to coincide with the display time of the second field of the field pair in output order, and the spatial locations of the samples in each individual field should be interpreted for display purposes as representing complete pictures as shown in Figure 6-1 rather than as spatially-distinct fields within a frame as shown in Figure 6-2.

NOTE 2 – The display process is not specified in this Recommendation | International Standard.

top_field_is_left_view_flag equal to 1 indicates that the top fields in the coded video sequence represent a left view and the bottom fields in the coded video sequence represent a right view. top_field_is_left_view_flag equal to 0 indicates that the bottom fields in the coded video sequence represent a left view and the top fields in the coded video sequence represent a right view. When present, the value of top_field_is_left_view_flag shall be the same in all stereo video information SEI messages within a coded video sequence.

current_frame_is_left_view_flag equal to 1 indicates that the current picture is the left view of a stereo-view pair. current_frame_is_left_view_flag equal to 0 indicates that the current picture is the right view of a stereo-view pair.

next_frame_is_second_view_flag equal to 1 indicates that the current picture and the next picture in output order form a stereo-view pair, and the display time of the current picture should be delayed to coincide with the display time of the next picture in output order. next_frame_is_second_view_flag equal to 0 indicates that the current picture and the previous picture in output order form a stereo-view pair, and the display time of the current picture should not be delayed for purposes of stereo-view pairing.

left_view_self_contained_flag equal to 1 indicates that no inter prediction operations within the decoding process for the left-view pictures of the coded video sequence refer to reference pictures that are right-view pictures. left_view_self_contained_flag equal to 0 indicates that some inter prediction operations within the decoding process for the left-view pictures of the coded video sequence may or may not refer to reference pictures that are right-view pictures. Within a coded video sequence, the value of left_view_self_contained_flag in all stereo video information SEI messages shall be the same.

right_view_self_contained_flag equal to 1 indicates that no inter prediction operations within the decoding process for the right-view pictures of the coded video sequence refer to reference pictures that are left-view pictures. right_view_self_contained_flag equal to 0 indicates that some inter prediction operations within the decoding process for the right-view pictures of the coded video sequence may or may not refer to reference pictures that are left-view pictures. Within a coded video sequence, the value of right_view_self_contained_flag in all stereo video information SEI messages shall be the same.

D.2.23 Post-filter hint SEI message semantics

This SEI message provides the coefficients of a post-filter or correlation information for the design of a post-filter for potential use in post-processing of the output decoded pictures to obtain improved displayed quality.

filter_hint_size_y specifies the vertical size of the filter coefficient or correlation array. The value of filter_hint_size_y shall be in the range of 1 to 15, inclusive.

filter_hint_size_x specifies the horizontal size of the filter coefficient or correlation array. The value of filter_hint_size_x shall be in the range of 1 to 15, inclusive.

filter_hint_type identifies the type of the transmitted filter hints as specified in Table D-7. The value of filter_hint_type shall be in the range of 0 to 2, inclusive. Decoders shall ignore post-filter hint SEI messages having filter_hint_type equal to the reserved value 3.

Table D-7 – filter_hint_type values

Value	Description
0	coefficients of a 2D FIRfilter
1	coefficients of two 1D FIR filters
2	cross-correlation matrix
3	Reserved

filter_hint[colour_component **][** cy **][** cx **]** specifies a filter coefficient or an element of a cross-correlation matrix between original and decoded signal with 16-bit precision. The value of filter_hint[colour_component][cy][cx] shall be in the range of $-2^{31} + 1$ to $2^{31} - 1$, inclusive. colour_component specifies the related colour component. cy represents a counter in vertical direction, cx represents a counter in horizontal direction. Depending on filter_hint_type, the following applies:

– If filter_hint_type is equal to 0, the coefficients of a 2-dimensional FIR filter with the size of filter_hint_size_y * filter_hint_size_x are transmitted.

– Otherwise, if filter_hint_type is equal to 1, the filter coefficients of two 1-dimensional FIR filters are transmitted. In this case, filter_hint_size_y shall be equal to 2. The index cy = 0 specifies the filter coefficients of the horizontal filter and cy = 1 specifies the filter coefficients of the vertical filter. In the filtering process, the horizontal filter shall be applied first and the result shall be filtered by the vertical filter.

– Otherwise (filter_hint_type is equal to 2), the transmitted hints specify a cross-correlation matrix between the original signal s and the decoded signal s′.

NOTE 1 – The normalized cross-correlation matrix for a related colour component with the size of filter_hint_size_y * filter_hint_size_x is defined as follows:

$$\text{filter\_hint}(\text{cy},\text{cx}) = \frac{1}{(2^{8+bitDepth}-1)^2 \cdot h \cdot w} \sum_{m=0}^{h-1} \sum_{n=0}^{w-1} s(m,n) \cdot s'(m+\text{cy}-offset\_y, n+\text{cx}-offset\_x) \qquad \text{(D-27)}$$

where s denotes the original frame, s' denotes the decoded frame, h denotes the vertical height of the related colour component, w denotes the horizontal width of the related colour component, $bitDepth$ denotes the bit depth of the colour component, $offset\_y$ is equal to (filter_hint_size_y >> 1), $offset\_x$ is equal to (filter_hint_size_x >> 1), 0 <= cy < filter_hint_size_y and 0 <= cx < filter_hint_size_x.

NOTE 2 – A decoder can derive a Wiener post-filter from the cross-correlation matrix of original and decoded signal and the auto-correlation matrix of the decoded signal.

additional_extension_flag equal to 0 indicates that no additional data follows within the post-filter hint SEI message. The value of additional_extension_flag shall be equal to 0. The value of 1 for additional_extension_flag is reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore all data that follows the value of 1 for additional_extension_flag in a post-filter hint SEI message.

D.2.24 Tone mapping information SEI message semantics

This SEI message provides information to enable remapping of the colour samples of the output decoded pictures for customization to particular display environments. The remapping process maps coded sample values in the RGB colour space (specified in Annex E) to target sample values. All mappings are expressed in the RGB colour space and should be applied to each RGB component separately.

tone_map_id contains an identifying number that may be used to identify the purpose of the tone mapping model. The value of tone_map_id shall be in the range of 0 to $2^{32} - 2$, inclusive.

Values of tone_map_id from 0 to 255 and from 512 to $2^{31} - 1$ may be used as determined by the application. Values of tone_map_id from 256 to 511 and from 2^{31} to $2^{32} - 2$ are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore (remove from the bitstream and discard) all tone mapping information SEI messages containing a value of tone_map_id in the range of 256 to 511 or in the range of 2^{31} to $2^{32} - 2$, and bitstreams shall not contain such values.

NOTE 1 – The tone_map_id can be used to support tone mapping operations that are suitable for different display scenarios. For example, different values of tone_map_id may correspond to different display bit depths.

tone_map_cancel_flag equal to 1 indicates that the tone mapping information SEI message cancels the persistence of any previous tone mapping information SEI message in output order. tone_map_cancel_flag equal to 0 indicates that tone mapping information follows.

tone_map_repetition_period specifies the persistence of the tone mapping information SEI message and may specify a picture order count interval within which another tone mapping information SEI message with the same value of tone_map_id or the end of the coded video sequence shall be present in the bitstream. The value of tone_map_repetition_period shall be in the range of 0 to 16 384, inclusive.

tone_map_repetition_period equal to 0 specifies that the tone map information applies to the current decoded picture only.

tone_map_repetition_period equal to 1 specifies that the tone map information persists in output order until any of the following conditions are true:

– A new coded video sequence begins.
– A picture in an access unit containing a tone mapping information SEI message with the same value of tone_map_id is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic).

tone_map_repetition_period equal to 0 or equal to 1 indicates that another tone mapping information SEI message with the same value of tone_map_id may or may not be present.

tone_map_repetition_period greater than 1 specifies that the tone map information persists until any of the following conditions are true:

– A new coded video sequence begins.
– A picture in an access unit containing a tone mapping information SEI message with the same value of tone_map_id is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + tone_map_repetition_period.

tone_map_repetition_period greater than 1 indicates that another tone mapping information SEI message with the same value of tone_map_id shall be present for a picture in an access unit that is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + tone_map_repetition_period; unless the bitstream ends or a new coded video sequence begins without output of such a picture.

coded_data_bit_depth specifies the $BitDepth_Y$ of the luma component of the coded video sequence. It is used to identify the tone mapping information SEI message that is intended for use with the coded video sequence. If tone mapping information SEI messages are present that have coded_data_bit_depth that is not equal to $BitDepth_Y$, these refer to the hypothetical result of a transcoding operation performed to convert the coded video to the $BitDepth_Y$ corresponding to the value of coded_data_bit_depth.

The value of coded_data_bit_depth shall be in the range of 8 to 14, inclusive. Values of coded_data_bit_depth from 0 to 7 and from 15 to 255 are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore (remove from the bitstream and discard) all tone mapping SEI messages that contain a coded_data_bit_depth in the range of 0 to 7 or in the range of 15 to 255, and bitstreams shall not contain such values.

target_bit_depth specifies the bit depth of the output of the dynamic range mapping function (or tone mapping function) described by the tone mapping information SEI message. The tone mapping function specified with a particular target_bit_depth is suggested to be reasonable for all display bit depths that are equal to or less than the target_bit_depth.

The value of target_bit_depth shall be in the range of 1 to 16, inclusive. Values of target_bit_depth equal to 0 and in the range of 17 to 255 are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore (remove from the bitstream and discard) all tone mapping SEI messages that contain a value of target_bit_depth equal to 0 or in the range of 17 to 255, and bitstreams shall not contain such values.

model_id specifies the model utilized for mapping the coded data into the target_bit_depth range. Values greater than 3 are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore (remove from the bitstream and discard) all tone mapping SEI messages that contain a value of model_id greater than 3, and bitstreams shall not contain such values.

NOTE 2 – A model_id of 0 corresponds to a linear mapping with clipping; a model_id of 1 corresponds to a sigmoidal mapping; a model_id of 2 corresponds to a user-defined table mapping, and a model_id of 3 corresponds to a piece-wise linear mapping.

min_value specifies the RGB sample value in the coded data that maps to the minimum value in the signalled target_bit_depth. It is used in combination with the max_value parameter. All values in the coded data that are less than or equal to min_value are mapped to this minimum value in the target_bit_depth representation.

max_value specifies the RGB sample value in the coded data that maps to the maximum value in the signalled target_bit_depth. It is used in combination with the min_value parameter. All values in the coded data that are larger than or equal to max_value are mapped to this maximum value in the target_bit_depth representation.

sigmoid_midpoint specifies the RGB sample value of the coded data that is mapped to the centre point of the target_bit_depth representation. It is used in combination with the sigmoid_width parameter.

sigmoid_width specifies the distance between two coded data values that approximately correspond to the 5% and 95% values of the target_bit_depth representation, respectively. It is used in combination with the sigmoid_midpoint parameter and is interpreted according to the following function:

$$f(i) = Round\left(\frac{2^{target\_bit\_depth} - 1}{1 + \exp\left(\frac{-6 * (i - sigmoid\_midpoint)}{sigmoid\_width} \right)} \right) \qquad for\ i = 0, \cdots, (2^{coded\_bit\_depth} - 1) \qquad \text{(D-28)}$$

where *f*(i) denotes the function that maps an RGB sample value i from the coded data to a resulting RGB sample value in the target_bit_depth representation.

start_of_coded_interval[i] specifies the beginning point of an interval in the coded data such that all RGB sample values that are greater than or equal to start_of_coded_interval[i] and less than start_of_coded_interval[i + 1] are mapped to i in the target bit depth representation. The value of start_of_coded_interval[$2^{target\_bit\_depth}$] is equal to $2^{coded\_bit\_depth}$. The number of bits used for the representation of the start_of_coded_interval is ((coded_data_bit_depth + 7) >> 3) << 3.

num_pivots specifies the number of pivot points in the piece-wise linear mapping function without counting the two default end points, (0, 0) and ($2^{coded\_data\_bit\_depth} - 1$, $2^{target\_bit\_depth} - 1$) .

coded_pivot_value[i] specifies the value in the coded_data_bit_depth corresponding to the i-th pivot point. The number of bits used for the representation of the coded_pivot_value is ((coded_data_bit_depth + 7) >> 3) << 3.

target_pivot_value[i] specifies the value in the reference target_bit_depth corresponding to the i-th pivot point. The number of bits used for the representation of the target_pivot_value is ((target_bit_depth + 7) >> 3) << 3.

D.2.25 Frame packing arrangement SEI message semantics

This SEI message informs the decoder that the output cropped decoded picture contains samples of multiple distinct spatially packed constituent frames that are packed into one frame using an indicated frame packing arrangement scheme. This information can be used by the decoder to appropriately rearrange the samples and process the samples of the constituent frames appropriately for display or other purposes (which are outside the scope of this Specification).

This SEI message may be associated with pictures that are either frames or fields. The frame packing arrangement of the samples is specified in terms of the sampling structure of a frame in order to define a frame packing arrangement structure that is invariant with respect to whether a picture is a single field of such a packed frame or is a complete packed frame.

frame_packing_arrangement_id contains an identifying number that may be used to identify the usage of the frame packing arrangement SEI message. The value of frame_packing_arrangement_id shall be in the range of 0 to $2^{32} - 2$, inclusive.

Values of frame_packing_arrangement_id from 0 to 255 and from 512 to $2^{31} - 1$ may be used as determined by the application. Values of frame_packing_arrangement_id from 256 to 511 and from 2^{31} to $2^{32} - 2$ are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore (remove from the bitstream and discard) all frame packing arrangement SEI messages containing a value of frame_packing_arrangement_id in the range of 256 to 511 or in the range of 2^{31} to $2^{32} - 2$, and bitstreams shall not contain such values.

frame_packing_arrangement_cancel_flag equal to 1 indicates that the frame packing arrangement SEI message cancels the persistence of any previous frame packing arrangement SEI message in output order. frame_packing_arrangement_cancel_flag equal to 0 indicates that frame packing arrangement information follows.

frame_packing_arrangement_type indicates the type of packing arrangement of the frames as specified in Table D-8.

Table D-8 – Definition of frame_packing_arrangement_type

Value	Interpretation
0	Each component plane of the decoded frames contains a "checkerboard" based interleaving of corresponding planes of two constituent frames as illustrated in Figure D-1.
1	Each component plane of the decoded frames contains a column based interleaving of corresponding planes of two constituent frames as illustrated in Figure D-2 and Figure D-3.
2	Each component plane of the decoded frames contains a row based interleaving of corresponding planes of two constituent frames as illustrated in Figure D-4 and Figure D-5.
3	Each component plane of the decoded frames contains a side-by-side packing arrangement of corresponding planes of two constituent frames as illustrated in Figure D-6, Figure D-7, and Figure D-10.
4	Each component plane of the decoded frames contains a top-bottom packing arrangement of corresponding planes of two constituent frames as illustrated in Figure D-8 and Figure D-9.
5	The component planes of the decoded frames in output order form a temporal interleaving of alternating first and second constituent frames as illustrated in Figure D-11.

NOTE 1 – Figure D-1 to Figure D-10 provide typical examples of rearrangement and upconversion processing for various packing arrangement schemes. Actual characteristics of the constituent frames are signalled in detail by the subsequent syntax elements of the frame packing arrangement SEI message. In Figure D-1 to Figure D-10, an upconversion processing is performed on each constituent frame to produce frames having the same resolution as that of the decoded frame. An example of the upsampling method to be applied to a quincunx sampled frame as shown in Figure D-1 or Figure D-10 is to fill in missing positions with an average of the available spatially neighbouring samples (the average of the values of the available samples above, below, to the left and to the right of each sample to be generated). The actual upconversion process to be performed, if any, is outside the scope of this Specification.

NOTE 2 – The sample aspect ratio (SAR) indicated in the VUI parameters should indicate the output picture shape for the packed decoded frame output by a decoder that does not interpret the frame packing arrangement SEI message. In the examples shown in Figure D-1 to Figure D-10, the SAR produced in each upconverted colour plane would be the same as the SAR indicated in the VUI parameters, since the illustrated upconversion process produces the same total number of samples from each constituent frame as existed in the packed decoded frame.

NOTE 3 – When the output time of the samples of constituent frame 0 differs from the output time of the samples of constituent frame 1 (i.e., when field_views_flag is equal to 1 or frame_packing_arrangement_type is equal to 5) and the display system in use presents two views simultaneously, the display time for constituent frame 0 should be delayed to coincide with the display time for constituent frame 1. (The display process is not specified in this Recommendation | International Standard.)

NOTE 4 – When field_views_flag is equal to 1 or frame_packing_arrangement_type is equal to 5, the value 0 for fixed_frame_rate_flag is not expected to be prevalent in industry use of this SEI message.

NOTE 5 – frame_packing_arrangement_type equal to 5 describes a temporal interleaving process of different views.

All other values of frame_packing_arrangement_type are reserved for future use by ITU-T | ISO/IEC. It is a requirement of bitstream conformance that the bitstreams shall not contain such other values of frame_packing_arrangement_type.

quincunx_sampling_flag equal to 1 indicates that each colour component plane of each constituent frame is quincunx sampled as illustrated in Figure D-1 or Figure D-10, and quincunx_sampling_flag equal to 0 indicates that the colour component planes of each constituent frame are not quincunx sampled.

When frame_packing_arrangement_type is equal to 0, it is a requirement of bitstream conformance that quincunx_sampling_flag shall be equal to 1. When frame_packing_arrangement_type is equal to 5, it is a requirement of bitstream conformance that quincunx_sampling_flag shall be equal to 0.

NOTE 6 – For any chroma format (4:2:0, 4:2:2, or 4:4:4), the luma plane and each chroma plane is quincunx sampled as illustrated in Figure D-1 when quincunx_sampling_flag is equal to 1.

content_interpretation_type indicates the intended interpretation of the constituent frames as specified in Table D-9. Values of content_interpretation_type that do not appear in Table D-9 are reserved for future specification by ITU-T | ISO/IEC.

For each specified frame packing arrangement scheme, there are two constituent frames that are referred to as frame 0 and frame 1.

Table D-9 – Definition of content_interpretation_type

Value	Interpretation
0	Unspecified relationship between the frame packed constituent frames
1	Indicates that the two constituent frames form the left and right views of a stereo view scene, with frame 0 being associated with the left view and frame 1 being associated with the right view
2	Indicates that the two constituent frames form the right and left views of a stereo view scene, with frame 0 being associated with the right view and frame 1 being associated with the left view

NOTE 7 – The value 2 for content_interpretation_type is not expected to be prevalent in industry use of this SEI message. However, the value was specified herein for purposes of completeness.

spatial_flipping_flag equal to 1, when frame_packing_arrangement_type is equal to 3 or 4, indicates that one of the two constituent frames is spatially flipped relative to its intended orientation for display or other such purposes.

When frame_packing_arrangement_type is equal to 3 or 4 and spatial_flipping_flag is equal to 1, the type of spatial flipping that is indicated is as follows:

– If frame_packing_arrangement_type is equal to 3, the indicated spatial flipping is horizontal flipping.

– Otherwise (frame_packing_arrangement_type is equal to 4), the indicated spatial flipping is vertical flipping.

When frame_packing_arrangement_type is not equal to 3 or 4, it is a requirement of bitstream conformance that spatial_flipping_flag shall be equal to 0. When frame_packing_arrangement_type is not equal to 3 or 4, the value 1 for spatial_flipping_flag is reserved for future use by ITU-T | ISO/IEC. When frame_packing_arrangement_type is not equal to 3 or 4, decoders shall ignore the value 1 for spatial_flipping_flag.

frame0_flipped_flag, when spatial_flipping_flag is equal to 1, indicates which one of the two constituent frames is flipped.

When spatial_flipping_flag is equal to 1, frame0_flipped_flag equal to 0 indicates that frame 0 is not spatially flipped and frame 1 is spatially flipped, and frame0_flipped_flag equal to 1 indicates that frame 0 is spatially flipped and frame 1 is not spatially flipped.

When spatial_flipping_flag is equal to 0, it is a requirement of bitstream conformance that frame0_flipped_flag shall be equal to 0. When spatial_flipping_flag is equal to 0, the value 1 for spatial_flipping_flag is reserved for future use by ITU-T | ISO/IEC. When spatial_flipping_flag is equal to 0, decoders shall ignore the value of frame0_flipped_flag.

field_views_flag equal to 1 indicates that all pictures in the current coded video sequence are coded as complementary field pairs. All fields of a particular parity are considered a first constituent frame and all fields of the opposite parity are considered a second constituent frame. When frame_packing_arrangement_type is not equal to 2, it is a requirement of bitstream conformance that the field_views_flag shall be equal to 0. When frame_packing_arrangement_type is not equal to 2, the value 1 for field_views_flag is reserved for future use by ITU-T | ISO/IEC. When frame_packing_arrangement_type is not equal to 2, decoders shall ignore the value of field_views_flag.

current_frame_is_frame0_flag equal to 1, when frame_packing_arrangement is equal to 5, indicates that the current decoded frame is constituent frame 0 and the next decoded frame in output order is constituent frame 1, and the display time of the constituent frame 0 should be delayed to coincide with the display time of constituent frame 1. current_frame_is_frame0_flag equal to 0, when frame_packing_arrangement is equal to 5, indicates that the current decoded frame is constituent frame 1 and the previous decoded frame in output order is constituent frame 0, and the display time of the constituent frame 1 should not be delayed for purposes of stereo-view pairing.

When frame_packing_arrangement_type is not equal to 5, the constituent frame associated with the upper-left sample of the decoded frame is considered to be consitutuent frame 0 and the other constituent frame is considered to be constituent frame 1. When frame_packing_arrangement_type is not equal to 5, it is a requirement of bitstream conformance that current_frame_is_frame0_flag shall be equal to 0. When frame_packing_arrangement_type is not equal to 5, the value 1 for current_frame_is_frame0_flag is reserved for future use by ITU-T | ISO/IEC. When frame_packing_arrangement_type is not equal to 5, decoders shall ignore the value of current_frame_is_frame0_flag.

frame0_self_contained_flag equal to 1 indicates that no inter prediction operations within the decoding process for the samples of constituent frame 0 of the coded video sequence refer to samples of any constituent frame 1. frame0_self_contained_flag equal to 0 indicates that some inter prediction operations within the decoding process for the samples of constituent frame 0 of the coded video sequence may or may not refer to samples of some constituent frame 1. When frame_packing_arrangement_type is equal to 0 or 1, it is a requirement of bitstream conformance that frame0_self_contained_flag shall be equal to 0. When frame_packing_arrangement_type is equal to 0 or 1, the value 1 for frame0_self_contained_flag is reserved for future use by ITU-T | ISO/IEC. When frame_packing_arrangement_type is equal to 0 or 1, decoders shall ignore the value of frame0_self_contained_flag. Within a coded video sequence, the value of frame0_self_contained_flag in all frame packing arrangement SEI messages shall be the same.

frame1_self_contained_flag equal to 1 indicates that no inter prediction operations within the decoding process for the samples of constituent frame 1 of the coded video sequence refer to samples of any constituent frame 0. frame1_self_contained_flag equal to 0 indicates that some inter prediction operations within the decoding process for the samples of constituent frame 1 of the coded video sequence may or may not refer to samples of some constituent frame 0. When frame_packing_arrangement_type is equal to 0 or 1, it is a requirement of bitstream conformance that frame1_self_contained_flag shall be equal to 0. When frame_packing_arrangement_type is equal to 0 or 1, the value 1 for frame1_self_contained_flag is reserved for future use by ITU-T | ISO/IEC. When frame_packing_arrangement_type is equal to 0 or 1, decoders shall ignore the value of frame1_self_contained_flag. Within a coded video sequence, the value of frame1_self_contained_flag in all frame packing arrangement SEI messages shall be the same.

NOTE 8 – When frame0_self_contained_flag is equal to 1 or frame1_self_contained_flag is equal to 1, and frame_packing_arrangement_type is equal to 2, it is expected that the decoded frame should not be an MBAFF frame.

When quincunx_sampling_flag is equal to 0 and frame_packing_arrangement_type is not equal to 5, two (x, y) coordinate pairs are specified to determine the indicated luma sampling grid alignment for constituent frame 0 and constituent frame 1, relative to the upper left corner of the rectangular area represented by the samples of the corresponding constituent frame.

NOTE 9 – The location of chroma samples relative to luma samples can be indicated by the chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field syntax elements in the VUI parameters.

frame0_grid_position_x (when present) specifies the x component of the (x, y) coordinate pair for constituent frame 0.

frame0_grid_position_y (when present) specifies the y component of the (x, y) coordinate pair for constituent frame 0.

frame1_grid_position_x (when present) specifies the x component of the (x, y) coordinate pair for constituent frame 1.

frame1_grid_position_y (when present) specifies the y component of the (x, y) coordinate pair for constituent frame 1.

When quincunx_sampling_flag is equal to 0 and frame_packing_arrangement_type is not equal to 5 the (x, y) coordinate pair for each constituent frame is interpreted as follows:

– If the (x, y) coordinate pair for a constituent frame is equal to (0, 0), this indicates a default sampling grid alignment specified as follows:
  – If frame_packing_arrangement_type is equal to 1 or 3, the indicated position is the same as for the (x, y) coordinate pair value (4, 8), as illustrated in Figure D-2 and Figure D-6.
  – Otherwise (frame_packing_arrangement_type is equal to 2 or 4), the indicated position is the same as for the (x, y) coordinate pair value (8, 4), as illustrated in Figure D-4 and Figure D-8.
– Otherwise, if the (x, y) coordinate pair for a constituent frame is equal to (15, 15), this indicates that the sampling grid alignment is unknown or unspecified or specified by other means not specified in this Recommendation | International Standard.
– Otherwise, the x and y elements of the (x, y) coordinate pair specify the indicated horizontal and vertical sampling grid alignment positioning to the right of and below the upper left corner of the rectangular area represented by the corresponding constituent frame, respectively, in units of one sixteenth of the luma sample grid spacing between the samples of the columns and rows of the constituent frame that are present in the decoded frame (prior to any upsampling for display or other purposes).

NOTE 10 – The spatial location reference information frame0_grid_position_x, frame0_grid_position_y, frame1_grid_position_x, and frame1_grid_position_y is not provided when quincunx_sampling_flag is equal to 1 because the spatial alignment in this case is assumed to be such that constituent frame 0 and constituent frame 1 cover corresponding spatial areas with interleaved quincunx sampling patterns as illustrated in Figure D-1 and Figure D-10.

NOTE 11 – When frame_packing_arrangement_type is equal to 2 and field_views_flag is equal to 1, it is suggested that frame0_grid_position_y should be equal to frame1_grid_position_y.

frame_packing_arrangement_reserved_byte is reserved for future use by ITU-T | ISO/IEC. It is a requirement of bitstream conformance that the value of frame_packing_arrangement_reserved_byte shall be equal to 0. All other values of frame_packing_arrangement_reserved_byte are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore (remove from the bitstream and discard) the value of frame_packing_arrangement_reserved_byte.

frame_packing_arrangement_repetition_period specifies the persistence of the frame packing arrangement SEI message and may specify a frame order count interval within which another frame packing arrangement SEI message with the same value of frame_packing_arrangement_id or the end of the coded video sequence shall be present in the bitstream. The value of frame_packing_arrangement_repetition_period shall be in the range of 0 to 16 384, inclusive.

frame_packing_arrangement_repetition_period equal to 0 specifies that the frame packing arrangement SEI message applies to the current decoded frame only.

frame_packing_arrangement_repetition_period equal to 1 specifies that the frame packing arrangement SEI message persists in output order until any of the following conditions are true:

– A new coded video sequence begins.
– A frame in an access unit containing a frame packing arrangement SEI message with the same value of frame_packing_arrangement_id is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic).

frame_packing_arrangement_repetition_period equal to 0 or equal to 1 indicates that another frame packing arrangement SEI message with the same value of frame_packing_arrangement_id may or may not be present.

frame_packing_arrangement_repetition_period greater than 1 specifies that the frame packing arrangement SEI message persists until any of the following conditions are true:

– A new coded video sequence begins.
– A frame in an access unit containing a frame packing arrangement SEI message with the same value of frame_packing_arrangement_id is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + frame_packing_arrangement_repetition_period.

frame_packing_arrangement_repetition_period greater than 1 indicates that another frame packing arrangement SEI message with the same value of frame_packing_arrangement_frames_id shall be present for a frame in an access unit that is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + frame_packing_arrangement_repetition_period; unless the bitstream ends or a new coded video sequence begins without output of such a frame.

frame_packing_arrangement_extension_flag equal to 0 indicates that no additional data follows within the frame packing arrangement SEI message. It is a requirement of bitstream conformance that the value of frame_packing_arrangement_extension_flag shall be equal to 0. The value 1 for frame_packing_arrangement_extension_flag is reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore the value 1 for frame_packing_arrangement_extension_flag in a frame packing arrangement SEI message and shall ignore all data that follows within a frame packing arrangement SEI message after the value 1 for frame_packing_arrangement_extension_flag.

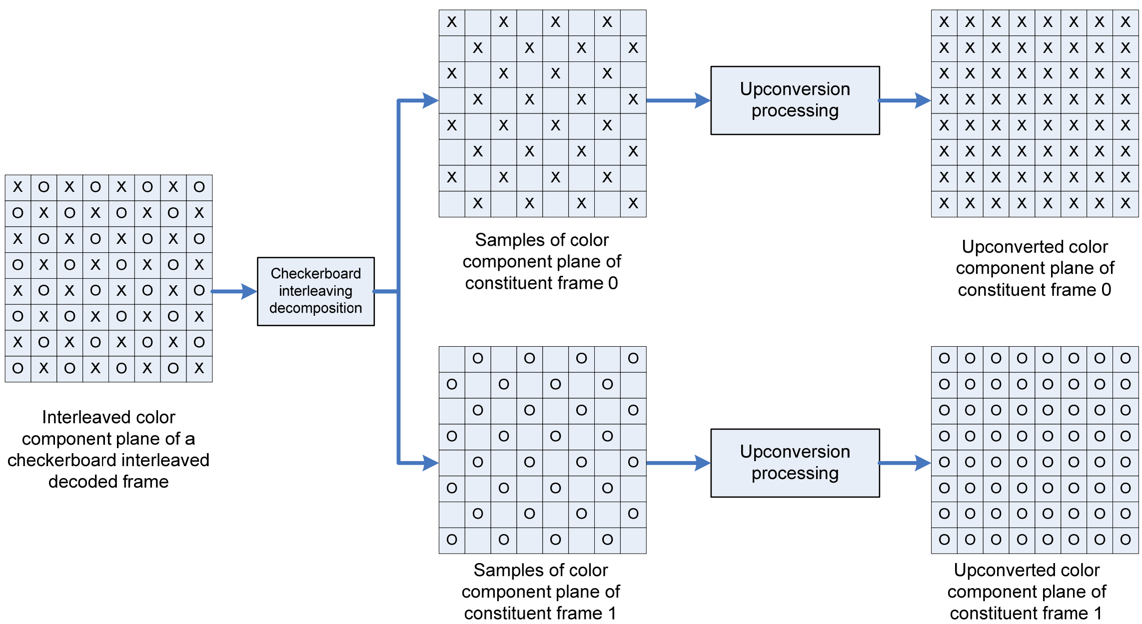


Figure D-1 – Rearrangement and upconversion of checkerboard interleaving (frame_packing_arrangement_type equal to 0)

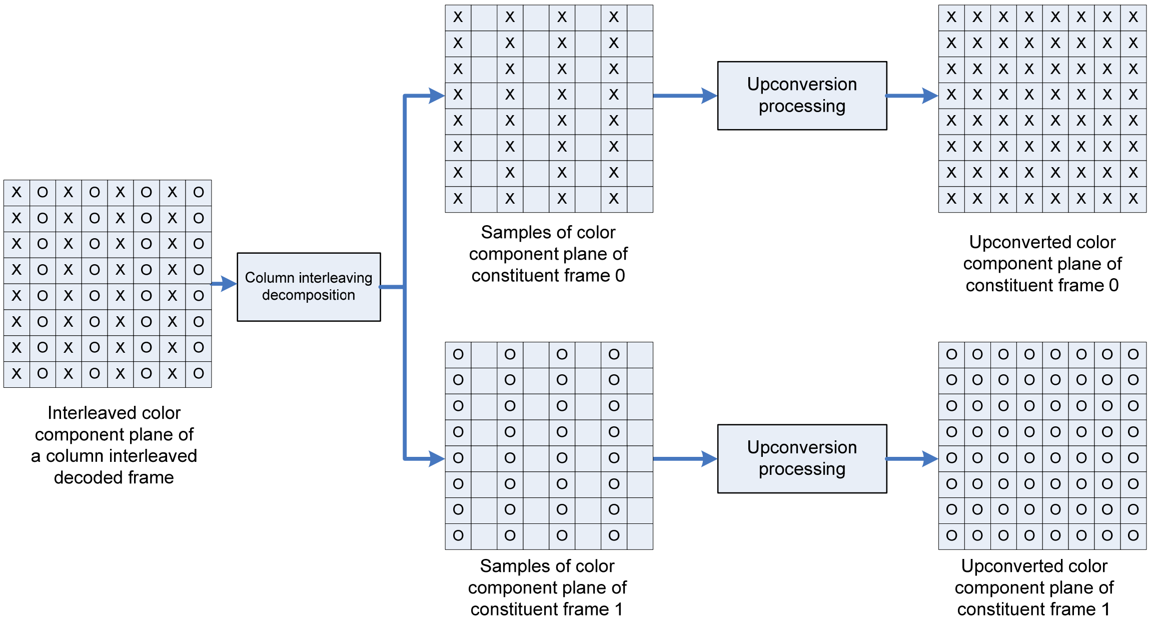


Figure D-2 – Rearrangement and upconversion of column interleaving with frame_packing_arrangement_type equal to 1, quincunx_sampling_flag equal to 0, and (x, y) equal to (0, 0) or (4, 8) for both constituent frames

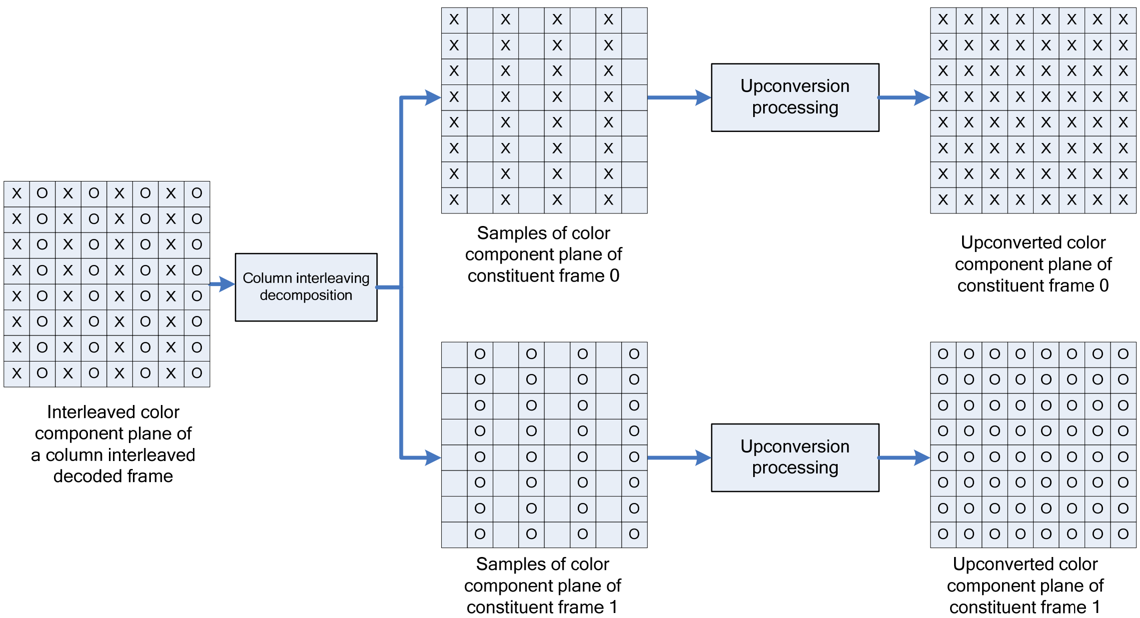


Figure D-3 – Rearrangement and upconversion of column interleaving with frame_packing_arrangement_type equal to 1, quincunx_sampling_flag equal to 0, (x, y) equal to (0, 0) or (4, 8) for constituent frame 0 and (x, y) equal to (12, 8) for constituent frame 1

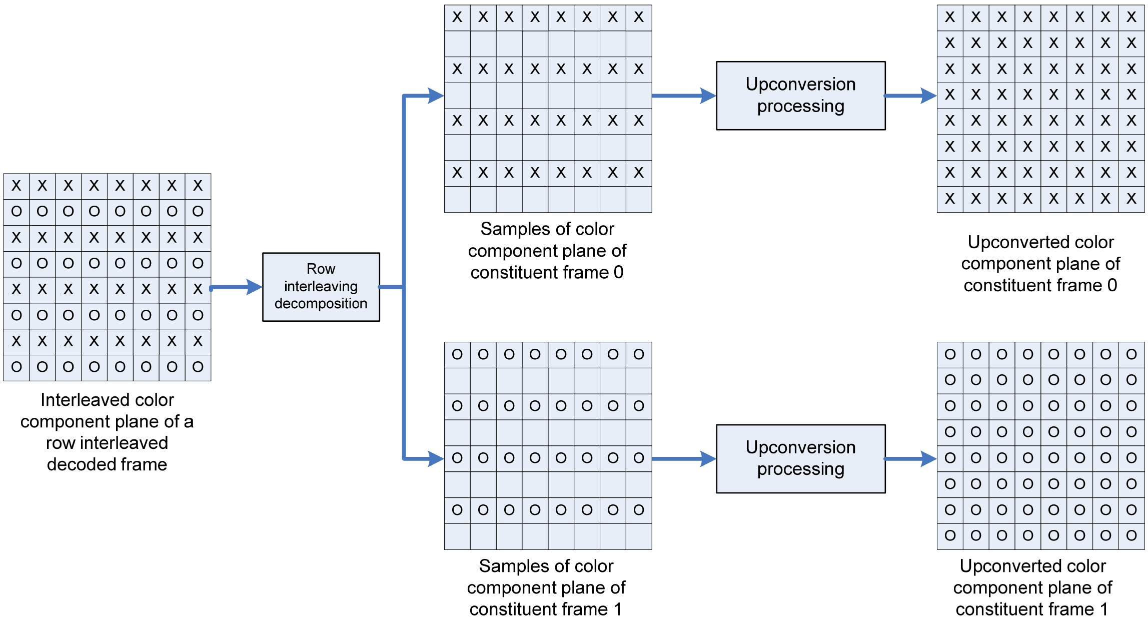


Figure D-4 – Rearrangement and upconversion of row interleaving with frame_packing_arrangement_type equal to 2, quincunx_sampling_flag equal to 0, and (x, y) equal to (0, 0) or (8, 4) for both constituent frames

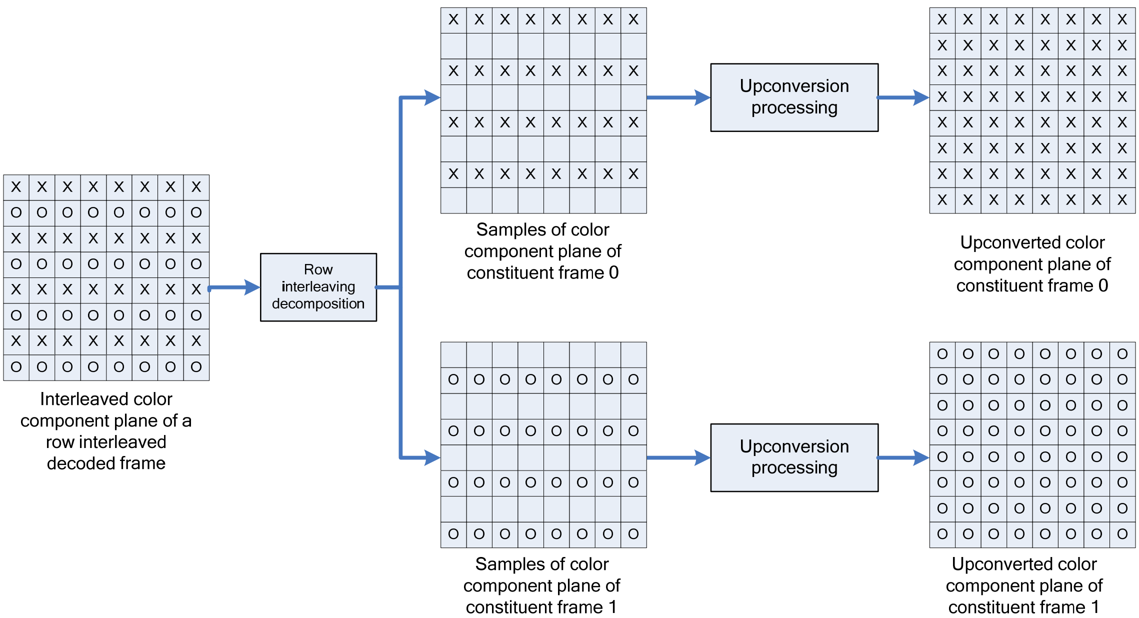


Figure D-5 – Rearrangement and upconversion of row interleaving with frame_packing_arrangement_type equal to 2, quincunx_sampling_flag equal to 0, (x, y) equal to (0, 0) or (8, 4) for constituent frame 0, and (x, y) equal to (8, 12) for constituent frame 1

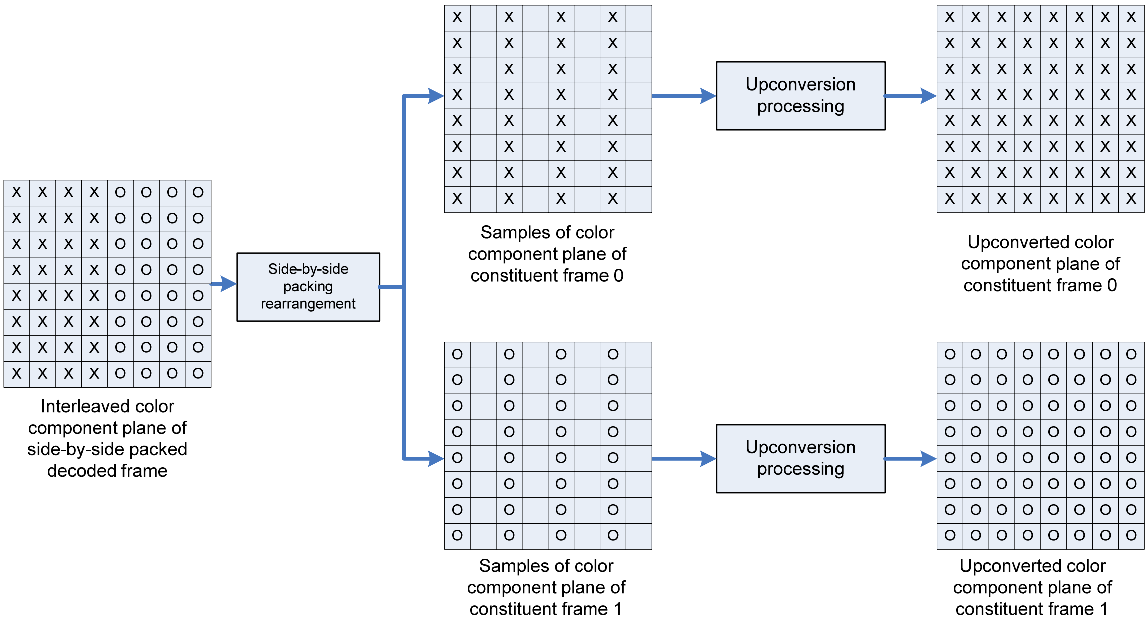


Figure D-6 – Rearrangement and upconversion of side-by-side packing arrangement with frame_packing_arrangement_type equal to 3, quincunx_sampling_flag equal to 0, and (x, y) equal to (0, 0) or (4, 8) for both constituent frames

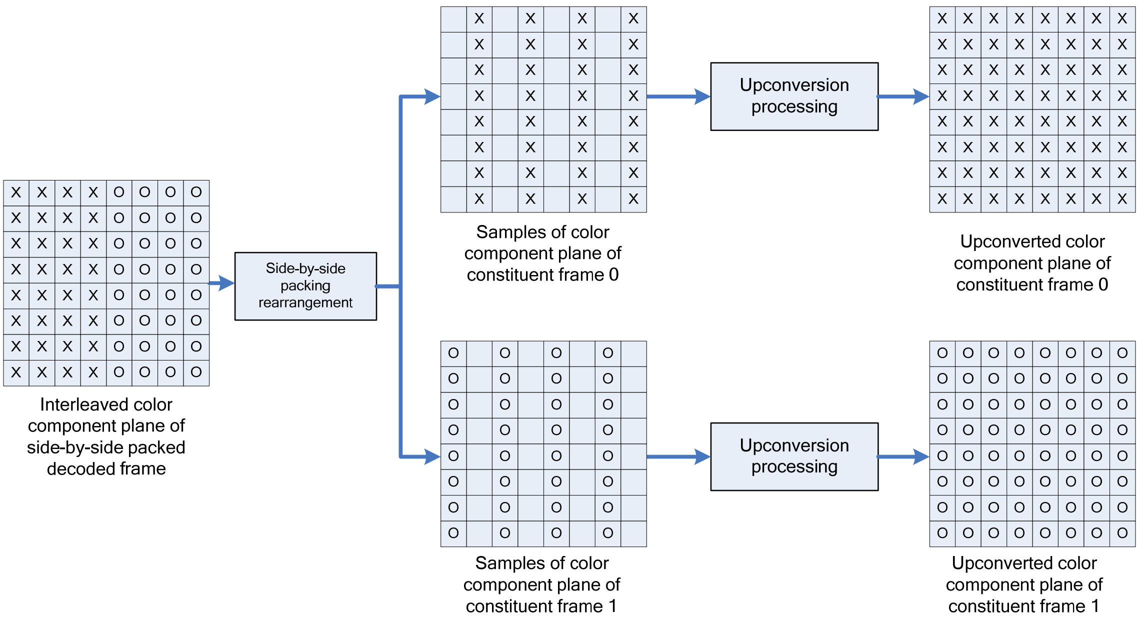


Figure D-7 – Rearrangement and upconversion of side-by-side packing arrangement with frame_packing_arrangement_type equal to 3, quincunx_sampling_flag equal to 0, (x, y) equal to (12, 8) for constituent frame 0, and (x, y) equal to (0, 0) or (4, 8) for constituent frame 1

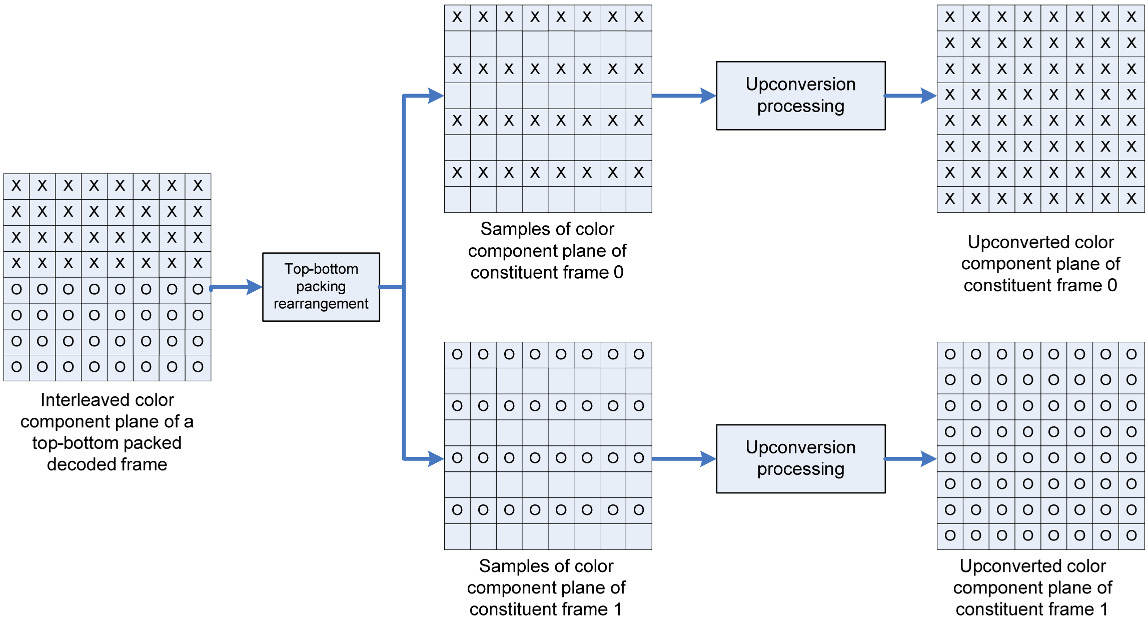


Figure D-8 – Rearrangement and upconversion of top-bottom packing arrangement with frame_packing_arrangement_type equal to 4, quincunx_sampling_flag equal to 0, and (x, y) equal to (0, 0) or (8, 4) for both constituent frames

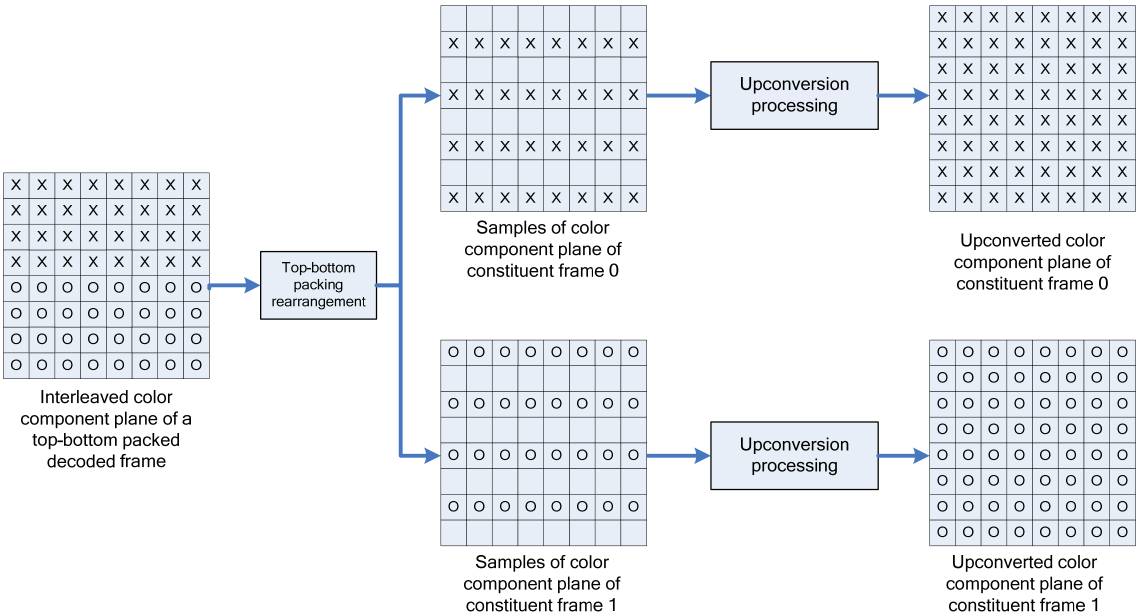


Figure D-9 – Rearrangement and upconversion of top-bottom packing arrangement with frame_packing_arrangement_type equal to 4, quincunx_sampling_flag equal to 0, (x, y) equal to (8, 12) for constituent frame 0, and (x, y) equal to (0, 0) or (8, 4) for constituent frame 1

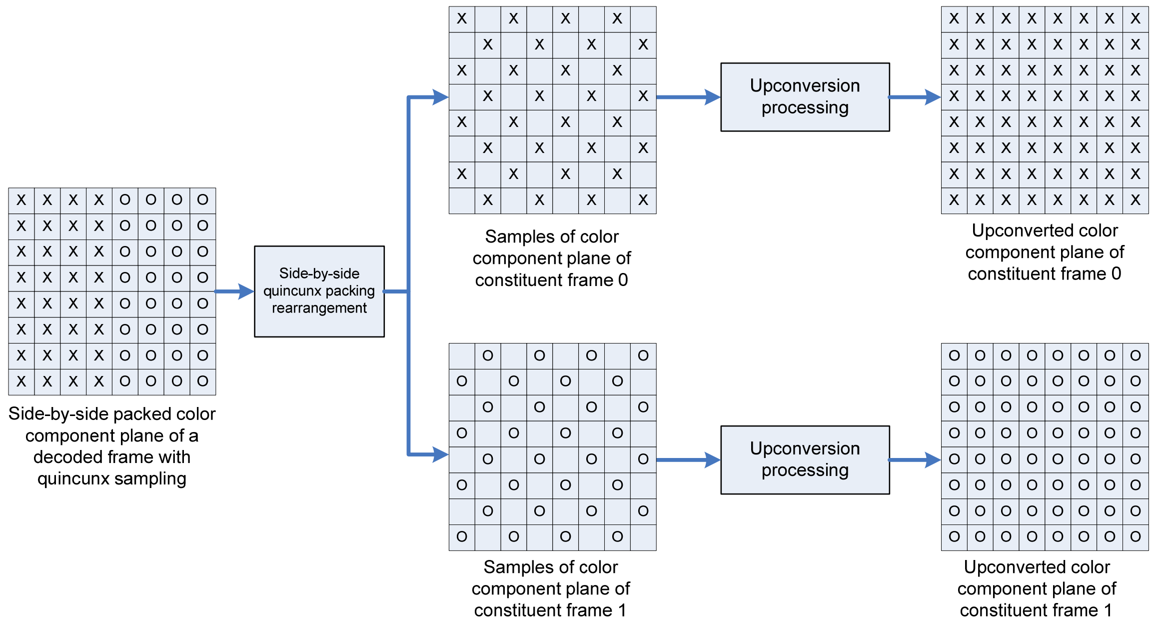


Figure D-10 – Rearrangement and upconversion of side-by-side packing arrangement with quincunx sampling (frame_packing_arrangement_type equal to 3 with quincunx_sampling_flag equal to 1)

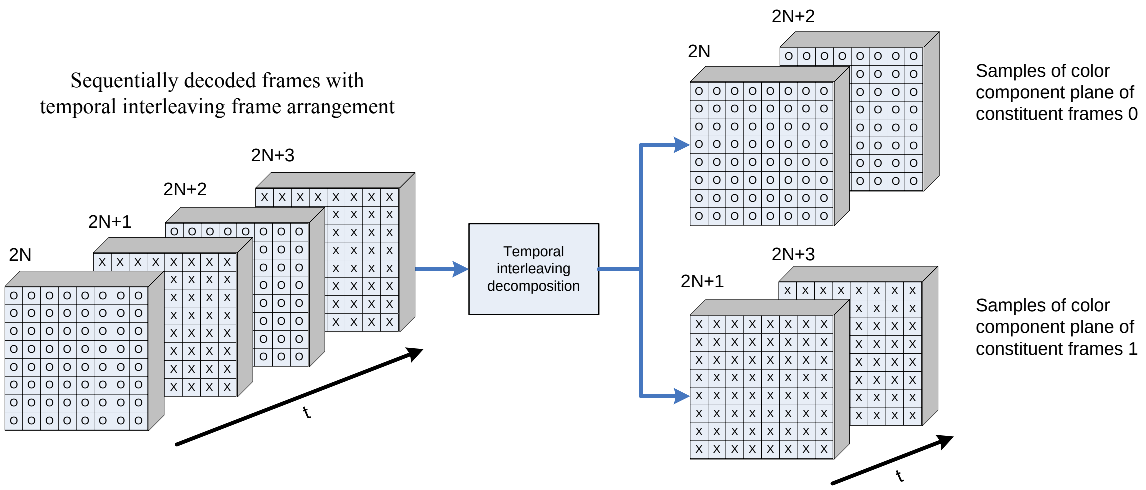


Figure D-11 – Rearrangement of a temporal interleaving frame arrangement (frame_packing_arrangement_type equal to 5)

D.2.26 Reserved SEI message semantics

This message consists of data reserved for future backward-compatible use by ITU-T | ISO/IEC. Encoders conforming to this Recommendation | International Standard shall not send reserved SEI messages until and unless the use of such messages has been specified by ITU-T | ISO/IEC. Decoders that encounter reserved SEI messages shall discard their content without effect on the decoding process, except as specified in future Recommendations | International Standards specified by ITU-T | ISO/IEC.

reserved_sei_message_payload_byte is a byte reserved for future use by ITU-T | ISO/IEC.

Annex E

Video usability information

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies syntax and semantics of the VUI parameters of the sequence parameter sets.

VUI parameters are not required for constructing the luma or chroma samples by the decoding process. Conforming decoders are not required to process this information for output order conformance to this Recommendation | International Standard (see Annex C for the specification of conformance). Some VUI parameters are required to check bitstream conformance and for output timing decoder conformance.

In Annex E, specification for presence of VUI parameters is also satisfied when those parameters (or some subset of them) are conveyed to decoders (or to the HRD) by other means not specified by this Recommendation | International Standard. When present in the bitstream, VUI parameters shall follow the syntax and semantics specified in subclauses 7.3.2.1 and 7.4.2.1 and this annex. When the content of VUI parameters is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the VUI parameters is not required to use the same syntax specified in this annex. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

E.1 VUI syntax

E.1.1 VUI parameters syntax

vui_parameters( ) {	**C**	**Descriptor**
aspect_ratio_info_present_flag	0	u(1)
if(aspect_ratio_info_present_flag) {		
aspect_ratio_idc	0	u(8)
if(aspect_ratio_idc = = Extended_SAR) {		
sar_width	0	u(16)
sar_height	0	u(16)
}		
}		
overscan_info_present_flag	0	u(1)
if(overscan_info_present_flag)		
overscan_appropriate_flag	0	u(1)
video_signal_type_present_flag	0	u(1)
if(video_signal_type_present_flag) {		
video_format	0	u(3)
video_full_range_flag	0	u(1)
colour_description_present_flag	0	u(1)
if(colour_description_present_flag) {		
colour_primaries	0	u(8)
transfer_characteristics	0	u(8)
matrix_coefficients	0	u(8)
}		
}		
chroma_loc_info_present_flag	0	u(1)
if(chroma_loc_info_present_flag) {		
chroma_sample_loc_type_top_field	0	ue(v)

chroma_sample_loc_type_bottom_field	0	ue(v)
}		
timing_info_present_flag	0	u(1)
if(timing_info_present_flag) {		
num_units_in_tick	0	u(32)
time_scale	0	u(32)
fixed_frame_rate_flag	0	u(1)
}		
nal_hrd_parameters_present_flag	0	u(1)
if(nal_hrd_parameters_present_flag)		
hrd_parameters( )	0	
vcl_hrd_parameters_present_flag	0	u(1)
if(vcl_hrd_parameters_present_flag)		
hrd_parameters( )	0	
if(nal_hrd_parameters_present_flag || vcl_hrd_parameters_present_flag)		
low_delay_hrd_flag	0	u(1)
pic_struct_present_flag	0	u(1)
bitstream_restriction_flag	0	u(1)
if(bitstream_restriction_flag) {		
motion_vectors_over_pic_boundaries_flag	0	u(1)
max_bytes_per_pic_denom	0	ue(v)
max_bits_per_mb_denom	0	ue(v)
log2_max_mv_length_horizontal	0	ue(v)
log2_max_mv_length_vertical	0	ue(v)
max_num_reorder_frames	0	ue(v)
max_dec_frame_buffering	0	ue(v)
}		
}		

E.1.2 HRD parameters syntax

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

E.2 VUI semantics

E.2.1 VUI parameters semantics

aspect_ratio_info_present_flag equal to 1 specifies that aspect_ratio_idc is present. aspect_ratio_info_present_flag equal to 0 specifies that aspect_ratio_idc is not present.

aspect_ratio_idc specifies the value of the sample aspect ratio of the luma samples. Table E-1 shows the meaning of the code. When aspect_ratio_idc indicates Extended_SAR, the sample aspect ratio is represented by sar_width : sar_height. When the aspect_ratio_idc syntax element is not present, aspect_ratio_idc value shall be inferred to be equal to 0.

Table E-1 – Meaning of sample aspect ratio indicator

aspect_ratio_idc	Sample aspect ratio	(informative) Examples of use
0	Unspecified	
1	1:1 ("square")	1280x720 16:9 frame without horizontal overscan 1920x1080 16:9 frame without horizontal overscan (cropped from 1920x1088) 640x480 4:3 frame without horizontal overscan
2	12:11	720x576 4:3 frame with horizontal overscan 352x288 4:3 frame without horizontal overscan
3	10:11	720x480 4:3 frame with horizontal overscan 352x240 4:3 frame without horizontal overscan
4	16:11	720x576 16:9 frame with horizontal overscan 528x576 4:3 frame without horizontal overscan
5	40:33	720x480 16:9 frame with horizontal overscan 528x480 4:3 frame without horizontal overscan
6	24:11	352x576 4:3 frame without horizontal overscan 480x576 16:9 frame with horizontal overscan
7	20:11	352x480 4:3 frame without horizontal overscan 480x480 16:9 frame with horizontal overscan
8	32:11	352x576 16:9 frame without horizontal overscan
9	80:33	352x480 16:9 frame without horizontal overscan
10	18:11	480x576 4:3 frame with horizontal overscan
11	15:11	480x480 4:3 frame with horizontal overscan
12	64:33	528x576 16:9 frame without horizontal overscan
13	160:99	528x480 16:9 frame without horizontal overscan
14	4:3	1440x1080 16:9 frame without horizontal overscan
15	3:2	1280x1080 16:9 frame without horizontal overscan
16	2:1	960x1080 16:9 frame without horizontal overscan
17..254	Reserved	
255	Extended_SAR	

NOTE 1 – For the examples in Table E-1, the term "without horizontal overscan" refers to display processes in which the display area matches the area of the cropped decoded pictures and the term "with horizontal overscan" refers to display processes in which some parts near the left and/or right border of the cropped decoded pictures are not visible in the display area. As an example, the entry "720x576 4:3 frame with horizontal overscan" for aspect_ratio_idc equal to 2 refers to having an area of 704x576 luma samples (which has an aspect ratio of 4:3) of the cropped decoded frame (720x576 luma samples) that is visible in the display area.

sar_width indicates the horizontal size of the sample aspect ratio (in arbitrary units).

sar_height indicates the vertical size of the sample aspect ratio (in the same arbitrary units as sar_width).

sar_width and sar_height shall be relatively prime or equal to 0. When aspect_ratio_idc is equal to 0 or sar_width is equal to 0 or sar_height is equal to 0, the sample aspect ratio shall be considered unspecified by this Recommendation | International Standard.

overscan_info_present_flag equal to 1 specifies that the overscan_appropriate_flag is present. When overscan_info_present_flag is equal to 0 or is not present, the preferred display method for the video signal is unspecified.

overscan_appropriate_flag equal to 1 indicates that the cropped decoded pictures output are suitable for display using overscan. overscan_appropriate_flag equal to 0 indicates that the cropped decoded pictures output contain visually important information in the entire region out to the edges of the cropping rectangle of the picture, such that the cropped decoded pictures output should not be displayed using overscan. Instead, they should be displayed using either an exact match between the display area and the cropping rectangle, or using underscan. As used in this paragraph, the term "overscan" refers to display processes in which some parts near the borders of the cropped decoded pictures are not visible in the display area. The term "underscan" describes display processes in which the entire cropped decoded pictures are visible in the display area, but they do not cover the entire display area. For display processes that neither use overscan nor underscan, the display area exactly matches the area of the cropped decoded pictures.

NOTE 2 – For example, overscan_appropriate_flag equal to 1 might be used for entertainment television programming, or for a live view of people in a videoconference, and overscan_appropriate_flag equal to 0 might be used for computer screen capture or security camera content.

video_signal_type_present_flag equal to 1 specifies that video_format, video_full_range_flag and colour_description_present_flag are present. video_signal_type_present_flag equal to 0, specify that video_format, video_full_range_flag and colour_description_present_flag are not present.

video_format indicates the representation of the pictures as specified in Table E-2, before being coded in accordance with this Recommendation | International Standard. When the video_format syntax element is not present, video_format value shall be inferred to be equal to 5.

Table E-2 – Meaning of video_format

video_format	Meaning
0	Component
1	PAL
2	NTSC
3	SECAM
4	MAC
5	Unspecified video format
6	Reserved
7	Reserved

video_full_range_flag indicates the black level and range of the luma and chroma signals as derived from E'_Y, E'_{PB}, and E'_{PR} or E'_R, E'_G, and E'_B real-valued component signals.

When the video_full_range_flag syntax element is not present, the value of video_full_range_flag shall be inferred to be equal to 0.

colour_description_present_flag equal to 1 specifies that colour_primaries, transfer_characteristics and matrix_coefficients are present. colour_description_present_flag equal to 0 specifies that colour_primaries, transfer_characteristics and matrix_coefficients are not present.

colour_primaries indicates the chromaticity coordinates of the source primaries as specified in Table E-3 in terms of the CIE 1931 definition of x and y as specified by ISO 11664-1.

When the colour_primaries syntax element is not present, the value of colour_primaries shall be inferred to be equal to 2 (the chromaticity is unspecified or is determined by the application).

Table E-3 – Colour primaries

Value	Primaries			Informative Remark
0	Reserved			For future use by ITU-T | ISO/IEC
1	primary green blue red white D65	x 0.300 0.150 0.640 0.3127	y 0.600 0.060 0.330 0.3290	ITU-R Rec. BT.709-5 ITU-R Rec. BT.1361 conventional colour gamut system and extended colour gamut system IEC 61966-2-4 Society of Motion Picture and Television Engineers RP 177 (1993) Annex B
2	Unspecified			Image characteristics are unknown or are determined by the application.
3	Reserved			For future use by ITU-T | ISO/IEC
4	primary green blue red white C	x 0.21 0.14 0.67 0.310	y 0.71 0.08 0.33 0.316	ITU-R Rec. BT.470-6 System M (historical) United States National Television System Committee 1953 Recommendation for transmission standards for colour television United States Federal Communications Commission Title 47 Code of Federal Regulations (2003) 73.682 (a) (20)
5	primary green blue red white D65	x 0.29 0.15 0.64 0.3127	y 0.60 0.06 0.33 0.3290	ITU-R Rec. BT.470-6 System B, G (historical) ITU-R Rec. BT.601-6 625 ITU-R Rec. BT.1358 625 ITU-R Rec. BT.1700 625 PAL and 625 SECAM
6	primary green blue red white D65	x 0.310 0.155 0.630 0.3127	y 0.595 0.070 0.340 0.3290	ITU-R Rec. BT.601-6 525 ITU-R Rec. BT.1358 525 ITU-R Rec. BT.1700 NTSC Society of Motion Picture and Television Engineers 170M (2004) (functionally the same as the value 7)
7	primary green blue red white D65	x 0.310 0.155 0.630 0.3127	y 0.595 0.070 0.340 0.3290	Society of Motion Picture and Television Engineers 240M (1999) (functionally the same as the value 6)
8	primary green blue red white C	x 0.243 0.145 0.681 0.310	y 0.692 (Wratten 58) 0.049 (Wratten 47) 0.319 (Wratten 25) 0.316	Generic film (colour filters using Illuminant C)
9..255	Reserved			For future use by ITU-T | ISO/IEC

transfer_characteristics indicates the opto-electronic transfer characteristic of the source picture as specified in Table E-4 as a function of a linear optical intensity input L_c with a nominal real-valued range of 0 to 1.

When the transfer_characteristics syntax element is not present, the value of transfer_characteristics shall be inferred to be equal to 2 (the transfer characteristics are unspecified or are determined by the application).

Table E-4 – Transfer characteristics

Value	Transfer Characteristic		Informative Remark
0	Reserved		For future use by ITU-T | ISO/IEC
1	$V = 1.099 * L_c^{0.45} - 0.099$ $V = 4.500 * L_c$	for $1 >= L_c >= 0.018$ for $0.018 > L_c >= 0$	ITU-R Rec. BT.709-5 ITU-R Rec. BT.1361 conventional colour gamut system (functionally the same as the value 6)
2	Unspecified		Image characteristics are unknown or are determined by the application.
3	Reserved		For future use by ITU-T | ISO/IEC
4	Assumed display gamma 2.2		ITU-R Rec. BT.470-6 System M (historical) United States National Television System Committee 1953 Recommendation for transmission standards for colour television United States Federal Communications Commission Title 47 Code of Federal Regulations (2003) 73.682 (a) (20) ITU-R Rec. BT.1700 (2007 revision) 625 PAL and 625 SECAM
5	Assumed display gamma 2.8		ITU-R Rec. BT.470-6 System B, G (historical)
6	$V = 1.099 * L_c^{0.45} - 0.099$ $V = 4.500 * L_c$	for $1 >= L_c >= 0.018$ for $0.018 > L_c >= 0$	ITU-R Rec. BT.601-6 525 or 625 ITU-R Rec. BT.1358 525 or 625 ITU-R Rec. BT.1700 NTSC Society of Motion Picture and Television Engineers 170M (2004) (functionally the same as the value 1)
7	$V = 1.1115 * L_c^{0.45} - 0.1115$ $V = 4.0 * L_c$	for $1 >= L_c >= 0.0228$ for $0.0228 > L_c >= 0$	Society of Motion Picture and Television Engineers 240M (1999)
8	$V = L_c$	for $1 > L_c >= 0$	Linear transfer characteristics
9	$V = 1.0 + \text{Log10}(L_c) \div 2$ $V = 0.0$	for $1 >= L_c >= 0.01$ for $0.01 > L_c >= 0$	Logarithmic transfer characteristic (100:1 range)
10	$V = 1.0 + \text{Log10}(L_c) \div 2.5$ $V = 0.0$	for $1 >= L_c >= \text{Sqrt}(10) / 1000$ for $\text{Sqrt}(10) / 1000 > L_c >= 0$	Logarithmic transfer characteristic (100 * Sqrt(10) : 1 range)
11	$V = 1.099 * L_c^{0.45} - 0.099$ $V = 4.500 * L_c$ $V = -1.099 * (-L_c)^{0.45} + 0.099$	for $L_c >= 0.018$ for $0.018 > L_c > -0.018$ for $-0.018 >= L_c$	IEC 61966-2-4
12	$V = 1.099 * L_c^{0.45} - 0.099$ $V = 4.500 * L_c$ $V = -(1.099 * (-4 * L_c)^{0.45} - 0.099) \div 4$	for $1.33 > L_c >= 0.018$ for $0.018 > L_c >= -0.0045$ for $-0.0045 > L_c >= -0.25$	ITU-R Rec. BT.1361 extended colour gamut system
13..255	Reserved		For future use by ITU-T | ISO/IEC

matrix_coefficients describes the matrix coefficients used in deriving luma and chroma signals from the green, blue, and red primaries, as specified in Table E-5.

matrix_coefficients shall not be equal to 0 unless both of the following conditions are true:

– BitDepth_C is equal to BitDepth_Y,

– chroma_format_idc is equal to 3 (4:4:4).

The specification of the use of matrix_coefficients equal to 0 under all other conditions is reserved for future use by ITU-T | ISO/IEC.

matrix_coefficients shall not be equal to 8 unless one of the following conditions is true:

– $BitDepth_C$ is equal to $BitDepth_Y$,

– $BitDepth_C$ is equal to $BitDepth_Y + 1$ and chroma_format_idc is equal to 3 (4:4:4).

The specification of the use of matrix_coefficients equal to 8 under all other conditions is reserved for future use by ITU-T | ISO/IEC.

When the matrix_coefficients syntax element is not present, the value of matrix_coefficients shall be inferred to be equal to 2 (unspecified).

For the interpretation of matrix_coefficients, the following is specified:

1. The range of E'_R, E'_G, and E'_B is specified as follows:
   – If transfer_characteristics is not equal to 11 or 12, E'_R, E'_G, and E'_B are real numbers with values in the range of 0 to 1.
   – Otherwise (transfer_characteristics is equal to 11 (IEC 61966-2-4) or 12 (ITU-R BT.1361 extended colour gamut system)), E'_R, E'_G and E'_B are real numbers with a larger range not specified in this Recommendation.
2. Nominal white is specified as having E'_R equal to 1, E'_G equal to 1, and E'_B equal to 1.
3. Nominal black is specified as having E'_R equal to 0, E'_G equal to 0, and E'_B equal to 0.

The interpretation of matrix_coefficients is specified as follows:

– If video_full_range_flag is equal to 0, the following applies:
  – If matrix_coefficients is equal to 1, 4, 5, 6, or 7, the following equations apply:

$$Y = \mathrm{Clip1}_Y(\ \mathrm{Round}(\ (\ 1 \ll (\ BitDepth_Y - 8\ )\ ) * (\ 219 * E'_Y + 16\ )\ )\ ) \quad \text{(E-1)}$$

$$Cb = \mathrm{Clip1}_C(\ \mathrm{Round}(\ (\ 1 \ll (\ BitDepth_C - 8\ )\ ) * (\ 224 * E'_{PB} + 128\ )\ )\ ) \quad \text{(E-2)}$$

$$Cr = \mathrm{Clip1}_C(\ \mathrm{Round}(\ (\ 1 \ll (\ BitDepth_C - 8\ )\ ) * (\ 224 * E'_{PR} + 128\ )\ )\ ) \quad \text{(E-3)}$$

  – Otherwise, if matrix_coefficients is equal to 0 or 8, the following equations apply:

$$R = \mathrm{Clip1}_Y(\ (\ 1 \ll (\ BitDepth_Y - 8\ )\ ) * (\ 219 * E'_R + 16\ )\ ) \quad \text{(E-4)}$$

$$G = \mathrm{Clip1}_Y(\ (\ 1 \ll (\ BitDepth_Y - 8\ )\ ) * (\ 219 * E'_G + 16\ )\ ) \quad \text{(E-5)}$$

$$B = \mathrm{Clip1}_Y(\ (\ 1 \ll (\ BitDepth_Y - 8\ )\ ) * (\ 219 * E'_B + 16\ )\ ) \quad \text{(E-6)}$$

  – Otherwise, if matrix_coefficients is equal to 2, the interpretation of the matrix_coefficients syntax element is unknown or is determined by the application.
  – Otherwise (matrix_coefficients is not equal to 0, 1, 2, 4, 5, 6, 7, or 8), the interpretation of the matrix_coefficients syntax element is reserved for future definition by ITU-T | ISO/IEC.

– Otherwise (video_full_range_flag is equal to 1), the following applies:
  – If matrix_coefficients is equal to 1, 4, 5, 6, or 7, the following equations apply:

$$Y = \mathrm{Clip1}_Y(\ \mathrm{Round}(\ (\ (\ 1 \ll BitDepth_Y\ ) - 1\ ) * E'_Y\ )\ ) \quad \text{(E-7)}$$

$$Cb = \mathrm{Clip1}_C(\ \mathrm{Round}(\ (\ (\ 1 \ll BitDepth_C\ ) - 1\ )\ \ * E'_{PB} + (\ 1 \ll (\ BitDepth_C - 1\ )\ )\ )\ ) \quad \text{(E-8)}$$

$$Cr = \mathrm{Clip1}_C(\ \mathrm{Round}(\ (\ (\ 1 \ll BitDepth_C\ ) - 1\ )\ \ * E'_{PR} + (\ 1 \ll (\ BitDepth_C - 1\ )\ )\ )\ ) \quad \text{(E-9)}$$

  – Otherwise, if matrix_coefficients is equal to 0 or 8, the following equations apply:

$$R = \mathrm{Clip1}_Y(\ (\ (\ 1 \ll BitDepth_Y\ ) - 1\ ) * E'_R\ ) \quad \text{(E-10)}$$

$$G = \mathrm{Clip1}_Y(\ (\ (\ 1 \ll BitDepth_Y\ ) - 1\ ) * E'_G\ ) \quad \text{(E-11)}$$

$$B = \mathrm{Clip1}_Y(\ (\ (\ 1 \ll BitDepth_Y\ ) - 1\ ) * E'_B\ ) \quad \text{(E-12)}$$

– Otherwise, if matrix_coefficients is equal to 2, the interpretation of the matrix_coefficients syntax element is unknown or is determined by the application.

– Otherwise (matrix_coefficients is not equal to 0, 1, 2, 4, 5, 6, 7, or 8), the interpretation of the matrix_coefficients syntax element is reserved for future definition by ITU-T | ISO/IEC.

The variables E'_Y, E'_{PB}, and E'_{PR} (for matrix_coefficients not equal to 0 or 8) or Y, Cb, and Cr (for matrix_coefficients equal to 0 or 8) are specified as follows:

– If matrix_coefficients is not equal to 0 or 8, the following equations apply:

$$E'_Y = K_R * E'_R + (1 - K_R - K_B) * E'_G + K_B * E'_B \quad \text{(E-13)}$$

$$E'_{PB} = 0.5 * (E'_B - E'_Y) \div (1 - K_B) \quad \text{(E-14)}$$

$$E'_{PR} = 0.5 * (E'_R - E'_Y) \div (1 - K_R) \quad \text{(E-15)}$$

NOTE 3 – E'_Y is a real number with the value 0 associated with nominal black and the value 1 associated with nominal white. E'_{PB} and E'_{PR} are real numbers with the value 0 associated with both nominal black and nominal white. When transfer_characteristics is not equal to 11 or 12, E'_Y is a real number with values in the range of 0 to 1. When transfer_characteristics is not equal to 11 or 12, E'_{PB} and E'_{PR} are real numbers with values in the range of −0.5 to 0.5. When transfer_characteristics is equal to 11 (IEC 61966-2-4), or 12 (ITU-R BT.1361 extended colour gamut system), E'_Y, E'_{PB} and E'_{PR} are real numbers with a larger range not specified in this Recommendation.

– Otherwise, if matrix_coefficients is equal to 0, the following equations apply:

$$Y = \text{Round}(G) \quad \text{(E-16)}$$

$$Cb = \text{Round}(B) \quad \text{(E-17)}$$

$$Cr = \text{Round}(R) \quad \text{(E-18)}$$

– Otherwise (matrix_coefficients is equal to 8), the following applies:

– If $BitDepth_C$ is equal to $BitDepth_Y$, the following equations apply:

$$Y = \text{Round}(0.5 * G + 0.25 * (R + B)) \quad \text{(E-19)}$$

$$Cb = \text{Round}(0.5 * G - 0.25 * (R + B)) + (1 \ll (BitDepth_C - 1)) \quad \text{(E-20)}$$

$$Cr = \text{Round}(0.5 * (R - B)) + (1 \ll (BitDepth_C - 1)) \quad \text{(E-21)}$$

NOTE 4 – For purposes of the YCgCo nomenclature used in Table E-5, Cb and Cr of Equations E-20 and E-21 may be referred to as Cg and Co, respectively. The inverse conversion for the above three equations should be computed as:

$$t = Y - (Cb - (1 \ll (BitDepth_C - 1))) \quad \text{(E-22)}$$

$$G = \text{Clip1}_Y(Y + (Cb - (1 \ll (BitDepth_C - 1)))) \quad \text{(E-23)}$$

$$B = \text{Clip1}_Y(t - (Cr - (1 \ll (BitDepth_C - 1)))) \quad \text{(E-24)}$$

$$R = \text{Clip1}_Y(t + (Cr - (1 \ll (BitDepth_C - 1)))) \quad \text{(E-25)}$$

– Otherwise ($BitDepth_C$ is not equal to $BitDepth_Y$), the following equations apply:

$$Cr = \text{Round}(R) - \text{Round}(B) + (1 \ll (BitDepth_C - 1)) \quad \text{(E-26)}$$

$$t = \text{Round}(B) + ((Cr - (1 \ll (BitDepth_C - 1))) \gg 1) \quad \text{(E-27)}$$

$$Cb = \text{Round}(G) - t + (1 \ll (BitDepth_C - 1)) \quad \text{(E-28)}$$

$$Y = t + ((Cb - (1 \ll (BitDepth_C - 1))) \gg 1) \quad \text{(E-29)}$$

NOTE 5 – For purposes of the YCgCo nomenclature used in Table E-5, Cb and Cr of Equations E-28 and E-26 may be referred to as Cg and Co, respectively. The inverse conversion for the above four equations should be computed as.

$$t = Y - ((Cb - (1 \ll (BitDepth_C - 1))) \gg 1) \quad \text{(E-30)}$$

$$G = \text{Clip1}_Y(t + (Cb - (1 \ll (BitDepth_C - 1)))) \quad \text{(E-31)}$$

$$B = \text{Clip1}_Y(t - ((Cr - (1 \ll (BitDepth_C - 1))) \gg 1)) \quad \text{(E-32)}$$

$$R = \text{Clip1}_Y(B + (Cr - (1 \ll (BitDepth_C - 1)))) \quad \text{(E-33)}$$

Table E-5 – Matrix coefficients

Value	Matrix	Informative remark
0	GBR	Typically referred to as RGB; see Equations E-16 to E-18
1	$K_R = 0.2126$; $K_B = 0.0722$	ITU-R Rec. BT.709-5 ITU-R Rec. BT.1361 conventional colour gamut system and extended colour gamut system IEC 61966-2-4 $xvYCC_{709}$ Society of Motion Picture and Television Engineers RP 177 (1993) Annex B
2	Unspecified	Image characteristics are unknown or are determined by the application.
3	Reserved	For future use by ITU-T | ISO/IEC
4	$K_R = 0.30$; $K_B = 0.11$	United States Federal Communications Commission Title 47 Code of Federal Regulations (2003) 73.682 (a) (20)
5	$K_R = 0.299$; $K_B = 0.114$	ITU-R Rec. BT.470-6 System B, G (historical) ITU-R Rec. BT.601-6 625 ITU-R Rec. BT.1358 625 ITU-R Rec. BT.1700 625 PAL and 625 SECAM IEC 61966-2-4 $xvYCC_{601}$ (functionally the same as the value 6)
6	$K_R = 0.299$; $K_B = 0.114$	ITU-R Rec. BT.601-6 525 ITU-R Rec. BT.1358 525 ITU-R Rec. BT.1700 NTSC Society of Motion Picture and Television Engineers 170M (2004) (functionally the same as the value 5)
7	$K_R = 0.212$; $K_B = 0.087$	Society of Motion Picture and Television Engineers 240M (1999)
8	YCgCo	See Equations E-19 to E-33
9..255	Reserved	For future use by ITU-T | ISO/IEC

chroma_loc_info_present_flag equal to 1 specifies that chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field are present. chroma_loc_info_present_flag equal to 0 specifies that chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field are not present.

When chroma_format_idc is not equal to 1, chroma_loc_info_present_flag should be equal to 0.

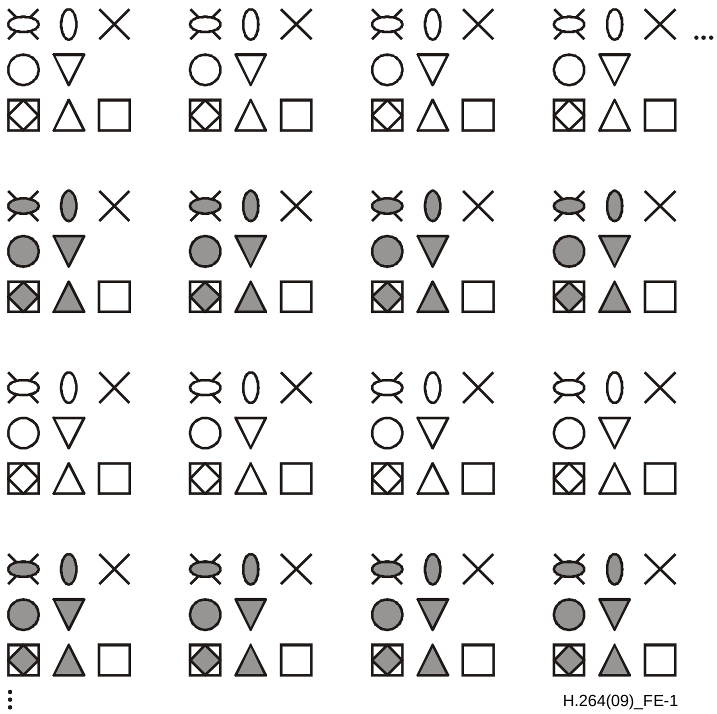
chroma_sample_loc_type_top_field and **chroma_sample_loc_type_bottom_field** specify the location of chroma samples as follows:

– If chroma_format_idc is equal to 1 (4:2:0 chroma format), chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field specify the location of chroma samples for the top field and the bottom field, respectively, as shown in Figure E-1.

– Otherwise (chroma_format_idc is not equal to 1), the values of the syntax elements chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field shall be ignored. When chroma_format_idc is equal to 2 (4:2:2 chroma format) or 3 (4:4:4 chroma format), the location of chroma samples is specified in subclause 6.2. When chroma_format_idc is equal to 0, there is no chroma sample array.

The value of chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field shall be in the range of 0 to 5, inclusive. When the chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field are not present, the values of chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field shall be inferred to be equal to 0.

NOTE 6 – When coding progressive source material, chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field should have the same value.



Interpretation of symbols

Luma sample position indications:

Luma sample top field

Luma sample bottom field

Chroma sample position indications, where grey fill indicates a bottom field sample type and no fill indicates a top field sample type:

Chroma sample type 2

Chroma sample type 3

Chroma sample type 0

Chroma sample type 1

Chroma sample type 4

Chroma sample type 5

Figure E-1 – Location of chroma samples for top and bottom fields for chroma_format_idc equal to 1 (4:2:0 chroma format) as a function of chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field

timing_info_present_flag equal to 1 specifies that num_units_in_tick, time_scale and fixed_frame_rate_flag are present in the bitstream. timing_info_present_flag equal to 0 specifies that num_units_in_tick, time_scale and fixed_frame_rate_flag are not present in the bitstream.

num_units_in_tick is the number of time units of a clock operating at the frequency time_scale Hz that corresponds to one increment (called a clock tick) of a clock tick counter. num_units_in_tick shall be greater than 0. A clock tick is the minimum interval of time that can be represented in the coded data. For example, when the frame rate of a video signal is 30 000 ÷ 1001 Hz, time_scale may be equal to 60 000 and num_units_in_tick may be equal to 1001. See Equation C-1.

time_scale is the number of time units that pass in one second. For example, a time coordinate system that measures time using a 27 MHz clock has a time_scale of 27 000 000. time_scale shall be greater than 0.

fixed_frame_rate_flag equal to 1 indicates that the temporal distance between the HRD output times of any two consecutive pictures in output order is constrained as follows. fixed_frame_rate_flag equal to 0 indicates that no such constraints apply to the temporal distance between the HRD output times of any two consecutive pictures in output order.

When fixed_frame_rate_flag is not present, it shall be inferred to be equal to 0.

For each picture n where n indicates the n-th picture (in output order) that is output and picture n is not the last picture in the bitstream (in output order) that is output, the value of $\Delta t_{fi,dpb}(\,n\,)$ is specified by

$$\Delta t_{fi,dpb}(\,n\,) = \Delta t_{o,dpb}(\,n\,) \div \text{DeltaTfiDivisor} \qquad \text{(E-34)}$$

where $\Delta t_{o,dpb}(n)$ is specified in Equation C-13 and DeltaTfiDivisor is specified by Table E-6 based on the value of pic_struct_present_flag, field_pic_flag, and pic_struct for the coded video sequence containing picture n. Entries marked "-" in Table E-6 indicate a lack of dependence of DeltaTfiDivisor on the corresponding syntax element.

When fixed_frame_rate_flag is equal to 1 for a coded video sequence containing picture n, the value computed for $\Delta t_{fi,dpb}(n)$ shall be equal to t_c as specified in Equation C-1 (using the value of t_c for the coded video sequence containing picture n) when either or both of the following conditions are true for the following picture n_n that is specified for use in Equation C-13:

– picture n_n is in the same coded video sequence as picture n.

– picture n_n is in a different coded video sequence and fixed_frame_rate_flag is equal to 1 in the coded video sequence containing picture n_n and the value of num_units_in_tick ÷ time_scale is the same for both coded video sequences.

Table E-6 – Divisor for computation of $\Delta t_{fi,dpb}(n)$

pic_struct_present_flag	field_pic_flag	pic_struct	DeltaTfiDivisor
0	1	-	1
1	-	1	1
1	-	2	1
0	0	-	2
1	-	0	2
1	-	3	2
1	-	4	2
1	-	5	3
1	-	6	3
1	-	7	4
1	-	8	6

NOTE 7 – In order to produce a DeltaTfiDivisor other than 2 for a picture with field_pic_flag equal to 0, pic_struct_present_flag must be equal to 1.

nal_hrd_parameters_present_flag equal to 1 specifies that NAL HRD parameters (pertaining to Type II bitstream conformance) are present. nal_hrd_parameters_present_flag equal to 0 specifies that NAL HRD parameters are not present.

NOTE 8 – When nal_hrd_parameters_present_flag is equal to 0, the conformance of the bitstream cannot be verified without provision of the NAL HRD parameters and all buffering period and picture timing SEI messages, by some means not specified in this Recommendation | International Standard.

When nal_hrd_parameters_present_flag is equal to 1, NAL HRD parameters (subclauses E.1.2 and E.2.2) immediately follow the flag.

The variable NalHrdBpPresentFlag is derived as follows:

– If any of the following is true, the value of NalHrdBpPresentFlag shall be set equal to 1:
  – nal_hrd_parameters_present_flag is present in the bitstream and is equal to 1,
  – the need for presence of buffering periods for NAL HRD operation to be present in the bitstream in buffering period SEI messages is determined by the application, by some means not specified in this Recommendation | International Standard.
– Otherwise, the value of NalHrdBpPresentFlag shall be set equal to 0.

vcl_hrd_parameters_present_flag equal to 1 specifies that VCL HRD parameters (pertaining to all bitstream conformance) are present. vcl_hrd_parameters_present_flag equal to 0 specifies that VCL HRD parameters are not present.

NOTE 9 – When vcl_hrd_parameters_present_flag is equal to 0, the conformance of the bitstream cannot be verified without provision of the VCL HRD parameters and all buffering period and picture timing SEI messages, by some means not specified in this Recommendation | International Standard.

When vcl_hrd_parameters_present_flag is equal to 1, VCL HRD parameters (subclauses E.1.2 and E.2.2) immediately follow the flag.

The variable VclHrdBpPresentFlag is derived as follows:

– If any of the following is true, the value of VclHrdBpPresentFlag shall be set equal to 1:
  – vcl_hrd_parameters_present_flag is present in the bitstream and is equal to 1,
  – the need for presence of buffering periods for VCL HRD operation to be present in the bitstream in buffering period SEI messages is determined by the application, by some means not specified in this Recommendation | International Standard.
– Otherwise, the value of VclHrdBpPresentFlag shall be set equal to 0.

The variable CpbDpbDelaysPresentFlag is derived as follows:

– If any of the following is true, the value of CpbDpbDelaysPresentFlag shall be set equal to 1:
  – nal_hrd_parameters_present_flag is present in the bitstream and is equal to 1,
  – vcl_hrd_parameters_present_flag is present in the bitstream and is equal to 1,
  – the need for presence of CPB and DPB output delays to be present in the bitstream in picture timing SEI messages is determined by the application, by some means not specified in this Recommendation | International Standard.
– Otherwise, the value of CpbDpbDelaysPresentFlag shall be set equal to 0.

low_delay_hrd_flag specifies the HRD operational mode as specified in Annex C. When fixed_frame_rate_flag is equal to 1, low_delay_hrd_flag shall be equal to 0. When low_delay_hrd_flag is not present, its value shall be inferred to be equal to 1 − fixed_frame_rate_flag.

NOTE 10 – When low_delay_hrd_flag is equal to 1, "big pictures" that violate the nominal CPB removal times due to the number of bits used by an access unit are permitted. It is expected, but not required, that such "big pictures" occur only occasionally.

pic_struct_present_flag equal to 1 specifies that picture timing SEI messages (subclause D.2.2) are present that include the pic_struct syntax element. pic_struct_present_flag equal to 0 specifies that the pic_struct syntax element is not present in picture timing SEI messages. When pic_struct_present_flag is not present, its value shall be inferred to be equal to 0.

bitstream_restriction_flag equal to 1, specifies that the following coded video sequence bitstream restriction parameters are present. bitstream_restriction_flag equal to 0, specifies that the following coded video sequence bitstream restriction parameters are not present.

motion_vectors_over_pic_boundaries_flag equal to 0 indicates that no sample outside the picture boundaries and no sample at a fractional sample position for which the sample value is derived using one or more samples outside the picture boundaries is used for inter prediction of any sample. motion_vectors_over_pic_boundaries_flag equal to 1 indicates that one or more samples outside picture boundaries may be used in inter prediction. When the motion_vectors_over_pic_boundaries_flag syntax element is not present, motion_vectors_over_pic_boundaries_flag value shall be inferred to be equal to 1.

max_bytes_per_pic_denom indicates a number of bytes not exceeded by the sum of the sizes of the VCL NAL units associated with any coded picture in the coded video sequence.

The number of bytes that represent a picture in the NAL unit stream is specified for this purpose as the total number of bytes of VCL NAL unit data (i.e., the total of the NumBytesInNALunit variables for the VCL NAL units) for the picture. The value of max_bytes_per_pic_denom shall be in the range of 0 to 16, inclusive.

Depending on max_bytes_per_pic_denom the following applies:

– If max_bytes_per_pic_denom is equal to 0, no limits are indicated.
– Otherwise (max_bytes_per_pic_denom is not equal to 0), it is a requirement of bitstream conformance that no coded picture shall be represented in the coded video sequence by more than the following number of bytes.

$$(\text{PicSizeInMbs} * \text{RawMbBits}) \div (8 * \text{max\_bytes\_per\_pic\_denom}) \quad \text{(E-35)}$$

When the max_bytes_per_pic_denom syntax element is not present, the value of max_bytes_per_pic_denom shall be inferred to be equal to 2.

max_bits_per_mb_denom indicates an upper bound for the number of coded bits of macroblock_layer( ) data for any macroblock in any picture of the coded video sequence. The value of max_bits_per_mb_denom shall be in the range of 0 to 16, inclusive.

Depending on max_bits_per_mb_denom the following applies:

– If max_bits_per_mb_denom is equal to 0, no limit is specified by this syntax element.

– Otherwise (max_bits_per_mb_denom is not equal to 0), it is a requirement of bitstream conformance that no coded macroblock_layer( ) shall be represented in the bitstream by more than the following number of bits.

$$(128 + \text{RawMbBits}) \div \text{max\_bits\_per\_mb\_denom} \quad \text{(E-36)}$$

Depending on entropy_coding_mode_flag, the bits of macroblock_layer( ) data are counted as follows:

– If entropy_coding_mode_flag is equal to 0, the number of bits of macroblock_layer( ) data is given by the number of bits in the macroblock_layer( ) syntax structure for a macroblock.

– Otherwise (entropy_coding_mode_flag is equal to 1), the number of bits of macroblock_layer( ) data for a macroblock is given by the number of times read_bits(1) is called in subclauses 9.3.3.2.2 and 9.3.3.2.3 when parsing the macroblock_layer( ) associated with the macroblock.

When the max_bits_per_mb_denom is not present, the value of max_bits_per_mb_denom shall be inferred to be equal to 1.

log2_max_mv_length_horizontal and **log2_max_mv_length_vertical** indicate the maximum absolute value of a decoded horizontal and vertical motion vector component, respectively, in ¼ luma sample units, for all pictures in the coded video sequence. A value of n asserts that no value of a motion vector component shall exceed the range from -2^n to $2^n - 1$, inclusive, in units of ¼ luma sample displacement. The value of log2_max_mv_length_horizontal shall be in the range of 0 to 16, inclusive. The value of log2_max_mv_length_vertical shall be in the range of 0 to 16, inclusive. When log2_max_mv_length_horizontal is not present, the values of log2_max_mv_length_horizontal and log2_max_mv_length_vertical shall be inferred to be equal to 16.

NOTE 11 – The maximum absolute value of a decoded vertical or horizontal motion vector component is also constrained by profile and level limits as specified in Annex A and subclauses G.10 and H.10.

max_num_reorder_frames indicates an upper bound for the number of frames buffers, in the decoded picture buffer (DPB), that are required for storing frames, complementary field pairs, and non-paired fields before output. It is a requirement of bitstream conformance that the maximum number of frames, complementary field pairs, or non-paired fields that precede any frame, complementary field pair, or non-paired field in the coded video sequence in decoding order and follow it in output order shall be less than or equal to max_num_reorder_frames. The value of max_num_reorder_frames shall be in the range of 0 to max_dec_frame_buffering, inclusive. When the max_num_reorder_frames syntax element is not present, the value of max_num_reorder_frames value shall be inferred as follows:

– If profile_idc is equal to 44, 86, 100, 110, 122, or 244 and constraint_set3_flag is equal to 1, the value of max_num_reorder_frames shall be inferred to be equal to 0.

– Otherwise (profile_idc is not equal to 44, 86, 100, 110, 122, or 244 or constraint_set3_flag is equal to 0), the value of max_num_reorder_frames shall be inferred to be equal to MaxDpbFrames.

max_dec_frame_buffering specifies the required size of the HRD decoded picture buffer (DPB) in units of frame buffers. It is a requirement of bitstream conformance that the coded video sequence shall not require a decoded picture buffer with size of more than Max(1, max_dec_frame_buffering) frame buffers to enable the output of decoded pictures at the output times specified by dpb_output_delay of the picture timing SEI messages. The value of max_dec_frame_buffering shall be greater than or equal to max_num_ref_frames. An upper bound for the value of max_dec_frame_buffering is specified by the level limits in subclauses A.3.1, A.3.2, G.10.2.1, and H.10.2.

When the max_dec_frame_buffering syntax element is not present, the value of max_dec_frame_buffering shall be inferred as follows:

– If profile_idc is equal to 44, 86, 100, 110, 122, or 244 and constraint_set3_flag is equal to 1, the value of max_dec_frame_buffering shall be inferred to be equal to 0.

– Otherwise (profile_idc is not equal to 44, 86, 100, 110, 122, or 244 or constraint_set3_flag is equal to 0), the value of max_dec_frame_buffering shall be inferred to be equal to MaxDpbFrames.

E.2.2 HRD parameters semantics

The syntax category of the HRD parameters syntax structure shall be inferred as follows:

– If the HRD parameters syntax structure is not part of an SEI message, the syntax category of the HRD parameters syntax structure is inferred to be equal to 0.

– Otherwise (the HRD parameters syntax structure is part of the base layer temporal HRD SEI message as specified in subclause G.13 or the base view temporal HRD SEI message as specified in subclause H.13), the syntax category of the HRD parameters syntax structure is inferred to be equal to 5.

cpb_cnt_minus1 plus 1 specifies the number of alternative CPB specifications in the bitstream. The value of cpb_cnt_minus1 shall be in the range of 0 to 31, inclusive. When low_delay_hrd_flag is equal to 1, cpb_cnt_minus1 shall be equal to 0. When cpb_cnt_minus1 is not present, it shall be inferred to be equal to 0.

bit_rate_scale (together with bit_rate_value_minus1[SchedSelIdx]) specifies the maximum input bit rate of the SchedSelIdx-th CPB.

cpb_size_scale (together with cpb_size_value_minus1[SchedSelIdx]) specifies the CPB size of the SchedSelIdx-th CPB.

bit_rate_value_minus1[SchedSelIdx **]** (together with bit_rate_scale) specifies the maximum input bit rate for the SchedSelIdx-th CPB. bit_rate_value_minus1[SchedSelIdx] shall be in the range of 0 to $2^{32} - 2$, inclusive. For any SchedSelIdx > 0, bit_rate_value_minus1[SchedSelIdx] shall be greater than bit_rate_value_minus1[SchedSelIdx − 1]. The bit rate in bits per second is given by

$$\text{BitRate[SchedSelIdx]} = (\text{ bit\_rate\_value\_minus1[SchedSelIdx]} + 1\ ) * 2^{(6 + \text{bit\_rate\_scale})} \quad \text{(E-37)}$$

When the bit_rate_value_minus1[SchedSelIdx] syntax element is not present, the value of BitRate[SchedSelIdx] shall be inferred as follows:

– If profile_idc is equal to 66, 77, or 88, BitRate[SchedSelIdx] shall be inferred to be equal to 1000 * MaxBR for VCL HRD parameters and to be equal to 1200 * MaxBR for NAL HRD parameters, where MaxBR is specified in subclause A.3.1.

– Otherwise, BitRate[SchedSelIdx] shall be inferred to be equal to cpbBrVclFactor * MaxBR for VCL HRD parameters and to be equal to cpbBrNalFactor * MaxBR for NAL HRD parameters, where MaxBR is specified in subclause A.3.1 and cpbBrVclFactor and cpbBrNalFactor are specified in subclause A.3.3 (for profiles specified in Annex A) or subclause G.10.2.2 (for profiles specified in Annex G) or subclause H.10.2 (for profiles specified in Annex H).

cpb_size_value_minus1[SchedSelIdx **]** is used together with cpb_size_scale to specify the SchedSelIdx-th CPB size. cpb_size_value_minus1[SchedSelIdx] shall be in the range of 0 to $2^{32} - 2$, inclusive. For any SchedSelIdx greater than 0, cpb_size_value_minus1[SchedSelIdx] shall be less than or equal to cpb_size_value_minus1[SchedSelIdx −1].

The CPB size in bits is given by

$$\text{CpbSize[SchedSelIdx]} = (\text{ cpb\_size\_value\_minus1[SchedSelIdx]} + 1\ ) * 2^{(4 + \text{cpb\_size\_scale})} \quad \text{(E-38)}$$

When the cpb_size_value_minus1[SchedSelIdx] syntax element is not present, the value of CpbSize[SchedSelIdx] shall be inferred as follows:

– If profile_idc is equal to 66, 77, or 88, CpbSize[SchedSelIdx] shall be inferred to be equal to 1000 * MaxCPB for VCL HRD parameters and to be equal to 1200 * MaxCPB for NAL HRD parameters, where MaxCPB is specified in subclause A.3.1.

– Otherwise, CpbSize[SchedSelIdx] shall be inferred to be equal to cpbBrVclFactor * MaxCPB for VCL HRD parameters and to be equal to cpbBrNalFactor * MaxCPB for NAL HRD parameters, where MaxCPB is specified in subclause A.3.1 and cpbBrVclFactor and cpbBrNalFactor are specified in subclause A.3.3 (for profiles specified in Annex A) or subclause G.10.2.2 (for profiles specified in Annex G) or subclause H.10.2 (for profiles specified in Annex H).

cbr_flag[SchedSelIdx **]** equal to 0 specifies that to decode this bitstream by the HRD using the SchedSelIdx-th CPB specification, the hypothetical stream delivery scheduler (HSS) operates in an intermittent bit rate mode. cbr_flag[SchedSelIdx] equal to 1 specifies that the HSS operates in a constant bit rate (CBR) mode. When the cbr_flag[SchedSelIdx] syntax element is not present, the value of cbr_flag shall be inferred to be equal to 0.

initial_cpb_removal_delay_length_minus1 specifies the length in bits of the initial_cpb_removal_delay[SchedSelIdx] and initial_cpb_removal_delay_offset[SchedSelIdx] syntax elements of the buffering period SEI message. The length of initial_cpb_removal_delay[SchedSelIdx] and of initial_cpb_removal_delay_offset[SchedSelIdx] is initial_cpb_removal_delay_length_minus1 + 1. When the initial_cpb_removal_delay_length_minus1 syntax element is present in more than one hrd_parameters( ) syntax structure within the VUI parameters syntax structure, the value of the initial_cpb_removal_delay_length_minus1 parameters shall be equal in both hrd_parameters( ) syntax structures. When the initial_cpb_removal_delay_length_minus1 syntax element is not present, it shall be inferred to be equal to 23.

cpb_removal_delay_length_minus1 specifies the length in bits of the cpb_removal_delay syntax element. The length of the cpb_removal_delay syntax element of the picture timing SEI message is cpb_removal_delay_length_minus1 + 1. When the cpb_removal_delay_length_minus1 syntax element is present in more than one hrd_parameters( ) syntax structure within the VUI parameters syntax structure, the value of the cpb_removal_delay_length_minus1 parameters shall be equal in both hrd_parameters( ) syntax structures. When the cpb_removal_delay_length_minus1 syntax element is not present, it shall be inferred to be equal to 23.

dpb_output_delay_length_minus1 specifies the length in bits of the dpb_output_delay syntax element. The length of the dpb_output_delay syntax element of the picture timing SEI message is dpb_output_delay_length_minus1 + 1. When the dpb_output_delay_length_minus1 syntax element is present in more than one hrd_parameters( ) syntax structure within the VUI parameters syntax structure, the value of the dpb_output_delay_length_minus1 parameters shall be equal in both hrd_parameters( ) syntax structures. When the dpb_output_delay_length_minus1 syntax element is not present, it shall be inferred to be equal to 23.

time_offset_length greater than 0 specifies the length in bits of the time_offset syntax element. time_offset_length equal to 0 specifies that the time_offset syntax element is not present. When the time_offset_length syntax element is present in more than one hrd_parameters( ) syntax structure within the VUI parameters syntax structure, the value of the time_offset_length parameters shall be equal in both hrd_parameters( ) syntax structures. When the time_offset_length syntax element is not present, it shall be inferred to be equal to 24.

Annex G

Scalable video coding

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies scalable video coding, referred to as SVC.

G.1 Scope

Bitstreams and decoders conforming to one or more of the profiles specified in this annex are completely specified in this annex with reference made to clauses 2-9 and Annexes A-E.

G.2 Normative references

The specifications in clause 2 apply with the following additions.

– ISO/IEC 10646:2003, *Information technology − Universal Multiple-Octet Coded Character Set (UCS)*.

– IETF RFC 3986 (2005), *Uniform Resource Identifiers (URI): Generic Syntax.*

G.3 Definitions

For the purpose of this annex, the following definitions apply in addition to the definitions in clause 3. These definitions are either not present in clause 3 or replace definitions in clause 3.

G.3.1 **arbitrary slice order (ASO)**: A *decoding order* of *slices* in which the *macroblock address* of the first *macroblock* of some *slice* of a *slice group* within a *layer representation* may be less than the *macroblock address* of the first *macroblock* of some other preceding *slice* of the same *slice group* within the same *layer representation* or in which the *slices* of a *slice group* within a *layer representation* may be interleaved with the *slices* of one or more other *slices groups* within the same *layer representation*.

G.3.2 **associated NAL unit**: A *NAL unit* that directly succeeds a *prefix NAL unit* in *decoding order*.

G.3.3 **B slice**: A *slice* that may be decoded using *intra-layer intra prediction* or *inter prediction* using at most two *motion vectors* and *reference indices* to *predict* the sample values of each *block*.

G.3.4 **base layer**: A *bitstream subset* that contains all *NAL units* with the nal_unit_type *syntax element* equal to 1 and 5 of the *bitstream* and does not contain any *NAL unit* with the nal_unit_type *syntax element* equal to 14, 15, or 20 and conforms to one or more of the profiles specified in Annex A.

G.3.5 **base quality layer representation**: The *layer representation* of the *target dependency representation* of an *access unit* that is associated with the quality_id *syntax element* equal to 0.

G.3.6 **bitstream subset**: A *bitstream* that is derived as a *subset* from a *bitstream* by discarding zero or more *NAL units*. A *bitstream subset* is also referred to as *sub-bitstream*.

G.3.7 **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* or *layer bottom field* of the *frame* or *layer frame*, respectively, that lie within the spatial region of the *macroblock pair.* For a *frame macroblock pair*, the bottom macroblock represents the samples of the *frame* or *layer frame* that lie within the bottom half of the spatial region of the *macroblock pair.*

G.3.8 **coded slice in scalable extension NAL unit**: A *coded slice NAL unit* that contains an *EI slice*, *EP slice*, or an *EB slice*.

G.3.9 **complementary reference field pair**: A collective term for two *reference fields* that are in consecutive *access units* in *decoding order* as two *coded fields*, where the *target dependency representations* of the *fields* share the same value of the frame_num *syntax element* and where the second *field* in *decoding order* is not an *IDR picture* and the *target dependency representation* of the second *field* does not include a memory_management_control_operation *syntax element* equal to 5, or a *complementary reference base field pair*.

G.3.10 **complementary reference base field pair**: Two *reference base fields* that are associated with two *coded fields* that are in consecutive *access units* in *decoding order*, where the *target dependency representations* of the *coded fields* share the same value of the frame_num *syntax element* and where the second *coded field* in *decoding order* is not an *IDR picture* and the *target dependency representation* of the second *coded field* does not include a memory_management_control_operation *syntax element* equal to 5. A *complementary reference base field pair* is a *complementary reference field pair*.

G.3.11 **dependency representation**: A *subset* of *VCL NAL units* within an *access unit* that are associated with the same value of the dependency_id *syntax element*, which is provided as part of the *NAL unit* header or by an associated *prefix NAL unit,* and the same value of the redundant_pic_cnt *syntax element*. A *dependency representation* consists of one or more *layer representations*.

G.3.12 **EB slice**: A *slice* that may be decoded using *intra prediction* or *inter prediction* or *inter-layer prediction* from *syntax elements* and derived variables of the *reference layer representation*. For *inter-prediction* of EB slices at most two *motion vectors* and *reference indices* are used to *predict* the sample values of each *block*.

G.3.13 **EI slice**: A *slice* that is not an *I slice* or *SI slice* that is decoded using *intra prediction* only.

G.3.14 **EP slice**: A *slice* that may be decoded using *intra prediction* or *inter prediction* or *inter-layer prediction* from *syntax elements* and derived variables of the *reference layer representation*. For *inter-prediction* of EP slices at most one *motion vector* and *reference index* is used to *predict* the sample values of each *block.*

G.3.15 **field macroblock**: A *macroblock* containing samples from a single *field* or *layer field*.

G.3.16 **frame macroblock**: A *macroblock* containing samples from the two *fields* or *layer fields* of a *frame* or *layer frame*, respectively.

G.3.17 **I slice**: A *slice* that is decoded using *intra-layer intra prediction* only.

G.3.18 **instantaneous decoding refresh (IDR) picture**: A *coded picture* for which the variable IdrPicFlag is equal to 1 for the *target dependency representation*. An IDR picture causes the *decoding process* to mark all *reference pictures* as "unused for reference" immediately after the decoding of the IDR picture. All *coded pictures* that follow an IDR picture in *decoding order* can be decoded without *inter prediction* from any *picture* that precedes the IDR picture in *decoding order*. The first *picture* of each *coded video sequence* in *decoding order* is an IDR picture.

G.3.19 **inter-layer intra prediction**: An *inter-layer prediction* derived from decoded samples of *intra-coded macroblocks* of the *reference layer representation*.

G.3.20 **inter-layer prediction**: A *prediction* derived from *syntax elements*, derived variables, or decoded samples of the *reference layer representation*.

G.3.21 **intra-layer intra prediction**: A *prediction* derived from decoded samples of the same decoded *slice*.

G.3.22 **intra prediction**: A collective term for *intra-layer intra prediction* or *inter-layer intra prediction* or a combination of *intra-layer intra prediction* together with *inter-layer prediction* from *syntax elements* and derived variables of the *reference layer representation*.

G.3.23 **intra slice**: A collective term for *I slice* or *EI slice*.

G.3.24 **layer bottom field**: One of two *layer fields* that comprise a *layer frame*. Each row of a layer bottom field is spatially located immediately below a corresponding row of a *layer top field*.

G.3.25 **layer field**: An assembly of alternate rows of a *layer frame*. A *layer frame* is composed of two *layer fields*, a *layer top field* and a *layer bottom field*.

G.3.26 **layer frame**: A *layer frame* contains an array of *luma* samples that represents an intermediate decoding result for a *field* or a *frame* in monochrome format or an array of *luma* samples and two corresponding arrays of *chroma* samples that represent an intermediate decoding result for a *field* or a *frame* in 4:2:0, 4:2:2, and 4:4:4 colour format. A *layer frame* consists of two *layer fields*, a *layer top field* and a *layer bottom field*.

G.3.27 **layer picture**: A collective term for a *layer field* or a *layer frame*.

G.3.28 **layer top field**: One of two *layer fields* that comprise a *layer frame*. Each row of a *layer top field* is spatially located immediately above a corresponding row of a *layer bottom field*.

G.3.29 **layer representation**: A subset of *VCL NAL units* within an *access unit* that are associated with the same values of the dependency_id and quality_id *syntax elements*, which are provided as part of the *VCL NAL unit* header or by an associated *prefix NAL unit*, and the same value of the redundant_pic_cnt *syntax element*. One or more *layer representations* represent a *dependency representation*.

G.3.30 layer representation identifier: An integer value by which a particular *layer representation* inside a *coded picture* is uniquely identified.

G.3.31 macroblock: A 16x16 *block* of *luma* samples and two corresponding *blocks* of *chroma* samples of a *picture* or *layer picture* that has three sample arrays, or a 16x16 *block* of samples of a monochrome *picture* or *layer picture*. The division of a *slice* or a *macroblock pair* into macroblocks is a *partitioning*.

G.3.32 macroblock-adaptive frame/field decoding: A *decoding process* for *coded frames* or *layer representations* in which some *macroblocks* may be decoded as *frame macroblocks* and others may be decoded as *field macroblocks.*

G.3.33 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* or *layer picture* starting with zero for the top-left *macroblock* in a *picture* or *layer 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* or *layer 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.

G.3.34 macroblock location: The two-dimensional coordinates of a *macroblock* in a *picture* or *layer picture* denoted by (x, y). For the top left *macroblock* of the *picture* or *layer 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*.

G.3.35 macroblock pair: A pair of vertically contiguous *macroblocks* in a *frame* or *layer frame* that is coupled for use in *macroblock-adaptive frame/field decoding*. The division of a *slice* into macroblock pairs is a *partitioning*.

G.3.36 macroblock to slice group map: A means of mapping *macroblocks* of a *picture* or *layer 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.

G.3.37 map unit to slice group map: A means of mapping *slice group map units* of a *picture* or *layer 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 to.

G.3.38 non-paired reference base field: A *reference base field* that is not part of a *complementary reference base field pair*. A non-paired reference base field is a *non-paired reference field*.

G.3.39 P slice: A *slice* that may be decoded using *intra-layer intra prediction* or *inter prediction* using at most one *motion vector* and *reference index* to *predict* the sample values of each *block*.

G.3.40 parameter: A *syntax element* of an *SVC sequence parameter set* or a *picture parameter set*. Parameter is also used as part of the defined term *quantisation parameter*.

G.3.41 picture parameter set: A *syntax structure* containing *syntax elements* that apply to zero or more *layer representations* as determined by the pic_parameter_set_id *syntax element* found in each *slice header.*

G.3.42 prefix NAL unit: A *NAL unit* with nal_unit_type equal to 14 that immediately precedes in *decoding order* a *NAL unit* with nal_unit_type equal to 1 or 5. The *NAL unit* that immediately succeeds the prefix NAL unit in *decoding order* is referred to as the *associated NAL unit*. The prefix NAL unit contains data associated with the *associated NAL unit*, which are considered to be part of the *associated NAL unit*.

G.3.43 reference base field: A *reference field* that is obtained by decoding a *base quality layer representation* with the nal_ref_idc *syntax element* not equal to 0, the store_ref_base_pic_flag *syntax element* equal to 1, and the field_pic_flag *syntax element* equal to 1 of a *coded picture* and all *layer representations* of the *coded picture* that are referred to by *inter-layer prediction* in the *base quality layer representation*. A reference base field is not a *decoded picture* and it is not an output of the *decoding process*, but may be used for *inter prediction* when *P*, *B*, *EP*, and *EB slices* of a *coded field* or a *field macroblock* of a *coded frame* are decoded. See also *reference base picture*.

G.3.44 reference base frame: A *reference frame* that is obtained by decoding a *base quality layer representation* with the nal_ref_idc *syntax element* not equal to 0, the store_ref_base_pic_flag *syntax element* equal to 1, and the field_pic_flag *syntax element* equal to 0 of a *coded picture* and all *layer representations* of the *coded picture* that are referred to by *inter-layer prediction* of the *base quality layer representation*. A reference base frame is not a *decoded picture* and it is not an output of the *decoding process*, but may be used for *inter prediction* when *P*, *B*, *EP*, and *EB slices* of a *coded frame* are decoded. See also *reference base picture*.

G.3.45 reference base picture: A collective term for a *reference base field* or a *reference base frame*.

G.3.46 reference field: A *reference field* may be used for *inter prediction* when *P*, *B, EP,* or *EB slices* of a *coded field* or *field macroblocks* of a *coded frame* are decoded. See also *reference picture*.

G.3.47 reference frame: A *reference frame* may be used for *inter prediction* when *P*, *B, EP,* or *EB slices* of a *coded frame* are decoded. See also *reference picture*.

G.3.48 reference layer macroblock: A *macroblock* of a *reference layer representation*.

G.3.49 reference layer representation: A reference layer representation for a particular *layer representation* of a *coded picture* is the *layer representation* that is used for *inter-layer prediction* of the particular *layer representation*. The reference layer representation belongs to the same *access unit* as the *layer representation* that uses the reference layer representation for *inter-layer prediction*.

G.3.50 reference picture: A collective term for a *decoded picture* that is obtained by decoding a *coded picture* for which the nal_ref_idc *syntax element* that is associated with the *target dependency representation* is not equal to 0 or a *reference base picture*. A reference picture contains samples that may be used for *inter prediction* in the *decoding process* of subsequent *pictures* in *decoding order*.

G.3.51 reference picture list: A list of *reference pictures* that is used for *inter prediction* of a *P, B, EP,* or *EB slice.* For the *decoding process* of a *P* or *EP slice,* there is one reference picture list. For the *decoding process* of a *B* or *EB slice*, there are two reference picture lists.

G.3.52 reference picture list 0: A *reference picture list* used for *inter prediction* of a *P*, *B*, *EP,* or *EB slice*. All *inter prediction* used for *P* and *EP slices* uses reference picture list 0. Reference picture list 0 is one of two *reference picture lists* used for *inter prediction* for a *B* or *EB slice*, with the other being *reference picture list 1*.

G.3.53 reference picture list 1: A *reference picture list* used for *inter prediction* of a *B* or *EB slice*. Reference picture list 1 is one of two *reference picture lists* used for *inter prediction* for a *B* or *EB slice*, with the other being *reference picture list 0*.

G.3.54 scalable bitstream: A *bitstream* with the property that one or more *bitstream subsets* that are not identical to the scalable bitstream form another *bitstream* that conforms to this specification.

G.3.55 sequence parameter set: A *syntax structure* containing *syntax elements* that apply to zero or more *layer representations* with the dependency_id *syntax element* equal to 0 and the quality_id *syntax element* equal to 0 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* of *I*, *P*, and *B slices.*

G.3.56 slice: An integer number of *macroblocks* or *macroblock pairs* ordered consecutively in the *raster scan* within a particular *slice group*. Each *macroblock* or *macroblock pair* of a *picture* or *layer picture* shall not be contained in more than one slice of a particular *layer representation*. 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* or *layer picture*. The *macroblock addresses* are derived from the first *macroblock address* in a slice (as represented in the *slice header*) and the *macroblock to slice group map.*

G.3.57 slice group: A subset of the *macroblocks* or *macroblock pairs* of a *picture* or *layer picture*. The division of the *picture* or *layer picture* into slice groups is a *partitioning* of the *picture* or *layer picture.* The partitioning is specified by the *macroblock to slice group map*.

G.3.58 spatial intra prediction: See *intra-layer intra prediction*.

G.3.59 sub-bitstream: A *subset* of a *bitstream*. A sub-bitstream is also referred to as *bitstream subset*.

G.3.60 subset: A subset contains only elements that are also contained in the set from which the subset is derived. The subset may be identical to the set from which it is derived.

G.3.61 subset sequence parameter set: A *syntax structure* containing *syntax elements* that apply to zero or more *layer representations* with the dependency_id *syntax element* not equal to 0 or the quality_id *syntax element* not equal to 0 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* of *EI*, *EP*, and *EB slices.*

G.3.62 SVC sequence parameter set: A collective term for *sequence parameter set* or *subset sequence parameter set*.

G.3.63 SVC sequence parameter set RBSP: A collective term for sequence parameter set RBSP or subset sequence parameter set RBSP.

G.3.64 target dependency representation: The *dependency representation* of a *coded picture* that is associated with the largest value of the dependency_id *syntax element* for all *dependency representations* of the *coded picture*.

G.3.65 target layer representation: The *layer representation* of the *target dependency representation* of a *coded picture* that is associated with the largest value of the quality_id *syntax element* for all *layer representations* of the *target dependency representation* of the *coded picture*.

G.3.66 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* or *layer top field* of the *frame* or *layer 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* or *layer frame* that lie within the top half of the spatial region of the *macroblock pair.*

G.3.67 VCL NAL unit: A collective term for *coded slice NAL units* and *prefix NAL units*.

G.4 Abbreviations

The specifications in clause 4 apply.

G.5 Conventions

The specifications in clause 5 apply.

G.6 Source, coded, decoded and output data formats, scanning processes, neighbouring and reference layer relationships

The specifications in clause 6 apply with substituting SVC sequence parameter set for sequence parameter set. The specification in subclause 6.3 also applies to layer pictures. Additionally, the following processes are specified.

G.6.1 Derivation process for reference layer macroblocks

This process is only invoked when no_inter_layer_pred_flag is equal to 0.

Inputs to this process are:

– a luma location (xP, yP) relative to the upper-left luma sample of the current macroblock,

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying macroblock types for the macroblocks of the reference layer representation.

Outputs of this process are:

– the macroblock address mbAddrRefLayer specifying the reference layer macroblock,

– a luma location (xB, yB) relative to the upper-left luma sample of the reference layer macroblock.

Let currDQId be the current value of DQId and let levelIdc be the value of level_idc in the SVC sequence parameter set that is referred to in coded slice NAL units with DQId equal to ((currDQId >> 4) << 4).

The variables shiftX, shiftY, scaleX, scaleY, offsetX, and offsetY are derived as specified in the following ordered steps:

1. The variables refW, refH, scaledW, scaledH, offsetX, and offsetY are derived by

$$\mathrm{refW} = \mathrm{RefLayerPicWidthInSamples}_L \quad \text{(G-1)}$$
$$\mathrm{refH} = \mathrm{RefLayerPicHeightInSamples}_L \quad \text{(G-2)}$$
$$\mathrm{scaledW} = \mathrm{ScaledRefLayerPicWidthInSamples}_L \quad \text{(G-3)}$$
$$\mathrm{scaledH} = \mathrm{ScaledRefLayerPicHeightInSamples}_L \quad \text{(G-4)}$$
$$\mathrm{offsetX} = \mathrm{ScaledRefLayerLeftOffset} \quad \text{(G-5)}$$
$$\mathrm{offsetY} = \mathrm{ScaledRefLayerTopOffset} / (1 + \mathrm{field\_pic\_flag}) \quad \text{(G-6)}$$

2. The variables shiftX and shiftY are derived by

$$\text{shiftX} = ((\text{levelIdc} <= 30) \ ? \ 16 \ : \ (31 - \text{Ceil}(\text{Log2}(\text{refW})))) \quad \text{(G-7)}$$
$$\text{shiftY} = ((\text{levelIdc} <= 30) \ ? \ 16 \ : \ (31 - \text{Ceil}(\text{Log2}(\text{refH})))) \quad \text{(G-8)}$$

3. The variables scaleX and scaleY are derived by

$$\text{scaleX} = ((\text{refW} << \text{shiftX}) + (\text{scaledW} >> 1)) / \text{scaledW} \quad \text{(G-9)}$$
$$\text{scaleY} = ((\text{refH} << \text{shiftY}) + (\text{scaledH} >> 1)) / \text{scaledH} \quad \text{(G-10)}$$

NOTE 1 – The variables shiftX, shiftY, scaleX, scaleY, offsetX, and offsetY do not depend on the luma location (xP, yP), the variable fieldMbFlag, or the current macroblock.

The reference layer luma location (xRef, yRef) relative to the upper-left sample of the reference layer picture is derived as specified by the following ordered steps:

1. The inverse macroblock scanning process as specified in subclause 6.4.1 is invoked with CurrMbAddr as the input and the output is assigned to (xM, yM). For this invocation of the process in subclause 6.4.1, the current macroblock is treated as field macroblock when fieldMbFlag is equal to 1, and it is treated as frame macroblock when fieldMbFlag is equal to 0.

2. The luma location (xC, yC) is derived by

$$\text{xC} = \text{xM} + \text{xP} \quad \text{(G-11)}$$
$$\text{yC} = \text{yM} + \text{yP} * (1 + \text{fieldMbFlag} - \text{field\_pic\_flag}) \quad \text{(G-12)}$$

3. The reference layer luma location is derived by

$$\text{xRef} = ((\text{xC} - \text{offsetX}) * \text{scaleX} + (1 << (\text{shiftX} - 1))) >> \text{shiftX} \quad \text{(G-13)}$$
$$\text{yRef} = ((\text{yC} - \text{offsetY}) * \text{scaleY} + (1 << (\text{shiftY} - 1))) >> \text{shiftY} \quad \text{(G-14)}$$

4. The reference layer luma location is modified by

$$\text{xRef} = \text{Min}(\text{RefLayerPicWidthInSamples}_L - 1, \text{xRef}) \quad \text{(G-15)}$$
$$\text{yRef} = \text{Min}(\text{RefLayerPicHeightInSamples}_L - 1, \text{yRef}) \quad \text{(G-16)}$$

The reference layer macroblock address mbAddrRefLayer and a luma location (xB, yB) relative to the upper-left sample of the reference layer macroblock mbAddrRefLayer are derived as follows:

– If MbaffFrameFlag is equal to 0 and RefLayerMbaffFrameFlag is equal to 0, the following ordered steps are specified:

1. The reference layer macroblock address mbAddrRefLayer is derived by

$$\text{mbAddrRefLayer} = (\text{yRef} / 16) * \text{RefLayerPicWidthInMbs} + (\text{xRef} / 16) \quad \text{(G-17)}$$

2. The luma location (xB, yB) is derived as follows:

   – If mbAddrRefLayer is not available, (xB, yB) is marked as not available.

   – Otherwise (mbAddrRefLayer is available), (xB, yB) is set equal to (xRef % 16, yRef % 16).

– Otherwise (MbaffFrameFlag is equal to 1 or RefLayerMbaffFrameFlag is equal to 1), the following ordered steps are specified:

NOTE 2 – When MbaffFrameFlag is equal to 1 or RefLayerMbaffFrameFlag is equal to 1, field_pic_flag and RefLayerFieldPicFlag are both equal to 0 (see subclause G.7.4.3.4).

1. A virtual reference layer macroblock address virtMbAddrRefLayer is derived as follows:

   – If RefLayerMbaffFrameFlag is equal to 1, virtMbAddrRefLayer is derived by

$$\begin{aligned}\text{virtMbAddrRefLayer} = 2 * ((\text{yRef} / 32) * \text{RefLayerPicWidthInMbs} + (\text{xRef} / 16)) +\\ (\text{yRef} \% 32) / 16\end{aligned} \quad \text{(G-18)}$$

   – Otherwise (RefLayerMbaffFrameFlag is equal to 0), virtMbAddrRefLayer is derived by

$$\text{virtMbAddrRefLayer} = (\text{yRef} / 16) * \text{RefLayerPicWidthInMbs} + (\text{xRef} / 16) \quad \text{(G-19)}$$

2. The reference layer macroblock address mbAddrRefLayer and the luma location (xB, yB) are derived as follows:

   – If fieldMbFlag is equal to 0 and refLayerFieldMbRef[virtMbAddrRefLayer] is equal to 1, the field-to-frame reference layer macroblock conversion process as specified in subclause G.6.1.1 is invoked with virtMbAddrRefLayer, (xRef, yRef), and refLayerMbType as the inputs and the outputs are assigned to mbAddrRefLayer and (xB, yB).

   – Otherwise, if fieldMbFlag is equal to 1 and refLayerFieldMbRef[virtMbAddrRefLayer] is equal to 0, the frame-to-field reference layer macroblock conversion process as specified in subclause G.6.1.2 is invoked with virtMbAddrRefLayer and (xRef, yRef) as the inputs and the outputs are assigned to mbAddrRefLayer and (xB, yB).

   – Otherwise (fieldMbFlag is equal to refLayerFieldMbRef[virtMbAddrRefLayer]), mbAddrRefLayer and (xB, yB) are derived by

$$\text{mbAddrRefLayer} = ((\text{virtMbAddrRefLayer} >> \text{fieldMbFlag}) << \text{fieldMbFlag}) + (\text{CurrMbAddr} \% 2) * \text{fieldMbFlag} \quad \text{(G-20)}$$
$$\text{xB} = (\text{xRef} \% 16) \quad \text{(G-21)}$$
$$\text{yB} = (\text{yRef} \% (16 << \text{fieldMbFlag})) >> \text{fieldMbFlag} \quad \text{(G-22)}$$

G.6.1.1 Field-to-frame reference layer macroblock conversion process

Inputs to this process are:

– a virtual reference layer macroblock address virtMbAddrRefLayer,

– a reference layer luma location (xRef, yRef) relative to the upper-left luma sample of the reference layer picture,

– a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying macroblock types for the macroblocks of the reference layer representation.

Outputs of this process are:

– the macroblock address mbAddrRefLayer of the reference layer macroblock,

– a luma location (xB, yB) relative to the upper-left luma sample of the reference layer macroblock.

The macroblock addresses mbAddrRefLayerTop and mbAddrRefLayerBot are derived by

$$\text{mbAddrRefLayerTop} = \text{virtMbAddrRefLayer} - (\text{virtMbAddrRefLayer} \% 2) \quad \text{(G-23)}$$
$$\text{mbAddrRefLayerBot} = \text{mbAddrRefLayerTop} + 1 \quad \text{(G-24)}$$

The reference layer macroblock address mbAddrRefLayer is derived as follows:

– If refLayerMbType[mbAddrRefLayerTop] is equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL, mbAddrRefLayer is set equal to mbAddrRefLayerBot.

– Otherwise (refLayerMbType[mbAddrRefLayerTop] is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL), mbAddrRefLayer is set equal to mbAddrRefLayerTop.

The luma location (xB, yB) is derived by

$$\text{xB} = \text{xRef} \% 16 \quad \text{(G-25)}$$
$$\text{yB} = 8 * ((\text{yRef} / 16) \% 2) + 4 * ((\text{yRef} \% 16) / 8) \quad \text{(G-26)}$$

G.6.1.2 Frame-to-field reference layer macroblock conversion process

Inputs to this process are:

– a virtual reference layer macroblock address virtMbAddrRefLayer,

– a virtual reference layer luma location (xRef, yRef) relative to the upper-left luma sample of the reference layer picture.

Outputs of this process are:

– the macroblock address mbAddrRefLayer of the reference layer macroblock,

– a luma location (xB, yB) relative to the upper-left luma sample of the reference layer macroblock.

The reference layer macroblock address mbAddrRefLayer and the luma location (xB, yB) are derived by

$$\text{mbAddrRefLayer} = \text{virtMbAddrRefLayer} \quad \text{(G-27)}$$
$$\text{xB} = \text{xRef} \,\%\, 16 \quad \text{(G-28)}$$
$$\text{yB} = \text{yRef} \,\%\, 16 \quad \text{(G-29)}$$

G.6.2 Derivation process for reference layer partitions

This process is only invoked when no_inter_layer_pred_flag is equal to 0.

Inputs to this process are:

- a luma location (xP, yP) relative to the upper-left luma sample of the current macroblock,
- a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,
- a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,
- a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying macroblock types for the macroblocks of the reference layer representation,
- a (RefLayerPicSizeInMbs)x4 array refLayerSubMbType specifying the sub-macroblock types for the macroblocks of the reference layer representation.

Outputs of this process are:

- the macroblock address mbAddrRefLayer specifying the reference layer macroblock,
- the macroblock partition index mbPartIdxRefLayer specifying the reference layer macroblock partition inside the reference layer macroblock mbAddrRefLayer,
- the sub-macroblock partition index subMbPartIdxRefLayer specifying the reference layer sub-macroblock partition inside the macroblock partition mbPartIdxRefLayer of the reference layer macroblock mbAddrRefLayer.

The derivation process for reference layer macroblocks as specified in subclause G.6.1 is invoked with the luma location (xP, yP), fieldMbFlag, refLayerFieldMbFlag, and refLayerMbType as the input and the outputs are assigned to mbAddrRefLayer and (xB, yB).

The SVC derivation process for macroblock and sub-macroblock partition indices as specified in subclause G.6.4 is invoked with currDQId set equal to ref_layer_dq_id, the luma location (xB, yB), the macroblock type refLayerMbType[mbAddrRefLayer], and, when refLayerMbType[mbAddrRefLayer] is equal to P_8x8, P_8x8ref0, or B_8x8, the list of sub-macroblock types refLayerSubMbType[mbAddrRefLayer] as the inputs and the outputs are the reference layer macroblock partition index mbPartIdxRefLayer and the reference layer sub-macroblock partition index subMbPartIdxRefLayer.

G.6.3 Derivation process for reference layer sample locations in resampling

Inputs to this process are:

- a variable chromaFlag specifying whether the luma or a chroma component is subject to the resampling process,
- a sample location (xP, yP) relative to the upper-left sample of the current macroblock,
- a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,
- a variable botFieldFlag specifying whether a top or a bottom field is subject to the resampling process (when RefLayerFrameMbsOnlyFlag is equal to 0 or frame_mbs_only_flag is equal to 0).

Output of this process is a reference layer sample location (xRef16, yRef16), which specifies the following:

- If RefLayerFrameMbsOnlyFlag is equal to 1 or RefLayerFieldPicFlag is equal to 1, (xRef16, yRef16) specifies the reference layer sample location in units of 1/16-th sample relative to the upper-left sample of the reference layer picture.
- Otherwise (RefLayerFrameMbsOnlyFlag is equal to 0 and RefLayerFieldPicFlag is equal to 0), (xRef16, yRef16) specifies the reference layer sample location in units of 1/16-th field sample relative to the upper-left sample of the field specified by botFieldFlag of the reference layer picture.

Let currDQId be the current value of DQId and let levelIdc be the value of level_idc in the SVC sequence parameter set that is referred to in coded slice NAL units with DQId equal to ((currDQId >> 4) << 4).

The variables subW, subH, shiftX, shiftY, scaleX, scaleY, offsetX, offsetY, addX, addY, deltaX, and deltaY are derived as specified in the following ordered steps:

1. With Z being replaced by L for chromaFlag equal to 0 and C for chromaFlag equal to 1, the variables refW, refH, scaledW, and scaledH are derived by

$$\text{refW} = \text{RefLayerPicWidthInSamples}_Z \quad \text{(G-30)}$$
$$\text{refH} = \text{RefLayerPicHeightInSamples}_Z * (1 + \text{RefLayerFieldPicFlag}) \quad \text{(G-31)}$$
$$\text{scaledW} = \text{ScaledRefLayerPicWidthInSamples}_Z \quad \text{(G-32)}$$
$$\text{scaledH} = \text{ScaledRefLayerPicHeightInSamples}_Z * (1 + \text{field\_pic\_flag}) \quad \text{(G-33)}$$

2. When frame_mbs_only_flag is equal to 0 and RefLayerFrameMbsOnlyFlag is equal to 1, the variable scaledH is modified by

$$\text{scaledH} = \text{scaledH} / 2 \quad \text{(G-34)}$$

3. The variables refPhaseX, refPhaseY, phaseX, phaseY, subW, and subH are derived by

$$\text{refPhaseX} = ((\text{chromaFlag} == 0) \ ? \ 0 : (\text{ref\_layer\_chroma\_phase\_x\_plus1\_flag} - 1)) \quad \text{(G-35)}$$
$$\text{refPhaseY} = ((\text{chromaFlag} == 0) \ ? \ 0 : (\text{ref\_layer\_chroma\_phase\_y\_plus1} - 1)) \quad \text{(G-36)}$$
$$\text{phaseX} = ((\text{chromaFlag} == 0) \ ? \ 0 : (\text{chroma\_phase\_x\_plus1\_flag} - 1)) \quad \text{(G-37)}$$
$$\text{phaseY} = ((\text{chromaFlag} == 0) \ ? \ 0 : (\text{chroma\_phase\_y\_plus1} - 1)) \quad \text{(G-38)}$$
$$\text{subW} = ((\text{chromaFlag} == 0) \ ? \ 1 : \text{SubWidthC}) \quad \text{(G-39)}$$
$$\text{subH} = ((\text{chromaFlag} == 0) \ ? \ 1 : \text{SubHeightC}) \quad \text{(G-40)}$$

4. When RefLayerFrameMbsOnlyFlag is equal to 0 or frame_mbs_only_flag is equal to 0, the following applies:

   – If RefLayerFrameMbsOnlyFlag is equal to 1, the variables phaseY and refPhaseY are modified by

$$\text{phaseY} = \text{phaseY} + 4 * \text{botFieldFlag} + 3 - \text{subH} \quad \text{(G-41)}$$
$$\text{refPhaseY} = 2 * \text{refPhaseY} + 2 \quad \text{(G-42)}$$

   – Otherwise (RefLayerFrameMbsOnlyFlag is equal to 0), the variables phaseY and refPhaseY are modified by

$$\text{phaseY} = \text{phaseY} + 4 * \text{botFieldFlag} \quad \text{(G-43)}$$
$$\text{refPhaseY} = \text{refPhaseY} + 4 * \text{botFieldFlag} \quad \text{(G-44)}$$

5. The variables shiftX and shiftY are derived by

$$\text{shiftX} = ((\text{levelIdc} <= 30) \ ? \ 16 : (31 - \text{Ceil}(\text{Log2}(\text{refW})))) \quad \text{(G-45)}$$
$$\text{shiftY} = ((\text{levelIdc} <= 30) \ ? \ 16 : (31 - \text{Ceil}(\text{Log2}(\text{refH})))) \quad \text{(G-46)}$$

6. The variables scaleX and scaleY are derived by

$$\text{scaleX} = ((\text{refW} << \text{shiftX}) + (\text{scaledW} >> 1)) / \text{scaledW} \quad \text{(G-47)}$$
$$\text{scaleY} = ((\text{refH} << \text{shiftY}) + (\text{scaledH} >> 1)) / \text{scaledH} \quad \text{(G-48)}$$

7. The variables offsetX, addX, and deltaX are derived by

$$\text{offsetX} = \text{ScaledRefLayerLeftOffset} / \text{subW} \quad \text{(G-49)}$$
$$\text{addX} = (((\text{refW} * (2 + \text{phaseX})) << (\text{shiftX} - 2)) + (\text{scaledW} >> 1)) / \text{scaledW} + (1 << (\text{shiftX} - 5)) \quad \text{(G-50)}$$
$$\text{deltaX} = 4 * (2 + \text{refPhaseX}) \quad \text{(G-51)}$$

8. The variables offsetY, addY, and deltaY are derived as follows:

   – If RefLayerFrameMbsOnlyFlag is equal to 1 and frame_mbs_only_flag is equal to 1, the variables offsetY, addY, and deltaY are derived by

$$\text{offsetY} = \text{ScaledRefLayerTopOffset} / \text{subH} \quad \text{(G-52)}$$
$$\text{addY} = (((\text{refH} * (2 + \text{phaseY})) << (\text{shiftY} - 2)) + (\text{scaledH} >> 1)) / \text{scaledH} + (1 << (\text{shiftY} - 5)) \quad \text{(G-53)}$$
$$\text{deltaY} = 4 * (2 + \text{refPhaseY}) \quad \text{(G-54)}$$

– Otherwise (RefLayerFrameMbsOnlyFlag is equal to 0 or frame_mbs_only_flag is equal to 0), the variables offsetY, addY, and deltaY are derived by

$$\text{offsetY} = \text{ScaledRefLayerTopOffset} / (2 * \text{subH}) \quad \text{(G-55)}$$

$$\text{addY} = (((\text{refH} * (2 + \text{phaseY})) << (\text{shiftY} - 3)) + (\text{scaledH} >> 1)) / \text{scaledH} + (1 << (\text{shiftY} - 5)) \quad \text{(G-56)}$$

$$\text{deltaY} = 2 * (2 + \text{refPhaseY}) \quad \text{(G-57)}$$

NOTE – The variables subW, subH, shiftX, shiftY, scaleX, scaleY, offsetX, offsetY, addX, addY, deltaX, and deltaY do not depend on the input sample location (xP, yP), the input variable fieldMbFlag, or the current macroblock address CurrMbAddr.

The sample location (xC, yC) is derived as specified in the following ordered steps:

1. The inverse macroblock scanning process as specified in subclause 6.4.1 is invoked with CurrMbAddr as input and the output is assigned to (xM, yM). For this invocation of the process in subclause 6.4.1, the current macroblock is treated as field macroblock when fieldMbFlag is equal to 1 and it is treated as frame macroblock when fieldMbFlag is equal to 0.
2. The sample location (xC, yC) is derived by

$$\text{xC} = \text{xP} + (\text{xM} >> (\text{subW} - 1)) \quad \text{(G-58)}$$

$$\text{yC} = \text{yP} + (\text{yM} >> (\text{subH} - 1 + \text{fieldMbFlag} - \text{field\_pic\_flag})) \quad \text{(G-59)}$$

3. When RefLayerFrameMbsOnlyFlag is equal to 0 or frame_mbs_only_flag is equal to 0, the vertical component of the sample location (xC, yC) is modified by

$$\text{yC} = \text{yC} >> (1 - \text{fieldMbFlag}) \quad \text{(G-60)}$$

The reference layer sample location (xRef16 yRef16) is derived by

$$\text{xRef16} = (((\text{xC} - \text{offsetX}) * \text{scaleX} + \text{addX}) >> (\text{shiftX} - 4)) - \text{deltaX} \quad \text{(G-61)}$$

$$\text{yRef16} = (((\text{yC} - \text{offsetY}) * \text{scaleY} + \text{addY}) >> (\text{shiftY} - 4)) - \text{deltaY} \quad \text{(G-62)}$$

G.6.4 SVC derivation process for macroblock and sub-macroblock partition indices

Inputs to this process are:

– a variable currDQId specifying an identifier for a layer representation,
– a luma location (xP, yP) relative to the upper-left luma sample of a macroblock,
– a macroblock type mbType,
– when mbType is equal to P_8x8, P_8x8ref0, or B_8x8, a list of sub-macroblock types subMbType with 4 elements.

Outputs of this process are:

– a macroblock partition index mbPartIdx,
– a sub-macroblock partition index subMbPartIdx.

The variable svcDirectModeFlag is derived as follows:

– If currDQId is greater than 0 and any of the following conditions are true, svcDirectModeFlag is set equal to 1.
  – mbType is equal to B_Skip or B_Direct_16x16
  – mbType is equal to B_8x8 and subMbType[2 * (yP / 8) + (xP / 8)] is equal to B_Direct_8x8
– Otherwise, svcDirectModeFlag is set equal to 0.

Depending on svcDirectModeFlag, the following applies:

– If svcDirectModeFlag is equal to 0, the derivation process for macroblock and sub-macroblock partition indices as specified in subclause 6.4.13.4 is invoked with the luma location (xP, yP), the macroblock type mbType, and, when mbType is equal to P_8x8, P_8x8ref0, or B_8x8, the list of sub-macroblock types subMbType as the inputs and the outputs are the macroblock partition index mbPartIdx and the sub-macroblock partition index subMbPartIdx.

– Otherwise, if mbType is equal to B_Skip or B_Direct_16x16, mbPartIdx is set equal to 0 and subMbPartIdx is set equal to 0.

– Otherwise (currDQId is greater than 0, mbType is equal to B_8x8, and subMbType[2 * (yP / 8) + (xP / 8)] is equal to B_Direct_8x8), mbPartIdx is set equal to (2 * (yP / 8) + (xP / 8)) and subMbPartIdx is set equal to 0.

G.7 Syntax and semantics

This clause specifies syntax and semantics for coded video sequences that conform to one or more of the profiles specified in this annex.

G.7.1 Method of specifying syntax in tabular form

The specifications in subclause 7.1 apply.

G.7.2 Specification of syntax functions, categories, and descriptors

The specifications in subclause 7.2 apply.

G.7.3 Syntax in tabular form

G.7.3.1 NAL unit syntax

The syntax table is specified in subclause 7.3.1.

G.7.3.1.1 NAL unit header SVC extension syntax

nal_unit_header_svc_extension( ) {	**C**	**Descriptor**
idr_flag	All	u(1)
priority_id	All	u(6)
no_inter_layer_pred_flag	All	u(1)
dependency_id	All	u(3)
quality_id	All	u(4)
temporal_id	All	u(3)
use_ref_base_pic_flag	All	u(1)
discardable_flag	All	u(1)
output_flag	All	u(1)
reserved_three_2bits	All	u(2)
}		

G.7.3.2 Raw byte sequence payloads and RBSP trailing bits syntax

G.7.3.2.1 Sequence parameter set RBSP syntax

The syntax table is specified in subclause 7.3.2.1.

G.7.3.2.1.1 Sequence parameter set data syntax

The syntax table is specified in subclause 7.3.2.1.1.

G.7.3.2.1.1.1 Scaling list syntax

The syntax table is specified in subclause 7.3.2.1.1.1.

G.7.3.2.1.2 Sequence parameter set extension RBSP syntax

The syntax table is specified in subclause 7.3.2.1.2.

G.7.3.2.1.3 Subset sequence parameter set RBSP syntax

The syntax table is specified in subclause 7.3.2.1.3.

G.7.3.2.1.4 Sequence parameter set SVC extension syntax

seq_parameter_set_svc_extension( ) {	C	Descriptor
inter_layer_deblocking_filter_control_present_flag	0	u(1)
extended_spatial_scalability_idc	0	u(2)
if(ChromaArrayType = = 1 || ChromaArrayType = = 2)		
chroma_phase_x_plus1_flag	0	u(1)
if(ChromaArrayType = = 1)		
chroma_phase_y_plus1	0	u(2)
if(extended_spatial_scalability_idc = = 1) {		
if(ChromaArrayType > 0) {		
seq_ref_layer_chroma_phase_x_plus1_flag	0	u(1)
seq_ref_layer_chroma_phase_y_plus1	0	u(2)
}		
seq_scaled_ref_layer_left_offset	0	se(v)
seq_scaled_ref_layer_top_offset	0	se(v)
seq_scaled_ref_layer_right_offset	0	se(v)
seq_scaled_ref_layer_bottom_offset	0	se(v)
}		
seq_tcoeff_level_prediction_flag	0	u(1)
if(seq_tcoeff_level_prediction_flag) {		
adaptive_tcoeff_level_prediction_flag	0	u(1)
}		
slice_header_restriction_flag	0	u(1)
}		

G.7.3.2.2 Picture parameter set RBSP syntax

The syntax table is specified in subclause 7.3.2.2.

G.7.3.2.3 Supplemental enhancement information RBSP syntax

The syntax table is specified in subclause 7.3.2.3.

G.7.3.2.3.1 Supplemental enhancement information message syntax

The syntax table is specified in subclause 7.3.2.3.1.

G.7.3.2.4 Access unit delimiter RBSP syntax

The syntax table is specified in subclause 7.3.2.4.

G.7.3.2.5 End of sequence RBSP syntax

The syntax table is specified in subclause 7.3.2.5.

G.7.3.2.6 End of stream RBSP syntax

The syntax table is specified in subclause 7.3.2.6.

G.7.3.2.7 Filler data RBSP syntax

The syntax table is specified in subclause 7.3.2.7.

G.7.3.2.8 Slice layer without partitioning RBSP syntax

The syntax table is specified in subclause 7.3.2.8.

G.7.3.2.9 Slice data partition RBSP syntax

Slice data partition syntax is not present in coded video sequences conforming to any of the profiles specified in this annex.

G.7.3.2.10 RBSP slice trailing bits syntax

The syntax table is specified in subclause 7.3.2.10.

G.7.3.2.11 RBSP trailing bits syntax

The syntax table is specified in subclause 7.3.2.11.

G.7.3.2.12 Prefix NAL unit RBSP syntax

The syntax table is specified in subclause 7.3.2.12.

G.7.3.2.12.1 Prefix NAL unit SVC syntax

prefix_nal_unit_svc( ) {	**C**	**Descriptor**
if(nal_ref_idc != 0) {		
store_ref_base_pic_flag	2	u(1)
if((use_ref_base_pic_flag || store_ref_base_pic_flag) && !idr_flag)		
dec_ref_base_pic_marking( )	2	
additional_prefix_nal_unit_extension_flag	2	u(1)
if(additional_prefix_nal_unit_extension_flag = = 1)		
while(more_rbsp_data( ))		
additional_prefix_nal_unit_extension_data_flag	2	u(1)
rbsp_trailing_bits( )	2	
} else if(more_rbsp_data( )) {		
while(more_rbsp_data( ))		
additional_prefix_nal_unit_extension_data_flag	2	u(1)
rbsp_trailing_bits( )	2	
}		
}		

G.7.3.2.13 Slice layer extension RBSP syntax

The syntax table is specified in subclause 7.3.2.13.

G.7.3.3 Slice header syntax

The syntax table is specified in subclause 7.3.3.

G.7.3.3.1 Reference picture list modification syntax

The syntax table is specified in subclause 7.3.3.1.

G.7.3.3.2 Prediction weight table syntax

The syntax table is specified in subclause 7.3.3.2.

G.7.3.3.3 Decoded reference picture marking syntax

The syntax table is specified in subclause 7.3.3.3.

G.7.3.3.4 Slice header in scalable extension syntax

slice_header_in_scalable_extension( ) {	C	**Descriptor**
first_mb_in_slice	2	ue(v)
slice_type	2	ue(v)
pic_parameter_set_id	2	ue(v)
if(separate_colour_plane_flag = = 1)		
colour_plane_id	2	u(2)
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(idr_flag = = 1)		
idr_pic_id	2	ue(v)
if(pic_order_cnt_type = = 0) {		
pic_order_cnt_lsb	2	u(v)
if(bottom_field_pic_order_in_frame_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(bottom_field_pic_order_in_frame_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(quality_id = = 0) {		
if(slice_type = = EB)		
direct_spatial_mv_pred_flag	2	u(1)
if(slice_type = = EP || slice_type = = EB) {		
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 = = EB)		
num_ref_idx_l1_active_minus1	2	ue(v)
}		
}		
ref_pic_list_modification( )	2	
if((weighted_pred_flag && slice_type = = EP) || (weighted_bipred_idc = = 1 && slice_type = = EB)) {		
if(!no_inter_layer_pred_flag)		
base_pred_weight_table_flag	2	u(1)
if(no_inter_layer_pred_flag || !base_pred_weight_table_flag)		
pred_weight_table( )	2	
}		
if(nal_ref_idc != 0) {		

dec_ref_pic_marking( )	2	
if(!slice_header_restriction_flag) {		
store_ref_base_pic_flag	2	u(1)
if ((use_ref_base_pic_flag || store_ref_base_pic_flag) && !idr_flag)		
dec_ref_base_pic_marking( )	2	
}		
}		
}		
if(entropy_coding_mode_flag && slice_type != EI)		
cabac_init_idc	2	ue(v)
slice_qp_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)
if(!no_inter_layer_pred_flag && quality_id = = 0) {		
ref_layer_dq_id	2	ue(v)
if(inter_layer_deblocking_filter_control_present_flag) {		
disable_inter_layer_deblocking_filter_idc	2	ue(v)
if(disable_inter_layer_deblocking_filter_idc != 1) {		
inter_layer_slice_alpha_c0_offset_div2	2	se(v)
inter_layer_slice_beta_offset_div2	2	se(v)
}		
}		
constrained_intra_resampling_flag	2	u(1)
if(extended_spatial_scalability_idc = = 2) {		
if(ChromaArrayType > 0) {		
ref_layer_chroma_phase_x_plus1_flag	2	u(1)
ref_layer_chroma_phase_y_plus1	2	u(2)
}		
scaled_ref_layer_left_offset	2	se(v)
scaled_ref_layer_top_offset	2	se(v)
scaled_ref_layer_right_offset	2	se(v)
scaled_ref_layer_bottom_offset	2	se(v)
}		
}		
if(!no_inter_layer_pred_flag) {		
slice_skip_flag	2	u(1)
if(slice_skip_flag)		
num_mbs_in_slice_minus1	2	ue(v)
else {		
adaptive_base_mode_flag	2	u(1)

if(!adaptive_base_mode_flag)		
default_base_mode_flag	2	u(1)
if(!default_base_mode_flag) {		
adaptive_motion_prediction_flag	2	u(1)
if(!adaptive_motion_prediction_flag)		
default_motion_prediction_flag	2	u(1)
}		
adaptive_residual_prediction_flag	2	u(1)
if(!adaptive_residual_prediction_flag)		
default_residual_prediction_flag	2	u(1)
}		
if(adaptive_tcoeff_level_prediction_flag)		
tcoeff_level_prediction_flag	2	u(1)
}		
if(!slice_header_restriction_flag && !slice_skip_flag) {		
scan_idx_start	2	u(4)
scan_idx_end	2	u(4)
}		
}		

G.7.3.3.5 Decoded reference base picture marking syntax

dec_ref_base_pic_marking( ) {	C	Descriptor
adaptive_ref_base_pic_marking_mode_flag	2	u(1)
if(adaptive_ref_base_pic_marking_mode_flag)		
do {		
memory_management_base_control_operation	2	ue(v)
if(memory_management_base_control_operation = = 1)		
difference_of_base_pic_nums_minus1	2	ue(v)
if(memory_management_base_control_operation = = 2)		
long_term_base_pic_num	2	ue(v)
} while(memory_management_base_control_operation != 0)		
}		

G.7.3.4 Slice data syntax

The syntax table is specified in subclause 7.3.4.

G.7.3.4.1 Slice data in scalable extension syntax

slice_data_in_scalable_extension( ) {	C	Descriptor
if(entropy_coding_mode_flag)		
while(!byte_aligned( ))		
cabac_alignment_one_bit	2	f(1)
CurrMbAddr = first_mb_in_slice * (1 + MbaffFrameFlag)		
moreDataFlag = 1		
prevMbSkipped = 0		
do {		
if(slice_type != EI)		
if(!entropy_coding_mode_flag) {		
mb_skip_run	2	ue(v)
prevMbSkipped = (mb_skip_run > 0)		
for(i = 0; i < mb_skip_run; i++)		
CurrMbAddr = NextMbAddress(CurrMbAddr)		
if(mb_skip_run > 0)		
moreDataFlag = more_rbsp_data( )		
} else {		
mb_skip_flag	2	ae(v)
moreDataFlag = !mb_skip_flag		
}		
if(moreDataFlag) {		
if(MbaffFrameFlag && ((CurrMbAddr % 2) = = 0 || ((CurrMbAddr % 2) = = 1 && prevMbSkipped)))		
mb_field_decoding_flag	2	u(1) | ae(v)
macroblock_layer_in_scalable_extension( )	2 | 3 | 4	
}		
if(!entropy_coding_mode_flag)		
moreDataFlag = more_rbsp_data( )		
else {		
if(slice_type != EI)		
prevMbSkipped = mb_skip_flag		
if(MbaffFrameFlag && (CurrMbAddr % 2) = = 0)		
moreDataFlag = 1		
else {		
end_of_slice_flag	2	ae(v)
moreDataFlag = !end_of_slice_flag		
}		
}		
CurrMbAddr = NextMbAddress(CurrMbAddr)		
} while(moreDataFlag)		
}		

G.7.3.5 Macroblock layer syntax

The syntax table is specified in subclause 7.3.5.

G.7.3.5.1 Macroblock prediction syntax

The syntax table is specified in subclause 7.3.5.1.

G.7.3.5.2 Sub-macroblock prediction syntax

The syntax table is specified in subclause 7.3.5.2.

G.7.3.5.3 Residual data syntax

The syntax table is specified in subclause 7.3.5.3.

G.7.3.5.3.1 Residual luma syntax

The syntax table is specified in subclause 7.3.5.3.1.

G.7.3.5.3.2 Residual block CAVLC syntax

The syntax table is specified in subclause 7.3.5.3.2.

G.7.3.5.3.3 Residual block CABAC syntax

The syntax table is specified in subclause 7.3.5.3.3.

G.7.3.6 Macroblock layer in scalable extension syntax

macroblock_layer_in_scalable_extension( ) {	**C**	**Descriptor**
if(InCropWindow(CurrMbAddr) && adaptive_base_mode_flag)		
base_mode_flag	2	u(1) | ae(v)
if(!base_mode_flag)		
mb_type	2	ue(v) | ae(v)
if(mb_type = = I_PCM) {		
while(!byte_aligned( ))		
pcm_alignment_zero_bit	3	f(1)
for(i = 0; i < 256; i++)		
pcm_sample_luma[i **]**	3	u(v)
for(i = 0; i < 2 * MbWidthC * MbHeightC; i++)		
pcm_sample_chroma[i **]**	3	u(v)
} else {		
if(!base_mode_flag) {		
noSubMbPartSizeLessThan8x8Flag = 1		
if(mb_type != I_NxN && MbPartPredMode(mb_type, 0) != Intra_16x16 && NumMbPart(mb_type) = = 4) {		
sub_mb_pred_in_scalable_extension(mb_type)	2	
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8) {		
if(NumSubMbPart(sub_mb_type [mbPartIdx]) > 1)		
noSubMbPartSizeLessThan8x8Flag = 0		
} else if(!direct_8x8_inference_flag)		
noSubMbPartSizeLessThan8x8Flag = 0		
} else {		
if(transform_8x8_mode_flag && mb_type = = I_NxN)		
transform_size_8x8_flag	2	u(1) | ae(v)
mb_pred_in_scalable_extension(mb_type)	2	
}		
}		

if(adaptive_residual_prediction_flag && slice_type != EI && InCropWindow(CurrMbAddr) && (base_mode_flag || (MbPartPredMode(mb_type, 0) != Intra_16x16 && MbPartPredMode(mb_type, 0) != Intra_8x8 && MbPartPredMode(mb_type, 0) != Intra_4x4)))		
residual_prediction_flag	2	u(1) | ae(v)
if(scan_idx_end >= scan_idx_start) {		
if(base_mode_flag || MbPartPredMode(mb_type, 0) != Intra_16x16) {		
coded_block_pattern	2	me(v) | ae(v)
if(CodedBlockPatternLuma > 0 && transform_8x8_mode_flag && (base_mode_flag || (mb_type != I_NxN && noSubMbPartSizeLessThan8x8Flag && (mb_type != B_Direct_16x16 || direct_8x8_inference_flag))))		
transform_size_8x8_flag	2	u(1) | ae(v)
}		
if(CodedBlockPatternLuma > 0 || CodedBlockPatternChroma > 0 || MbPartPredMode(mb_type, 0) = = Intra_16x16) {		
mb_qp_delta	2	se(v) | ae(v)
residual(scan_idx_start, scan_idx_end)	3 | 4	
}		
}		
}		
}		

G.7.3.6.1 Macroblock prediction in scalable extension syntax

mb_pred_in_scalable_extension(mb_type) {	C	Descriptor
if(MbPartPredMode(mb_type, 0) = = Intra_4x4 || MbPartPredMode(mb_type, 0) = = Intra_8x8 || MbPartPredMode(mb_type, 0) = = Intra_16x16) {		
if(MbPartPredMode(mb_type, 0) = = Intra_4x4)		
for(luma4x4BlkIdx = 0; luma4x4BlkIdx < 16; luma4x4BlkIdx++) {		
prev_intra4x4_pred_mode_flag[luma4x4BlkIdx **]**	2	u(1) | ae(v)
if(!prev_intra4x4_pred_mode_flag**[** luma4x4BlkIdx **]**)		
rem_intra4x4_pred_mode[luma4x4BlkIdx **]**	2	u(3) | ae(v)
}		
if(MbPartPredMode(mb_type, 0) = = Intra_8x8)		
for(luma8x8BlkIdx = 0; luma8x8BlkIdx < 4; luma8x8BlkIdx++) {		
prev_intra8x8_pred_mode_flag[luma8x8BlkIdx **]**	2	u(1) | ae(v)
if(!prev_intra8x8_pred_mode_flag[luma8x8BlkIdx])		
rem_intra8x8_pred_mode[luma8x8BlkIdx **]**	2	u(3) | ae(v)
}		
if(ChromaArrayType != 0)		
intra_chroma_pred_mode	2	ue(v) | ae(v)
} else if(MbPartPredMode(mb_type, 0) != Direct) {		
if(InCropWindow(CurrMbAddr) && adaptive_motion_prediction_flag) {		

for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode(mb_type, mbPartIdx) != Pred_L1)		
motion_prediction_flag_l0[mbPartIdx **]**	2	u(1) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode(mb_type, mbPartIdx) != Pred_L0)		
motion_prediction_flag_l1[mbPartIdx **]**	2	u(1) | ae(v)
}		
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if((num_ref_idx_l0_active_minus1 > 0 || mb_field_decoding_flag != field_pic_flag) && MbPartPredMode(mb_type, mbPartIdx) != Pred_L1 && !motion_prediction_flag_l0[mbPartIdx])		
ref_idx_l0[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if((num_ref_idx_l1_active_minus1 > 0 || mb_field_decoding_flag != field_pic_flag) && MbPartPredMode(mb_type, mbPartIdx) != Pred_L0 && !motion_prediction_flag_l1[mbPartIdx])		
ref_idx_l1[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode (mb_type, mbPartIdx) != Pred_L1)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l0[mbPartIdx **][** 0 **][** compIdx **]**	2	se(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode(mb_type, mbPartIdx) != Pred_L0)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l1[mbPartIdx **][** 0 **][** compIdx **]**	2	se(v) | ae(v)
}		
}		

G.7.3.6.2 Sub-macroblock prediction in scalable extension syntax

sub_mb_pred_in_scalable_extension(mb_type) {	**C**	**Descriptor**
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
sub_mb_type[mbPartIdx **]**	2	ue(v) | ae(v)
if(InCropWindow(CurrMbAddr) && adaptive_motion_prediction_flag) {		
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(SubMbPredMode(sub_mb_type[mbPartIdx]) != Direct && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L1)		
motion_prediction_flag_l0[mbPartIdx]	2	u(1) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(SubMbPredMode(sub_mb_type[mbPartIdx]) != Direct && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L0)		
motion_prediction_flag_l1[mbPartIdx]	2	u(1) | ae(v)
}		
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if((num_ref_idx_l0_active_minus1 > 0 || mb_field_decoding_flag != field_pic_flag) && mb_type != P_8x8ref0 && sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L1 && !motion_prediction_flag_l0[mbPartIdx])		
ref_idx_l0[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if((num_ref_idx_l1_active_minus1 > 0 || mb_field_decoding_flag != field_pic_flag) && sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L0 && !motion_prediction_flag_l1[mbPartIdx])		
ref_idx_l1[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L1)		
for(subMbPartIdx = 0; subMbPartIdx < NumSubMbPart(sub_mb_type[mbPartIdx]); subMbPartIdx++)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l0[mbPartIdx **][** subMbPartIdx **][** compIdx **]**	2	se(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L0)		
for(subMbPartIdx = 0; subMbPartIdx < NumSubMbPart(sub_mb_type[mbPartIdx]); subMbPartIdx++)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l1[mbPartIdx **][** subMbPartIdx **][** compIdx **]**	2	se(v) | ae(v)
}		

G.7.4 Semantics

Semantics associated with the syntax structures and syntax elements within these structures (in subclause G.7.3 and in subclause 7.3 by reference in subclause G.7.3) are specified in this subclause and by reference to subclause 7.4. When the semantics of a syntax element are specified using a table or a set of tables, any values that are not specified in the table(s) shall not be present in the bitstream unless otherwise specified in this Recommendation | International Standard.

Sub-bitstreams that are derived according to the process specified in subclause G.8.8.1 shall conform to one or more of the profiles specified in Annex A or one or more of the profiles specified in this annex.

One or more sub-bitstreams shall conform to one or more of the profiles specified in Annex A. The decoding for these sub-bitstreams is specified in clauses 2-9 and Annexes B-E.

The decoding for bitstreams conforming to one or more of the profiles specified in this annex is completely specified in this annex with reference made to clauses 2-9 and Annexes B-E.

A specification or a process in clauses 2-9 and Annexes B-E may be used as is or by specifying assignments or alternative meanings of certain parts.

This subclause describes the semantics of syntax elements. The syntax elements appear multiple times in the bitstream and in each access unit. The meaning of each syntax element and derived variables depends on the position of the syntax structure in the bitstream in which it is contained. A decoder conforming to this Recommendation | International Standard processes the syntax structures in decoding order and determines the semantics according to the position derived from that.

G.7.4.1 NAL unit semantics

The semantics for the syntax elements in subclause G.7.3.1 are specified in subclause 7.4.1. The following specifications additionally apply.

For NAL units with nal_unit_type equal to 14, nal_ref_idc shall be identical to nal_ref_idc of the associated NAL unit, which succeeds the NAL unit with nal_unit_type equal to 14 in decoding order.

The value of nal_ref_idc shall be the same for all VCL NAL units of a dependency representation.

The variable refNalRefIdc is derived as follows:

– If nal_unit_type is not equal to 20 or dependency_id is equal to the minimum value of dependency_id for all VCL NAL units of the coded picture, refNalRefIdc is set equal to 0.

– Otherwise (nal_unit_type is equal to 20 and dependency_id is not equal to the minimum value of dependency_id for all VCL NAL units of the coded picture), refNalRefIdc is set equal to the maximum value of nal_ref_idc for all VCL NAL units of the coded picture with a value of dependency_id less than the current value of dependency_id.

When refNalRefIdc is greater than 0, the value of nal_ref_idc shall not be equal to 0.

nal_ref_idc equal to 0 for a NAL unit containing a slice and having a value of dependency_id that is equal to the maximum value of dependency_id in the coded picture indicates that all coded slice NAL units of the coded picture are coded slice NAL units of a non-reference picture.

nal_ref_idc greater than 0 for a NAL unit containing a slice and having a value of dependency_id that is equal to the maximum value of dependency_id in the coded picture indicates that all coded slice NAL units of the coded picture are coded slice NAL units of a reference picture.

G.7.4.1.1 NAL unit header SVC extension semantics

The syntax elements idr_flag, priority_id, no_inter_layer_pred_flag, dependency_id, quality_id, temporal_id, use_ref_base_pic_flag, discardable_flag, and output_flag, when present in a prefix NAL unit, are considered as if they were present in the associated NAL unit.

idr_flag equal to 1 specifies that the current coded picture is an IDR picture when the value of dependency_id for the NAL unit is equal to the maximum value of dependency_id in the coded picture. idr_flag equal to 0 specifies that the current coded picture is not an IDR picture when the value of dependency_id for the NAL unit is equal to the maximum value of dependency_id in the coded picture. The value of idr_flag shall be the same for all NAL units of a dependency representation.

NOTE 1 – The classification of a coded picture as IDR picture and the partitioning of a sequence of access units in coded video sequences depends on the maximum value of dependency_id that is present in the associated NAL units. When NAL units are removed from a bitstream, e.g. in order to adjust the bitstream to the capabilities of a receiving device, the maximum value of dependency_id in the coded pictures may change and hence the classification of coded pictures as IDR pictures may change and with that the partitioning of the sequence of access units into coded video sequences may change.

When idr_flag is equal to 1 for a prefix NAL unit, the associated NAL unit shall have nal_unit_type equal to 5. When idr_flag is equal to 0 for a prefix NAL unit, the associated NAL unit shall have nal_unit_type equal to 1.

When nal_ref_idc is equal to 0, the value of idr_flag shall be equal to 0.

For NAL units, in which idr_flag is present, the variable IdrPicFlag derived in subclause 7.4.1 is modified by setting it equal to idr_flag.

priority_id specifies a priority identifier for the NAL unit. The assignment of values to priority_id is constrained by the sub-bitstream extraction process as specified in subclause G.8.8.1.

NOTE 2 – The syntax element priority_id is not required by the decoding process specified in this Recommendation | International Standard. The syntax element priority_id may be used as determined by the application within the specified constraints.

no_inter_layer_pred_flag specifies whether inter-layer prediction may be used for decoding the coded slice. When no_inter_layer_pred_flag is equal to 1, inter-layer prediction is not used for decoding the coded slice. When no_inter_layer_pred_flag is equal to 0, inter-layer prediction may be used for decoding the coded slice as signalled in the macroblock layer.

For prefix NAL units, no_inter_layer_pred_flag shall be equal to 1. When nal_unit_type is equal to 20 and quality_id is greater than 0, no_inter_layer_pred_flag shall be equal to 0.

The variable MinNoInterLayerPredFlag is set equal to the minimum value of no_inter_layer_pred_flag for the slices of the layer representation.

dependency_id specifies a dependency identifier for the NAL unit. dependency_id shall be equal to 0 in prefix NAL units. The assignment of values to dependency_id is constrained by the sub-bitstream extraction process as specified in subclause G.8.8.1.

quality_id specifies a quality identifier for the NAL unit. quality_id shall be equal to 0 in prefix NAL units. The assignment of values to quality_id is constrained by the sub-bitstream extraction process as specified in subclause G.8.8.1.

The variable DQId is derived by

$$\text{DQId} = (\text{ dependency\_id} \ll 4 \text{ }) + \text{quality\_id} \quad \text{(G-63)}$$

When nal_unit_type is equal to 20, the bitstream shall not contain data that result in DQId equal to 0.

temporal_id specifies a temporal identifier for the NAL unit. The assignment of values to temporal_id is constrained by the sub-bitstream extraction process as specified in subclause G.8.8.1.

The value of temporal_id shall be the same for all prefix NAL units and coded slice in scalable extension NAL units of an access unit. When an access unit contains any NAL unit with nal_unit_type equal to 5 or idr_flag equal to 1, temporal_id shall be equal to 0.

use_ref_base_pic_flag equal to 1 specifies that reference base pictures (when present) and decoded pictures (when reference base pictures are not present) are used as reference pictures for inter prediction as specified in subclause G.8.2.3. use_ref_base_pic_flag equal to 0 specifies that reference base pictures are not used as reference pictures for inter prediction (i.e., only decoded pictures are used for inter prediction).

The values of use_ref_base_pic_flag shall be the same for all NAL units of a dependency representation.

discardable_flag equal to 1 specifies that the current NAL unit is not used for decoding dependency representations that are part of the current coded picture or any subsequent coded picture in decoding order and have a greater value of dependency_id than the current NAL unit. discardable_flag equal to 0 specifies that the current NAL unit may be used for decoding dependency representations that are part of the current coded picture or any subsequent coded picture in decoding order and have a greater value of dependency_id than the current NAL unit.

output_flag affects the decoded picture output and removal processes as specified in Annex C. The value of output_flag shall be the same for all NAL units of a dependency representation. For any particular value of dependency_id, the value of output_flag shall be the same for both fields of a complementary field pair.

reserved_three_2bits shall be equal to 3. Other values of reserved_three_2bits may be specified in the future by ITU-T | ISO/IEC. Decoders shall ignore the value of reserved_three_2bits.

G.7.4.1.2 Order of NAL units and association to coded pictures, access units, and video sequences

This subclause specifies constraints on the order of NAL units in the bitstream. Any order of NAL units in the bitstream obeying these constraints is referred to in the text as the decoding order of NAL units. Within a NAL unit, the syntax in subclauses 7.3, D.1, E.1, G.7.3, G.13.1, and G.14.1 specifies the decoding order of syntax elements. Decoders shall be capable of receiving NAL units and their syntax elements in decoding order.

G.7.4.1.2.1 Order of SVC sequence parameter set RBSPs and picture parameter set RBSPs and their activation

NOTE 1 – The sequence and picture parameter set mechanism decouples the transmission of infrequently changing information from the transmission of coded macroblock data. Sequence and picture parameter sets may, in some applications, be conveyed "out-of-band" using a reliable transport mechanism.

A picture parameter set RBSP includes parameters that can be referred to by the coded slice NAL units of one or more layer representations of one or more coded pictures.

Each picture parameter set RBSP is initially considered not active at the start of the operation of the decoding process. At most one picture parameter set RBSP is considered as the active picture parameter set RBSP at any given moment during the operation of the decoding process, and when any particular picture parameter set RBSP becomes the active picture parameter set RBSP, the previously-active picture parameter set RBSP (if any) is deactivated.

In addition to the active picture parameter set RBSP, zero or more picture parameter set RBSPs may be specifically active for layer representations (with a particular value of DQId less than DQIdMax) that may be referred to through inter-layer prediction in decoding the target layer representation. Such a picture parameter set RBSP is referred to as active layer picture parameter set RBSP for the particular value of DQId (less than DQIdMax). The restrictions on active picture parameter set RBSPs also apply to active layer picture parameter set RBSPs with a particular value of DQId.

When a picture parameter set RBSP (with a particular value of pic_parameter_set_id) is not the active picture parameter set RBSP and it is referred to by a coded slice NAL unit with DQId equal to DQIdMax (using that value of pic_parameter_set_id), it is activated. This picture parameter set RBSP is called the active picture parameter set RBSP until it is deactivated when another picture parameter set RBSP becomes the active picture parameter set RBSP. A picture parameter set RBSP, with that particular value of pic_parameter_set_id, shall be available to the decoding process prior to its activation.

When a picture parameter set RBSP (with a particular value of pic_parameter_set_id) is not the active layer picture parameter set for a particular value of DQId less than DQIdMax and it is referred to by a coded slice NAL unit with the particular value of DQId (using that value of pic_parameter_set_id), it is activated for layer representations with the particular value of DQId. This picture parameter set RBSP is called the active layer picture parameter set RBSP for the particular value of DQId until it is deactivated when another picture parameter set RBSP becomes the active layer picture parameter set RBSP for the particular value of DQId or when decoding an access unit with DQIdMax less than or equal to the particular value of DQId. A picture parameter set RBSP, with that particular value of pic_parameter_set_id, shall be available to the decoding process prior to its activation.

Any picture parameter set NAL unit containing the value of pic_parameter_set_id for the active picture parameter set RBSP for a coded picture shall have the same content as that of the active picture parameter set RBSP for the coded picture unless it follows the last VCL NAL unit of the coded picture and precedes the first VCL NAL unit of another coded picture. Any picture parameter set NAL unit containing the value of pic_parameter_set_id for the active layer picture parameter set RBSP for a particular value of DQId less than DQIdMax for a coded picture shall have the same content as that of the active layer picture parameter set RBSP for the particular value of DQId for the coded picture unless it follows the last VCL NAL unit of the coded picture and precedes the first VCL NAL unit of another coded picture.

When a picture parameter set NAL unit with a particular value of pic_parameter_set_id is received, its content replaces the content of the previous picture parameter set NAL unit, in decoding order, with the same value of pic_parameter_set_id (when a previous picture parameter set NAL unit with the same value of pic_parameter_set_id was present in the bitstream).

NOTE 2 – A decoder must be capable of simultaneously storing the contents of the picture parameter sets for all values of pic_parameter_set_id. The content of the picture parameter set with a particular value of pic_parameter_set_id is overwritten when a new picture parameter set NAL unit with the same value of pic_parameter_set_id is received.

An SVC sequence parameter set RBSP includes parameters that can be referred to by one or more picture parameter set RBSPs or one or more SEI NAL units containing a buffering period SEI message.

Each SVC sequence parameter set RBSP is initially considered not active at the start of the operation of the decoding process. At most one SVC sequence parameter set RBSP is considered as the active SVC sequence parameter set RBSP at any given moment during the operation of the decoding process, and when any particular SVC sequence parameter set RBSP becomes the active SVC sequence parameter set RBSP, the previously-active SVC sequence parameter set RBSP (if any) is deactivated.

In addition to the active SVC sequence parameter set RBSP, zero or more SVC sequence parameter set RBSPs may be specifically active for layer representations (with a particular value of DQId less than DQIdMax) that may be referred to through inter-layer prediction in decoding the target layer representation. Such an SVC sequence parameter set RBSP is referred to as active layer SVC sequence parameter set RBSP for the particular value of DQId (less than DQIdMax). The restrictions on active SVC sequence parameter set RBSPs also apply to active layer SVC sequence parameter set RBSPs with a particular value of DQId.

For the following specification in this subclause, the activating buffering period SEI message is specified as follows:

– If the access unit contains one or more buffering period SEI messages that are included in a scalable nesting SEI message and are associated with values of DQId in the range of ((DQIdMax >> 4) << 4) to (((DQIdMax >> 4) << 4) + 15), inclusive, the last of these buffering period SEI messages in decoding order is the activating buffering period SEI message.

– Otherwise, if DQIdMax is equal to 0 and the access unit contains a buffering period SEI message that is not included in a scalable nesting SEI message, this buffering period SEI message is the activating buffering period SEI message.

– Otherwise, the access unit does not contain an activating buffering period SEI message.

When a sequence parameter set RBSP (nal_unit_type is equal to 7) with a particular value of seq_parameter_set_id is not already the active SVC sequence parameter set RBSP and it is referred to by activation of a picture parameter set RBSP (using that value of seq_parameter_set_id) and the picture parameter set RBSP is activated by a coded slice NAL unit with nal_unit_type equal to 1 or 5 (the picture parameter set RBSP becomes the active picture parameter set RBSP and DQIdMax is equal to 0) and the access unit does not contain an activating buffering period SEI message, the sequence parameter set RBSP is activated. This sequence parameter set RBSP, is called the active SVC sequence parameter set RBSP until it is deactivated when another SVC sequence parameter set RBSP becomes the active SVC sequence parameter set RBSP. A sequence parameter set RBSP, with that particular value of seq_parameter_set_id, shall be available to the decoding process prior to its activation.

When a sequence parameter set RBSP (nal_unit_type is equal to 7) with a particular value of seq_parameter_set_id is not already the active SVC sequence parameter set RBSP and it is referred to by an activating buffering period SEI message (using that value of seq_parameter_set_id) that is not included in a scalable nesting SEI message (DQIdMax is equal to 0), the sequence parameter set RBSP is activated. This sequence parameter set RBSP is called the active SVC sequence parameter set RBSP until it is deactivated when another SVC sequence parameter set RBSP becomes the active SVC sequence parameter set RBSP. A sequence parameter set RBSP, with that particular value of seq_parameter_set_id, shall be available to the decoding process prior to its activation.

When a subset sequence parameter set RBSP (nal_unit_type is equal to 15) with a particular value of seq_parameter_set_id is not already the active SVC sequence parameter set RBSP and it is referred to by activation of a picture parameter set RBSP (using that value of seq_parameter_set_id) and the picture parameter set RBSP is activated by a coded slice in scalable extension NAL unit (nal_unit_type is equal to 20) with DQId equal to DQIdMax (the picture parameter set RBSP becomes the active picture parameter set RBSP) and the access unit does not contain an activating buffering period SEI message, the subset sequence parameter set RBSP is activated. This subset sequence parameter set RBSP is called the active SVC sequence parameter set RBSP until it is deactivated when another SVC sequence parameter set RBSP becomes the active SVC sequence parameter set RBSP. A subset sequence parameter set RBSP, with that particular value of seq_parameter_set_id, shall be available to the decoding process prior to its activation.

When a subset sequence parameter set RBSP (nal_unit_type is equal to 15) with a particular value of seq_parameter_set_id is not already the active SVC sequence parameter set RBSP and it is referred to by an activating buffering period SEI message (using that value of seq_parameter_set_id) that is included in a scalable nesting SEI message, the subset sequence parameter set RBSP is activated. This subset sequence parameter set RBSP, is called the active SVC sequence parameter set RBSP until it is deactivated when another SVC sequence parameter set RBSP becomes the active SVC sequence parameter set RBSP. A subset sequence parameter set RBSP, with that particular value of seq_parameter_set_id, shall be available to the decoding process prior to its activation.

NOTE 3 – The active SVC sequence parameter set RBSP is either a sequence parameter set RBSP or a subset sequence parameter set RBSP. Sequence parameter set RBSPs are activated by coded slice NAL units with nal_unit_type equal to 1 or 5 or buffering period SEI messages that are not included in a scalable nesting SEI message. Subset sequence parameter set RBSPs are activated by coded slice in scalable extension NAL units (nal_unit_type equal to 20) or buffering period SEI messages that are included in a scalable nesting SEI message. A sequence parameter set RBSP and a subset sequence parameter set RBSP may have the same value of seq_parameter_set_id.

NOTE 4 – Buffering period SEI messages have a higher priority for activating SVC sequence parameter sets than coded slice NAL units. When an SVC sequence parameter set RBSP is referred to by activation of a picture parameter set RBSP inside a particular access unit and this picture parameter set RBSP is activated by a coded slice NAL unit with DQId equal to DQIdMax (the picture parameter set RBSP becomes the active picture parameter set RBSP) and this particular access unit also contains an activating buffering period SEI message that refers to an SVC sequence parameter set RBSP that is different than the SVC sequence parameter set RBSP referred to by the activation of the picture parameter set RBSP, the SVC sequence parameter set RBSP that is referred to by the activating buffering period SEI message becomes the active SVC sequence parameter set.

NOTE 5 – Compared to the specifications for profiles specified in Annex A, where an activated sequence parameter set RBSP must remain active for the entire coded video sequence, the specification for profiles specified in this annex differs. When an SVC sequence parameter set RBSP is already active (as the active SVC sequence parameter set RBSP), another SVC sequence parameter set RBSP becomes the active SVC sequence parameter set RBSP in a non-IDR access unit when it is referred to by an activating buffering period SEI message or by the activation of a picture parameter set RBSP (as the active picture parameter set RBSP). In this case, the contents of the de-activated and activated SVC sequence parameter set RBSP are mutually restricted as described below. Hence, within a coded video sequence, multiple successively activated/de-activated SVC sequence parameter set RBSPs can be present.

For the following specification in this subclause, the activating layer buffering period SEI message for a particular value of DQId is specified as follows:

– If the access unit contains a buffering period SEI messages that is included in a scalable nesting SEI message and is associated with the particular value of DQId, this buffering period SEI message is the activating layer buffering period SEI message for the particular value of DQId.

– Otherwise, if the particular value of DQId is equal to 0 and the access unit contains a buffering period SEI message that is not included in a scalable nesting SEI message, this buffering period SEI message is the activating layer buffering period SEI message for the particular value of DQId.

– Otherwise, the access unit does not contain an activating layer buffering period SEI message for the particular value of DQId.

When a sequence parameter set RBSP (nal_unit_type is equal to 7) with a particular value of seq_parameter_set_id is not already the active layer SVC sequence parameter set RBSP for DQId equal to 0 and it is referred to by activation of a picture parameter set RBSP (using that value of seq_parameter_set_id) and the picture parameter set RBSP is activated by a coded slice NAL unit with nal_unit_type equal to 1 or 5 and DQIdMax is greater than 0 (the picture parameter set RBSP becomes the active layer picture parameter set RBSP for DQId equal to 0) and the access unit does not contain an activating layer buffering period SEI message for DQId equal to 0, the sequence parameter set RBSP is activated for layer representations with DQId equal to 0. This sequence parameter set RBSP is called the active layer SVC sequence parameter set RBSP for DQId equal to 0 until it is deactivated when another SVC sequence parameter set RBSP becomes the active layer SVC sequence parameter set RBSP for DQId equal to 0 or when decoding an access unit with DQIdMax equal to 0. A sequence parameter set RBSP, with that particular value of seq_parameter_set_id, shall be available to the decoding process prior to its activation.

When a sequence parameter set RBSP (nal_unit_type is equal to 7) with a particular value of seq_parameter_set_id is not already the active layer SVC sequence parameter set RBSP for DQId equal to 0 and it is referred to by an activating layer buffering period SEI message for DQId equal to 0 (using that value of seq_parameter_set_id) that is not included in a scalable nesting SEI message and DQIdMax is greater than 0, the sequence parameter set RBSP is activated for layer representations with DQId equal to 0. This sequence parameter set RBSP is called the active layer SVC sequence parameter set RBSP for DQId equal to 0 until it is deactivated when another SVC sequence parameter set RBSP becomes the active layer SVC sequence parameter set RBSP for DQId equal to 0 or when decoding an access unit with DQIdMax equal to 0. A sequence parameter set RBSP, with that particular value of seq_parameter_set_id, shall be available to the decoding process prior to its activation.

When a subset sequence parameter set RBSP (nal_unit_type is equal to 15) with a particular value of seq_parameter_set_id is not already the active layer SVC sequence parameter set RBSP for a particular value of DQId less than DQIdMax and it is referred to by activation of a picture parameter set RBSP (using that value of seq_parameter_set_id) and the picture parameter set RBSP is activated by a coded slice in scalable extension NAL unit (nal_unit_type is equal to 20) with the particular value of DQId (the picture parameter set RBSP becomes the active layer picture parameter set RBSP for the particular value of DQId) and the access unit does not contain an activating layer buffering period SEI message for the particular value of DQId, the subset sequence parameter set is activated for layer representations with the particular value of DQId. This subset sequence parameter set RBSP is called the active layer SVC sequence parameter set RBSP for the particular value of DQId until it is deactivated when another SVC sequence parameter set RBSP becomes the active layer SVC sequence parameter set RBSP for the particular value of DQId or when decoding an access unit with DQIdMax less than or equal to the particular value of DQId. A subset sequence parameter set RBSP, with that particular value of seq_parameter_set_id, shall be available to the decoding process prior to its activation.

When a subset sequence parameter set RBSP (nal_unit_type is equal to 15) with a particular value of seq_parameter_set_id is not already the active layer SVC sequence parameter set RBSP for a particular value of DQId less than DQIdMax and it is referred to by an activating layer buffering period SEI message for the particular value of DQId (using that value of seq_parameter_set_id) that is included in a scalable nesting SEI message, the subset sequence parameter set RBSP is activated for layer representations with the particular value of DQId. This subset sequence parameter set RBSP is called the active layer SVC sequence parameter set RBSP for the particular value of DQId until it is deactivated when another SVC sequence parameter set RBSP becomes the active layer SVC sequence parameter set RBSP for the particular value of DQId or when decoding an access unit with DQIdMax less than or equal to the particular value of DQId. A subset sequence parameter set RBSP, with that particular value of seq_parameter_set_id, shall be available to the decoding process prior to its activation.

A sequence parameter set RBSP or a subset sequence parameter set RBSP that includes a value of profile_idc not specified in Annex A or G shall not be referred to by activation of a picture parameter set RBSP as the active picture parameter set RBSP or as active layer picture parameter set RBSP (using that value of seq_parameter_set_id) or referred to by a buffering period SEI message (using that value of seq_parameter_set_id). A sequence parameter set RBSP or a subset sequence parameter set RBSP including a value of profile_idc not specified in Annex A or G is ignored in the decoding for profiles specified in Annex A or G.

Let spsA and spsB be two SVC sequence parameter set RBSPs with one of the following properties:

– spsA is the SVC sequence parameter set RBSP that is referred to by the coded slice NAL units (via the picture parameter set) of a layer representation with a particular value of dependency_id and quality_id equal to 0 and spsB is the SVC sequence parameter set RBSP that is referred to by the coded slice NAL units (via the picture parameter set) of another layer representation, in the same access unit, with the same value of dependency_id and quality_id greater than 0,

– spsA is the active SVC sequence parameter set RBSP for an access unit and spsB is the SVC sequence parameter set RBSP that is referred to by the coded slice NAL units (via the picture parameter set) of the layer representation with DQId equal to DQIdMax,

– spsA is the active SVC sequence parameter set RBSP for an IDR access unit and spsB is the active SVC sequence parameter set RBSP for any non-IDR access unit of the same coded video sequence.

The SVC sequence parameter set RBSPs spsA and spsB are restricted with regards to their contents as specified in the following.

– The values of the syntax elements in the seq_parameter_set_data( ) syntax structure of spsA and spsB may only differ for the following syntax elements and shall be the same otherwise: profile_idc, constraint_setX_flag (with X being equal to 0 to 5, inclusive), reserved_zero_2bits, level_idc, seq_parameter_set_id, timing_info_present_flag, num_units_in_tick, time_scale, fixed_frame_rate_flag, nal_hrd_parameters_present_flag, vcl_hrd_parameters_present_flag, low_delay_hrd_flag, pic_struct_present_flag, and the hrd_parameters( ) syntax structures.

– When spsA is the active SVC sequence parameter set RBSP and spsB is the SVC sequence parameter set RBSP that is referred to by the coded slice NAL units of the layer representation with DQId equal to DQIdMax, the level specified by level_idc (or level_idc and constraint_set3_flag) in spsA shall not be less than the level specified by level_idc (or level_idc and constraint_set3_flag) in spsB.

– When the seq_parameter_set_svc_extension( ) syntax structure is present in both spsA and spsB, the values of all syntax elements in the seq_parameter_set_svc_extension( ) syntax structure shall be the same.

It is a requirement of bitstream conformance that the following constraints are obeyed:

– For each particular value of DQId, all coded slice NAL units of a coded video sequence shall refer to the same value of seq_parameter_set_id (via the picture parameter set RBSP that is referred to by the value of pic_parameter_set_id).

– The value of seq_parameter_set_id in a buffering period SEI message that is not included in a scalable nesting SEI message shall be identical to the value of seq_parameter_set_id in the picture parameter set RBSP that is referred to by coded slice NAL units with nal_unit_type equal to 1 or 5 (via the value of pic_parameter_set_id) in the same access unit.

– The value of seq_parameter_set_id in a buffering period SEI message that is included in a scalable nesting SEI message and is associated with a particular value of DQId shall be identical to the value of seq_parameter_set_id in the picture parameter set RBSP that is referred to by coded slice NAL units with the particular value of DQId (via the value of pic_parameter_set_id) in the same access unit.

The active layer SVC sequence parameter set RBSPs for different values of DQId may be the same SVC sequence parameter set RBSP. The active SVC sequence parameter set RBSP and an active layer SVC sequence parameter set RBSP for a particular value of DQId may be the same SVC sequence parameter set RBSP.

When the active SVC sequence parameter set RBSP for a coded picture is a sequence parameter set RBSP, any sequence parameter set RBSP with the value of seq_parameter_set_id for the active SVC sequence parameter set RBSP for the coded picture shall have the same content as that of the active SVC sequence parameter set RBSP for the coded picture unless it follows the last access unit of the coded video sequence containing the coded picture and precedes the first VCL NAL unit and the first SEI NAL unit containing a buffering period SEI message (when present) of another coded video sequence.

When the active SVC sequence parameter set RBSP for a coded picture is a subset sequence parameter set RBSP, any subset sequence parameter set RBSP with the value of seq_parameter_set_id for the active SVC sequence parameter set RBSP for the coded picture shall have the same content as that of the active SVC sequence parameter set RBSP for the coded picture unless it follows the last access unit of the coded video sequence containing the coded picture and precedes the first VCL NAL unit and the first SEI NAL unit containing a buffering period SEI message (when present) of another coded video sequence.

For each particular value of DQId, the following applies:

– When the active layer SVC sequence parameter set RBSP for a coded picture is a sequence parameter set RBSP, any sequence parameter set RBSP with the value of seq_parameter_set_id for the active layer SVC sequence parameter set RBSP for the coded picture shall have the same content as that of the active layer SVC sequence parameter set RBSP for the coded picture unless it follows the last access unit of the coded video sequence containing the coded picture and precedes the first VCL NAL unit and the first SEI NAL unit containing a buffering period SEI message (when present) of another coded video sequence.

– When the active layer SVC sequence parameter set RBSP for a coded picture is a subset sequence parameter set RBSP, any subset sequence parameter set RBSP with the value of seq_parameter_set_id for the active layer SVC sequence parameter set RBSP for the coded picture shall have the same content as that of the active layer SVC sequence parameter set RBSP for the coded picture unless it follows the last access unit of the coded video sequence containing the coded picture and precedes the first VCL NAL unit and the first SEI NAL unit containing a buffering period SEI message (when present) of another coded video sequence.

NOTE 6 – If picture parameter set RBSP or SVC sequence parameter set RBSP are conveyed within the bitstream, these constraints impose an order constraint on the NAL units that contain the picture parameter set RBSP or SVC sequence parameter set RBSP, respectively. Otherwise (picture parameter set RBSP or SVC sequence parameter set RBSP are conveyed by other means not specified in this Recommendation | International Standard), they must be available to the decoding process in a timely fashion such that these constraints are obeyed.

When a sequence parameter set NAL unit with a particular value of seq_parameter_set_id is received, its content replaces the content of the previous sequence parameter set NAL unit, in decoding order, with the same value of seq_parameter_set_id (when a previous sequence parameter set NAL unit with the same value of seq_parameter_set_id was present in the bitstream). When a subset sequence parameter set NAL unit with a particular value of seq_parameter_set_id is received, its content replaces the content of the previous subset sequence parameter set NAL unit, in decoding order, with the same value of seq_parameter_set_id (when a previous subset sequence parameter set NAL unit with the same value of seq_parameter_set_id was present in the bitstream).

NOTE 7 – A decoder must be capable of simultaneously storing the contents of the sequence parameter sets and subset sequence parameter sets for all values of seq_parameter_set_id. The content of the sequence parameter set with a particular value of seq_parameter_set_id is overwritten when a new sequence parameter set NAL unit with the same value of seq_parameter_set_id is received, and the content of the subset sequence parameter set with a particular value of seq_parameter_set_id is overwritten when a new subset sequence parameter set NAL unit with the same value of seq_parameter_set_id is received.

When present, a sequence parameter set extension RBSP includes parameters having a similar function to those of a sequence parameter set RBSP. For purposes of establishing constraints on the syntax elements of the sequence parameter set extension RBSP and for purposes of determining activation of a sequence parameter set extension RBSP, the sequence parameter set extension RBSP shall be considered part of the preceding sequence parameter set RBSP with the same value of seq_parameter_set_id. When a sequence parameter set RBSP is present that is not followed by a sequence parameter set extension RBSP with the same value of seq_parameter_set_id prior to the activation of the sequence parameter set RBSP, the sequence parameter set extension RBSP and its syntax elements shall be considered not present for the active SVC sequence parameter set RBSP. The contents of sequence parameter set extension RBSPs only apply when the base layer, which conforms to one or more of the profiles specified in Annex A, of a coded video sequence conforming to one or more of the profiles specified in Annex G is decoded. Subset sequence parameter set RBSPs shall not be followed by a sequence parameter set extension RBSP.

NOTE 8 – Sequence parameter sets extension RBSPs are not considered to be part of a subset sequence parameter set RBSP and subset sequence parameter set RBSPs must not be followed by a sequence parameter set extension RBSP.

For layer representations with DQId equal to DQIdMax, all constraints that are expressed on the relationship between the values of the syntax elements (and the values of variables derived from those syntax elements) in SVC sequence parameter sets and picture parameter sets and other syntax elements are expressions of constraints that apply only to the active SVC sequence parameter set and the active picture parameter set. For layer representations with a particular value of DQId less than DQIdMax, all constraints that are expressed on the relationship between the values of the syntax elements (and the values of variables derived from those syntax elements) in SVC sequence parameter sets and picture parameter sets and other syntax elements are expressions of constraints that apply only to the active layer SVC sequence parameter set and the active layer picture parameter set for the particular value of DQId. If any SVC sequence parameter set RBSP having profile_idc equal to one of the profile_idc values specified in Annex A or G is present that is never activated in the bitstream (i.e., it never becomes the active SVC sequence parameter set or an active layer SVC sequence parameter set), its syntax elements shall have values that would conform to the specified constraints if it were activated by reference in an otherwise-conforming bitstream. If any picture parameter set RBSP is present that is never activated in the bitstream (i.e., it never becomes the active picture parameter set or an active layer picture parameter set), its syntax elements shall have values that would conform to the specified constraints if it were activated by reference in an otherwise-conforming bitstream.

During operation of the decoding process (see subclause G.8), for layer representations with DQId equal to DQIdMax, the values of parameters of the active picture parameter set and the active SVC sequence parameter set shall be considered in effect. For layer representations with a particular value of DQId less than DQIdMax, the values of the parameters of the active layer picture parameter set and the active layer SVC sequence parameter set for the particular value of DQId shall be considered in effect. For interpretation of SEI messages that apply to access units or dependency representations with dependency_id equal to DependencyIdMax or layer representation with DQId equal to DQIdMax, the values of the parameters of the active picture parameter set and the active SVC sequence parameter set for the access unit shall be considered in effect unless otherwise specified in the SEI message semantics. For interpretation of SEI messages that apply to dependency representations with a particular value of dependency_id less than DependencyIdMax, the values of the parameters of the active layer picture parameter set and the active layer SVC sequence parameter set for the layer representation with DQId equal to (dependency_id << 4) of the access unit shall be considered in effect unless otherwise specified in the SEI message semantics. For interpretation of SEI messages that apply to layer representations with a particular value of DQId less than DQIdMax, the values of the parameters of the active layer picture parameter set and the active layer SVC sequence parameter set for the layer representation with the particular value of DQId of the access unit shall be considered in effect unless otherwise specified in the SEI message semantics.

G.7.4.1.2.2 Order of access units and association to coded video sequences

The specification of subclause 7.4.1.2.2 applies with the following modifications.

The first access unit of the bitstream shall only contain coded slice NAL units with nal_unit_type equal to 5 or idr_flag equal to 1.

The order of NAL units and coded pictures and their association to access units is described in subclause G.7.4.1.2.3.

G.7.4.1.2.3 Order of NAL units and coded pictures and association to access units

The specification of subclause 7.4.1.2.3 applies with the following modifications.

NOTE – Some bitstreams that conform to one or more profiles specified in this annex do not conform to any profile specified in Annex A (prior to operation of the base layer extraction process specified in subclause G.8.8.2). As specified in subclauses 7.4.1 and 7.4.1.2.3, for the profiles specified in Annex A, NAL units with nal_unit_type equal to 20 are classified as non-VCL NAL units that must be preceded within each access unit by at least one NAL unit with nal_unit_type in the range of 1 to 5, inclusive. For this reason, any bitstream that conforms to one or more profiles specified in this annex does not conform to any profile specified in Annex A when it contains any of the following:

- any access unit that does not contain any NAL units with nal_unit_type equal to 1 or 5, but contains one or more NAL units with nal_unit_type equal to 6, 7, 8, 9, or 15;
- any access unit in which one or more NAL units with nal_unit_type equal to 7, 8, or 15 is present after the last NAL unit in the access unit with nal_unit_type equal to 1 or 5.

The association of VCL NAL units to primary or redundant coded pictures is specified in subclause G.7.4.1.2.5. When the primary coded picture does not contain a layer representation with a particular value of DQId, all redundant coded pictures (when present) in the same access unit shall not contain a layer representation with the particular value of DQId.

The constraints for the detection of the first VCL NAL unit of a primary coded picture are specified in subclause G.7.4.1.2.4.

The constraint expressed in subclause 7.4.1.2.3 on the order of a buffering period SEI message is replaced by the following constraints.

- When an SEI NAL unit containing a buffering period SEI message is present, the following applies:
  - If the buffering period SEI message is the only buffering period SEI message in the access unit and it is not included in a scalable nesting SEI message, the buffering period SEI message shall be the first SEI message payload of the first SEI NAL unit in the access unit.
  - Otherwise (the buffering period SEI message is not the only buffering period SEI message in the access unit or it is included in a scalable nesting SEI message), the following constraints are specified:
    - When a buffering period SEI message that is not included in a scalable nesting SEI message is present, this buffering period SEI message shall be the only SEI message payload of the first SEI NAL unit in the access unit.
    - A scalable nesting SEI message that includes a buffering period SEI message shall not include any other SEI messages and the scalable nesting SEI message that includes a buffering period SEI message shall be the only SEI message inside an SEI NAL unit.
    - All SEI NAL units that precede an SEI NAL unit that contains a scalable nesting SEI message with a buffering period SEI message as payload in an access unit shall only contain buffering period SEI messages or scalable nesting SEI messages with a buffering period SEI message as payload.
    - When present, a scalable nesting SEI message with all_layer_representations_in_au_flag equal to 1 and a buffering period SEI message as payload shall be the first scalable nesting SEI message in an access unit.

– Any scalable nesting SEI message with a buffering period SEI message as payload that immediately precedes another scalable nesting SEI message with a buffering period SEI message as payload shall have values of 128 * sei_dependency_id[i]) + 8 * sei_quality_id[i] + sei_temporal_id, for all present i, that are less than any of the values of 128 * sei_dependency_id[i] + 8 * sei_quality_id[i] + sei_temporal_id in the immediately following scalable nesting SEI message with a buffering period SEI message as payload.

The following additional constraints shall be obeyed:

– Each NAL unit with nal_unit_type equal to 1 or 5 shall be immediately preceded by a prefix NAL unit.

– In bitstreams conforming to this Recommendation | International Standard, each prefix NAL unit shall be immediately followed by a NAL unit with nal_unit_type equal to 1 or 5.

G.7.4.1.2.4 Detection of the first VCL NAL unit of a primary coded picture

This subclause specifies constraints on VCL NAL unit syntax that are sufficient to enable the detection of the first VCL NAL unit of each primary coded picture.

The first VCL NAL unit of the primary coded picture of the current access unit, in decoding order, shall be different from the last VCL NAL unit of the primary coded picture of the previous access unit, in decoding order, in one or more of the following ways:

– dependency_id of the first VCL NAL unit of the primary coded picture of the current access unit is less than dependency_id of the last VCL NAL unit of the primary coded picture of the previous access unit

– dependency_id of the first VCL NAL unit of the primary coded picture of the current access unit is equal to dependency_id of the last VCL NAL unit of the primary coded picture of the previous access unit and any of the following conditions are true

  – quality_id of the first VCL NAL unit of the primary coded picture of the current access unit is less than quality_id of the last VCL NAL unit of the primary coded picture of the previous access unit

  – quality_id of the first VCL NAL unit of the primary coded picture of the current access unit and the last VCL NAL unit of the primary coded picture of the previous access unit is equal to 0, and any of the conditions specified in subclause 7.4.1.2.4 is fulfilled

G.7.4.1.2.5 Order of VCL NAL units and association to coded pictures

Each VCL NAL unit is part of a coded picture.

Let dId be the value of dependency_id and let qId be the value of quality_id of any particular VCL NAL unit. The order of the VCL NAL units within a coded picture is constrained as follows:

– For all VCL NAL units following this particular VCL NAL unit, the value of dependency_id shall be greater than or equal to dId.

– For all VCL NAL units with a value of dependency_id equal to dId following this particular VCL NAL unit, the value of quality_id shall be greater than or equal to qId.

For each set of VCL NAL units within a layer representation, the following applies:

– If arbitrary slice order, as specified in Annex A or subclause G.10, is allowed, coded slice NAL units of a layer representation may have any order relative to each other.

– Otherwise (arbitrary slice order is not allowed), coded slice NAL units of a slice group shall not be interleaved with coded slice NAL units of another slice group and the order of coded slice NAL units within a slice group shall be in the order of increasing macroblock address for the first macroblock of each coded slice NAL unit of the same slice group.

NAL units having nal_unit_type equal to 12 may be present in the access unit but shall not precede the first VCL NAL unit of the primary coded picture within the access unit.

NAL units having nal_unit_type equal to 0 or in the range of 24 to 31, inclusive, which are unspecified, may be present in the access unit but shall not precede the first VCL NAL unit of the primary coded picture within the access unit.

NAL units having nal_unit_type in the range of 21 to 23, inclusive, which are reserved, shall not precede the first VCL NAL unit of the primary coded picture within the access unit (when specified in the future by ITU-T | ISO/IEC).

G.7.4.2 Raw byte sequence payloads and RBSP trailing bits semantics

G.7.4.2.1 Sequence parameter set RBSP semantics

The semantics specified in subclause 7.4.2.1 apply.

G.7.4.2.1.1 Sequence parameter set data semantics

The semantics specified in subclause 7.4.2.1.1 apply with substituting SVC sequence parameter set for sequence parameter set. Additionally, the following applies.

profile_idc and **level_idc** indicate the profile and level to which the coded video sequence conforms when the SVC sequence parameter set is the active SVC sequence parameter set.

constraint_set0_flag is specified as follows:

– If the sequence parameter set data syntax structure is included in a sequence parameter set RBSP, the semantics specified in subclause 7.4.2.1.1 apply.

– Otherwise (the sequence parameter set data syntax structure is included in a subset sequence parameter set RBSP), constraint_set0_flag equal to 1 specifies that all of the following conditions are obeyed:

  – the coded video sequence obeys all constraints specified in subclause G.10.1.1,

  – the output of the decoding process as specified in subclause G.8 is identical to the output of the decoding process that is obtained when profile_idc would be set equal to 83.

  constraint_set0_flag equal to 0 specifies that the coded video sequence may or may not obey all constraints specified in subclause G.10.1.1 and that the output of the decoding process as specified in subclause G.8 may or may not be identical to the output of the decoding process that is obtained when profile_idc would be set equal to 83.

  NOTE 1 – The output of the decoding process may be different, if the array sTCoeff contains non-zero scaled luma transform coefficient values for a transform block of a macroblock that is coded in an Inter macroblock prediction mode, but all reconstructed luma residual samples of the array rS_L that are associated with the transform blocks are equal to 0. In this case, the boundary filter strength that is derived as specified in subclause G.8.7.4.3 can depend on the value of profile_idc.

constraint_set1_flag is specified as follows:

– If the sequence parameter set data syntax structure is included in a sequence parameter set RBSP, the semantics specified in subclause 7.4.2.1.1 apply.

– Otherwise (the sequence parameter set data syntax structure is included in a subset sequence parameter set RBSP), constraint_set1_flag equal to 1 specifies that all of the following conditions are obeyed:

  – the coded video sequence obeys all constraints specified in subclause G.10.1.2,

  – the output of the decoding process as specified in subclause G.8 is identical to the output of the decoding process that is obtained when profile_idc would be set equal to 86.

  constraint_set1_flag equal to 0 specifies that the coded video sequence may or may not obey all constraints specified in subclause G.10.1.2 and that the output of the decoding process as specified in subclause G.8 may or may not be identical to the output of the decoding process that is obtained when profile_idc would be set equal to 86.

  NOTE 2 – The output of the decoding process may be different, if the array sTCoeff contains non-zero scaled luma transform coefficient values for a transform block of a macroblock that is coded in an Inter macroblock prediction mode, but all reconstructed luma residual samples of the array rS_L that are associated with the transform blocks are equal to 0. In this case, the boundary filter strength that is derived as specified in subclause G.8.7.4.3 can depend on the value of profile_idc.

constraint_set2_flag is specified as follows:

– If the sequence parameter set data syntax structure is included in a sequence parameter set RBSP, the semantics specified in subclause 7.4.2.1.1 apply.

– Otherwise (the sequence parameter set data syntax structure is included in a subset sequence parameter set RBSP), the value of 1 for constraint_set2_flag is reserved for future use by ITU-T | ISO/IEC. constraint_set2_flag shall be equal to 0 for coded video sequences with profile_idc equal to 83 and 86 in bitstreams conforming to this Recommendation | International Standard. Decoders shall ignore the value of constraint_set2_flag when profile_idc is equal to 83 or 86.

constraint_set3_flag is specified as follows:

– If the sequence parameter set data syntax structure is included in a sequence parameter set RBSP, the semantics specified in subclause 7.4.2.1.1 apply.

– Otherwise (the sequence parameter set data syntax structure is included in a subset sequence parameter set RBSP), the following applies:
  – If profile_idc is equal to 86, constraint_set3_flag equal to 1 specifies that the coded video sequence obeys all constraints specified in subclause G.10.1.3, and constraint_set3_flag equal to 0 specifies that the coded video sequence may or may not obey these corresponding constraints.
  – Otherwise (profile_idc is not equal to 86), the value of 1 for constraint_set3_flag is reserved for future use by ITU-T | ISO/IEC. constraint_set3_flag shall be equal to 0 for coded video sequences with profile_idc not equal to 86 in bitstreams conforming to this Recommendation | International Standard. Decoders shall ignore the value of constraint_set3_flag when profile_idc is not equal to 86.

The value of separate_colour_plane_flag shall be equal to 0 and the value of qpprime_y_zero_transform_bypass_flag shall be equal to 0.

When the seq_parameter_set_data( ) syntax structure is present in a subset sequence parameter set RBSP and vui_parameters_present_flag is equal to 1, timing_info_present_flag shall be equal to 0, nal_hrd_parameters_present_flag shall be equal to 0, vcl_hrd_parameters_present_flag shall be equal to 0, and pic_struct_present_flag shall be equal to 0. The value of 1 for timing_info_present_flag, nal_hrd_parameters_present_flag, vcl_hrd_parameters_present_flag, and pic_struct_present_flag for subset sequence parameter set RBSPs is reserved for future use by ITU-T | ISO/IEC. When timing_info_present_flag is equal to 1, decoders shall ignore the values of the directly following num_units_in_tick, time_scale, fixed_frame_rate_flag syntax elements. When nal_hrd_parameters_present_flag is equal to 1, decoders shall ignore the value of the syntax elements in the directly following hrd_parameters( ) syntax structure. When vcl_hrd_parameters_present_flag is equal to 1, decoders shall ignore the value of the syntax elements in the directly following hrd_parameters( ) syntax structure.

When the seq_parameter_set_data( ) syntax structure is present in a sequence parameter set RBSP and vui_parameters_present_flag is equal to 1, the values of timing_info_present_flag, num_units_in_tick, time_scale, fixed_frame_rate_flag, nal_hrd_parameters_present_flag, vcl_hrd_parameters_present_flag, low_delay_hrd_flag, pic_struct_present_flag and the values of syntax elements included in the hrd_parameters( ) syntax structures, when present, shall be such that the bitstream activating the sequence parameter set is conforming to one or more of the profiles specified in Annex A.

max_num_ref_frames specifies the maximum number of short-term and long-term reference frames, complementary reference field pairs, and non-paired reference fields that may be used by the decoding process for inter prediction of any picture in the coded video sequence. max_num_ref_frames also determines the size of the sliding window operation as specified in subclause G.8.2.4.2. The value of max_num_ref_frames shall be in the range of 0 to MaxDpbFrames (as specified in subclause G.10), inclusive.

The allowed range of values for pic_width_in_mbs_minus1, pic_height_in_map_units_minus1, and frame_mbs_only_flag is specified by constraints in subclause G.10.

G.7.4.2.1.1.1 Scaling list semantics

The semantics specified in subclause 7.4.2.1.1.1 apply.

G.7.4.2.1.2 Sequence parameter set extension RBSP semantics

The semantics specified in subclause 7.4.2.1.2 apply. Additionally, the following applies.

Sequence parameter set extension RBSPs can only follow sequence parameter set RBSPs in decoding order. Subset sequence parameter set RBSPs shall not be followed by a sequence parameter set extension RBSP. The contents of sequence parameter set extension RBSPs only apply when the base layer, which conforms to one or more of the profiles specified in Annex A, of a coded video sequence conforming to one or more of the profiles specified in Annex G is decoded.

G.7.4.2.1.3 Subset sequence parameter set RBSP semantics

The semantics specified in subclause 7.4.2.1.3 apply.

G.7.4.2.1.4 Sequence parameter set SVC extension semantics

inter_layer_deblocking_filter_control_present_flag equal to 1 specifies that a set of syntax elements controlling the characteristics of the deblocking filter for inter-layer prediction is present in the slice header. inter_layer_deblocking_filter_control_present_flag equal to 0 specifies that the set of syntax elements controlling the characteristics of the deblocking filter for inter-layer prediction is not present in the slice headers and their inferred values are in effect.

extended_spatial_scalability_idc specifies the presence of syntax elements related to geometrical parameters for the resampling processes. The value of extended_spatial_scalability_idc shall be in the range of 0 to 2, inclusive, and the following applies:

- If extended_spatial_scalability_idc is equal to 0, no geometrical parameters are present in the subset sequence parameter set and the slice headers referring to this subset sequence parameter set.
- Otherwise, if extended_spatial_scalability_idc is equal to 1, geometrical parameters are present in the subset sequence parameter set, but not in the slice headers referring to this subset sequence parameter set.
- Otherwise (extended_spatial_scalability_idc is equal to 2), geometrical parameters are not present in the subset sequence parameter set, but they are present in the slice headers with no_inter_layer_pred_flag equal to 0 and quality_id equal to 0 that refer to this subset sequence parameter set.

chroma_phase_x_plus1_flag specifies the horizontal phase shift of the chroma components in units of half luma samples of a frame or layer frame. When chroma_phase_x_plus1_flag is not present, it shall be inferred to be equal to 1.

When ChromaArrayType is equal to 1 and chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field are present, the following applies:

- If chroma_phase_x_plus1_flag is equal to 0, chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field should be equal to 0, 2, or 4.
- Otherwise (chroma_phase_x_plus1_flag is equal to 1), chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field should be equal to 1, 3, or 5.

When ChromaArrayType is equal to 2, chroma_phase_x_plus1_flag should be equal to 1.

chroma_phase_y_plus1 specifies the vertical phase shift of the chroma components in units of half luma samples of a frame or layer frame. When chroma_phase_y_plus1 is not present, it shall be inferred to be equal to 1. The value of chroma_phase_y_plus1 shall be in the range of 0 to 2, inclusive.

When ChromaArrayType is equal to 1 and chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field are present, the following applies:

- If chroma_phase_y_plus1 is equal to 0, chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field should be equal to 2 or 3.
- Otherwise, if chroma_phase_y_plus1 is equal to 1, chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field should be equal to 0 or 1.
- Otherwise (chroma_phase_y_plus1 is equal to 2), chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field should be equal to 4 or 5.

seq_ref_layer_chroma_phase_x_plus1_flag specifies the horizontal phase shift of the chroma components in units of half luma samples of a layer frame for the layer pictures that may be used for inter-layer prediction. When seq_ref_layer_chroma_phase_x_plus1_flag is not present, it shall be inferred to be equal to chroma_phase_x_plus1_flag.

seq_ref_layer_chroma_phase_y_plus1 specifies the vertical phase shift of the chroma components in units of half luma samples of a layer frame for the layer pictures that may be used for inter-layer prediction. When seq_ref_layer_chroma_phase_y_plus1 is not present, it shall be inferred to be equal to chroma_phase_y_plus1. The value of seq_ref_layer_chroma_phase_y_plus1 shall be in the range of 0 to 2, inclusive.

seq_scaled_ref_layer_left_offset specifies the horizontal offset between the upper-left luma sample of a resampled layer picture used for inter-layer prediction and the upper-left luma sample of the current picture or current layer picture in units of two luma samples. When seq_scaled_ref_layer_left_offset is not present, it shall be inferred to be equal to 0. The value of seq_scaled_ref_layer_left_offset shall be in the range of -2^{15} to $2^{15} - 1$, inclusive.

seq_scaled_ref_layer_top_offset specifies the vertical offset between the upper-left luma sample of a resampled layer picture used for inter-layer prediction and the upper-left luma sample of the current picture or current layer picture. Depending on the value of frame_mbs_only_flag, the following applies:

- If frame_mbs_only_flag is equal to 1, the vertical offset is specified in units of two luma samples.
- Otherwise (frame_mbs_only_flag is equal to 0), the vertical offset is specified in units of four luma samples.

When seq_scaled_ref_layer_top_offset is not present, it shall be inferred to be equal to 0. The value of seq_scaled_ref_layer_top_offset shall be in the range of -2^{15} to $2^{15} - 1$, inclusive.

seq_scaled_ref_layer_right_offset specifies the horizontal offset between the bottom-right luma sample of a resampled layer picture used for inter-layer prediction and the bottom-right luma sample of the current picture or current layer picture in units of two luma samples. When seq_scaled_ref_layer_right_offset is not present, it shall be inferred to be equal to 0. The value of seq_scaled_ref_layer_right_offset shall be in the range of -2^{15} to $2^{15}-1$, inclusive.

seq_scaled_ref_layer_bottom_offset specifies the vertical offset between the bottom-right luma sample of a resampled layer picture used for inter-layer prediction and the bottom-right luma sample of the current picture or current layer picture. Depending on the value of frame_mbs_only_flag, the following applies:

– If frame_mbs_only_flag is equal to 1, the vertical offset is specified in units of two luma samples.

– Otherwise (frame_mbs_only_flag is equal to 0), the vertical offset is specified in units of four luma samples.

When seq_scaled_ref_layer_bottom_offset is not present, it shall be inferred to be equal to 0. The value of seq_scaled_ref_layer_bottom_offset shall be in the range of -2^{15} to $2^{15}-1$, inclusive.

seq_tcoeff_level_prediction_flag specifies the presence of the syntax element adaptive_tcoeff_level_prediction_flag in the subset sequence parameter set.

adaptive_tcoeff_level_prediction_flag specifies the presence of tcoeff_level_prediction_flag in slice headers that refer to the subset sequence parameter set. When adaptive_tcoeff_level_prediction_flag is not present, it shall be inferred to be equal to 0.

slice_header_restriction_flag specifies the presence of syntax elements in slice headers that refer to the subset sequence parameter set.

G.7.4.2.2 Picture parameter set RBSP semantics

The semantics specified in subclause 7.4.2.2 apply with substituting "SVC sequence parameter set" for "sequence parameter set" and substituting "active SVC sequence parameter set or active layer SVC sequence parameter set" for "active sequence parameter set". Additionally, the following applies.

num_slice_groups_minus1 plus 1 specifies the number of slice groups for a picture. When num_slice_groups_minus1 is equal to 0, all slices of the picture belong to the same slice group. The allowed range of num_slice_groups_minus1 is specified in subclause G.10.

G.7.4.2.3 Supplemental enhancement information RBSP semantics

The semantics specified in subclause 7.4.2.3 apply.

G.7.4.2.3.1 Supplemental enhancement information message semantics

The semantics specified in subclause 7.4.2.3.1 apply.

G.7.4.2.4 Access unit delimiter RBSP semantics

The semantics specified in subclause 7.4.2.4 apply.

NOTE – The value of primary_pic_type applies to the slice_type values in all slice headers of the primary coded picture, including the slice_type syntax elements in all NAL units with nal_unit_type equal to 1, 5, or 20. NAL units with nal_unit_type equal to 2 are not present in bitstreams conforming to any of the profiles specified in this annex.

G.7.4.2.5 End of sequence RBSP semantics

The end of sequence RBSP specifies that the next subsequent access unit in the bitstream in decoding order (if any) shall be an access unit for which all layer representation of the primary coded picture have IdrPicFlag equal to 1. The syntax content of the SODB and RBSP for the end of sequence RBSP are empty. No normative decoding process is specified for an end of sequence RBSP.

G.7.4.2.6 End of stream RBSP semantics

The semantics specified in subclause 7.4.2.6 apply.

G.7.4.2.7 Filler data RBSP semantics

The semantics specified in subclause 7.4.2.7 apply with the following addition.

Filler data NAL units shall be considered to contain the syntax elements dependency_id, quality_id, temporal_id, and priority_id with values that are inferred as follows:

1. Let prevSvcNalUnit be the most recent NAL unit in decoding order that has nal_unit_type equal to 14 or 20.

NOTE – The most recent NAL unit in decoding order with nal_unit_type equal to 14 or 20 always belongs to the same access unit as the filler data NAL unit.

2. The values of dependency_id, quality_id, temporal_id, and priority_id for the filler data NAL unit are inferred to be equal to the values of dependency_id, quality_id, temporal_id, and priority_id, respectively, of the NAL unit prevSvcNalUnit.

G.7.4.2.8 Slice layer without partitioning RBSP semantics

The semantics specified in subclause 7.4.2.8 apply.

G.7.4.2.9 Slice data partition RBSP semantics

Slice data partition syntax is not present in bitstreams conforming to any of the profiles specified in Annex G.

G.7.4.2.10 RBSP slice trailing bits semantics

The semantics specified in subclause 7.4.2.10 apply with the following modifications.

Let NumBytesInVclNALunits be the sum of the values of NumBytesInNALunit for all VCL NAL units of a layer representation and let BinCountsInNALunits be the number of times that the parsing process function DecodeBin( ), specified in subclause 9.3.3.2, is invoked to decode the contents of all VCL NAL units of the layer representation. When entropy_coding_mode_flag is equal to 1, it is a requirement of bitstream conformance that BinCountsInNALunits shall not exceed $(32 \div 3) * \text{NumBytesInVclNALunits} + (\text{RawMbBits} * \text{PicSizeInMbs}) \div 32$.

NOTE – The constraint on the maximum number of bins resulting from decoding the contents of the slice layer NAL units of a layer representation can be met by inserting a number of cabac_zero_word syntax elements to increase the value of NumBytesInVclNALunits. Each cabac_zero_word is represented in a NAL unit by the three-byte sequence 0x000003 (as a result of the constraints on NAL unit contents that result in requiring inclusion of an emulation_prevention_three_byte for each cabac_zero_word).

G.7.4.2.11 RBSP trailing bits semantics

The semantics specified in subclause 7.4.2.11 apply.

G.7.4.2.12 Prefix NAL unit RBSP semantics

The semantics specified in subclause 7.4.2.12 apply.

G.7.4.2.12.1 Prefix NAL unit SVC semantics

The syntax element store_ref_base_pic_flag is considered as if it was present in the associated NAL unit.

store_ref_base_pic_flag equal to 1 specifies that, when the value of dependency_id as specified in the NAL unit header is equal to the maximum value of dependency_id for the VCL NAL units of the current coded picture, an additional representation of the coded picture that may or may not be identical to the decoded picture is marked as "used for reference". This additional representation is also referred to as reference base picture and may be used for inter prediction of following pictures in decoding order, but it is not output. When store_ref_base_pic_flag is not present, it shall be inferred to be equal to 0.

The syntax element store_ref_base_pic_flag shall have the same value for all VCL NAL units of a dependency representation. When nal_ref_idc is equal to 0, store_ref_base_pic_flag shall be equal to 0.

When max_num_ref_frames is less than 2 in the SVC sequence parameter set that is referred to by the associated NAL unit, store_ref_base_pic_flag shall be equal to 0.

additional_prefix_nal_unit_extension_flag equal to 0 specifies that the prefix_nal_unit_svc( ) syntax structure does not contain any additional_prefix_nal_unit_extension_data_flag syntax elements. additional_prefix_nal_unit_extension_flag shall be equal to 0 in bitstreams conforming to this Recommendation | International Standard. The value of 1 for additional_prefix_nal_unit_extension_flag is reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore all data that follow the value 1 for additional_prefix_nal_unit_extension_flag in a prefix_nal_unit_svc( ) syntax structure.

additional_prefix_nal_unit_extension_data_flag may have any value.

NOTE – The syntax elements additional_prefix_nal_unit_extension_flag and additional_prefix_nal_unit_extension_data_flag are not used by the decoding process specified in this Recommendation | International Standard.

G.7.4.2.13 Slice layer extension RBSP semantics

The semantics specified in subclause 7.4.2.13 apply.

G.7.4.3 Slice header semantics

The semantics specified in subclause 7.4.3 apply with the following modifications.

a) All referenced syntax elements and variables are syntax elements and variables for the dependency representation with dependency_id equal to 0.

b) A frame, field, top field, bottom field, picture, and decoded picture is interpreted as layer frame, layer field, layer top field, layer bottom field, layer picture, and decoded layer picture, respectively, that represent an intermediate decoding result for the dependency representation with dependency_id equal to 0.

c) An IDR picture is interpreted as layer picture with IdrPicFlag equal to 1 for the dependency representation with dependency_id equal to 0.

d) An IDR access unit is interpreted as an access unit containing a primary coded picture with IdrPicFlag equal to 1 for the dependency representation with dependency_id equal to 0.

e) A reference frame, reference field, and reference picture is interpreted as layer frame, layer field, and layer picture with nal_ref_idc greater than 0 for the dependency representation with dependency_id equal to 0.

f) A non-reference frame, non-reference field, and non-reference picture is interpreted as layer frame, layer field, and layer picture with nal_ref_idc equal to 0 for the dependency representation with dependency_id equal to 0.

g) All constraints specified in subclause 7.4.3 apply only to layer representations with DQId equal to 0.

h) The slice_header( ) syntax structure shall be considered to contain the following syntax elements with the following inferred values:

   – ref_layer_dq_id is inferred to be equal to −1.

   – scan_idx_start is inferred to be equal to 0.

   – scan_idx_end is inferred to be equal to 15.

i) References to the decoded reference picture marking process as specified in subclause 8.2.5 are replaced with reference to the SVC decoded reference picture marking process as specified in subclause G.8.2.4.

j) The value of direct_spatial_mv_pred_flag shall be equal to 1.

k) The variable MaxRefLayerDQId is set equal to −1.

l) The variable CroppingChangeFlag is set equal to 0.

m) The variable SpatialResolutionChangeFlag is set equal to 0.

n) In the semantics of first_mb_in_slice, the reference to Annex A is substituted with a reference to subclause G.10.

G.7.4.3.1 Reference picture list modification semantics

The semantics specified in subclause 7.4.3.1 apply. For this specification, the modifications a) to f) specified in subclause G.7.4.3 apply. When quality_id is greater than 0, all syntax elements of the ref_pic_list_modification( ) syntax structure are inferred as specified in the beginning of subclause G.7.4.3.4.

G.7.4.3.2 Prediction weight table semantics

The semantics specified in subclause 7.4.3.2 apply. When quality_id is greater than 0, all syntax elements of the pred_weight_table( ) syntax structure are inferred as specified in the beginning of subclause G.7.4.3.4.

G.7.4.3.3 Decoded reference picture marking semantics

The semantics specified in subclause 7.4.3.3 apply with substituting "SVC sequence parameter set" for "sequence parameter set" and with considering the reference pictures marked as "reference base pictures" as not present. The constraints specified in subclause 7.4.3.3 apply only to the dependency representation with dependency_id equal to the current value of dependency_id and the modifications a) and b) specified in subclause G.8.2 apply with currDependencyId being equal to the current value of dependency_id.

When quality_id is greater than 0, all syntax elements of the dec_ref_pic_marking( ) syntax structure are inferred as specified in the beginning of subclause G.7.4.3.4.

In addition to the constraints specified in subclause 7.4.3.3, the following constraints are specified:

a) When decoding a frame, the dec_ref_pic_marking( ) syntax structure shall not contain a memory_management_control_operation command equal to 3 that assigns a long-term frame index to a complementary reference field pair (not marked as "reference base picture") when any of the following conditions are true (when processing the memory_management_control_operation command equal to 3):

   – there exists a non-paired reference base field (marked as "reference base picture") that is associated with one of the fields of the complementary reference field pair and that is marked as "used for reference",

   – there exists a complementary reference base field pair (marked as "reference base picture") that is associated with the complementary reference field pair and in which one field is marked as "used for reference" and the other field is marked as "unused for reference".

b) When decoding a field, the dec_ref_pic_marking( ) syntax structure shall not contain a memory_management_control_operation command equal to 3 that assigns a long-term frame index to a field (not marked as "reference base picture") of a reference frame or a complementary reference field pair when both of the following conditions are true (when processing the memory_management_control_operation command equal to 3):

   – the other field of the reference frame or complementary reference field pair is marked as "unused for reference",

   – there exists a reference base frame or a complementary reference base field pair (marked as "reference base picture") that is associated with the reference frame or complementary reference field pair, respectively, and in which both fields are marked as "used for reference".

c) When decoding the second field (in decoding order) of a complementary reference field pair, the dec_ref_pic_marking( ) syntax structure shall not contain a memory_management_control_operation command equal to 6 that assigns a long-term frame index to this field when both of the following conditions are true:

   – there exists a reference base field (marked as "reference base picture") that is associated with the first field of the complementary reference field pair and that is marked as "used for short-term reference" when the memory_management_control_operation command equal to 6 is processed,

   – the dec_ref_pic_marking( ) syntax structure does not contain a memory_management_control_operation command equal to 3 that assigns the same long-term frame index to the first field of the complementary reference field pair.

NOTE – The additional constraints specified above (in connection with the constraints specified in subclause 7.4.3.3) ensure that after processing all memory_management_control_operation commands of the decoded reference picture marking syntax structure the following applies, with reference entry being a collective term for a non-paired reference field, a reference frame, or a complementary reference field pair (not marked as "reference base picture") and reference base entry being a collective term for a non-paired reference base field, a reference base frame, or a complementary reference base field pair (marked as "reference base picture"): When one or more fields of a reference entry are marked as "used for reference" and there exists a reference base entry that is associated with the reference entry or one field of the reference entry and one or more fields of the reference base entry are marked as "used for reference", either all fields of the reference entry and the reference base entry that are marked as "used for reference" must be marked as "used for short-term reference" or all fields of the reference entry and the reference base entry that are marked as "used for reference" must be marked as "used for long-term reference". When these fields are marked as "used for long-term reference", the same value of long-term frame index must be assigned to all fields of the reference entry and the reference base entry that are marked as "used for reference".

G.7.4.3.4 Slice header in scalable extension semantics

Unless stated otherwise, for all references to subclause 7.4.3 inside this subclause, the following modifications apply.

a) All referenced syntax elements and variables are syntax elements and variables for the dependency representation with dependency_id equal to the current value of dependency_id.

b) A frame, field, top field, bottom field, picture, and decoded picture is interpreted as layer frame, layer field, layer top field, layer bottom field, layer picture, and decoded layer picture, respectively, that represent an intermediate decoding result for the dependency representation with dependency_id equal to the current value of dependency_id.

c) An IDR picture is interpreted as layer picture with IdrPicFlag equal to 1 for the dependency representation with dependency_id equal to the current value of dependency_id.

d) An IDR access unit is interpreted as an access unit containing a primary coded picture with IdrPicFlag equal to 1 for the dependency representation with dependency_id equal to the current value of dependency_id.

e) A reference frame, reference field, and reference picture is interpreted as layer frame, layer field, and layer picture with nal_ref_idc greater than 0 for the dependency representation with dependency_id equal to the current value of dependency_id.

f) A non-reference frame, non-reference field, and non-reference picture is interpreted as layer frame, layer field, and layer picture with nal_ref_idc equal to 0 for the dependency representation with dependency_id equal to the current value of dependency_id.

g) References to the decoded reference picture marking process as specified in subclause 8.2.5 are replaced with reference to the SVC decoded reference picture marking process as specified in subclause G.8.2.4.

When quality_id is greater than 0, the following syntax elements (which are not present) shall be inferred to be equal to the corresponding syntax elements of the slice header of the slice with dependency_id equal to the current value of dependency_id and quality_id equal to 0, in the same coded picture, that covers the macroblock with the macroblock address (first_mb_in_slice * (1 + MbaffFrameFlag)), when present in this slice: direct_spatial_mv_pred_flag, num_ref_idx_active_override_flag, num_ref_idx_l0_active_minus1, num_ref_idx_l1_active_minus1, all syntax elements of the syntax structure ref_pic_list_modification( ), base_pred_weight_table_flag, all syntax elements of the syntax structure pred_weight_table( ), all syntax elements of the syntax structure dec_ref_pic_marking( ), all syntax elements of the syntax structure dec_ref_base_pic_marking( ), and store_ref_base_pic_flag.

The value of the following SVC sequence parameter set syntax elements shall be the same across all coded slice NAL units of an access unit: bit_depth_luma_minus8, bit_depth_chroma_minus8, and chroma_format_idc.

The value of the following picture parameter set syntax elements shall be the same across all coded slice NAL units of a dependency representation: bottom_field_pic_order_in_frame_present_flag, num_ref_idx_l0_default_active_minus1, num_ref_idx_l1_default_active_minus1, weighted_pred_flag, and weighted_bipred_idc.

For all coded slice NAL units of a coded picture in which the syntax element field_pic_flag is present, field_pic_flag shall have the same value.

For all coded slice NAL units of a coded picture in which the syntax element bottom_field_flag is present, bottom_field_flag shall have the same value.

When present in any coded slice NAL unit of a dependency representation, the value of the following slice header syntax elements shall be the same across all slices of the dependency representation including slice headers of NAL units with nal_unit_type equal to 1 or 5: frame_num, idr_pic_id, pic_order_cnt_lsb, delta_pic_order_cnt_bottom, delta_pic_order_cnt[0], and delta_pic_order_cnt[1].

When present, the value of the following slice header syntax elements shall be the same across all slices of a layer representation: pic_parameter_set_id, ref_layer_dq_id, disable_inter_layer_deblocking_filter_idc, inter_layer_slice_alpha_c0_offset_div2, inter_layer_slice_beta_offset_div2, constrained_intra_resampling_flag, ref_layer_chroma_phase_x_plus1_flag, ref_layer_chroma_phase_y_plus1, scaled_ref_layer_left_offset, scaled_ref_layer_top_offset, scaled_ref_layer_right_offset, scaled_ref_layer_bottom_offset, slice_group_change_cycle, store_ref_base_pic_flag, tcoeff_level_prediction_flag, and all syntax elements of the syntax structures dec_ref_pic_marking( ) and dec_ref_base_pic_marking( ).

Let setOfRefLayerSlices be the set of slices with dependency_id equal to the current value of dependency_id and quality_id equal to 0, inside the current coded picture, that are covered or partly covered by the macroblocks of the current slice.

When quality_id is greater than 0, the value of (slice_type % 5) for all slices in the set setOfRefLayerSlices shall be the same as the value of (slice_type % 5) for the current slice.

When quality_id is greater than 0 and setOfRefLayerSlices contains more than one slice, the following constraints shall be obeyed:

a) When slice_type specifies an EP or EB slice, the value of num_ref_idx_l0_active_minus1 (either the value transmitted in the slice header when num_ref_idx_active_override_flag is equal to 1 or the inferred value when num_ref_idx_active_override_flag is equal to 0) shall be the same across all slices of the set setOfRefLayerSlices.

b) When slice_type specifies an EB slice, the value of num_ref_idx_l1_active_minus1 (either the value transmitted in the slice header when num_ref_idx_active_override_flag is equal to 1 or the inferred value when num_ref_idx_active_override_flag is equal to 0) shall be the same across all slices of the set setOfRefLayerSlices.

c) All elements of the syntax structure ref_pic_list_modification( ) shall be the same across all slices of the set setOfRefLayerSlices.

d) When slice_type specifies an EP slice, the following applies:

   i) When weighted_pred_flag is equal to 1, the value of base_pred_weight_table_flag shall be the same across all slices of the set setOfRefLayerSlices.

   ii) When weighted_pred_flag is equal to 1 and pred_weight_table( ) is present in the slices of the set setOfRefLayerSlices, the values of all syntax elements inside the syntax structure pred_weight_table( ) shall be the same across all slices of the set setOfRefLayerSlices.

e) When slice_type specifies an EB slice, the following applies:

   i) When weighted_bipred_idc is equal to 1, the value of base_pred_weight_table_flag shall be the same across all slices of the set setOfRefLayerSlices.

   ii) When weighted_bipred_idc is equal to 1 and pred_weight_table( ) is present in the slices of the set setOfRefLayerSlices, the values of all syntax elements inside the syntax structure pred_weight_table( ) shall be the same across all slices of the set setOfRefLayerSlices.

first_mb_in_slice has the same semantics as specified in subclause 7.4.3 with the term current picture being substituted by the term current layer representation and with the reference to Annex A being substituted by a reference to subclause G.10.

slice_type specifies the coding type of the slice according to Table G-1.

Table G-1 – Name association to slice_type for NAL units with nal_unit_type equal to 20.

slice_type	Name of slice_type
0, 5	EP (P slice in scalable extension)
1, 6	EB (B slice in scalable extension)
2, 7	EI (I slice in scalable extension)

When slice_type has a value in the range 5..7, it is a requirement of bitstream conformance that all other slices of the current layer representation shall have a value of slice_type equal to the current value of slice_type or equal to the current value of slice_type minus 5.

NOTE 1 – Values of slice_type in the range 5..7 can be used by an encoder to indicate that all slices of a layer representation have the same value of (slice_type % 5). Values of slice_type in the range 5..7 are otherwise equivalent to corresponding values in the range 0..2.

When idr_flag is equal to 1 or max_num_ref_frames is equal to 0, slice_type shall be equal to 2 or 7.

In the text (in particular when the clauses 7-9 are referenced in this annex), slices with (slice_type % 5) equal to 0, 1, and 2 may be collectively referred to as P, B, and I slices, respectively, regardless of whether the slices are coded using NAL units with nal_unit_type equal to 20 (slice_type is present in the slice_header_in_scalable_extension( ) syntax structure) or NAL units with nal_unit_type in the range of 1 to 5, inclusive (slice_type is present in the slice_header( ) syntax structure).

pic_parameter_set_id has the same semantics as specified in subclause 7.4.3.

colour_plane_id has the same semantics as specified in subclause 7.4.3.

frame_num is used as an identifier for dependency representations and shall be represented by log2_max_frame_num_minus4 + 4 bits in the bitstream.

frame_num is constrained as specified in subclause 7.4.3. For this specification, the modifications a) to f) specified in the first paragraph of this subclause apply.

field_pic_flag and **bottom_field_flag** have the same semantics as specified in subclause 7.4.3. For this specification, the modifications a) to d) specified in the first paragraph of this subclause apply.

idr_pic_id identifies an IDR picture when dependency_id is equal to the maximum present value of dependency_id in the VCL NAL units of the current coded picture. The value of idr_pic_id shall be in the range of 0 to 65535, inclusive.

When two consecutive access units in decoding order are both IDR access units, the value of idr_pic_id in the slices of the target dependency representation in the primary coded pictures of the first such IDR access unit shall differ from the idr_pic_id in the slices of the target dependency representation in the primary coded pictures of the second such IDR access unit.

NOTE 2 – The classification of an access unit as IDR access unit depends on the maximum present value of dependency_id. When NAL units are removed from a bitstream, e.g. in order to adjust the bitstream to the capabilities of a receiving device, the classification of access units as IDR access units may change. Since all bitstreams for different conformance points supported in a scalable bitstream (in particular for different maximum values of dependency_id) must conform to this Recommendation | International Standard (as specified in subclause G.8.8.1), the constraints on idr_pic_id must be obeyed for all conformance points contained in a scalable bitstream.

pic_order_cnt_lsb, **delta_pic_order_cnt_bottom**, **delta_pic_order_cnt**[0], and **delta_pic_order_cnt**[1] have the same semantics as specified in subclause 7.4.3. For this specification, the modifications a) to f) specified in the first paragraph of this subclause apply.

redundant_pic_cnt has the same semantics as specified in subclause 7.4.3. For this specification, the modifications a) to g) specified in the first paragraph of this subclause apply.

direct_spatial_mv_pred_flag specifies the method used in the decoding process to derive motion vectors and reference indices for inter prediction. When quality_id is greater than 0, direct_spatial_mv_pred_flag is inferred as specified in the beginning of this subclause. The value of direct_spatial_mv_pred_flag shall be equal to 1.

num_ref_idx_active_override_flag, **num_ref_idx_l0_active_minus1**, and **num_ref_idx_l1_active_minus1** have the same semantics as specified in subclause 7.4.3. When quality_id is greater than 0, num_ref_idx_active_override_flag, num_ref_idx_l0_active_minus1, and num_ref_idx_l1_active_minus1 are inferred as specified in the beginning of this subclause.

base_pred_weight_table_flag equal to 1 specifies that the variables for weighted prediction are inferred. When base_pred_weight_table_flag is not present, it shall be inferred as follows:

– If quality_id is greater than 0, base_pred_weight_table_flag is inferred as specified in the beginning of this subclause.

– Otherwise (quality_id is equal to 0), base_pred_weight_table_flag is inferred to be equal to 0.

When base_pred_weight_table_flag is equal to 1 and quality_id is equal to 0, let refSetOfSlices be the set of slices that is represented by the VCL NAL units with dependency_id equal to (ref_layer_dq_id >> 4) and quality_id equal to 0 inside the current coded picture.

When base_pred_weight_table_flag is equal to 1 and quality_id is equal to 0, the following constraints shall be obeyed:

a) For all slices in refSetOfSlices, the value of (slice_type % 5) shall be equal to (slice_type % 5) of the current slice.

b) base_pred_weight_table_flag shall have the same value in all slices in refSetOfSlices.

c) When the syntax structure pred_weight_table( ) is present in the slices of the set refSetOfSlices, the values of all syntax elements inside the syntax structure pred_weight_table( ) shall be the same for all slices in refSetOfSlices.

d) When the current slice is an EP slice, the following applies:

   i) The value of num_ref_idx_l0_active_minus1 of all slices in refSetOfSlices shall be identical to the value of num_ref_idx_l0_active_minus1 of the current slice.

   ii) For each slice in refSetOfSlices, the syntax elements inside the syntax structure ref_pic_list_modification( ) shall be the same, and the syntax structure ref_pic_list_modification( ) for the slices in refSetOfSlices shall contain syntax elements so that for useRefBasePicFlag equal to 0 and 1, an invocation of subclause G.8.2.3 with currDependencyId set equal to (ref_layer_dq_id >> 4), useRefBasePicFlag, and any slice of refSetOfSlices as the inputs derives a reference picture list refPicList0RefLayer that is identical to the reference picture list refPicList0, which is derived by invoking subclause G.8.2.3 with currDependencyId set equal to dependency_id, useRefBasePicFlag, and the current slice as the inputs. The entries of two reference picture lists are considered the same when they represent entries that correspond to same coded frame, the same complementary reference field pair, the same coded field, or the same field of a coded frame.

   iii) weighted_pred_flag shall be equal to 1 for the slices in refSetOfSlices.

e) When the current slice is an EB slice, the following applies:

   i) The values of num_ref_idx_l0_active_minus1 and num_ref_idx_l1_active_minus1 of all slices in refSetOfSlices shall be identical to the values of num_ref_idx_l0_active_minus1 and num_ref_idx_l1_active_minus1, respectively, of the current slice.

   ii) For each slice in refSetOfSlices, the syntax elements inside the syntax structure ref_pic_list_modification( ) shall be the same, and the syntax structure ref_pic_list_modification( ) for the slices in refSetOfSlices shall contain syntax elements so that for useRefBasePicFlag equal to 0 and 1, an invocation of subclause G.8.2.3 with currDependencyId set equal to (ref_layer_dq_id >> 4), useRefBasePicFlag, and any slice of refSetOfSlices as the inputs derives reference picture lists refPicList0RefLayer and refPicList1RefLayer that are identical to the reference picture lists refPicList0 and refPicList1, respectively, which are derived by invoking subclause G.8.2.3 with currDependencyId set equal to dependency_id, useRefBasePicFlag, and the current slice as the inputs. The entries of two reference picture lists are considered the same when they represent entries that correspond to same coded frame, the same complementary reference field pair, the same coded field, or the same field of a coded frame.

iii) weighted_bipred_idc shall be equal to 1 for the slices in refSetOfSlices.

store_ref_base_pic_flag equal to 1 specifies that, when the value of dependency_id is equal to the maximum value of dependency_id for the VCL NAL units of the current coded picture, an additional representation of the coded picture that may or may not be identical to the decoded picture is marked as "used for reference". This additional representation is also referred to as reference base picture and may be used for inter prediction of following pictures in decoding order, but it is not output. When store_ref_base_pic_flag is not present, it shall be inferred as follows:

– If quality_id is equal to 0, store_ref_base_pic_flag is inferred to be equal to 0.

– Otherwise (quality_id is greater than 0), store_ref_base_pic_flag is inferred as specified in the beginning of this subclause.

The syntax element store_ref_base_pic_flag shall have the same value for all VCL NAL units of a dependency representation. When nal_ref_idc is equal to 0, store_ref_base_pic_flag shall be equal to 0.

When max_num_ref_frames is less than 2, store_ref_base_pic_flag shall be equal to 0.

cabac_init_idc and **slice_qp_delta** have the same semantics as specified in subclause 7.4.3.

disable_deblocking_filter_idc specifies whether the operation of the deblocking filter shall be disabled across some block edges of the slice, specifies for which edges the filtering is disabled, and specifies the order of deblocking filter operations. When disable_deblocking_filter_idc is not present in the slice header, the value of disable_deblocking_filter_idc shall be inferred to be equal to 0.

The value of disable_deblocking_filter_idc shall be in the range of 0 to 6, inclusive. disable_deblocking_filter_idc equal to 0 specifies that all luma and chroma block edges of the slice are filtered. disable_deblocking_filter_idc equal to 1 specifies that deblocking is disabled for all block edges of the slice. disable_deblocking_filter_idc equal to 2 specifies that all luma and chroma block edges of the slice are filtered with exception of the block edges that coincide with slice boundaries. disable_deblocking_filter_idc equal to 3 specifies a two stage deblocking filter process for the slice: After filtering all block luma and chroma block edges that do not coincide with slice boundaries (as if disable_deblocking_filter_idc were equal to 2), the luma and chroma block edges that coincide with slice boundaries are filtered. disable_deblocking_filter_idc equal to 4 specifies that all luma block edges of the slice are filtered, but the deblocking of the chroma block edges is disabled. disable_deblocking_filter_idc equal to 5 specifies that all luma block edges of the slice are filtered with exception of the block edges that coincide with slice boundaries (as if disable_deblocking_filter_idc were equal to 2), and that deblocking for chroma block edges of the slice is disabled. disable_deblocking_filter_idc equal to 6 specifies that the deblocking for chroma block edges is disabled and that the two stage deblocking filter process is used for luma block edges of the slice: After filtering all block luma block edges that do not coincide with slice boundaries (as if disable_deblocking_filter_idc were equal to 2), the luma block edges that coincide with slice boundaries are filtered.

When no_inter_layer_pred_flag is equal to 1 or tcoeff_level_prediction_flag is equal to 1, the value of disable_deblocking_filter_idc shall be in the range of 0 to 2, inclusive.

slice_alpha_c0_offset_div2, and **slice_beta_offset_div2** have the same semantics as specified in subclause 7.4.3.

slice_group_change_cycle has the same semantics as specified in subclause 7.4.3.

ref_layer_dq_id specifies the layer representation inside the current coded picture that is used for inter-layer prediction of the current layer representation. When present, the value of ref_layer_dq_id shall be in the range of 0 to DQId − 1, inclusive. When ref_layer_dq_id is not present, it shall be inferred as follows:

– If quality_id is greater than 0, ref_layer_dq_id is inferred to be equal to (DQId − 1).

– Otherwise (quality_id is equal to 0), ref_layer_dq_id is inferred to be equal to −1.

When quality_id is equal to 0, the NAL units with DQId equal to ref_layer_dq_id shall have discardable_flag equal to 0.

The variable MaxRefLayerDQId is set equal to the maximum value of ref_layer_dq_id for the slices of the current layer representation.

When MinNoInterLayerPredFlag is equal to 0, the layer representation inside the current coded picture that has a value of DQId equal MaxRefLayerDQId is also referred to as reference layer representation.

When MaxRefLayerDQId is not equal to −1, the following variables are derived as follows:

– RefLayerPicSizeInMbs is set equal to the value of the variable PicSizeInMbs for the reference layer representation.
– RefLayerPicWidthInMbs is set equal to the value of the variable PicWidthInMbs for the reference layer representation.
– RefLayerPicHeightInMbs is set equal to the value of the variable PicHeightInMbs for the reference layer representation.
– RefLayerChromaFormatIdc is set equal to the value of the syntax element chroma_format_idc for the reference layer representation.
– RefLayerChromaArrayType is set equal to the value of ChromaArrayType for the reference layer representation.
– $RefLayerPicWidthInSamples_L$ is set equal to the value of the variable $PicWidthInSamples_L$ for the reference layer representation.
– $RefLayerPicHeightInSamples_L$ is set equal to the value of the variable $PicHeightInSamples_L$ for the reference layer representation.
– $RefLayerPicWidthInSamples_C$ is set equal to the value of the variable $PicWidthInSamples_C$ for the reference layer representation.
– $RefLayerPicHeightInSamples_C$ is set equal to the value of the variable $PicHeightInSamples_C$ for the reference layer representation.
– RefLayerMbWidthC is set equal to the value of the variable MbWidthC for the reference layer representation.
– RefLayerMbHeightC is set equal to the value of the variable MbHeightC for the reference layer representation.
– RefLayerFrameMbsOnlyFlag is set equal to the value of the syntax element frame_mbs_only_flag for the reference layer representation.
– RefLayerFieldPicFlag is set equal to the value of the syntax element field_pic_flag for the reference layer representation.
– RefLayerBottomFieldFlag is set equal to the value of the syntax element bottom_field_flag for the reference layer representation.
– RefLayerMbaffFrameFlag is set equal to the value of the variable MbaffFrameFlag for the reference layer representation.

disable_inter_layer_deblocking_filter_idc specifies whether the operation of the deblocking filter for inter-layer intra prediction is disabled across some block edges of the reference layer representation, specifies for which edges the filtering is disabled, and specifies the order of deblocking filter operations for inter-layer intra prediction. When disable_inter_layer_deblocking_filter_idc is not present in the slice header, the value of disable_inter_layer_deblocking_filter_idc shall be inferred to be equal to 0. The value of disable_inter_layer_deblocking_filter_idc shall be in the range of 0 to 6, inclusive. The values 0 to 6 of disable_inter_layer_deblocking_filter_idc specify the same deblocking filter operations as the corresponding values of disable_deblocking_filter_idc, but for the deblocking of the intra macroblocks of the reference layer representation specified by ref_layer_dq_id before resampling.

When disable_inter_layer_deblocking_filter_idc is present, quality_id is equal to 0, and SpatialResolutionChangeFlag as specified in the following paragraphs is equal to 0, disable_inter_layer_deblocking_filter_idc shall be equal to 1.

inter_layer_slice_alpha_c0_offset_div2 specifies the offset used in accessing the α and t_{C0} deblocking filter tables for filtering operations of the intra macroblocks of the reference layer representation before resampling. From this value, the offset that is applied when addressing these tables shall be computed as:

$$\text{InterlayerFilterOffsetA} = \text{inter\_layer\_slice\_alpha\_c0\_offset\_div2} \ll 1 \quad \text{(G-64)}$$

The value of inter_layer_slice_alpha_c0_offset_div2 shall be in the range of −6 to +6, inclusive. When inter_layer_slice_alpha_c0_offset_div2 is not present in the slice header, the value of inter_layer_slice_alpha_c0_offset_div2 shall be inferred to be equal to 0.

inter_layer_slice_beta_offset_div2 specifies the offset used in accessing the β deblocking filter table for filtering operations of the intra macroblocks of the reference layer representation before resampling. From this value, the offset that is applied when addressing the β table of the deblocking filter is computed as:

InterlayerFilterOffsetB = inter_layer_slice_beta_offset_div2 << 1 (G-65)

The value of inter_layer_slice_beta_offset_div2 shall be in the range of −6 to +6, inclusive. When inter_layer_slice_beta_offset_div2 is not present in the slice header the value of inter_layer_slice_beta_offset_div2 shall be inferred to be equal to 0.

constrained_intra_resampling_flag specifies whether slice boundaries in the layer picture that is used for inter-layer prediction (as specified by ref_layer_dq_id) are treated similar to layer picture boundaries for the intra resampling process. When constrained_intra_resampling_flag is equal to 1, disable_inter_layer_deblocking_filter_idc shall be equal to 1, 2, or 5.

When constrained_intra_resampling_flag is equal to 1, a macroblock cannot be coded using the Intra_Base macroblock prediction mode when it covers more than one slice in the layer picture that is used for inter-layer prediction, as specified in subclause G.8.6.2.

When constrained_intra_resampling_flag is not present, it shall be inferred to be equal to 0.

ref_layer_chroma_phase_x_plus1_flag specifies the horizontal phase shift of the chroma components in units of half luma samples of a layer frame for the layer pictures that may be used for inter-layer prediction.

When ref_layer_chroma_phase_x_plus1_flag is not present, it shall be inferred as follows:

– If quality_id is greater than 0, ref_layer_chroma_phase_x_plus1_flag is inferred to be equal to chroma_phase_x_plus1_flag.

– Otherwise (quality_id is equal to 0), ref_layer_chroma_phase_x_plus1_flag is inferred to be equal to seq_ref_layer_chroma_phase_x_plus1_flag.

When no_inter_layer_pred_flag is equal to 0, the following is specified:

a) When ref_layer_dq_id is greater than 0, ref_layer_chroma_phase_x_plus1_flag should be equal to chroma_phase_x_plus1_flag of the subset sequence parameter set RBSP that is referred to by the reference layer representation (with DQId equal to ref_layer_dq_id).

b) When RefLayerChromaArrayType is equal to 1 and chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field are present in the SVC sequence parameter set that is referred to by the reference layer representation (with DQId equal to ref_layer_dq_id), the following applies:

   – If ref_layer_chroma_phase_x_plus1_flag is equal to 0, chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field of the SVC sequence parameter set that is referred to by the reference layer representation should be equal to 0, 2, or 4.

   – Otherwise (ref_layer_chroma_phase_x_plus1_flag is equal to 1), chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field of the SVC sequence parameter set that is referred to by the reference layer representation should be equal to 1, 3, or 5.

c) When RefLayerChromaArrayType is not equal to 1, ref_layer_chroma_phase_x_plus1_flag should be equal to 1.

ref_layer_chroma_phase_y_plus1 specifies the vertical phase shift of the chroma components in units of half luma samples of a layer frame for the layer pictures that may be used for inter-layer prediction.

When ref_layer_chroma_phase_y_plus1 is not present, it shall be inferred as follows:

– If quality_id is greater than 0, ref_layer_chroma_phase_y_plus1 is inferred to be equal to chroma_phase_y_plus1.

– Otherwise (quality_id is equal to 0), ref_layer_chroma_phase_y_plus1 is inferred to be equal to seq_ref_layer_chroma_phase_y_plus1.

The value of ref_layer_chroma_phase_y_plus1 shall be in the range of 0 to 2, inclusive.

When no_inter_layer_pred_flag is equal to 0, the following applies:

a) When ref_layer_dq_id is greater than 0, ref_layer_chroma_phase_y_plus1 should be equal to chroma_phase_y_plus1 of the subset sequence parameter set RBSP that is referred to by the reference layer representation (with DQId equal to ref_layer_dq_id).

b) When RefLayerChromaArrayType is equal to 1 and chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field are present in the SVC sequence parameter set that is referred to by the reference layer representation (with DQId equal to ref_layer_dq_id), the following applies:

– If ref_layer_chroma_phase_y_plus1 is equal to 0, chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field of the SVC sequence parameter set that is referred to by the reference layer representation should be equal to 2 or 3.

– Otherwise, if ref_layer_chroma_phase_y_plus1 is equal to 1, chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field of the SVC sequence parameter set that is referred to by the reference layer representation should be equal to 0 or 1.

– Otherwise (chroma_phase_y_plus1 is equal to 2), chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field of the SVC sequence parameter set that is referred to by the reference layer representation should be equal to 4 or 5.

c) When RefLayerChromaArrayType is not equal to 1, ref_layer_chroma_phase_y_plus1 should be equal to 1.

scaled_ref_layer_left_offset specifies the horizontal offset between the upper-left luma sample of a resampled layer picture used for inter-layer prediction and the upper-left luma sample of the current picture or current layer picture in units of two luma samples.

When scaled_ref_layer_left_offset is not present, it shall be inferred as follows:

– If quality_id is greater than 0, scaled_ref_layer_left_offset is inferred to be equal to 0.

– Otherwise (quality_id is equal to 0), scaled_ref_layer_left_offset is inferred to be equal to seq_scaled_ref_layer_left_offset.

The value of scaled_ref_layer_left_offset shall be in the range of -2^{15} to $2^{15} - 1$, inclusive.

scaled_ref_layer_top_offset specifies the vertical offset between the upper-left luma sample of a resampled layer picture used for inter-layer prediction and the upper-left luma sample of the current picture or current layer picture. The vertical offset is specified in units of two luma samples when frame_mbs_only_flag is equal to 1, and it is specified in units of four luma samples when frame_mbs_only_flag is equal to 0.

When scaled_ref_layer_top_offset is not present, it shall be inferred as follows:

– If quality_id is greater than 0, scaled_ref_layer_top_offset is inferred to be equal to 0.

– Otherwise (quality_id is equal to 0), scaled_ref_layer_top_offset is inferred to be equal to seq_scaled_ref_layer_top_offset.

The value of scaled_ref_layer_top_offset shall be in the range of -2^{15} to $2^{15} - 1$, inclusive.

scaled_ref_layer_right_offset specifies the horizontal offset between the bottom-right luma sample of a resampled layer picture used for inter-layer prediction and the bottom-right luma sample of the current picture or current layer picture in units of two luma samples.

When scaled_ref_layer_right_offset is not present, it shall be inferred as follows:

– If quality_id is greater than 0, scaled_ref_layer_right_offset is inferred to be equal to 0.

– Otherwise (quality_id is equal to 0), scaled_ref_layer_right_offset is inferred to be equal to seq_scaled_ref_layer_right_offset.

The value of scaled_ref_layer_right_offset shall be in the range of -2^{15} to $2^{15} - 1$, inclusive.

scaled_ref_layer_bottom_offset specifies the vertical offset between the bottom-right luma sample of a resampled layer picture used for inter-layer prediction and the bottom-right luma sample of the current picture or current layer picture. The vertical offset is specified in units of two luma samples when frame_mbs_only_flag is equal to 1, and it is specified in units of four luma samples when frame_mbs_only_flag is equal to 0.

When scaled_ref_layer_bottom_offset is not present, it shall be inferred as follows:

– If quality_id is greater than 0, scaled_ref_layer_bottom_offset is inferred to be equal to 0.

– Otherwise (quality_id is equal to 0), scaled_ref_layer_bottom_offset is inferred to be equal to seq_scaled_ref_layer_bottom_offset.

The value of scaled_ref_layer_bottom_offset shall be in the range of -2^{15} to $2^{15} - 1$, inclusive.

The variables scaledLeftOffset, scaledRightOffset, scaledTopOffset, and scaledBottomOffset are derived as follows:

– If MinNoInterLayerPredFlag is equal to 0, scaledLeftOffset, scaledRightOffset, scaledTopOffset, and scaledBottomOffset are set equal to the values of scaled_ref_layer_left_offset, scaled_ref_layer_right_offset, scaled_ref_layer_top_offset, and scaled_ref_layer_bottom_offset, respectively, for the slices of the current layer representation that have no_inter_layer_pred_flag equal to 0.

– Otherwise (MinNoInterLayerPredFlag is equal to 1), scaledLeftOffset, scaledRightOffset, scaledTopOffset, and scaledBottomOffset are set equal to the values of scaled_ref_layer_left_offset, scaled_ref_layer_right_offset, scaled_ref_layer_top_offset, and scaled_ref_layer_bottom_offset, respectively.

The variables ScaledRefLayerLeftOffset, ScaledRefLayerRightOffset, ScaledRefLayerTopOffset, ScaledRefLayerBottomOffset, $\text{ScaledRefLayerPicWidthInSamples}_L$, and $\text{ScaledRefLayerPicHeightInSamples}_L$ are derived by

$$\text{ScaledRefLayerLeftOffset} = 2 * \text{scaledLeftOffset} \tag{G-66}$$
$$\text{ScaledRefLayerRightOffset} = 2 * \text{scaledRightOffset} \tag{G-67}$$
$$\text{ScaledRefLayerTopOffset} = 2 * \text{scaledTopOffset} * (\ 2 - \text{frame\_mbs\_only\_flag}\ ) \tag{G-68}$$
$$\text{ScaledRefLayerBottomOffset} = 2 * \text{scaledBottomOffset} * (\ 2 - \text{frame\_mbs\_only\_flag}\ ) \tag{G-69}$$
$$\text{ScaledRefLayerPicWidthInSamples}_L = \text{PicWidthInMbs} * 16 - \text{ScaledRefLayerLeftOffset} - \text{ScaledRefLayerRightOffset} \tag{G-70}$$
$$\text{ScaledRefLayerPicHeightInSamples}_L = \text{PicHeightInMbs} * 16 - (\ \text{ScaledRefLayerTopOffset} + \text{ScaledRefLayerBottomOffset}\ ) / (\ 1 + \text{field\_pic\_flag}\ ) \tag{G-71}$$

When no_inter_layer_pred_flag is equal to 0, the following constraints shall be obeyed:

a) The bitstream shall not contain data that result in $\text{ScaledRefLayerPicWidthInSamples}_L$ less than $\text{RefLayerPicWidthInSamples}_L$.

b) When RefLayerFrameMbsOnlyFlag is equal to 0 or frame_mbs_only_flag is equal to 1, the bitstream shall not contain data that result in ($\text{ScaledRefLayerPicHeightInSamples}_L$ * (1 + field_pic_flag)) less than ($\text{RefLayerPicHeightInSamples}_L$ * (1 + RefLayerFieldPicFlag)).

c) When RefLayerFrameMbsOnlyFlag is equal to 1 and frame_mbs_only_flag is equal to 0, the bitstream shall not contain data that result in ($\text{ScaledRefLayerPicHeightInSamples}_L$ * (1 + field_pic_flag)) less than (2 * $\text{RefLayerPicHeightInSamples}_L$).

When ChromaArrayType is not equal to 0, the variables $\text{ScaledRefLayerPicWidthInSamples}_C$, and $\text{ScaledRefLayerPicHeightInSamples}_C$ are derived by

$$\text{ScaledRefLayerPicWidthInSamples}_C = \text{ScaledRefLayerPicWidthInSamples}_L\ /\ \text{SubWidthC} \tag{G-72}$$
$$\text{ScaledRefLayerPicHeightInSamples}_C = \text{ScaledRefLayerPicHeightInSamples}_L\ /\ \text{SubHeightC} \tag{G-73}$$

The variable CroppingChangeFlag is derived as follows:

– If MinNoInterLayerPredFlag is equal to 0, quality_id is equal to 0, and extended_spatial_scalability_idc is equal to 2, CroppingChangeFlag is set equal to 1.

– Otherwise (MinNoInterLayerPredFlag is equal to 1, quality_id is greater than 0, or extended_spatial_scalability_idc is less than 2), CroppingChangeFlag is set equal to 0.

NOTE 3 – Encoder designers are encouraged to set the value of no_inter_layer_pred_flag equal to 0 for at least one slice of each layer representation with extended_spatial_scalability_idc equal to 2 and quality_id equal to 0.

The variable SpatialResolutionChangeFlag is derived as follows:

– If MinNoInterLayerPredFlag is equal to 1, quality_id is greater than 0, or all of the following conditions are true, SpatialResolutionChangeFlag is set equal to 0:

  – CroppingChangeFlag is equal to 0,

  – $\text{ScaledRefLayerPicWidthInSamples}_L$ is equal to $\text{RefLayerPicWidthInSamples}_L$,

  – $\text{ScaledRefLayerPicHeightInSamples}_L$ is equal to $\text{RefLayerPicHeightInSamples}_L$,

  – (ScaledRefLayerLeftOffset % 16) is equal to 0,

  – (ScaledRefLayerTopOffset % (16 * (1 + field_pic_flag + MbaffFrameFlag))) is equal to 0,

  – field_pic_flag is equal to RefLayerFieldPicFlag,

  – MbaffFrameFlag is equal to RefLayerMbaffFrameFlag,

  – chroma_format_idc is equal to RefLayerChromaFormatIdc,

– chroma_phase_x_plus1_flag is equal to ref_layer_chroma_phase_x_plus1_flag for the slices with no_inter_layer_pred_flag equal to 0,

– chroma_phase_y_plus1 is equal to ref_layer_chroma_phase_y_plus1 for the slices with no_inter_layer_pred_flag equal to 0.

– Otherwise, SpatialResolutionChangeFlag is set equal to 1.

The variable RestrictedSpatialResolutionChangeFlag is derived as follows:

– If SpatialResolutionChangeFlag is equal to 0 or all of the following conditions are true, RestrictedSpatialResolutionChangeFlag is set equal to 1:

  – $\text{ScaledRefLayerPicWidthInSamples}_L$ is equal to $\text{RefLayerPicWidthInSamples}_L$ or $(2 * \text{RefLayerPicWidthInSamples}_L)$,

  – $\text{ScaledRefLayerPicHeightInSamples}_L$ is equal to $\text{RefLayerPicHeightInSamples}_L$ or $(2 * \text{RefLayerPicHeightInSamples}_L)$,

  – (ScaledRefLayerLeftOffset % 16) is equal to 0,

  – (ScaledRefLayerTopOffset % (16 * (1 + field_pic_flag))) is equal to 0,

  – MbaffFrameFlag is equal to 0,

  – RefLayerMbaffFrameFlag is equal to 0,

  – field_pic_flag is equal to RefLayerFieldPicFlag.

– Otherwise, RestrictedSpatialResolutionChangeFlag is set equal to 0.

slice_skip_flag specifies the presence of the slice data in scalable extension syntax structure. When slice_skip_flag is not present, it shall be inferred to be equal to 0. slice_skip_flag equal to 0 specifies that the slice data in scalable extension syntax structure is present in the NAL unit. slice_skip_flag equal to 1 specifies that the slice data in scalable extension syntax structure is not present in the NAL unit and that the syntax elements for the macroblock layer of the slice are derived by the following process:

1. CurrMbAddr is derived by

$$\text{CurrMbAddr} = \text{first\_mb\_in\_slice} * (1 + \text{MbaffFrameFlag}) \quad \text{(G-74)}$$

2. The variable mbIdx proceeds over the values 0..num_mbs_in_slice_minus1, and for each value of mbIdx, the following ordered steps are specified:

   a. The bitstream shall not contain data that result in InCropWindow(CurrMbAddr) equal to 0.

   b. For the macroblock with address CurrMbAddr, the syntax elements mb_skip_flag (when applicable), mb_skip_run (when applicable), mb_field_decoding_flag, base_mode_flag, residual_prediction_flag and coded_block_pattern shall be inferred as follows:

      – mb_skip_flag (when applicable) and mb_skip_run (when applicable) are inferred to be equal to 0.

      – mb_field_decoding_flag is inferred to be equal to 0.

        NOTE 4 – The frame/field mode used for decoding is inferred in subclause G.8.1.5.1.

      – base_mode_flag is inferred to be equal to 1.

      – residual_prediction_flag is inferred to be equal to 1.

      – coded_block_pattern is inferred to be equal to 0.

      – QP_Y is inferred to be equal to SliceQP_Y.

      – QP'_Y is inferred to be equal to $(QP_Y + \text{QpBdOffset}_Y)$.

   c. When the variable mbIdx is less than num_mbs_in_slice_minus1, CurrMbAddr is set to NextMbAddress(CurrMbAddr). The bitstream shall not contain data that result in CurrMbAddr being set equal to a value that is not less than PicSizeInMbs.

num_mbs_in_slice_minus1 plus 1specifies the number of macroblocks for a slice with slice_skip_flag equal to 1.

adaptive_base_mode_flag specifies the presence of syntax elements in the slice header and in the macroblock layer in scalable extension. When adaptive_base_mode_flag is not present, it shall be inferred to be equal to 0.

default_base_mode_flag specifies how base_mode_flag is inferred when it is not present in macroblock layer in scalable extension. When default_base_mode_flag is not present, it shall be inferred to be equal to 0.

adaptive_motion_prediction_flag specifies the presence of syntax elements in the macroblock layer in scalable extension. When adaptive_motion_prediction_flag is not present, it shall be inferred to be equal to 0.

default_motion_prediction_flag specifies how motion_prediction_flag_l0[ ] and motion_prediction_flag_l1[ ] are inferred when they are not present in macroblock layer in scalable extension. When default_motion_prediction_flag is not present, it shall be inferred to be equal to 0.

adaptive_residual_prediction_flag specifies the presence of syntax elements in the macroblock layer in scalable extension. When adaptive_residual_prediction_flag is not present, it shall be inferred to be equal to 0.

default_residual_prediction_flag specifies how residual_prediction_flag is inferred when it is not present in the macroblock layer in scalable extension. When default_residual_prediction_flag is not present, it shall be inferred to be equal to 0.

tcoeff_level_prediction_flag equal to 1 specifies that an alternative inter-layer prediction process is applied as specified in subclause G.8. When tcoeff_level_prediction_flag is not present, it shall be inferred as follows:

– If no_inter_layer_pred_flag is equal to 1, tcoeff_level_prediction_flag is inferred to be equal to 0.

– Otherwise (no_inter_layer_pred_flag is equal to 0), tcoeff_level_prediction_flag is inferred to be equal to the value of seq_tcoeff_level_prediction_flag.

When SpatialResolutionChangeFlag is equal to 1, tcoeff_level_prediction_flag shall be equal to 0.

When tcoeff_level_prediction_flag is equal to 1, the following constraints shall be obeyed:

a) The slices of the reference layer representation (with DQId equal to ref_layer_dq_id) shall have no_inter_layer_pred_flag equal to 1 or tcoeff_level_prediction_flag equal to 1.

b) All elements of ScalingList4x4 shall be the same for the slices of the current layer representation and all slices of the reference layer representation (with DQId equal to the value of ref_layer_dq_id).

c) All elements of ScalingList8x8 shall be the same for the slices of the current layer representation and all slices of the reference layer representation (with DQId equal to the value of ref_layer_dq_id).

d) The value of the syntax element use_ref_base_pic_flag shall be equal to 0 for the slices of the current layer representation and all slices of the reference layer representation (with DQId equal to the value of ref_layer_dq_id).

e) When slice_skip_flag is equal to 1, the value of constrained_intra_pred_flag for the current layer representation shall be identical to the value of constrained_intra_pred_flag for the reference layer representation (with DQId equal to ref_layer_dq_id).

The variable MaxTCoeffLevelPredFlag is set equal to the maximum value of tcoeff_level_prediction_flag for the slices of the current layer representation.

scan_idx_start specifies the first scanning position for the transform coefficient levels in the current slice. When scan_idx_start is not present, it shall be inferred to be equal to 0.

scan_idx_end specifies the last scanning position for the transform coefficient levels in the current slice. When scan_idx_end is not present, it shall be inferred to be equal to 15.

When default_base_mode_flag is equal to 1, (slice_type % 5) is equal to 2, and entropy_coding_mode_flag is equal to 0, it is a requirement of bitstream conformance that the value of scan_idx_end is greater than or equal to scan_idx_start.

G.7.4.3.5 Decoded reference base picture marking semantics

The specification of this subclause applies to the current dependency representation. The modifications a) and b) specified in subclause G.8.2 apply with currDependencyId being equal to the current value of dependency_id.

The syntax elements adaptive_ref_base_pic_marking_mode_flag, memory_management_base_control_operation, difference_of_base_pic_nums_minus1, and long_term_base_pic_num specify marking of reference base pictures as "unused for reference".

When present in a prefix NAL unit, all syntax elements of the dec_ref_base_pic_marking( ) syntax structure are considered as if they were present in the associated NAL unit.

When quality_id is greater than 0, all syntax elements of the dec_ref_base_pic_marking( ) syntax structure are inferred as specified in the beginning of subclause G.7.4.3.4.

The content of the decoded reference picture base marking syntax structure shall be the same in all slice headers of the primary coded picture. When one or more redundant coded pictures are present, the content of the decoded reference base picture marking syntax structure shall be the same in all slice headers of a redundant coded picture with a particular value of redundant_pic_cnt.

NOTE 1 – It is not required that the content of the decoded reference base picture marking syntax structure in a redundant coded picture with a particular value of redundant_pic_cnt is identical to the content of the decoded reference base picture marking syntax structure in the corresponding primary coded picture or a redundant coded picture with a different value of redundant_pic_cnt. However, as specified in subclause G.7.4.3.4 (by referencing subclause 7.4.3), the content of the decoded reference base picture marking syntax structure in a redundant coded picture is constrained in the way that the marking status of reference pictures and the value of frame_num after the SVC decoded reference picture marking process in subclause G.8.2.4 must be identical regardless whether the primary coded picture or any redundant coded picture of the access unit would be decoded.

The memory_management_base_control_operation commands of the dec_ref_base_pic_marking( ) syntax structure are processed by the decoding process before the memory_management_control_operation commands of the dec_ref_pic_marking( ) syntax structure are processed.

adaptive_ref_base_pic_marking_mode_flag selects the reference base picture marking mode for the current picture or layer picture as specified in Table G-2. When adaptive_ref_base_pic_marking_mode_flag is not present and quality_id is equal to 0, it shall be inferred to be equal to 0.

Table G-2 – Interpretation of adaptive_ref_base_pic_marking_mode_flag

adaptive_ref_base_pic_marking_mode_flag	Reference base picture marking mode specified
0	Sliding window reference picture marking mode: A marking mode providing a first-in, first-out mechanism for short-term reference pictures
1	Adaptive reference base picture marking mode: A reference picture marking mode providing syntax elements to specify marking of reference base pictures as "unused for reference"

memory_management_base_control_operation specifies a control operation to be applied to affect the marking of reference base pictures. The memory_management_base_control_operation syntax element is followed by data necessary for the operation specified by the value of memory_management_base_control_operation. The values and control operations associated with memory_management_base_control_operation are specified in Table G-3. The memory_management_base_control_operation syntax elements are processed by the decoding process in the order in which they appear, and the semantics constraints expressed for each memory_management_base_control_operation apply at the specific position in that order at which that individual memory_management_base_control_operation is processed.

For interpretation of memory_management_base_control_operation, the terms reference picture and reference base picture are interpreted as follows:

– If the current picture is a frame, the term reference picture refers either to a reference frame or a complementary reference field pair and the term reference base picture refers either to a reference base frame or a complementary reference base field pair.

– Otherwise (the current picture is a field), the term reference picture refers either to a reference field or a field of a reference frame and the term reference base picture refers either to a reference base field or a field of a reference base frame.

memory_management_base_control_operation shall not be equal to 1 unless the specified reference base picture is marked as "used for short-term reference" (and as "reference base picture") when the memory_management_base_control_operation is processed by the decoding process.

memory_management_base_control_operation shall not be equal to 2 unless the specified long-term picture number refers to a reference base picture that is marked as "used for long-term reference" (and as "reference base picture") when the memory_management_base_control_operation is processed by the decoding process.

When the dec_ref_pic_marking( ) syntax structure contains a memory_management_control_operation equal to 5, memory_management_base_control_operation shall not be equal to 1 or 2.

Table G-3 – Memory management base control operation (memory_management_base_control_operation) values

memory_management_base_control_operation	Memory Management Base Control Operation
0	End memory_management_base_control_operation syntax element loop
1	Mark a short-term reference base picture as "unused for reference"
2	Mark a long-term reference base picture as "unused for reference"

difference_of_base_pic_nums_minus1 is used (with memory_management_base_control_operation equal to 1) to mark a short-term reference base picture as "unused for reference". When the associated memory_management_base_control_operation is processed by the decoding process, the resulting picture number derived from difference_of_base_pic_nums_minus1 shall be a picture number assigned to one of the reference pictures marked as "used for short-term reference" and as "reference base picture".

The resulting picture number is constrained as follows:

– If field_pic_flag is equal to 0, the resulting picture number shall be one of the set of picture numbers assigned to reference frames or complementary reference field pairs marked as "reference base picture".

  NOTE 2 – When field_pic_flag is equal to 0, the resulting picture number must be a picture number assigned to a complementary reference field pair in which both fields are marked as "used for short-term reference" and "reference base picture" or a reference frame in which both fields are marked as "used for short-term reference" and "reference base picture".

– Otherwise (field_pic_flag is equal to 1), the resulting picture number shall be one of the set of picture numbers assigned to reference fields marked as "reference base picture".

long_term_base_pic_num is used (with memory_management_base_control_operation equal to 2) to mark a long-term reference base picture as "unused for reference". When the associated memory_management_base_control_operation is processed by the decoding process, long_term_base_pic_num shall be equal to a long-term picture number assigned to one of the reference pictures marked as "used for long-term reference" and as "reference base picture".

The resulting long-term picture number is constrained as follows:

– If field_pic_flag is equal to 0, the resulting long-term picture number shall be one of the set of long-term picture numbers assigned to reference frames or complementary reference field pairs marked as "reference base picture".

  NOTE 3 – When field_pic_flag is equal to 0, the resulting long-term picture number must be a long-term picture number assigned to a complementary reference field pair in which both fields are marked as "used for long-term reference" and "reference base picture" or a reference frame in which both fields are marked as "used for long-term reference" and "reference base picture".

– Otherwise (field_pic_flag is equal to 1), the resulting long-term picture number shall be one of the set of long-term picture numbers assigned to reference fields marked as "reference base picture".

G.7.4.4 Slice data semantics

The semantics specified in subclause 7.4.4 apply.

G.7.4.4.1 Slice data in scalable extension semantics

The semantics specified in subclause 7.4.4 apply with the following modifications.

mb_skip_run specifies the number of consecutive skipped macroblocks for which, when decoding an EP slice, mb_type shall be inferred to be P_Skip and the macroblock type is collectively referred to as a P macroblock type, or for which, when decoding an EB slice, mb_type shall be inferred to be B_Skip and the macroblock type is collectively referred to as a B macroblock type. The value of mb_skip_run shall be in the range of 0 to PicSizeInMbs − CurrMbAddr, inclusive.

mb_skip_flag equal to 1 specifies that for the current macroblock, when decoding an EP slice, mb_type shall be inferred to be P_Skip and the macroblock type is collectively referred to as P macroblock type, or for which, when decoding an EB slice, mb_type shall be inferred to be B_Skip and the macroblock type is collectively referred to as B macroblock type. mb_skip_flag equal to 0 specifies that the current macroblock is not skipped.

G.7.4.5 Macroblock layer semantics

The semantics specified in subclause 7.4.5 apply. Additionally, the following applies.

The macroblock_layer( ) syntax structure shall be considered to contain the following syntax elements with the following inferred values:

– base_mode_flag is inferred to be equal to 0.

– residual_prediction_flag is inferred to be equal to 0.

G.7.4.5.1 Macroblock prediction semantics

The semantics specified in subclause 7.4.5.1 apply. Additionally, the following applies.

The range of the components of mvd_l0[mbPartIdx][0][compIdx] and mvd_l1[mbPartIdx][0][compIdx] is specified by constraints on the motion vector variable values derived from it as specified in subclause G.10.

The mb_pred( ) syntax structure shall be considered to contain the following syntax elements with the following inferred values:

– motion_prediction_flag_l0[mbPartIdx] is inferred to be equal to 0 for each value of mbPartIdx in the range of 0 to NumMbPart(mb_type) − 1, inclusive.

– motion_prediction_flag_l1[mbPartIdx] is inferred to be equal to 0 for each value of mbPartIdx in the range of 0 to NumMbPart(mb_type) − 1, inclusive.

G.7.4.5.2 Sub-macroblock prediction semantics

The semantics specified in subclause 7.4.5.2 apply. Additionally, the following applies.

The range of the components of mvd_l0[mbPartIdx][subMbPartIdx][compIdx] and mvd_l1[mbPartIdx][subMbPartIdx][compIdx] is specified by constraints on the motion vector variable values derived from it as specified in subclause G.10.

The sub_mb_pred( ) syntax structure shall be considered to contain the following syntax elements with the following inferred values:

– motion_prediction_flag_l0[mbPartIdx] is inferred to be equal to 0 for each value of mbPartIdx in the range of 0 to 3, inclusive.

– motion_prediction_flag_l1[mbPartIdx] is inferred to be equal to 0 for each value of mbPartIdx in the range of 0 to 3, inclusive.

G.7.4.5.3 Residual data semantics

The semantics specified in subclause 7.4.5.3 apply.

G.7.4.5.3.1 Residual luma semantics

The semantics specified in subclause 7.4.5.3.1 apply.

G.7.4.5.3.2 Residual block CAVLC semantics

The semantics specified in subclause 7.4.5.3.2 apply.

G.7.4.5.3.3 Residual block CABAC semantics

The semantics specified in subclause 7.4.5.3.3 apply.

G.7.4.6 Macroblock layer in scalable extension semantics

The semantics specified in subclause 7.4.5 apply. Additionally, the following modifications and extensions are specified.

The function InCropWindow(mbAddr) is specified by the following ordered steps:

1. The variable mbX is set equal to ((mbAddr / (1 + MbaffFrameFlag)) % PicWidthInMbs).
2. The variables mbY0 and mbY1 are derived as follows:
   – If MbaffFrameFlag is equal to 0, mbY0 and mbY1 are set equal to (mbAddr / PicWidthInMbs).
   – Otherwise (MbaffFrameFlag is equal to 1), mbY0 is set equal to (2 * ((mbAddr / PicWidthInMbs) / 2)) and mbY1 is set equal to (mbY0 + 1).
3. The variable scalMbH is set equal to (16 * (1 + field_pic_flag)).

4. The return value of InCropWindow(mbAddr) is derived as follows:
   - If all of the following conditions are true, the return value of InCropWindow(mbAddr) is equal to TRUE.
     - no_inter_layer_pred_flag is equal to 0
     - mbX is greater than or equal to ((ScaledRefLayerLeftOffset + 15) / 16)
     - mbX is less than ((ScaledRefLayerLeftOffset + ScaledRefLayerPicWidthInSamples$_L$) / 16)
     - mbY0 is greater than or equal to ((ScaledRefLayerTopOffset + scalMbH − 1) / scalMbH)
     - mbY1 is less than ((ScaledRefLayerTopOffset + ScaledRefLayerPicHeightInSamples$_L$) / scalMbH)
   - Otherwise, the return value of InCropWindow(mbAddr) is equal to FALSE.

base_mode_flag equal to 1 specifies that the macroblock partitioning, the macroblock (partition) prediction mode(s), and the corresponding motion data (when applicable) are inferred as specified in subclause G.8. base_mode_flag equal to 0 specifies that the syntax element mb_type is present in the macroblock layer in scalable extension syntax structure or that mb_type shall be inferred as specified in subclause G.7.4.4.1.

When base_mode_flag is not present, base_mode_flag shall be inferred as follows:

- If InCropWindow(CurrMbAddr) is equal to 0, the value of base_mode_flag is inferred to be equal to 0.
- Otherwise, if the syntax element mb_skip_run (when entropy_coding_mode_flag is equal to 0) or mb_skip_flag (when entropy_coding_mode_flag is equal to 1) specifies that mb_type is inferred to be equal to P_Skip or B_Skip as specified in subclause G.7.4.4.1, the value of base_mode_flag is inferred to be equal to 0.
- Otherwise (InCropWindow(CurrMbAddr) is equal to 1 and the syntax element mb_skip_run (when entropy_coding_mode_flag is equal to 0) or mb_skip_flag (when entropy_coding_mode_flag is equal to 1) does not specify that mb_type is inferred to be equal to P_Skip or B_Skip), the value of base_mode_flag is inferred to be equal to default_base_mode_flag.

When store_ref_base_pic_flag is equal to 1 and quality_id is greater than 0, base_mode_flag shall be equal to 1.

mb_type specifies the macroblock type. The semantics of mb_type depend on the slice type.

When mb_type is not present, it shall be inferred as follows:

- If base_mode_flag is equal to 1, mb_type is inferred to be equal to Mb_Inferred.
- Otherwise, (base_mode_flag is equal to 0), mb_type is inferred as specified in subclause G.7.4.4.1.

The macroblock type Mb_Inferred specifies that the macroblock partitioning and the macroblock (partition) prediction mode(s) are not known during the parsing process. In the decoding process specified in subclause G.8, the macroblock type used for decoding is inferred to be equal to any of the macroblock types specified in Tables 7-11, 7-13, 7-14, or G-5. For the purpose of parsing the slice_data_in_scalable_extension( ) syntax structure including the processes specified in clause 9 and subclause G.9, Mb_Inferred shall be considered an additional macroblock type that is different from all macroblock types specified in Tables 7-11, 7-13, 7-14, and G-5 and the following applies:

- macroblocks with mb_type equal to Mb_Inferred are considered as coded in an Inter macroblock prediction mode and not coded in an Intra macroblock prediction mode,
- NumMbPart(Mb_Inferred) is considered to be equal to 1,
- MbPartWidth(Mb_Inferred) and MbPartHeight(Mb_Inferred) are considered to be equal to 16,
- MbPartPredMode(Mb_Inferred, 0) is considered to be not equal to Intra_4x4, Intra_8x8, Intra_16x16, Pred_L0, Pred_L1, BiPred, and Direct.

Tables and semantics are specified for the various macroblock types for EI, EP, and EB slices. Each table presents the value of mb_type, the name of mb_type, the number of macroblock partitions used (given by NumMbPart(mb_type) function), the prediction mode of the macroblock (when it is not partitioned) or the first partition (given by the MbPartPredMode(mb_type, 0) function) and the prediction mode of the second partition (given by the MbPartPredMode(mb_type, 1) function). When a value is not applicable it is designated by "na". In the text, the value of mb_type may be referred to as the macroblock type and a value X of MbPartPredMode( ) may be referred to in the text by "X macroblock (partition) prediction mode" or as "X prediction macroblocks". The tables do not include the macroblock type Mb_Inferred.

Table G-4 shows the allowed collective macroblock types for each slice_type.

Table G-4 – Allowed collective macroblock types for slice_type.

slice_type	allowed collective macroblock types
EI (slice)	I (see Table 7-11 and Table G-5) (macroblock types)
EP (slice)	P (see Table 7-13) and I (see Table 7-11 and Table G-5) (macroblock types)
EB (slice)	B (see Table 7-14) and I (see Table 7-11 and Table G-5) (macroblock types)

Macroblock types that may be collectively referred to as I macroblock types are specified in Tables G-5 and 7-11. mb_type values 0 to 25 are specified in Table 7-11. Table G-5 specifies the additional macroblock type I_BL that can be inferred in the decoding process specified in subclause G.8 for macroblocks with base_mode_flag equal to 1 (mb_type inferred to be equal to Mb_Inferred).

The macroblock types for EI slices are all I macroblock types.

Table G-5 – Inferred macroblock type I_BL for EI slices.

mb_type	Name of mb_type	transform_size_8x8_flag	MbPartPredMode (mb_type, 0)	Intra16x16PredMode	CodedBlockPatternChroma	CodedBlockPatternLuma
inferred	I_BL	na	Intra_Base	na	Equation 7-35	Equation 7-35

Intra_Base specifies the macroblock prediction mode and specifies that the intra prediction samples are derived using constructed intra samples of the reference layer representation as specified in subclause G.8. Intra_Base is an Intra macroblock prediction mode.

Macroblock types that may be collectively referred to as P macroblock types are specified in Table 7-13.

The macroblock types for EP slices are specified in Tables 7-13, 7-11, and G-5. mb_type values 0 to 4 are specified in Table 7-13 and mb_type values 5 to 30 are specified in Table 7-11, indexed by subtracting 5 from the value of mb_type. Table G-5 specifies the additional macroblock type I_BL that can be inferred in the decoding process specified in subclause G.8 for macroblocks with base_mode_flag equal to 1 (mb_type inferred to be equal to Mb_Inferred).

Macroblock types that may be collectively referred to as B macroblock types are specified in Table 7-14.

The macroblock types for EB slices are specified in Tables 7-14, 7-11, and G-5. mb_type values 0 to 22 are specified in Table 7-14 and mb_type values 23 to 48 are specified in Table 7-11, indexed by subtracting 23 from the value of mb_type. Table G-5 specifies the additional macroblock type I_BL that can be inferred in the decoding process specified in subclause G.8 for macroblocks with base_mode_flag equal to 1 (mb_type inferred to be equal to Mb_Inferred).

coded_block_pattern specifies which of the four 8x8 luma blocks and associated chroma blocks of a macroblock may contain non-zero transform coefficient values. When coded_block_pattern is present in the bitstream, the variables CodedBlockPatternLuma and CodedBlockPatternChroma are derived as specified by Equation 7-35.

When scan_idx_end is less than scan_idx_start and one of the following conditions is true, the variables CodedBlockPatternLuma and CodedBlockPatternChroma are set equal to 0:

– base_mode_flag is equal to 1,

– base_mode_flag is equal to 0, the macroblock type is not equal to P_Skip, B_Skip, or I_PCM, and the macroblock prediction mode is not equal to Intra_16x16.

When the macroblock type is not equal to P_Skip, B_Skip, or I_PCM, the following constraints shall be obeyed:

a) When scan_idx_end is less than scan_idx_start, and the macroblock prediction mode is equal to Intra_16x16, the bitstream shall not contain data that result in derived values of CodedBlockPatternLuma and CodedBlockPatternChroma that are not equal to 0.

b) When scan_idx_start is equal to 0, scan_idx_end is equal to 0, and the macroblock prediction mode is equal to Intra_16x16, the bitstream shall not contain data that result in a derived value of CodedBlockPatternLuma that is not equal to 0.

c) When scan_idx_start is equal to 0 and scan_idx_end is equal to 0, the bitstream shall not contain data that result in a derived value of CodedBlockPatternChroma that is equal to 2.

The meaning of CodedBlockPatternLuma and CodedBlockPatternChroma is specified in subclause 7.4.5.

residual_prediction_flag equal to 1 specifies that the residual signal of the current macroblock is predicted as specified in subclause G.8 using the reference layer representation specified by ref_layer_dq_id. residual_prediction_flag equal to 0 specifies that the residual signal of the current macroblock is not predicted.

When the syntax element residual_prediction_flag is not present, residual_prediction_flag shall be inferred as follows:

– If all of the following conditions are true, residual_prediction_flag is inferred to be equal to default_residual_prediction_flag:
  – slice_type is not equal to EI,
  – InCropWindow(CurrMbAddr) is equal to 1,
  – base_mode_flag is equal to 1 or mb_type does not specify an I macroblock type.
– Otherwise, residual_prediction_flag is inferred to be equal to 0.

All elements of the arrays LumaLevel4x4, LumaLevel8x8, Intra16x16DCLevel, Intra16x16ACLevel, CbLevel4x4, CbLevel8x8, CbIntra16x16DCLevel, CbIntra16x16ACLevel, CrLevel4x4, CrLevel8x8, CrIntra16x16DCLevel, CrIntra16x16ACLevel, ChromaDCLevel, and ChromaACLevel are set equal to 0 before parsing the residual( ) syntax structure. All elements of these arrays are also set equal to 0 when the residual( ) syntax structure is not present.

G.7.4.6.1 Macroblock prediction in scalable extension semantics

The semantics specified in subclause 7.4.5.1 apply. Additionally, the following semantics are specified.

motion_prediction_flag_l0[mbPartIdx **]** equal to 1 specifies that an alternative motion vector prediction process as specified in subclause G.8 is used for deriving the list 0 motion vector of the macroblock partition mbPartIdx and that the list 0 reference index of the macroblock partition mbPartIdx is inferred as specified in subclause G.8.

When motion_prediction_flag_l0[mbPartIdx] is not present, motion_prediction_flag_l0[mbPartIdx] shall be inferred as follows:

– If InCropWindow(CurrMbAddr) is equal to 0, motion_prediction_flag_l0[mbPartIdx] is inferred to be equal to 0.
– Otherwise (InCropWindow(CurrMbAddr) is equal to 1), motion_prediction_flag_l0[mbPartIdx] is inferred to be equal to default_motion_prediction_flag.

motion_prediction_flag_l1[mbPartIdx **]** has the same semantics as motion_prediction_flag_l0[mbPartIdx], with l0 and list 0 replaced by l1 and list 1, respectively.

G.7.4.6.2 Sub-macroblock prediction in scalable extension semantics

The semantics specified in subclause 7.4.5.2 apply. Additionally, the following semantics are specified.

motion_prediction_flag_l0[mbPartIdx **]** and **motion_prediction_flag_l1[** mbPartIdx **]** have the same semantics as specified in subclause G.7.4.6.1.

G.8 SVC decoding process

This subclause describes the decoding process for an access unit, given syntax elements and upper-case variables from subclause G.7 (with reference made to clause 7 in subclause G.7) that are derived from the bitstream.

NOTE 1 – All syntax elements and upper-case variables from subclause G.7 are available for the entire current access unit. When syntax elements or upper-case variables appear with identical names in subclause G.7 they are referred herein through unique identifiers.

Outputs of this process are decoded samples of the current primary coded picture.

The decoding process is specified such that all decoders shall produce numerically identical results. Any decoding process that produces identical results to the process described here conforms to the decoding process requirements of this Recommendation | International Standard.

All sub-bitstreams that can be derived using the sub-bitstream extraction process as specified in subclause G.8.8.1 with any combination of values for priority_id, temporal_id, dependency_id, or quality_id as the input shall result in a set of coded video sequences, with each coded video sequence conforming to one or more of the profiles specified in Annexes A and G.

This subclause specifies the decoding process for an access unit of a coded video sequence conforming to one or more of the profiles specified in subclause G.10.

Each picture referred to in this subclause is a complete primary coded picture or part of a primary coded picture. Each dependency representation referred to in this subclause is a dependency representation of a primary coded picture. Each layer representation referred to in this subclause is a layer representation of a primary coded picture. Each slice referred to in this subclause is a slice of a primary coded picture. All syntax elements and derived variables referred to in this subclause are syntax elements and derived variables for primary coded pictures.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the decoding process specified in this subclause and all child processes invoked from the process specified in this subclause are the syntax elements and derived upper-case variables for the current access unit.

The derivation process for the set of layer representations required for decoding as specified in subclause G.8.1.1 is invoked and the output is a list dqIdList of integer values specifying layer representation identifiers. The variables DQIdMin and DQIdMax are set equal to the minimum and maximum values, respectively, of the entries of the list dqIdList, and the variable DependencyIdMax is set equal to (DQIdMax >> 4). DependencyIdMax shall be the same for all access units of the coded video sequence.

At the start of the decoding process for an access unit, the following applies:

1. Variables and functions relating to picture order count are derived by invoking the SVC decoding process for picture order count as specified in subclause G.8.2.1 with dqIdList as the input.

2. The SVC decoding process for gaps in frame_num as specified in subclause G.8.2.5 is invoked with dqIdList as the input.

3. For each value of currDQId that is contained in the list dqIdList, the following applies:

   – The decoding process for macroblock to slice group map as specified in subclause 8.2.2 is invoked with the syntax elements of the NAL units with DQId equal to currDQId as the input. For this invocation of the process specified in subclause 8.2.2, when currDQId is less than DQIdMax, "active picture parameter set" is substituted with "active layer picture parameter set".

   – The function NextMbAddress( ) as specified in subclause 8.2.2 is used for parsing the slice data syntax structures of all NAL units with DQId equal to currDQId and for inferring slice data and macroblock layer syntax elements for slices with slice_skip_flag equal to 1 and DQId equal to currDQId (see subclause G.7.4.3.4).

The collective terms currentVars and refLayerVars are initially marked as not available.

The variable currDQId proceeds over the values DQIdMin..DQIdMax, and when a value of currDQId is present in the list dqIdList, the following ordered steps apply:

1. The variable spatResChangeFlag is set equal to the variable SpatialResolutionChangeFlag of the layer representation with DQId equal to currDQId.

2. Depending on spatResChangeFlag, the following applies:

   – If spatResChangeFlag is equal to 0, the base decoding process for layer representations without resolution change as specified in subclause G.8.1.3.1 is invoked with currDQId and currentVars as the inputs and the output is a modified version of currentVars.

   – Otherwise (spatResChangeFlag is equal to 1), the base decoding process for layer representations with resolution change as specified in subclause G.8.1.3.2 is invoked with currDQId and currentVars as the inputs and the outputs are variables assigned to the collective term refLayerVars and a modified version of currentVars.

3. When currDQId is equal to (DependencyIdMax << 4) and store_ref_base_pic_flag for the dependency representation with dependency_id equal to DependencyIdMax is equal to 1, the target layer representation decoding process as specified in subclause G.8.1.3.3 is invoked with currDQId, refLayerVars (when

spatResChangeFlag is equal to 1), and currentVars as the inputs and the outputs are assigned to the sample array B_L and, when ChromaArrayType is not equal to 0, the sample arrays B_{Cb} and B_{Cr}.

NOTE 2 – The sample arrays B_L, B_{Cb}, and B_{Cr} represent the reference base picture for an access unit with store_ref_base_pic_flag equal to 1 for the dependency representation with dependency_id equal to DependencyIdMax.

The target layer representation decoding process as specified in subclause G.8.1.3.3 is invoked with currDQId set equal to DQIdMax, refLayerVars (when the variable SpatialResolutionChangeFlag of the layer representation with DQId equal to DQIdMax is equal to 1), and currentVars as the inputs and the outputs are assigned to the sample array S_L and, when ChromaArrayType is not equal to 0, the sample arrays S_{Cb} and S_{Cr}.

NOTE 3 – The sample arrays S_L, S_{Cb}, and S_{Cr} represent the decoded picture for the access unit.

The SVC decoded reference picture marking process as specified in subclause G.8.2.4 is invoked with dqIdList as the input.

G.8.1 SVC initialisation and decoding processes

Subclause G.8.1.1 specifies the derivation process for the set of layer representations required for decoding.

Subclause G.8.1.2 specifies the array assignment, initialisation, and restructuring processes.

Subclause G.8.1.3 specifies the layer representation decoding processes.

Subclause G.8.1.4 specifies the slice decoding processes.

Subclause G.8.1.5 specifies the macroblock initialisation and decoding processes.

G.8.1.1 Derivation process for the set of layer representations required for decoding

Inputs to this process are the coded slice NAL units of an access unit.

Output of this process is a list dqIdList of integer values specifying layer representation identifiers.

With currDQId being set equal to the maximum value of DQId for all coded slice NAL units of the current access unit and with refLayerDQId(dqId) being a function that returns the value of MaxRefLayerDQId for the layer representation, of the current access unit, with DQId equal to dqId, the list dqIdList is derived as specified by the following pseudo-code.

```
numEntries = 0
dqIdList[ numEntries++ ] = currDQId
while( refLayerDQId( currDQId ) >= 0 )  {
    dqIdList[ numEntries++ ] = refLayerDQId( currDQId )                    (G-75)
    currDQId = dqIdList[ numEntries – 1 ]
}
```

G.8.1.2 Array assignment, initialisation, and restructuring processes

Subclause G.8.1.2.1 specifies the array assignment and initialisation process.

Subclause G.8.1.2.2 specifies the array restructuring process.

G.8.1.2.1 Array assignment and initialisation process

Output of this process is a set of arrays that are assigned to the collective term currentVars.

The following arrays are collectively referred to as currentVars:

– A one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr is referred to as sliceIdc[mbAddr]. All elements of the array sliceIdc are initially marked as unspecified.

– A one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of a layer representation are field macroblocks and which macroblocks are frame macroblocks. An element of this array for a macroblock with address mbAddr is referred to as fieldMbFlag[mbAddr]. All elements of the array fieldMbFlag are initially marked as unspecified.

– A one-dimensional array cTrafo with PicSizeInMbs elements specifying the luma and, when ChromaArrayType is equal to 3, chroma transform types for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr is referred to as cTrafo[mbAddr]. Unless marked as unspecified, each element of cTrafo is equal to T_4x4, T_8x8, T_16x16, or T_PCM. All elements of the array cTrafo are initially marked as unspecified.

– A one-dimensional array baseModeFlag with PicSizeInMbs elements specifying the syntax element base_mode_flag for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr is referred to as baseModeFlag[mbAddr]. All elements of the array baseModeFlag are initially marked as unspecified.

– A one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr is referred to as mbType[mbAddr]. Unless marked as unspecified, each element of mbType is equal to I_4x4, I_8x8, I_16x16, I_PCM, I_BL, or one of the Inter macroblock types specified in Tables 7-13 and 7-14. All elements of the array mbType are initially marked as unspecified.

– A (PicSizeInMbs)x4 array subMbType specifying sub-macroblock types for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr and a macroblock partition index mbPartIdx is referred to as subMbType[mbAddr][mbPartIdx]. A one-dimensional array specifying sub-macroblock types for the macroblock partitions of a macroblock with address mbAddr is referred to as subMbType[mbAddr]. Unless marked as unspecified, each element of subMbType is equal to one of the sub-macroblock types specified in Tables 7-17 and 7-18. All elements of the array subMbType are initially marked as unspecified.

– A one-dimensional array mvCnt with PicSizeInMbs elements specifying the number of motion vectors for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr is referred to as mvCnt[mbAddr]. All elements of the array mvCnt are initially set equal to 0.

– A one-dimensional array tQP_Y with PicSizeInMbs elements specifying luma quantisation parameters for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr is referred to as tQP_Y[mbAddr]. All elements of the array tQP_Y are initially set equal to 0.

– When ChromaArrayType is not equal to 0, two one-dimensional arrays tQP_{Cb} and tQP_{Cr} with PicSizeInMbs elements specifying Cb and Cr quantisation parameters, respectively, for the macroblocks of a layer representation. An element of these arrays for a macroblock with address mbAddr is referred to as tQP_{CX}[mbAddr] with CX being replaced by Cb or Cr. All elements of the arrays tQP_{Cb} and tQP_{Cr} are initially set equal to 0.

– A (PicSizeInMbs)x16 array ipred4x4 specifying Intra_4x4 prediction modes for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr and a 4x4 block with index c4x4BlkIdx is referred to as ipred4x4[mbAddr][c4x4BlkIdx]. A one-dimensional array specifying Intra_4x4 prediction modes for the 4x4 blocks of a macroblock with address mbAddr is referred to as ipred4x4[mbAddr]. All elements of the array ipred4x4 are initially marked as unspecified.

– A (PicSizeInMbs)x4 array ipred8x8 specifying Intra_8x8 prediction modes for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr and a 8x8 block with index c8x8BlkIdx is referred to as ipred8x8[mbAddr][c8x8BlkIdx]. A one-dimensional array specifying Intra_8x8 prediction modes for the 8x8 blocks of a macroblock with address mbAddr is referred to as ipred8x8[mbAddr]. All elements of the array ipred8x8 are initially marked as unspecified.

– A one-dimensional array ipred16x16 with PicSizeInMbs elements specifying Intra_16x16 prediction modes for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr is referred to as ipred16x16[mbAddr]. All elements of the array ipred16x16 are initially marked as unspecified.

– When ChromaArrayType is equal to 1 or 2, a one-dimensional array ipredChroma with PicSizeInMbs elements specifying intra chroma prediction modes for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr is referred to as ipredChroma[mbAddr]. All elements of the array ipredChroma are initially marked as unspecified.

– Two (PicSizeInMbs)x4 arrays predFlagL0 and predFlagL1 specifying prediction utilization flags for the macroblocks of a layer representation. An element of these arrays for a macroblock with address mbAddr and a macroblock partition index mbPartIdx is referred to as predFlagLX[mbAddr][mbPartIdx] with X being replaced by 0 or 1. A one-dimensional array specifying prediction utilization flags for the macroblock partitions of a macroblock with address mbAddr is referred to as predFlagLX[mbAddr] with X being replaced by 0 or 1. All elements of the arrays predFlagL0 and predFlagL1 are initially set equal to 0.

– Two (PicSizeInMbs)x4 arrays refIdxL0 and refIdxL1 specifying reference indices for the macroblocks of a layer representation. An element of these arrays for a macroblock with address mbAddr and a macroblock partition index mbPartIdx is referred to as refIdxLX[mbAddr][mbPartIdx] with X being replaced by 0 or 1. A one-dimensional array specifying reference indices for the macroblock partitions of a macroblock with address mbAddr is referred to as refIdxLX[mbAddr] with X being replaced by 0 or 1. All elements of the arrays refIdxL0 and refIdxL1 are initially set equal to −1.

– Two (PicSizeInMbs)x4x4x2 arrays mvL0 and mvL1 specifying motion vector components for the macroblocks of a layer representation. An element of these arrays for a macroblock with address mbAddr, a macroblock partition index mbPartIdx, a sub-macroblock partition index subMbPartIdx, and a motion vector component index c is referred to as mvLX[mbAddr][mbPartIdx][subMbPartIdx][c] with X being replaced by 0 or 1. A one-dimensional array with 2 elements representing the motion vector for a sub-macroblock partition subMbPartIdx of a macroblock partition mbPartIdx inside a macroblock mbAddr is referred to as mvLX[mbAddr][mbPartIdx][subMbPartIdx] with X being replaced by 0 or 1. A 4x2 array representing the motion vectors for a macroblock partition mbPartIdx inside a macroblock mbAddr is referred to as mvLX[mbAddr][mbPartIdx] with X being replaced by 0 or 1. A 4x4x2 array representing the motion vectors for a macroblock mbAddr is referred to as mvLX[mbAddr] with X being replaced by 0 or 1. A motion vector component with component index c for a macroblock partition mbPartIdx of a macroblock mbAddr that is not split into sub-macroblock partitions can also be referred to as mvLX[mbAddr][mbPartIdx][c] with X being replaced by 0 or 1, which is identical to mvLX[mbAddr][mbPartIdx][0][c]. A motion vector for a macroblock partition mbPartIdx of a macroblock mbAddr that is not split into sub-macroblock partitions can also be referred to as mvLX[mbAddr][mbPartIdx] with X being replaced by 0 or 1, which is identical to mvLX[mbAddr][mbPartIdx][0]. All elements of the arrays mvL0 and mvL1 are initially set equal to 0.

– A (PicSizeInMbs)x(256 + 2 * MbWidthC * MbHeightC) array tCoeffLevel specifying transform coefficient level values for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr and a transform coefficient index tCoeffIdx is referred to as tCoeffLevel[mbAddr][tCoeffIdx]. A one-dimensional array specifying the transform coefficient level values for a macroblock with address mbAddr is referred to as tCoeffLevel[mbAddr]. All elements of the array tCoeffLevel are initially set equal to 0.

– A (PicSizeInMbs)x(256 + 2 * MbWidthC * MbHeightC) array sTCoeff specifying scaled transform coefficient values for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr and a transform coefficient index tCoeffIdx is referred to as sTCoeff[mbAddr][tCoeffIdx]. A one-dimensional array specifying the scaled transform coefficient values for a macroblock with address mbAddr is referred to as sTCoeff[mbAddr]. All elements of the array sTCoeff are initially set equal to 0.

– A ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array rS_L specifying residual luma sample values for a layer picture. An element of this array for a luma location (x, y) relative to the upper-left luma sample of the macroblock with address 0 is referred to as rS_L[x, y]. All elements of the array rS_L are initially set equal to 0.

– When ChromaArrayType is not equal to 0, two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays rS_{Cb} and rS_{Cr} specifying residual chroma sample values for a layer picture. An element of these arrays for a chroma location (x, y) relative to the upper-left chroma sample of the macroblock with address 0 is referred to as rS_{CX}[x, y] with CX being replaced by Cb or Cr. All elements of the arrays rS_{Cb} and rS_{Cr} are initially set equal to 0.

– A ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array cS_L specifying constructed luma sample values for a layer picture. An element of this array for a luma location (x, y) relative to the upper-left luma sample of the macroblock with address 0 is referred to as cS_L[x, y]. All elements of the array cS_L are initially set equal to 0.

– When ChromaArrayType is not equal to 0, two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays cS_{Cb} and cS_{Cr} specifying constructed chroma sample values for a layer picture. An element of these arrays for a chroma location (x, y) relative to the upper-left chroma sample of the macroblock with address 0 is referred to as cS_{CX}[x, y] with CX being replaced by Cb or Cr. All elements of the arrays cS_{Cb} and cS_{Cr} are initially set equal to 0.

G.8.1.2.2 Array restructuring process

This process is only invoked when MinNoInterLayerPredFlag is equal to 0, SpatialResolutionChangeFlag is equal to 0, and any of the variables ScaledRefLayerLeftOffset, ScaledRefLayerRightOffset, ScaledRefLayerTopOffset, or ScaledRefLayerBottomOffset is not equal to 0.

Input to this process is a set of arrays collectively referred to as currentVars.

Output of this process is the set of arrays collectively referred to as currentVars with modifications related to the array sizes as well as the ordering of array elements.

The variables that are assigned to the collective term currentVars are assigned to the collective term refLayerVars.

The array assignment and initialisation process as specified in subclause G.8.1.2.1 is invoked and the output is the set of arrays collectively referred to as currentVars.

The variables xOffset, yOffset, xOffsetC, and yOffsetC are derived by

$$xOffset = ScaledRefLayerLeftOffset \tag{G-76}$$
$$yOffset = ScaledRefLayerTopOffset / (1 + field\_pic\_flag) \tag{G-77}$$

$$xOffsetC = (xOffset >> 4) * MbWidthC \tag{G-78}$$
$$yOffsetC = (yOffset >> 4) * MbHeightC \tag{G-79}$$

For the macroblock address mbAddr proceeding over the values 0..(PicSizeInMbs − 1), the following ordered steps are specified:

1. With eS set equal to (1 + MbaffFrameFlag), the variables refMbX and refMbY are derived by

$$refMbX = ((mbAddr / eS) \% PicWidthInMbs) − (xOffset / 16) \tag{G-80}$$
$$refMbY = ((mbAddr / eS) / PicWidthInMbs) * eS + (mbAddr \% eS) − (yOffset / 16) \tag{G-81}$$

2. The reference layer macroblock address refMbAddr is derived as follows:

   – If any of the following conditions are true, refMbAddr is marked as not available:

     – refMbX is less than 0 or refMbX is greater than or equal to RefLayerPicWidthInMbs,

     – refMbY is less than 0 or refMbY is greater than or equal to RefLayerPicHeightInMbs.

   – Otherwise, with bS set equal to (1 + RefLayerMbaffFrameFlag), refMbAddr is derived by

$$refMbAddr = (refMbY / bS) * bS * RefLayerPicWidthInMbs + (refMbY \% bS) + refMbX \tag{G-82}$$

3. When refMbAddr is available, for X being replaced by sliceIdc, fieldMbFlag, cTrafo, baseModeFlag, mbType, subMbType, mvCnt, tQP_Y, tQP_{Cb} (when ChromaArrayType is not equal to 0), tQP_{Cr} (when ChromaArrayType is not equal to 0), ipred4x4, ipred8x8, ipred16x16, ipredChroma (when ChromaArrayType is equal to 1 or 2), predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, tCoeffLevel, and sTCoeff and with currArray representing the array X of the collective term currentVars and refLayerArray representing the array X of the collective term refLayerVars, the array element currArray[mbAddr], which can be a scalar or an array, is set equal to the array element refLayerArray[refMbAddr].

For X being replaced by rS_L and cS_L and with currArray representing the array X of the collective term currentVars and refLayerArray representing the array X of the collective term refLayerVars, the array currArray is modified by

$$\begin{aligned}&currArray[x, y] = refLayerArray[x − xOffset, y − yOffset]\\&\text{with } x = Max(0, xOffset)..Min(PicWidthInSamples_L, RefLayerPicWidthInSamples_L + xOffset) − 1\\&\text{and } y = Max(0, yOffset)..Min(PicHeightInSamples_L, RefLayerPicHeightInSamples_L + yOffset) − 1\end{aligned} \tag{G-83}$$

When ChromaArrayType is not equal to 0, for X being replaced by rS_{Cb}, rS_{Cr}, cS_{Cb}, and cS_{Cr} and with currArray representing the array X of the collective term currentVars and refLayerArray representing the array X of the collective term refLayerVars, the array currArray is modified by

$$\begin{aligned}&currArray[x, y] = refLayerArray[x − xOffsetC, y − yOffsetC]\\&\text{with } x = Max(0, xOffsetC)..Min(PicWidthInSamples_C, RefLayerPicWidthInSamples_C + xOffsetC) − 1\\&\text{and } y = Max(0, yOffsetC)..Min(PicHeightInSamples_C, RefLayerPicHeightInSamples_C + yOffsetC) − 1\end{aligned} \tag{G-84}$$

G.8.1.3 Layer representation decoding processes

Subclause G.8.1.3.1 specifies the base decoding process for layer representations without resolution change.

Subclause G.8.1.3.2 specifies the base decoding process for layer representations with resolution change.

Subclause G.8.1.3.3 specifies the target layer representation decoding process

G.8.1.3.1 Base decoding process for layer representations without resolution change

Inputs to this process are:

– a variable currDQId specifying the current layer representation,

– a set of arrays collectively referred to as currentVars.

Output of this process is the modified set of arrays collectively referred to as currentVars.

This process modifies the variables assigned to currentVars using syntax elements and derived upper-case variables for the current layer representation with DQId equal to currDQId.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the current layer representation with DQId equal to currDQId.

The base decoding process for layer representations without resolution change proceeds in the following ordered steps:

1. Depending on MinNoInterLayerPredFlag, the following applies:
   - If MinNoInterLayerPredFlag is equal to 1, the array assignment and initialisation process as specified in subclause G.8.1.2.1 is invoked and the output is a modified set of arrays collectively referred to as currentVars.
   - Otherwise (MinNoInterLayerPredFlag is equal to 0), the following ordered steps are specified:
     a. When MaxTCoeffLevelPredFlag is equal to 0, the macroblock address mbAddr proceeds over the values 0..(RefLayerPicSizeInMbs − 1), and for each macroblock address mbAddr, the macroblock decoding process prior to decoding a layer representation without resolution change and MaxTCoeffLevelPredFlag equal to 0 as specified in subclause G.8.1.5.4 is invoked with currDQId set equal to MaxRefLayerDQId, mbAddr, and currentVars as the inputs and the output is a modified version of currentVars.
     b. When any of the variables ScaledRefLayerLeftOffset, ScaledRefLayerRightOffset, ScaledRefLayerTopOffset, or ScaledRefLayerBottomOffset is not equal to 0, the array restructuring process as specified in subclause G.8.1.2.2 is invoked with currentVars as the input and the output is a modified version of currentVars.
2. Let setOfSlices be the set of all slices of the current layer representation with DQId equal to currDQId. For each slice of the set setOfSlices, the base decoding process for slices without resolution change as specified in subclause G.8.1.4.1 is invoked with currSlice representing the currently processed slice, currDQId, and currentVars as the inputs and the output is a modified version of currentVars.
3. When currDQId is less than or equal to (DependencyIdMax << 4), with sliceIdc being the array sliceIdc of the collective term currentVars, the bitstream shall not contain data that result in any value of (sliceIdc[mbAddr] & 127) with mbAddr = 0..(PicSizeInMbs − 1) not equal to currDQId.

   NOTE – This constraint and a similar constraint in subclause G.8.1.3.2 specify that all layer representations with quality_id equal to 0 and all layer representations that are used for inter-layer prediction must be completely covered by the slices of the access unit. An additional constraint for layer representations with dependency_id equal to DependencyIdMax and quality_id greater than 0 is specified in subclause G.8.1.5.1.

G.8.1.3.2 Base decoding process for layer representations with resolution change

Inputs to this process are:

– a variable currDQId specifying the current layer representation,

– a set of arrays collectively referred to as currentVars.

Outputs of this process are:

– a set of arrays collectively referred to as refLayerVars,

– the modified set of arrays collectively referred to as currentVars.

This process modifies the variables assigned to currentVars using syntax elements and derived upper-case variables for the current layer representation with DQId equal to currDQId.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the current layer representation with DQId equal to currDQId.

The base decoding process for layer representations with resolution change proceeds in the following ordered steps:

1. The macroblock address mbAddr proceeds over the values 0..(RefLayerPicSizeInMbs − 1), and for each macroblock address mbAddr, the macroblock decoding process prior to resolution change as specified in subclause G.8.1.5.5 is invoked with currDQId set equal to MaxRefLayerDQId, mbAddr, and currentVars as the inputs and the output is a modified version of currentVars.
2. The deblocking filter process for Intra_Base prediction as specified in subclause G.8.7.1 is invoked with currDQId and currentVars as the inputs and the output is a modified version of currentVars.
3. The variables that are assigned to the collective term currentVars are assigned to the collective term refLayerVars.
4. The array assignment and initialisation process as specified in subclause G.8.1.2.1 is invoked and the output is assigned to the collective term currentVars.

5. Let setOfSlices be the set of all slices of the current layer representation with DQId equal to currDQId. For each slice of the set setOfSlices, the base decoding process for slices with resolution change as specified in subclause G.8.1.4.2 is invoked with currSlice representing the currently processed slice, currDQId, refLayerVars, and currentVars as the inputs and the output is a modified version of currentVars.

6. With sliceIdc being the array sliceIdc of the collective term currentVars, the bitstream shall not contain data that result in any value of (sliceIdc[mbAddr] & 127) with mbAddr = 0..(PicSizeInMbs − 1) not equal to currDQId.

   NOTE – This constraint and a similar constraint in subclause G.8.1.3.1 specify that all layer representations with quality_id equal to 0 and all layer representation that are used for inter-layer prediction must be completely covered by the slices of the access unit. An additional constraint for layer representations with dependency_id equal to DependencyIdMax and quality_id greater than 0 is specified in subclause G.8.1.5.1.

G.8.1.3.3 Target layer representation decoding process

Inputs to this process are:

– a variable currDQId specifying the current layer representation,

– when present, a set of arrays collectively referred to as refLayerVars,

– a set of arrays collectively referred to as currentVars.

Outputs of this process are:

– a $(PicWidthInSamples_L)x(PicHeightInSamples_L)$ array s_L containing constructed luma sample values,

– when ChromaArrayType is not equal to 0, two $(PicWidthInSamples_C)x(PicHeightInSamples_C)$ arrays s_{Cb} and s_{Cr} containing constructed chroma sample values.

In this process the constructed samples of the array s_L and, when ChromaArrayType is not equal to 0, the arrays s_{Cb} and s_{Cr} are derived using the variables that are assigned to currentVars.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the current layer representation with DQId equal to currDQId.

The target layer representation decoding process proceeds in the following ordered steps:

1. The variables that are assigned to the collective term currentVars are assigned to the collective term tempVars, and in the following of this subclause, the arrays that are collectively referred to as tempVars are referred to by their names as specified in subclause G.8.1.2.1.

   NOTE 1 – Any following modification of the variables assigned to the collective term tempVars does not influence the variables assigned to the collective term currentVars.

2. The macroblock address mbAddr proceeds over the values 0..(PicSizeInMbs − 1), and for each macroblock address mbAddr, the following ordered steps are specified:

   a. Let currSlice specify the slice of the layer representation with DQId equal to (((sliceIdc[mbAddr] & 127) >> 4) << 4) that covers the macroblock with macroblock address ((sliceIdc[mbAddr] >> 7) * (1 + MbaffFrameFlag)).

   b. Let firstMbInSlice and sliceType be the syntax elements first_mb_in_slice and slice_type of the slice currSlice.

   c. The variable firstMbAddrInSlice is set equal to (firstMbInSlice * (1 + MbaffFrameFlag)).

   d. The reference picture lists refPicList0 and refPicList1 are marked as not available.

   e. When (sliceType % 5) is less than 2, the following applies:

      – If mbAddr is greater than firstMbAddrInSlice, the reference picture list refPicList0 is set equal to the reference picture list refPicList0 that was derived for the macroblock address mbAddr equal to firstMbAddrInSlice inside this subclause and, when (sliceType % 5) is equal to 1, the reference picture list refPicList1 is set equal to the reference picture list refPicList1 that was derived for the macroblock address mbAddr equal to firstMbAddrInSlice inside this subclause.

      – Otherwise (mbAddr is equal to firstMbAddrInSlice), the SVC decoding process for reference picture lists construction as specified in subclause G.8.2.3 is invoked with currDependencyId set equal to dependency_id, useRefBasePicFlag set equal to use_ref_base_pic_flag, and the slice currSlice as the inputs and the outputs are the modified reference picture list refPicList0 and, when (sliceType % 5) is equal to 1, the modified reference picture list refPicList1.

NOTE 2 – The reference picture lists refPicList0 and refPicList1 are only constructed for the slices of the layer representation with dependency_id equal to DependencyIdMax and quality_id equal to 0. For slices with dependency_id equal to DependencyIdMax and quality_id greater than 0, the reference picture lists are inferred.

f. The target macroblock decoding process as specified in subclause G.8.1.5.6 is invoked with currDQId, mbAddr, refLayerVars (when present as input to this subclause), tempVars, refPicList0 (when available), and refPicList1 (when available) as the inputs and the output is a modified version of tempVars.

NOTE 3 – Although the target layer representation decoding process is invoked twice for pictures with store_ref_base_pic_flag equal to 1, only a single motion compensation operation is needed for each macroblock.

3. The deblocking filter process for target representations as specified in subclause G.8.7.2 is invoked with currDQId and tempVars as the inputs and the output is a modified version of tempVars.

4. All sample values of the array cS_L are copied to the array s_L, which is output of this subclause.

5. When ChromaArrayType is not equal to 0, all sample values of the arrays cS_{Cb} and cS_{Cr} are copied to the arrays s_{Cb} and s_{Cr}, respectively, which are output of this subclause.

G.8.1.4 Slice decoding processes

Subclause G.8.1.4.1 specifies the base decoding process for slices without resolution change.

Subclause G.8.1.4.2 specifies the base decoding process for slices with resolution change.

G.8.1.4.1 Base decoding process for slices without resolution change

Inputs to this process are:

– the current slice currSlice,

– a variable currDQId specifying the current layer representation,

– a set of arrays collectively referred to as currentVars.

Output of this process is the modified set of arrays collectively referred to as currentVars.

This process modifies the variables assigned to currentVars using syntax elements and derived upper-case variables for the current slice currSlice.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the slice header of the current slice currSlice, the current picture parameter, which is identified by the syntax element pic_parameter_set_id inside the slice header of the current slice currSlice, and the current sequence parameter, which is identified by the syntax element seq_parameter_set_id inside the current picture parameter set.

When currDQId is equal to 0 and (slice_type % 5) is equal to 1, the SVC decoding process for reference picture lists construction as specified in subclause G.8.2.3 is invoked with currDependencyId equal to 0, useRefBasePicFlag equal to use_ref_base_pic_flag, and the current slice as input and the output is the reference picture list refPicList1.

The macroblocks of the current slice currSlice are processed in increasing order of their macroblock addresses. For each macroblock with macroblock address mbAddr, the base decoding process for macroblocks in slices without resolution change as specified in subclause G.8.1.5.2 is invoked with currDQId, mbAddr, currentVars, and, when currDQId is equal to 0 and (slice_type % 5) is equal to 1, the reference picture list refPicList1 as the inputs and the output is a modified version of currentVars.

G.8.1.4.2 Base decoding process for slices with resolution change

Inputs to this process are:

– the current slice currSlice,

– a variable currDQId specifying the current layer representation,

– a set of arrays collectively referred to as refLayerVars,

– a set of arrays collectively referred to as currentVars.

Output of this process is the modified set of arrays collectively referred to as currentVars.

This process modifies the variables assigned to currentVars using syntax elements and derived upper-case variables for the current slice currSlice.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the slice header of the current slice currSlice, the current picture parameter, which is identified by the syntax element pic_parameter_set_id inside the slice header of the current slice currSlice, and the current sequence parameter, which is identified by the syntax element seq_parameter_set_id inside the current picture parameter set.

When CroppingChangeFlag is equal to 1 and (slice_type % 5) is less than 2, the SVC decoding process for reference picture lists construction as specified in subclause G.8.2.3 is invoked with currDependencyId equal to dependency_id, useRefBasePicFlag equal to use_ref_base_pic_flag, and the current slice as the inputs and the outputs are the reference picture list refPicList0 and, when (sliceType % 5) is equal to 1, the reference picture list refPicList1.

The macroblocks of the current slice currSlice are processed in increasing order of their macroblock addresses. For each macroblock with macroblock address mbAddr, the base decoding process for macroblocks in slices with resolution change as specified in subclause G.8.1.5.3 is invoked with currDQId, mbAddr, refLayerVars, currentVars, refPicList0 (when CroppingChangeFlag is equal to 1 and (slice_type % 5) is less than 2), and refPicList1 (when CroppingChangeFlag is equal to 1 and (slice_type % 5) is equal to 1) as the inputs and the output is a modified version of currentVars.

G.8.1.5 Macroblock initialisation and decoding processes

Subclause G.8.1.5.1 specifies the macroblock initialisation process.

Subclause G.8.1.5.2 specifies the base decoding process for macroblocks in slices without resolution change.

Subclause G.8.1.5.3 specifies the base decoding process for macroblocks in slices with resolution change.

Subclause G.8.1.5.4 specifies the macroblock decoding process prior to decoding a layer representation without resolution change and MaxTCoeffLevelPredFlag equal to 0.

Subclause G.8.1.5.5 specifies the macroblock decoding process prior to resolution change.

Subclause G.8.1.5.6 specifies the target macroblock decoding process.

G.8.1.5.1 Macroblock initialisation process

Inputs to this process are:

– a set of arrays collectively referred to as refLayerVars,
– when CroppingChangeFlag is equal to 1 and (slice_type % 5) is less than 2, the reference picture list refPicList0,
– when CroppingChangeFlag is equal to 1 and (slice_type % 5) is equal to 1, the reference picture list refPicList1.

Outputs of this process are:

– a variable sliceIdc specifying the slice identification for the current macroblock,
– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,
– a variable cTrafo specifying the transform type for the current macroblock,
– a variable baseModeFlag specifying the syntax element base_mode_flag of the current macroblock,
– a variable mbType specifying the macroblock type of the current macroblock,
– a list subMbType with 4 elements specifying the sub-macroblock types of the current macroblock,
– a variable mvCnt specifying an initialisation value for the motion vector count of the current macroblock,
– a variable tQP_Y specifying the luma quantisation parameter for the current macroblock,
– when ChromaArrayType is not equal to 0, two variables tQP_{Cb} and tQP_{Cr} specifying the chroma quantisation parameters for the current macroblock,
– two 2x2 arrays refIdxILPredL0 and refIdxILPredL1 specifying inter-layer predictors for the reference indices of the current macroblock,

– two 4x4x2 arrays mvILPredL0 and mvILPredL1 specifying inter-layer predictors for the motion vector components of the current macroblock.

Inside this subclause, the arrays sliceIdc, fieldMbFlag, cTrafo, mbType, subMbType, tQP_Y, predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, tCoeffLevel, and sTCoeff that are collectively referred to as refLayerVars are referred to as refLayerSliceIdc, refLayerFieldMbFlag, refLayerCTrafo, refLayerMbType, refLayerSubMbType, $refLayerQP_Y$, refLayerPredFlagL0, refLayerPredFlagL1, refLayerRefIdxL0, refLayerRefIdxL1, refLayerMvL0, refLayerMvL1, refLayerTCoeffLevel, and refLayerSTCoeff, respectively.

The variable sliceIdc is set equal to ((first_mb_in_slice << 7) + DQId).

The variable baseModeFlag is set equal to base_mode_flag.

The variable fieldMbFlag is derived as follows:

– If field_pic_flag is equal to 1, fieldMbFlag is set equal to 1.

– Otherwise, if SpatialResolutionChangeFlag is equal to 0 and slice_skip_flag is equal to 1, fieldMbFlag is set equal to refLayerFieldMbFlag[CurrMbAddr].

– Otherwise, fieldMbFlag is set equal to mb_field_decoding_flag.

The derivation process for macroblock type, sub-macroblock type, and inter-layer predictors for reference indices and motion vectors as specified in subclause G.8.1.5.1.1 is invoked with fieldMbFlag, refLayerFieldMbFlag, refLayerMbType, refLayerSubMbType, refLayerPredFlagL0, refLayerPredFlagL1, refLayerRefIdxL0, refLayerRefIdxL1, refLayerMvL0, refLayerMvL1, refPicList0 (when available), and refPicList1 (when available) as the inputs and the outputs are the variable mbType, the list subMbType, the 2x2 arrays refIdxILPredL0 and refIdxILPredL1, and the 4x4x2 arrays mvILPredL0 and mvILPredL1.

The derivation process for quantisation parameters and transform type as specified in subclause G.8.1.5.1.2 is invoked with mbType, subMbType, refLayerMbType, refLayerCTrafo, $refLayerQP_Y$, refLayerTCoeffLevel, and refLayerSTCoeff as the inputs and the outputs are cTrafo, tQP_Y, and, when ChromaArrayType is not equal to 0, tQP_{Cb} and tQP_{Cr}.

The variable mvCnt is set equal to 0.

When quality_id is greater than 0, the bitstream shall not contain data that result in (refLayerSliceIdc[CurrMbAddr] & 127) not equal to (DQId − 1).

When no_inter_layer_pred_flag is equal to 0, SpatialResolutionChangeFlag is equal to 0, and fieldMbFlag is not equal to refLayerFieldMbRef[CurrMbAddr], the following constraints shall be obeyed:

a) The bitstream shall not contain data that result in base_mode_flag equal to 1, or any motion_prediction_flag_lX[mbPartIdx] with X being replaced by 0 and 1 and mbPartIdx = 0..3 equal to 1.

b) When residual_prediction_flag is equal to 1, refLayerMbType[CurrMbAddr] is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL, and mbType is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL, the following applies:

  – If tcoeff_level_prediction_flag is equal to 0, the bitstream shall not contain data that result in any element refLayerSTCoeff[CurrMbAddr][i] not equal to 0 for i = 0..(255 + 2 * MbWidthC * MbHeightC).

  – Otherwise (tcoeff_level_prediction_flag is equal to 1), the bitstream shall not contain data that result in any element refLayerTCoeffLevel[CurrMbAddr][i] not equal to 0 for i = 0..(255 + 2 * MbWidthC * MbHeightC).

G.8.1.5.1.1 Derivation process for macroblock type, sub-macroblock type, and inter-layer predictors for reference indices and motion vectors

Inputs to this process are:

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying the macroblock types for the macroblocks of the reference layer representation,

– a (RefLayerPicSizeInMbs)x4 array refLayerSubMbType specifying the sub-macroblock types for the macroblocks of the reference layer representation,

– two (RefLayerPicSizeInMbs)x4 arrays refLayerPredFlagL0 and refLayerPredFlagL1 specifying prediction utilization flags for the macroblocks of the reference layer representation,

– two (RefLayerPicSizeInMbs)x4 arrays refLayerRefIdxL0 and refLayerRefIdxL1 specifying reference indices for the macroblocks of the reference layer representation,

– two (RefLayerPicSizeInMbs)x4x4x2 arrays refLayerMvL0 and refLayerMvL1 specifying motion vector components for the macroblocks of the reference layer representation,

– when CroppingChangeFlag is equal to 1 and (slice_type % 5) is less than 2, the reference picture list refPicList0,

– when CroppingChangeFlag is equal to 1 and (slice_type % 5) is equal to 1, the reference picture list refPicList1.

Outputs of this process are:

– a variable mbType specifying the macroblock type of the current macroblock,

– a list subMbType with 4 elements specifying the sub-macroblock types of the current macroblock,

– two 2x2 arrays refIdxILPredL0 and refIdxILPredL1 specifying inter-layer predictors for the reference indices of the current macroblock,

– two 4x4x2 arrays mvILPredL0 and mvILPredL1 specifying inter-layer predictors for the motion vector components of the current macroblock.

The variable mbTypeILPred, the list subMbTypeILPred, the 2x2 arrays refIdxILPredL0 and refIdxILPredL1, and the 4x4x2 arrays mvILPredL0 and mvILPredL1 are derived as follows:

– If base_mode_flag is equal to 1 or any syntax element motion_prediction_flag_lX[mbPartIdx] with X being replaced by 0 and 1 and mbPartIdx = 0..3 is equal to 1, the derivation process for inter-layer predictors for macroblock type, sub-macroblock type, reference indices, and motion vectors as specified in subclause G.8.6.1 is invoked with fieldMbFlag, refLayerFieldMbFlag, refLayerMbType, refLayerSubMbType, refLayerPredFlagL0, refLayerPredFlagL1, refLayerRefIdxL0, refLayerRefIdxL1, refLayerMvL0, refLayerMvL1, refPicList0 (when available), and refPicList1 (when available) as the inputs and the outputs are the variable mbTypeILPred, the list subMbTypeILPred, the 2x2 arrays refIdxILPredL0 and refIdxILPredL1, and the 4x4x2 arrays mvILPredL0 and mvILPredL1.

– Otherwise (base_mode_flag is equal to 0 and all syntax elements motion_prediction_flag_lX[mbPartIdx] with X being replaced by 0 and 1 and mbPartIdx = 0..3 are equal to 0), mbTypeILPred is marked as not available, all elements subMbTypeILPred[mbPartIdx] with mbPartIdx = 0..3 of the list subMbTypeILPred are marked as not available, all elements of the 2x2 arrays refIdxILPredL0 and refIdxILPredL1 are set equal to −1, and all elements of the 4x4x2 arrays mvILPredL0 and mvILPredL1 are set equal to 0.

Depending on base_mode_flag, mb_type, SpatialResolutionChangeFlag, refLayerMbType[CurrMbAddr], CodedBlockPatternLuma, and CodedBlockPatternChroma, the variable mbType is derived as follows:

– If base_mode_flag is equal to 1, the following applies:

  – If SpatialResolutionChangeFlag is equal to 0, refLayerMbType[CurrMbAddr] is equal to I_PCM, CodedBlockPatternLuma is equal to 0, and CodedBlockPatternChroma is equal to 0, mbType is set equal to I_PCM.

  – Otherwise (SpatialResolutionChangeFlag is equal to 1, refLayerMbType[CurrMbAddr] is not equal to I_PCM, CodedBlockPatternLuma is not equal to 0, or CodedBlockPatternChroma is not equal to 0), mbType is set equal to mbTypeILPred.

– Otherwise, if MbPartPredMode(mb_type, 0) is equal to Intra_4x4, mbType is set equal to I_4x4.

– Otherwise, if MbPartPredMode(mb_type, 0) is equal to Intra_8x8, mbType is set equal to I_8x8.

– Otherwise, if MbPartPredMode(mb_type, 0) is equal to Intra_16x16, mbType is set equal to I_16x16.

– Otherwise, if mb_type is equal to I_PCM, mbType is set equal to I_PCM.

– Otherwise (base_mode_flag is equal to 0 and mb_type specifies a P or B macroblock type), mbType is set equal to mb_type.

Depending on mbType and base_mode_flag, the list subMbType is derived as follows:

– If mbType is not equal to P_8x8, P_8x8ref0, or B_8x8, all elements subMbType[mbPartIdx] with mbPartIdx = 0..3 are marked as unspecified.

– Otherwise, if base_mode_flag is equal to 1, each element subMbType[mbPartIdx] with mbPartIdx = 0..3 is set equal to subMbTypeILPred[mbPartIdx].

– Otherwise (mbType is equal to P_8x8, P_8x8ref0, or B_8x8 and base_mode_flag is equal to 0), each element subMbType[mbPartIdx] with mbPartIdx = 0..3 is set equal to sub_mb_type[mbPartIdx].

When slice_type is equal to EP, base_mode_flag is equal to 1, and mbType is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL, the bitstream shall not contain data that result in any element refIdxILPredL0[mbPartIdx] with mbPartIdx = 0..3 that is less than 0.

When residual_prediction_flag equal to 1 is present in the bitstream, the bitstream shall not contain data that result in mbType being equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL.

G.8.1.5.1.2 Derivation process for quantisation parameters and transform type

Inputs to this process are:

– a variable mbType specifying the macroblock type for the current macroblock,

– a list subMbType with 4 elements specifying the sub-macroblock types for the current macroblock,

– a one-dimensional array refLayerMbType specifying macroblock types for the macroblocks of the reference layer representation,

– a one-dimensional array refLayerCTrafo specifying transform types for the macroblocks of the reference layer representation,

– a one-dimensional array refLayerQP_Y specifying luma quantisation parameters for the macroblocks of the reference layer representation,

– an (RefLayerPicSizeInMbs)x(256 + 2 * MbWidthC * MbHeightC) array refLayerTCoeffLevel specifying transform coefficient level values for the macroblocks of the reference layer representation,

– an (RefLayerPicSizeInMbs)x(256 + 2 * MbWidthC * MbHeightC) array refLayerSTCoeff specifying scaled transform coefficient values for the macroblocks of the reference layer representation.

Outputs of this process are:

– a variable cTrafo specifying the transform type for the current macroblock,

– a variable tQP_Y specifying the luma quantisation parameter for the current macroblock,

– when ChromaArrayType is not equal to 0, two variables tQP_{Cb} and tQP_{Cr} specifying the chroma quantisation parameters for the current macroblock.

The variable tQP_Y is derived as follows:

– If SpatialResolutionChangeFlag is equal to 0, and any of the following conditions are true, tQP_Y is set equal to refLayerQP_Y[CurrMbAddr]:

  – mbType is equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL, base_mode_flag is equal to 1, CodedBlockPatternLuma is equal to 0, and CodedBlockPatternChroma is equal to 0,

  – mbType is equal to P_Skip or B_Skip and residual_prediction_flag is equal to 1,

  – mbType is not equal to I_PCM, I_16x16, I_8x8, I_4x4, I_BL, P_Skip, or B_Skip, residual_prediction_flag is equal to 1, CodedBlockPatternLuma is equal to 0, and CodedBlockPatternChroma is equal to 0.

– Otherwise, tQP_Y is set equal to QP_Y.

When ChromaArrayType is not equal to 0, for CX being replaced by Cb and Cr, the variable tQP_{CX} is set equal to the value of QP_{CX} that corresponds to a value of tQP_Y for QP_Y as specified in subclause 8.5.8.

The variable predTrafoFlag is derived as follows:

– If SpatialResolutionChangeFlag is equal to 0 and any of the following conditions are true, predTrafoFlag is set equal to 1:

  – base_mode_flag is equal to 1, tcoeff_level_prediction_flag is equal to 0, refLayerMbType[CurrMbAddr] is equal to I_BL, and CodedBlockPatternLuma is equal to 0,

  – base_mode_flag is equal to 1, tcoeff_level_prediction_flag is equal to 0, refLayerMbType[CurrMbAddr] is equal to I_PCM, CodedBlockPatternLuma is equal to 0, and CodedBlockPatternChroma is equal to 0,

  - base_mode_flag is equal to 1, tcoeff_level_prediction_flag is equal to 0, refLayerMbType[CurrMbAddr] is equal to I_8x8 or I_4x4, and CodedBlockPatternLuma is equal to 0,
  - base_mode_flag is equal to 1, tcoeff_level_prediction_flag is equal to 1, and mbType is equal to I_PCM, I_16x16, I_8x8, or I_4x4,
  - residual_prediction_flag is equal to 1, refLayerMbType[CurrMbAddr] is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL, mbType is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL, and CodedBlockPatternLuma is equal to 0.
- Otherwise, predTrafoFlag is set equal to 0.

The variable cTrafo is derived as follows:

- If mbType is equal to I_PCM, cTrafo is set equal to T_PCM.
- Otherwise, if mbType is equal to I_16x16, cTrafo is set equal to T_16x16.
- Otherwise, if mbType is equal to I_8x8 or transform_size_8x8_flag is equal to 1, cTrafo is set equal to T_8x8.
- Otherwise, if predTrafoFlag is equal to 1, cTrafo is set equal to refLayerCTrafo[CurrMbAddr].
- Otherwise (predTrafoFlag is equal to 0, transform_size_8x8_flag is equal to 0, and mbType is not equal to I_PCM, I_16x16, or I_8x8), cTrafo is set equal to T_4x4.

When cTrafo is equal to T_8x8, the SVC sequence parameter set that is referred to by the coded slice NAL unit (via pic_parameter_set_id in the slice header and seq_parameter_set_id in referenced the picture parameter set) shall have transform_8x8_mode_flag equal to 1.

When base_mode_flag is equal to 1, the following constraints shall be obeyed:

a) When mbType is equal to P_8x8, P_8x8ref0, or B8x8 and NumSubMbPart(subMbType[mbPartIdx]) is not equal to 1 for any mbPartIdx = 0..3, the bitstream shall not contain transform_size_8x8_flag equal to 1.

b) When mbType is equal to I_PCM, the bitstream shall not contain data that result in CodedBlockPatternLuma not equal to 0 or CodedBlockPatternChroma not equal to 0.

c) When mbType is equal to I_16x16 or I_4x4, the bitstream shall not contain transform_size_8x8_flag equal to 1.

d) When mbType is equal to I_8x8 and transform_size_8x8_flag is equal to 0, the bitstream shall not contain data that result in CodedBlockPatternLuma not equal to 0.

The variable constrainedCoeffFlag is derived as follows:

- If SpatialResolutionChangeFlag is equal to 0 and any of the following conditions are true, constrainedCoeffFlag is set equal to 1:
  - base_mode_flag is equal to 1, tcoeff_level_prediction_flag is equal to 0, and refLayerMbType[CurrMbAddr] is equal to I_BL,
  - residual_prediction_flag is equal to 1, refLayerMbType[CurrMbAddr] is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL, and mbType is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL.
- Otherwise, constrainedCoeffFlag is set equal to 0.

When constrainedCoeffFlag is equal to 1, the following constraints shall be obeyed:

a) When refLayerCTrafo[CurrMbAddr] is equal to T_8x8 and transform_size_8x8_flag is equal 0, any of the following constraints shall be obeyed:

   i) The bitstream shall not contain data that result in CodedBlockPatternLuma not equal to 0.

   ii) Depending on tcoeff_level_prediction_flag, the following applies:

      - If tcoeff_level_prediction_flag is equal to 0, the bitstream shall not contain data that result in any element refLayerSTCoeff[CurrMbAddr][i] not equal to 0 for
        i = 0..((ChromaArrayType != 3) ? 255 : 767).
      - Otherwise (tcoeff_level_prediction_flag is equal to 1), the bitstream shall not contain data that result in any element refLayerTCoeffLevel[CurrMbAddr][i] not equal to 0 for
        i = 0..((ChromaArrayType != 3) ? 255 : 767).

b) When refLayerCTrafo[CurrMbAddr] is equal to T_4x4 and transform_size_8x8_flag equal to 1, the following applies:

   – If tcoeff_level_prediction_flag is equal to 0, the bitstream shall not contain data that result in any element refLayerSTCoeff[CurrMbAddr][i] not equal to 0 for
   i = 0..((ChromaArrayType != 3) ? 255 : 767).

   – Otherwise (tcoeff_level_prediction_flag is equal to 1), the bitstream shall not contain data that result in any element refLayerTCoeffLevel[CurrMbAddr][i] not equal to 0 for
   i = 0..((ChromaArrayType != 3) ? 255 : 767).

G.8.1.5.2 Base decoding process for macroblocks in slices without resolution change

Inputs to this process are:

– a variable currDQId specifying the current layer representation,

– a variable mbAddr specifying the current macroblock inside the current layer representation,

– a set of arrays collectively referred to as currentVars,

– when currDQId is equal to 0 and (slice_type % 5) is equal to 1, the reference picture list refPicList1.

Output of this process is the modified set of arrays collectively referred to as currentVars.

This process modifies the variables assigned to currentVars using syntax elements and derived upper-case variables for the current macroblock, which is specified by its address mbAddr and the layer representation identifier currDQId.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the current macroblock, which is the macroblock with address mbAddr inside the layer representation with DQId equal to currDQId, the slice header of the current slice, which is the slice that contains the current macroblock, the current picture parameter, which is identified by the syntax element pic_parameter_set_id inside the slice header of the current slice, and the current sequence parameter, which is identified by the syntax element seq_parameter_set_id inside the current picture parameter set.

Inside this subclause, the arrays that are collectively referred to as currentVars are referred to by their names as specified in subclause G.8.1.2.1.

The base decoding process for macroblocks in slices without resolution change is specified by the following ordered steps:

1. The variable CurrMbAddr is set equal to mbAddr.

2. When tcoeff_level_prediction_flag is equal to 1, the variable $refQP_Y$ is set equal to tQP_Y[mbAddr] and, when ChromaArrayType is not equal to 0, the variables $refQP_{Cb}$ and $refQP_{Cr}$ are set equal to tQP_{Cb}[mbAddr] and tQP_{Cr}[mbAddr], respectively.

3. When no_inter_layer_pred_flag is equal to 0, the variable refLayerIntraBLFlag is derived as follows:

   – If mbType[mbAddr] is equal to I_BL, refLayerIntraBLFlag is set equal to 1.

   – Otherwise (mbType[mbAddr] is not equal to I_BL), refLayerIntraBLFlag is set equal to 0.

4. The variable resPredFlag is derived as follows:

   – If residual_prediction_flag is equal to 1 and mbType[mbAddr] is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL, resPredFlag is set equal to 1.

   – Otherwise (residual_prediction_flag is equal to 0 or mbType[mbAddr] is equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL), resPredFlag is set equal to 0.

5. The macroblock initialisation process as specified in subclause G.8.1.5.1 is invoked with refLayerVars set equal to currentVars as the input and the outputs are assigned to sliceIdc[mbAddr], fieldMbFlag[mbAddr], cTrafo[mbAddr], baseModeFlag[mbAddr], mbType[mbAddr], subMbType[mbAddr], mvCnt[mbAddr], tQP_Y[mbAddr], tQP_{Cb}[mbAddr] (when ChromaArrayType is not equal to 0), tQP_{Cr}[mbAddr] (when ChromaArrayType is not equal to 0), the 2x2 arrays refIdxILPredL0 and refIdxILPredL1, and the 4x4x2 arrays mvILPredL0 and mvILPredL1.

6. The SVC derivation process for motion vector components and reference indices as specified in subclause G.8.4.1 is invoked with sliceIdc, fieldMbFlag, mbType, subMbType, predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, mvCnt, refIdxILPredL0, refIdxILPredL1, mvILPredL0, mvILPredL1, and

refPicList1 (when available) as the inputs and the outputs are modified versions of the arrays predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, and mvCnt.

7. Depending on mbType[mbAddr], the following applies:

   – If mbType[mbAddr] is equal to I_PCM, I_16x16, I_8x8, or I_4x4, the following ordered steps are specified:

      a. When base_mode_flag is equal to 0, the SVC derivation process for intra prediction modes as specified in subclause G.8.3.1 is invoked with sliceIdc, fieldMbFlag, baseModeFlag, mbType, ipred4x4, ipred8x8, ipred16x16, and, when ChromaArrayType is equal to 1 or 2, ipredChroma as the inputs and the outputs are modified versions of ipred4x4, ipred8x8, ipred16x16, and, when ChromaArrayType is equal to 1 or 2, ipredChroma.

      b. When tcoeff_level_prediction_flag is equal to 1 and base_mode_flag is equal to 1, the transform coefficient level scaling process prior to transform coefficient refinement as specified in subclause G.8.5.2 is invoked with cTrafo[mbAddr], tCoeffLevel[mbAddr], tQP_Y[mbAddr], $refQP_Y$, and, when ChromaArrayType is not equal to 0, tQP_{Cb}[mbAddr], tQP_{Cr}[mbAddr], $refQP_{Cb}$, and $refQP_{Cr}$ as the inputs and the output is a modified version of tCoeffLevel[mbAddr].

      c. The transform coefficient scaling and refinement process as specified in subclause G.8.5.1 is invoked with refinementFlag set equal to base_mode_flag, fieldMbFlag[mbAddr], mbType[mbAddr], cTrafo[mbAddr], sTCoeff[mbAddr], tCoeffLevel[mbAddr], tQP_Y[mbAddr], and, when ChromaArrayType is not equal to 0, tQP_{Cb}[mbAddr] and tQP_{Cr}[mbAddr] as the inputs and the outputs are modified versions of sTCoeff[mbAddr] and tCoeffLevel[mbAddr].

      d. The sample array re-initialisation process as specified in subclause G.8.5.5 is invoked with fieldMbFlag[mbAddr], rS_L, and, when ChromaArrayType is not equal to 0, rS_{Cb} and rS_{Cr} as the inputs and the outputs are a modified version of rS_L and, when ChromaArrayType is not equal to 0, modified versions of rS_{Cb} and rS_{Cr}.

      e. The sample array re-initialisation process as specified in subclause G.8.5.5 is invoked with fieldMbFlag[mbAddr], cS_L, and, when ChromaArrayType is not equal to 0, cS_{Cb} and cS_{Cr} as the inputs and the outputs are a modified version of cS_L and, when ChromaArrayType is not equal to 0, modified versions of cS_{Cb} and cS_{Cr}.

   – Otherwise, if mbType[mbAddr] is equal to I_BL, the transform coefficient scaling and refinement process as specified in subclause G.8.5.1 is invoked with refinementFlag set equal to refLayerIntraBLFlag, fieldMbFlag[mbAddr], mbType[mbAddr], cTrafo[mbAddr], sTCoeff[mbAddr], tCoeffLevel[mbAddr], tQP_Y[mbAddr], and, when ChromaArrayType is not equal to 0, tQP_{Cb}[mbAddr] and tQP_{Cr}[mbAddr] as the inputs and the outputs are modified versions of sTCoeff[mbAddr] and tCoeffLevel[mbAddr].

   – Otherwise (mbType[mbAddr] is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL), the following ordered steps are specified:

      a. When tcoeff_level_prediction_flag is equal to 1 and resPredFlag is equal to 1, the transform coefficient level scaling process prior to transform coefficient refinement as specified in subclause G.8.5.2 is invoked with cTrafo[mbAddr], tCoeffLevel[mbAddr], tQP_Y[mbAddr], $refQP_Y$, and, when ChromaArrayType is not equal to 0, tQP_{Cb}[mbAddr], tQP_{Cr}[mbAddr], $refQP_{Cb}$, and $refQP_{Cr}$ as the inputs and the output is a modified version of tCoeffLevel[mbAddr].

      b. The transform coefficient scaling and refinement process as specified in subclause G.8.5.1 is invoked with refinementFlag set equal to resPredFlag, fieldMbFlag[mbAddr], mbType[mbAddr], cTrafo[mbAddr], sTCoeff[mbAddr], tCoeffLevel[mbAddr], tQP_Y[mbAddr], and, when ChromaArrayType is not equal to 0, tQP_{Cb}[mbAddr] and tQP_{Cr}[mbAddr] as the inputs and the outputs are modified versions of sTCoeff[mbAddr] and tCoeffLevel[mbAddr].

      c. When resPredFlag is equal to 0, the sample array re-initialisation process as specified in subclause G.8.5.5 is invoked with fieldMbFlag[mbAddr], rS_L, and, when ChromaArrayType is not equal to 0, rS_{Cb} and rS_{Cr} as the inputs and the outputs are a modified versions of rS_L and, when ChromaArrayType is not equal to 0, modified versions of rS_{Cb} and rS_{Cr}.

      d. When resPredFlag is equal to 0, the sample array re-initialisation process as specified in subclause G.8.5.5 is invoked with fieldMbFlag[mbAddr], cS_L, and, when ChromaArrayType is not equal to 0, cS_{Cb} and cS_{Cr} as the inputs and the outputs are a modified versions of cS_L and, when ChromaArrayType is not equal to 0, modified versions of cS_{Cb} and cS_{Cr}.

8. The variable MvCnt for the macroblock mbAddr is set equal to mvCnt[mbAddr].

G.8.1.5.3 Base decoding process for macroblocks in slices with resolution change

Inputs to this process are:

– a variable currDQId specifying the current layer representation,

– a variable mbAddr specifying the current macroblock inside the current layer representation,

– a set of arrays collectively referred to as refLayerVars,

– a set of arrays collectively referred to as currentVars,

– when CroppingChangeFlag is equal to 1 and (slice_type % 5) is less than 2, the reference picture list refPicList0,

– when CroppingChangeFlag is equal to 1 and (slice_type % 5) is equal to 1, the reference picture list refPicList1.

Output of this process is the modified set of arrays collectively referred to as currentVars.

This process modifies the variables assigned to currentVars using syntax elements and derived upper-case variables for the current macroblock, which is specified by its address mbAddr and the layer representation identifier currDQId, as well as variables assigned to refLayerVars.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the current macroblock, which is the macroblock with address mbAddr inside the layer representation with DQId equal to currDQId, the slice header of the current slice, which is the slice that contains the current macroblock, the current picture parameter, which is identified by the syntax element pic_parameter_set_id inside the slice header of the current slice, and the current sequence parameter, which is identified by the syntax element seq_parameter_set_id inside the current picture parameter set.

Inside this subclause, the arrays that are collectively referred to as currentVars are referred to by their names as specified in subclause G.8.1.2.1.

Inside this subclause, the arrays sliceIdc, fieldMbFlag, cTrafo, mbType, cS_L, cS_{Cb}, cS_{Cr}, rS_L, rS_{Cb}, and rS_{Cr} of the collective term refLayerVars are referred to as refLayerSliceIdc, refLayerFieldMbFlag, refLayerCTrafo, refLayerMbType, $refS_L$, $refS_{Cb}$, $refS_{Cr}$, $refR_L$, $refR_{Cb}$, and $refR_{Cr}$, respectively.

The base decoding process for macroblocks in slices with resolution change is specified by the following ordered steps:

1. The variable CurrMbAddr is set equal to mbAddr.

2. The macroblock initialisation process as specified in subclause G.8.1.5.1 is invoked with refLayerVars, refPicList0 (when available), and refPicList1 (when available) as the inputs and the outputs are assigned to sliceIdc[mbAddr], fieldMbFlag[mbAddr], cTrafo[mbAddr], baseModeFlag[mbAddr], mbType[mbAddr], subMbType[mbAddr], mvCnt[mbAddr], tQP_Y[mbAddr], tQP_{Cb}[mbAddr] (when ChromaArrayType is not equal to 0), tQP_{Cr}[mbAddr] (when ChromaArrayType is not equal to 0), the 2x2 arrays refIdxILPredL0 and refIdxILPredL1, and the 4x4x2 arrays mvILPredL0 and mvILPredL1.

3. The SVC derivation process for motion vector components and reference indices as specified in subclause G.8.4.1 is invoked with sliceIdc, fieldMbFlag, mbType, subMbType, predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, mvCnt, refIdxILPredL0, refIdxILPredL1, mvILPredL0, and mvILPredL1 as the inputs and the outputs are modified versions of the arrays predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, and mvCnt.

4. The variable intraResamplingFlag is derived as follows:

   – If any of the following conditions are true, intraResamplingFlag is set equal to 1:

     – mbType[mbAddr] is equal to I_BL,

     – RestrictedSpatialResolutionChangeFlag is equal to 0, MbaffFrameFlag is equal to 0, RefLayerMbaffFrameFlag is equal to 0, and base_mode_flag is equal to 1.

   – Otherwise, intraResamplingFlag is set equal to 0.

5. When intraResamplingFlag is equal to 1, the resampling process for intra samples as specified in subclause G.8.6.2 is invoked with fieldMbFlag[mbAddr], refLayerSliceIdc, refLayerFieldMbFlag, refLayerMbType, $refS_L$, cS_L, and, when ChromaArrayType is not equal to 0, $refS_{Cb}$, $refS_{Cr}$, cS_{Cb}, and cS_{Cr} as the inputs and the outputs are a modified version of the array cS_L and, when ChromaArrayType is not equal to 0, modified versions of the array cS_{Cb}, and cS_{Cr}.

6. Depending on mbType[mbAddr], the following applies:

– If mbType[mbAddr] is equal to I_PCM, I_16x16, I_8x8, or I_4x4, the SVC derivation process for intra prediction modes as specified in subclause G.8.3.1 is invoked with sliceIdc, fieldMbFlag, baseModeFlag, mbType, ipred4x4, ipred8x8, ipred16x16, and, when ChromaArrayType is equal to 1 or 2, ipredChroma as the inputs and the outputs are modified versions of ipred4x4, ipred8x8, ipred16x16, and, when ChromaArrayType is equal to 1 or 2, ipredChroma.

– Otherwise, if mbType[mbAddr] is not equal to I_BL and residual_prediction_flag is equal to 1, the resampling process for residual samples as specified in subclause G.8.6.3 is invoked with fieldMbFlag[mbAddr], refLayerFieldMbFlag, refLayerCTrafo, $refR_L$, rS_L, and, when ChromaArrayType is not equal to 0, $refR_{Cb}$, $refR_{Cr}$, rS_{Cb}, and rS_{Cr} as the inputs and the outputs are a modified version of the array rS_L and, when ChromaArrayType is not equal to 0, modified versions of the array rS_{Cb}, and rS_{Cr}.

– Otherwise, the arrays of the collective term currentVars are not modified.

7. The transform coefficient scaling and refinement process as specified in subclause G.8.5.1 is invoked with refinementFlag set equal to 0, fieldMbFlag[mbAddr], mbType[mbAddr], cTrafo[mbAddr], sTCoeff[mbAddr], tCoeffLevel[mbAddr], tQP_Y[mbAddr], and, when ChromaArrayType is not equal to 0, tQP_{Cb}[mbAddr] and tQP_{Cr}[mbAddr] as the inputs and the outputs are modified versions of sTCoeff[mbAddr] and tCoeffLevel[mbAddr].

8. The variable MvCnt for the macroblock mbAddr is set equal to mvCnt[mbAddr].

G.8.1.5.4 Macroblock decoding process prior to decoding a layer representation without resolution change and MaxTCoeffLevelPredFlag equal to 0

Inputs to this process are:

– a variable currDQId specifying the current layer representation,

– a variable mbAddr specifying the current macroblock inside the current layer representation,

– a set of arrays collectively referred to as currentVars.

Output of this process is the modified set of arrays collectively referred to as currentVars.

This process modifies the variables assigned to currentVars using syntax elements and derived upper-case variables for the current macroblock, which is specified by its address mbAddr and the layer representation identifier currDQId.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the current layer representation with DQId equal to currDQId.

Inside this subclause, the arrays that are collectively referred to as currentVars are referred to by their names as specified in subclause G.8.1.2.1.

The macroblock decoding process prior to decoding a layer representation without resolution change and MaxTCoeffLevelPredFlag equal to 0 is specified by the following ordered steps:

1. The variable CurrMbAddr is set equal to mbAddr.

2. The variable intraPredFlag is derived as follows:

   – If (sliceIdc[mbAddr] & 127) is equal to currDQId or MaxTCoeffLevelPredFlag is equal to 1, intraPredFlag is set equal to 1.

   – Otherwise ((sliceIdc[mbAddr] & 127) is not equal to currDQId and MaxTCoeffLevelPredFlag is equal to 0), intraPredFlag is set equal to 0.

3. When intraPredFlag is equal to 1 and mbType[mbAddr] is equal to I_PCM, I_16x16, I_8x8, or I_4x4, the SVC intra sample prediction and construction process as specified in subclause G.8.3.2 is invoked with sliceIdc, fieldMbFlag, baseModeFlag, mbType, ipred4x4[mbAddr], ipred8x8[mbAddr], ipred16x16[mbAddr], ipredChroma[mbAddr], cTrafo[mbAddr], sTCoeff[mbAddr], cS_L, and, when ChromaArrayType is not equal to 0, cS_{Cb} and cS_{Cr} as the inputs and the outputs are a modified version of the array cS_L and, when ChromaArrayType is not equal to 0, modified versions of the arrays cS_{Cb} and cS_{Cr}.

G.8.1.5.5 Macroblock decoding process prior to resolution change

Inputs to this process are:

– a variable currDQId specifying the current layer representation,

– a variable mbAddr specifying the current macroblock inside the current layer representation,

– a set of arrays collectively referred to as currentVars.

Output of this process is the modified set of arrays collectively referred to as currentVars.

This process modifies the variables assigned to currentVars using syntax elements and derived upper-case variables for the current macroblock, which is specified by its address mbAddr and the layer representation identifier currDQId.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the current macroblock, which is the macroblock with address mbAddr inside the layer representation with DQId equal to currDQId, the slice header of the current slice, which is the slice that contains the current macroblock, the current picture parameter, which is identified by the syntax element pic_parameter_set_id inside the slice header of the current slice, and the current sequence parameter, which is identified by the syntax element seq_parameter_set_id inside the current picture parameter set.

Inside this subclause, the arrays that are collectively referred to as currentVars are referred to by their names as specified in subclause G.8.1.2.1.

The macroblock decoding process prior to resolution change is specified by the following ordered steps:

1. The variable CurrMbAddr is set equal to mbAddr.
2. Depending on mbType[mbAddr], the following applies:
   - If mbType[mbAddr] is equal to I_PCM, I_16x16, I_8x8, or I_4x4, the SVC intra sample prediction and construction process as specified in subclause G.8.3.2 in invoked with sliceIdc, fieldMbFlag, baseModeFlag, mbType, ipred4x4[mbAddr], ipred8x8[mbAddr], ipred16x16[mbAddr], ipredChroma[mbAddr], cTrafo[mbAddr], sTCoeff[mbAddr], cS_L, and, when ChromaArrayType is not equal to 0, cS_{Cb} and cS_{Cr} as the inputs and the outputs are a modified version of the array cS_L and, when ChromaArrayType is not equal to 0, modified versions of the arrays cS_{Cb} and cS_{Cr}.
   - Otherwise, if mbType[mbAddr] is equal to I_BL, the following ordered steps are specified:
     a. The residual construction and accumulation process as specified in subclause G.8.5.3 is invoked with accumulationFlag set equal to 0, fieldMbFlag[mbAddr], cTrafo[mbAddr], sTCoeff[mbAddr], rS_L, and, when ChromaArrayType is not equal to 0, rS_{Cb} and rS_{Cr} as the inputs and the outputs are a modified version of rS_L and, when ChromaArrayType is not equal to 0, modified versions of rS_{Cb} and rS_{Cr}.
     b. The sample array accumulation process as specified in subclause G.8.5.4 is invoked with fieldMbFlag[mbAddr], rS_L, cS_L, and, when ChromaArrayType is not equal to 0, rS_{Cb}, rS_{Cr} cS_{Cb}, and cS_{Cr} as the inputs and the outputs are a modified version of cS_L and, when ChromaArrayType is not equal to 0, modified versions of cS_{Cb} and cS_{Cr}.
     c. The sample array re-initialisation process as specified in subclause G.8.5.5 is invoked with fieldMbFlag[mbAddr], rS_L, and, when ChromaArrayType is not equal to 0, rS_{Cb} and rS_{Cr} as the inputs and the outputs are a modified version of rS_L and, when ChromaArrayType is not equal to 0, modified versions of rS_{Cb} and rS_{Cr}.
   - Otherwise (mbType[mbAddr] is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL), the following ordered steps are specified:
     a. The residual construction and accumulation process as specified in subclause G.8.5.3 is invoked with accumulationFlag set equal to 1, fieldMbFlag[mbAddr], cTrafo[mbAddr], sTCoeff[mbAddr], rS_L, and, when ChromaArrayType is not equal to 0, rS_{Cb} and rS_{Cr} as the inputs and the outputs are a modified version of rS_L and, when ChromaArrayType is not equal to 0, modified versions of rS_{Cb} and rS_{Cr}.
     b. The sample array re-initialisation process as specified in subclause G.8.5.5 is invoked with fieldMbFlag[mbAddr], cS_L, and, when ChromaArrayType is not equal to 0, cS_{Cb} and cS_{Cr} as the inputs and the outputs are a modified version of cS_L and, when ChromaArrayType is not equal to 0, modified versions of cS_{Cb} and cS_{Cr}.

G.8.1.5.6 Target macroblock decoding process

Inputs to this process are:

– a variable currDQId specifying the current layer representation,

– a variable mbAddr specifying the current macroblock inside the current layer representation,

– when present, a set of arrays collectively referred to as refLayerVars,

– a set of arrays collectively referred to as currentVars,

– when (slice_type % 5) is less than 2, the reference picture list refPicList0,

– when (slice_type % 5) is equal to 1, the reference picture list refPicList1.

Output of this process is the modified set of arrays collectively referred to as currentVars.

This process modifies the variables assigned to currentVars using syntax elements and derived upper-case variables for the current macroblock, which is specified by its address mbAddr and the layer representation identifier currDQId.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the current layer representation with DQId equal to currDQId.

Inside this subclause, the arrays that are collectively referred to as currentVars are referred to by their names as specified in subclause G.8.1.2.1.

Inside this subclause, the following applies:

– If refLayerVars is present as input to this subclause, the arrays fieldMbFlag and mbType of the collective term refLayerVars are referred to as refLayerFieldMbFlag and refLayerMbType, respectively.

– Otherwise (refLayerVars are not present as input to this subclause), the variables refLayerFieldMbFlag and refLayerMbType are marked as not available.

The target macroblock decoding process is specified by the following ordered steps:

1. The variable CurrMbAddr is set equal to mbAddr.

2. When MaxTCoeffLevelPredFlag is equal to 1, (sliceIdc[mbAddr] &127) is not equal to currDQId, and ChromaArrayType is not equal to 0, the following ordered steps are specified:

   a. The variable cQP_Y is set equal to tQP_Y[mbAddr], and for CX being replaced by Cb and Cr, the variable cQP_{CX} is set equal to the value of QP_{CX} that corresponds to a value of cQP_Y for QP_Y as specified in subclause 8.5.8.

   b. The transform coefficient level scaling process prior to transform coefficient refinement as specified in subclause G.8.5.2 is invoked with cTrafo[mbAddr], tCoeffLevel[mbAddr], tQP_Y set equal to cQP_Y, $refQP_Y$ set equal to tQP_Y[mbAddr], and, when ChromaArrayType is not equal to 0, tQP_{Cb} set equal to cQP_{Cb}, tQP_{Cr} set equal to cQP_{Cr}, $refQP_{Cb}$ set equal to tQP_{Cb}[mbAddr], $refQP_{Cr}$ set equal to tQP_{Cr}[mbAddr] as the inputs and the output is a modified version of tCoeffLevel[mbAddr].

   c. The variables tQP_{Cb}[mbAddr] and tQP_{Cr}[mbAddr] are set equal to cQP_{Cb} and cQP_{Cr}, respectively.

   d. The transform coefficient scaling and refinement process as specified in subclause G.8.5.1 is invoked with refinementFlag equal to 1, fieldMbFlag[mbAddr], mbType[mbAddr], cTrafo[mbAddr], sTCoeff[mbAddr], tCoeffLevel[mbAddr], tQP_Y[mbAddr], tQP_{Cb}[mbAddr], and tQP_{Cr}[mbAddr] as the inputs and the outputs are modified versions of sTCoeff[mbAddr] and tCoeffLevel[mbAddr]. For this invocation of the process in subclause G.8.5.1, all elements of the arrays LumaLevel4x4, LumaLevel8x8, Intra16x16DCLevel, Intra16x16ACLevel, CbLevel4x4, CbLevel8x8, CbIntra16x16DCLevel, CbIntra16x16ACLevel, CrLevel4x4, CrLevel8x8, CrIntra16x16DCLevel, CrIntra16x16ACLevel, ChromaDCLevel, and ChromaACLevel are inferred to be equal to 0, tcoeff_level_prediction_flag is inferred to be equal to 1, and base_mode_flag is inferred to be equal to 1.

   NOTE – By the ordered steps specified above, the elements of the arrays tCoeffLevel[mbAddr] and sTCoeff[mbAddr] that are related to luma transform coefficients are not modified. The array elements that are related to chroma transform coefficients are only modified when the chroma quantisation parameter offsets of the current layer representation with DQId equal to currDQId and the layer representation with DQId equal to (sliceIdc[mbAddr] &127) are different.

3. Depending on mbType[mbAddr], the following applies:

   – If mbType[mbAddr] is equal to I_PCM, I_16x16, I_8x8, or I_4x4, the following ordered steps are specified:

     a. The variable intraPredFlag is derived as follows:

        – If (sliceIdc[mbAddr] & 127) is equal to currDQId or MaxTCoeffLevelPredFlag is equal to 1, intraPredFlag is set equal to 1.

– Otherwise ((sliceIdc[mbAddr] & 127) is not equal to currDQId and MaxTCoeffLevelPredFlag is equal to 0), intraPredFlag is set equal to 0.

b. When intraPredFlag is equal to 1, the SVC intra sample prediction and construction process as specified in subclause G.8.3.2 is invoked with sliceIdc, fieldMbFlag, baseModeFlag, mbType, ipred4x4[mbAddr], ipred8x8[mbAddr], ipred16x16[mbAddr], ipredChroma[mbAddr], cTrafo[mbAddr], sTCoeff[mbAddr], cS_L, and, when ChromaArrayType is not equal to 0, cS_{Cb} and cS_{Cr} as the inputs and the outputs are a modified version of the array cS_L and, when ChromaArrayType is not equal to 0, modified versions of the arrays cS_{Cb} and cS_{Cr}.

– Otherwise, if mbType[mbAddr] is equal to I_BL, the following ordered steps are specified:

a. The residual construction and accumulation process as specified in subclause G.8.5.3 is invoked with accumulationFlag set equal to 0, fieldMbFlag[mbAddr], cTrafo[mbAddr], sTCoeff[mbAddr], rS_L, and, when ChromaArrayType is not equal to 0, rS_{Cb} and rS_{Cr} as the inputs and the outputs are a modified version of rS_L and, when ChromaArrayType is not equal to 0, modified versions of rS_{Cb} and rS_{Cr}.

b. The sample array accumulation process as specified in subclause G.8.5.4 is invoked with fieldMbFlag[mbAddr], rS_L, cS_L, and, when ChromaArrayType is not equal to 0, rS_{Cb}, rS_{Cr} cS_{Cb}, and cS_{Cr} as the inputs and the outputs are a modified version of cS_L and, when ChromaArrayType is not equal to 0, modified versions of cS_{Cb} and cS_{Cr}.

– Otherwise (mbType[mbAddr] is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL), the following ordered steps are specified:

a. The SVC decoding process for Inter prediction samples as specified in subclause G.8.4.2 is invoked with targetQId set equal to (currDQId & 15), fieldMbFlag[mbAddr], sliceIdc[mbAddr], mbType[mbAddr], subMbType[mbAddr], predFlagL0[mbAddr], predFlagL1[mbAddr], refIdxL0[mbAddr], refIdxL1[mbAddr], mvL0[mbAddr], mvL1[mbAddr], refLayerFieldMbFlag (when available), refLayerMbType (when available), refPicList0, refPicList1 (when available), cS_L, rS_L, and, when ChromaArrayType is not equal to 0, cS_{Cb}, cS_{Cr}, rS_{Cb}, and rS_{Cr} as the inputs and the outputs are modified version of cS_L and rS_L, and, when ChromaArrayType is not equal to 0, modified versions of cS_{Cb}, cS_{Cr}, rS_{Cb}, and rS_{Cr}.

b. The residual construction and accumulation process as specified in subclause G.8.5.3 is invoked with accumulationFlag set equal to 1, fieldMbFlag[mbAddr], cTrafo[mbAddr], sTCoeff[mbAddr], rS_L, and, when ChromaArrayType is not equal to 0, rS_{Cb} and rS_{Cr} as the inputs and the outputs are a modified version of rS_L and, when ChromaArrayType is not equal to 0, modified versions of rS_{Cb} and rS_{Cr}.

c. The sample array accumulation process as specified in subclause G.8.5.4 is invoked with fieldMbFlag[mbAddr], rS_L, cS_L, and, when ChromaArrayType is not equal to 0, rS_{Cb}, rS_{Cr} cS_{Cb}, and cS_{Cr} as the inputs and the outputs are a modified version of cS_L and, when ChromaArrayType is not equal to 0, modified versions of cS_{Cb} and cS_{Cr}.

G.8.2 SVC reference picture lists construction and decoded reference picture marking process

The SVC decoding process for picture order count is specified in subclause G.8.2.1.

The SVC decoding process for picture numbers is specified in subclause G.8.2.2.

The SVC decoding process for reference picture lists construction is specified in subclause G.8.2.3.

The SVC decoded reference picture marking process is specified in subclause G.8.2.4.

The SVC decoding process for gaps in frame_num is specified in subclause G.8.2.5.

The decoding process for picture order counts is independently applied for different values of dependency_id. Syntax elements that are related to picture order count for a particular value of dependency_id do not influence the derivation of picture order counts for other values of dependency_id.

The reference picture marking is independently applied for different values of dependency_id. Syntax elements that are related to reference picture marking for a particular value of dependency_id do not influence the reference picture marking for other values of dependency_id.

The decoding process for gaps is frame_num is independently applied for different values of dependency_id.

Reference picture lists for different dependency representations are constructed independently. Syntax elements that are related to reference picture lists construction for a particular value of dependency_id do not influence the reference picture lists construction for other values of dependency_id. Reference picture lists for a particular value of dependency_id are constructed based on the reference picture marking for this particular value of dependency_id. The reference picture marking for a particular value of dependency_id does not influence the reference picture lists construction for a different value of dependency_id.

Only the elements of the reference picture lists for dependency_id equal to DependencyIdMax represent decoded pictures that are associated with decoded samples. Only the reference picture lists for dependency_id equal to DependencyIdMax are used for the derivation of inter prediction samples as specified in subclause G.8.4.2. The elements of the reference picture lists for dependency representation with dependency_id less than DependencyIdMax represent layer pictures, which are not associated with decoded samples. The elements of the reference picture lists for dependency_id equal to 0 are associated with the arrays fieldMbFlag, mbType, subMbType, predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, and mvL1 as specified in subclause G.8.1.2.1 that were derived when decoding the layer representation with dependency_id equal to 0 and quality_id equal to 0 for the corresponding access unit. These arrays are used for the derivation of motion vectors and reference indices for layer representation with dependency_id equal to 0 and quality_id equal to 0 as specified in subclause G.8.4.1.2. The elements of the reference picture lists for all dependency representations with dependency_id greater than 0 are associated with the variables ScaledRefLayerLeftOffset, ScaledRefLayerRightOffset, ScaledRefLayerTopOffset, and ScaledRefLayerBottomOffset. These variables are used for deriving inter-layer motion vector predictions as specified in subclause G.8.6.1.2.

NOTE – For each access unit, decoded samples only need to be stored for the dependency representation with dependency_id equal to DependencyIdMax and motion data arrays only need to be stored for the dependency representation with dependency_id equal to 0.

The SVC decoding processes for picture order count, reference picture lists construction, reference picture marking, and gaps in frame_num are specified using processes specified in clause 8. The following modifications to the processes specified in this subclause and the processes of clause 8 that are invoked from these processes apply with currDependencyId representing the value of dependency_id for the dependency representation for which the processes are invoked:

a) All syntax elements and derived upper-case variables that are referred to in this process or in a child process invoked from this process are syntax elements and upper-case variables for the dependency representation with dependency_id equal to currDependencyId.

b) When dependency_id is less than DependencyIdMax, the following applies:

   – A frame, field, top field, bottom field, picture, and decoded picture is interpreted as layer frame, layer field, layer top field, layer bottom field, layer picture, and decoded layer picture, respectively, for the dependency representation with dependency_id equal to currDependencyId. A decoded layer picture is not associated with the sample arrays S_L, S_{Cb}, or S_{Cr}.

   – An IDR picture is interpreted as layer picture with IdrPicFlag equal to 1 for the dependency representation with dependency_id equal to currDependencyId.

   – A reference frame, reference field, and reference picture is interpreted as layer frame, layer field, and layer picture with nal_ref_idc greater than 0 for the dependency representation with dependency_id equal to currDependencyId.

   – A non-reference frame, non-reference field, and non-reference picture is interpreted as layer frame, layer field, and layer picture with nal_ref_idc equal to 0 for the dependency representation with dependency_id equal to currDependencyId.

   – A complementary non-reference field pair is interpreted as complementary non-reference layer field pair for the dependency representation with dependency_id equal to currDependencyId. A complementary non-reference layer field pair for a particular value of dependency_id is a pair of two layer fields for the particular value of dependency_id with the following properties: (i) the layer fields are in consecutive access units containing a dependency representation with the particular value of dependency_id, (ii) the dependency representations with the particular value of dependency_id in these access units have nal_ref_idc equal to 0, field_pic_flag equal to 1, different values of bottom_field_flag, and they share the same value of the frame_num syntax element, (iii) the first layer field is not already a paired layer field.

   – A complementary reference field pair is interpreted as complementary reference layer field pair for the dependency representation with dependency_id equal to currDependencyId. A complementary reference layer field pair for a particular value of dependency_id is a pair of two layer fields for the particular value of dependency_id with the following properties: (i) the layer fields are in consecutive access units containing a dependency representation with the particular value of dependency_id, (ii) the dependency representations with the particular value of dependency_id in these access units have nal_ref_idc greater than 0, field_pic_flag equal to 1, and the same value of frame_num, (iii) the dependency representation with the particular value of dependency_id of the second access unit in decoding order has IdrPicFlag equal to 0 and does not contain a memory_management_control_operation syntax element equal to 5.

– A complementary field pair is interpreted as complementary layer field pair for the dependency representation with dependency_id equal to currDependencyId. A complementary layer field pair is a collective term for a complementary reference layer field pair and a complementary non-reference layer field pair.

– A non-paired non-reference field is interpreted as layer field with nal_ref_idc equal to 0 for the dependency representation with dependency_id equal to currDependencyId that is not part of a complementary non-reference layer field pair.

– A non-paired reference field is interpreted as layer field with nal_ref_idc greater than 0 for the dependency representation with dependency_id equal to currDependencyId that is not part of a complementary reference layer field pair.

– A non-paired field is interpreted as layer field for the dependency representation with dependency_id equal to currDependencyId that is not part of a complementary layer field pair.

– A reference base frame is interpreted as reference layer base frame for the dependency representation with dependency_id equal to currDependencyId. A reference layer base frame for a particular value of dependency_id represents a second representation of a layer frame for dependency representations with nal_ref_idc greater than 0, store_ref_base_pic_flag equal to 1, and field_pic_flag equal to 0.

– A reference base field is interpreted as reference layer base field for the dependency representation with dependency_id equal to currDependencyId. A reference layer base field for a particular value of dependency_id represents a second representation of a layer field for dependency representations with nal_ref_idc greater than 0, store_ref_base_pic_flag equal to 1, and field_pic_flag equal to 1.

– A reference base picture is interpreted as reference layer base picture for the dependency representation with dependency_id equal to currDependencyId. A reference layer base picture is a collective term for a reference layer base field or a reference layer base frame. A reference layer base picture is not associated with the sample arrays B_L, B_{Cb}, or B_{Cr}.

– A complementary reference base field pair is interpreted as complementary reference layer base field pair for the dependency representation with dependency_id equal to currDependencyId. A complementary reference layer base field pair for a particular value of dependency_id is a pair of two reference layer base fields for the particular value of dependency_id with the following properties: (i) the reference layer base fields are in consecutive access units containing a dependency representation with the particular value of dependency_id, (ii) the dependency representations with the particular value of dependency_id in these access units have nal_ref_idc greater than 0, store_ref_base_pic_flag equal to 1, field_pic_flag equal to 1 and the same value of frame_num, (iii) the dependency representation with the particular value of dependency_id of the second of these access units in decoding order has IdrPicFlag equal to 0 and does not contain a memory_management_control_operation syntax element equal to 5.

– A non-paired reference base field is interpreted as reference layer base field for the dependency representation with dependency_id equal to currDependencyId that is not part of a complementary reference layer base field pair.

G.8.2.1 SVC decoding process for picture order count

Input to this process is a list dqIdList of integer values specifying layer representation identifiers.

Outputs of this process are the variables TopFieldOrderCnt (if applicable) and BottomFieldOrderCnt (if applicable) for all dependency representations of the set depRepSet specified in the following.

Let depRepSet be the set of dependency representations for which (dependency_id << 4) is contained in the list dqIdList.

For all dependency representations of the set depRepSet, the variables TopFieldOrderCnt (if applicable) and BottomFieldOrderCnt (if applicable) are derived by invoking the decoding process for picture order count as specified in subclause 8.2.1. For these invocations of the process specified in subclause 8.2.1, the modifications a) and b) specified in subclause G.8.2 apply with currDependencyId being equal to dependency_id of the corresponding dependency representation.

For all dependency representations of the set depRepSet for an access unit, either TopFieldOrderCnt or BottomFieldOrderCnt or both are derived. When both are derived in two or more dependency representations of an access unit, their difference shall be the same in these dependency representations of the access unit.

The values of TopFieldOrderCnt and BottomFieldOrderCnt are restricted as specified in the following ordered steps:

1. The set depRepSet for an access unit is the set depRepSet that has been derived in the process specified in this subclause for the corresponding access unit.
2. For each access unit, the one-dimensional array picOrderCnt is derived as follows:
   – If TopFieldOrderCnt is derived for all dependency representations of the set depRepSet for an access unit, for each dependency representation of the set depRepSet for this access unit, the variable picOrderCnt[dId] is set equal to TopFieldOrderCnt with dId being the value of dependency_id for the dependency representation.
   – Otherwise (TopFieldOrderCnt is not derived for all dependency representations of the set depRepSet for an access unit), for each dependency representation of the set depRepSet for this access unit, the variable picOrderCnt[dId] is set equal to BottomFieldOrderCnt with dId being the value of dependency_id for the dependency representation.
3. Let au0 and au1 be any pair of access units in the bitstream with au1 being later in decoding order than au0.
4. Let the flag idrConditionFlag be derived for each dependency representation of the set depRepSet for an access unit as follows:
   – If the dependency representation in the access unit has IdrPicFlag equal to 1 or a memory_management_control_operation syntax element equal to 5, idrConditionFlag is set equal to 1.
   – Otherwise (the dependency representation in the access unit has IdrPicFlag equal to 0 and does not have a memory_management_control_operation syntax element equal to 5), idrConditionFlag is set equal to 0.
5. Let the set dIdSet0 be the set of all dependency_id values of the set depRepSet for au0.
6. Let the set dIdSet1 be the set of all dependency_id values of the set depRepSet for au1 for which idrConditionFlag is not equal to 1 in any access unit in decoding order between the access unit that follows au0 and the access unit au1, inclusive.
7. For all values of dId that are present in both sets dIdSet0 and dIdSet1, the differences between the value picOrderCnt[dId] in au0 and the value picOrderCnt[dId] in au1 shall be the same.

G.8.2.2 SVC decoding process for picture numbers

This process is invoked when the SVC decoding process for reference picture lists construction specified in subclause G.8.2.3, the SVC reference picture marking process for a dependency representation specified in subclause G.8.2.4.1, or the SVC decoding process for gaps in frame_num specified in subclause G.8.2.5 is invoked.

Inputs to this process are:

– a variable currDependencyId specifying a dependency representation,
– a variable refPicListConstructionFlag specifying whether this process is invoked for reference picture lists construction,
– when refPicListConstructionFlag is equal to 1, a variable useRefBasePicFlag specifying whether reference base pictures are considered for reference picture lists construction.

From here to the end of this subclause, the modifications a) and b) specified in subclause G.8.2 apply.

The variables FrameNum, FrameNumWrap, and PicNum are assigned to all short-term reference pictures and the variables LongTermFrameIdx and LongTermPicNum are assigned to all long-term reference pictures by invoking the decoding process for picture numbers as specified in subclause 8.2.4.1.

NOTE 1 – For this invocation of the process specified in subclause 8.2.4.1, the pictures marked as "reference base pictures" and the pictures not marked as "reference base pictures" are taken into account.

For the following specification of this subclause, reference frames, complementary reference field pairs, and non-paired reference fields with at least one field marked as "used for reference" are referred to as reference entries. When only one field of a reference entry is marked as "used for reference", the reference entry is considered to have the same marking(s) and the same assigned variables as its field marked as "used for reference". When a reference entry is marked as "not available for reference list construction" in the following process, both of its fields are also marked as "not available for reference list construction".

When refPicListConstructionFlag is equal to 1, the following applies:

– If useRefBasePicFlag is equal to 0, all reference entries that are marked as "reference base picture" are marked as "not available for reference list construction".

NOTE 2 – When useRefBasePicFlag is equal to 0, only reference entries that are not marked as "reference base picture" are considered as present for the purpose of reference picture lists construction.

– Otherwise (useRefBasePicFlag is equal to 1), all reference entries for which one of the following conditions is true are marked as "not available for reference list construction":

  – the reference entry is not marked as "reference base picture", the reference entry is marked as "used for short-term reference", and there exists a reference entry with the same value of FrameNum that is marked as "reference base picture" and "used for short-term reference",

  – the reference entry is not marked as "reference base picture", the reference entry is marked as "used for long-term reference", and there exists a reference entry with the same value of LongTermFrameIdx that is marked as "reference base picture" and "used for long-term reference".

NOTE 3 – When useRefBasePicFlag is equal to 1 and either two short-term reference entries have the same value of FrameNum or two long-term reference entries have the same value of LongTermFrameIdx (one of these reference entries is marked as "reference base picture" and the other reference entry is not marked as "reference base picture"), only the reference entry marked as "reference base picture" is considered as present for the purpose of reference picture lists construction.

G.8.2.3 SVC decoding process for reference picture lists construction

This process is invoked at the beginning of the decoding process for each P, EP, B, or EB slice.

Inputs to this process are:

– a variable currDependencyId,

– a variable useRefBasePicFlag,

– the current slice currSlice.

Outputs of this process are:

– a reference picture list refPicList0,

– for B and EB slices, a reference picture list refPicList1.

After applying the process described in this subclause, the output reference picture lists refPicList0 and refPicList1 (when applicable) shall not contain any pictures for which the syntax element temporal_id is greater than the syntax element temporal_id of the current picture.

From here to the end of this subclause, the modifications a) and b) specified in subclause G.8.2 apply.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the slice header of the current slice currSlice, the current picture parameter, which is identified by the syntax element pic_parameter_set_id inside the slice header of the current slice currSlice, and the current sequence parameter, which is identified by the syntax element seq_parameter_set_id inside the current picture parameter set.

A variable biPred is derived as follows:

– If the current slice currSlice is a B or EB slice, biPred is set equal to 1.

– Otherwise, biPred is set equal to 0.

Decoded reference pictures are marked as "used for short-term reference" or "used for long-term reference" as specified by the bitstream and specified in subclause G.8.2.4. Short-term reference pictures are identified by the value of frame_num that is decoded in the slice header(s) with dependency_id equal to currDependencyId. Long-term reference pictures are identified by a long-term frame index as specified by the bitstream and specified in subclause G.8.2.4.

Subclause G.8.2.2 is invoked with currDependencyId, refPicListConstructionFlag equal to 1, and useRefBasePicFlag as inputs to specify the following:

– the assignment of variables FrameNum, FrameNumWrap, and PicNum to each of the short-term reference pictures,

– the assignment of variables LongTermPicNum to each of the long-term reference pictures,

– the marking of reference pictures that are not used for reference picture lists construction as "not available for reference list construction" (depending on the value of useRefBasePicFlag).

NOTE 1 – The marking of reference pictures as "not available for reference list construction" is removed after construction of the reference picture lists.

Reference pictures are addressed through reference indices as specified in subclause 8.4.2.1 with the modification e) specified in subclause G.8.4.2. A reference index is an index into a reference picture list. When biPred is equal to 0, a single reference picture list refPicList0 is constructed. When decoding a B or EB slice (biPred is equal to 1), a second independent reference picture list refPicList1 is constructed in addition to refPicList0.

At the beginning of the decoding process for each slice, reference picture list refPicList0, and for biPred equal to 1 refPicList1, are derived as specified in the following ordered steps:

1. Initial reference picture lists RefPicList0 and, for biPred equal to 1, RefPicList1 are derived by invoking the initialisation process for reference picture lists as specified in subclause 8.2.4.2. During the initialisation process in subclause 8.2.4.2 all reference frames, complementary reference field pairs, and non-paired reference fields that have been marked as "not available for reference list construction" by the invocation of subclause G.8.2.2 are considered as not present.

2. When ref_pic_list_modification_flag_l0 is equal to 1 or, when decoding a B or EB slice (biPred is equal to 1), ref_pic_list_modification_flag_l1 is equal to 1, the initial reference picture list RefPicList0 and for biPred equal to 1 RefPicList1 are modified by invoking the modification process for reference picture lists as specified in subclause 8.2.4.3. During the modification process in subclause 8.2.4.3 all reference frames, complementary reference field pairs, and non-paired reference fields that have been marked as "not available for reference list construction" by the invocation of subclause G.8.2.2 are considered as not present.

3. RefPicList0 is assigned to refPicList0.

4. When biPred is equal to 1, RefPicList1 is assigned to refPicList1.

NOTE 2 – By the invocation of the process in subclause G.8.2.2 some reference frames, complementary reference field pairs, and non-paired reference fields might have been marked as "not available for reference list construction". Since, these pictures are not considered in the construction process for reference picture lists, the reference picture lists refPicList0 and, for biPred equal to 1, refPicList1 are dependent on the value of the input parameter useRefBasePicFlag.

The number of entries in the modified reference picture list refPicList0 is num_ref_idx_l0_active_minus1 + 1, and for biPred equal to 1 the number of entries in the modified reference picture list refPicList1 is num_ref_idx_l1_active_minus1 + 1. A reference picture may appear at more than one index in the modified reference picture lists refPicList0 or refPicList1.

For all reference frames, complementary reference field pairs, and non-paired reference fields that have been marked as "not available for reference list construction" during the invocation of subclause G.8.2.2, this marking is removed.

G.8.2.4 SVC decoded reference picture marking process

Input to this process is a list dqIdList of integer values specifying layer representation identifiers.

Let depRepSet be the set of dependency representations for which all of the following conditions are true:

– the list dqIdList contains the value (dependency_id << 4),

– nal_ref_idc is greater than 0.

For each dependency representation of the set depRepSet, the SVC reference picture marking process for a dependency representation as specified in subclause G.8.2.4.1 is invoked. For these invocations of the process specified in subclause G.8.2.4.1, the modifications a) and b) specified in subclause G.8.2 apply with currDependencyId being equal to dependency_id for the corresponding dependency representation.

G.8.2.4.1 SVC reference picture marking process for a dependency representation

Input to this process is a variable currDependencyId.

Output of this process is a modified reference picture marking for dependency representations with dependency_id equal to currDependencyId.

This process is invoked for a decoded picture when nal_ref_idc is not equal to 0 for the dependency representation with dependency_id being equal to currDependencyId.

All syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are syntax elements and derived upper-case variables for the dependency representation with dependency_id equal to currDependencyId.

A decoded picture with nal_ref_idc not equal to 0, referred to as a reference picture, is marked as "used for short-term reference" or "used for long-term reference". When store_ref_base_pic_flag is equal to 1, a second representation of the decoded picture also referred to as reference base picture is marked as "used for short-term reference" or "used for long-term reference" and additionally marked as "reference base picture". Pictures that are marked as "reference base picture" are only used as references for inter prediction of following pictures with use_ref_base_pic_flag equal to 1. These pictures are not used for inter prediction of pictures with use_ref_base_pic_flag equal to 0, and these pictures do not represent an output of the decoding process.

For a decoded reference frame, both of its fields are marked the same as the frame. For a complementary reference field pair, the pair is marked the same as both of its fields. A picture that is marked as "used for short-term reference" is identified by its FrameNum and, when it is a field, by its parity, and, when it is a reference base picture, by the marking "reference base picture". A picture that is marked as "used for long-term reference" is identified by its LongTermFrameIdx and, when it is a field, by its parity, and, when it is a reference base picture, by the marking "reference base picture".

While decoded pictures are represented by the sample arrays S_L and, when ChromaArrayType is not equal to 0, S_{Cb} and S_{Cr}, reference base pictures are represented by the sample arrays B_L and, when ChromaArrayType is not equal to 0, B_{Cb} and B_{Cr}. When reference base pictures are referenced in the inter prediction process via subclause 8.4.2.1, the samples arrays B_L, B_{Cb}, and B_{Cr} are referred to as S_L, S_{Cb}, and S_{Cr}, respectively. The sample arrays S_L, S_{Cb}, S_{Cr}, B_L, B_{Cb}, and B_{Cr} that referenced in the inter prediction process via subclause 8.4.2.1 are constructed as specified in subclause G.8. Reference base pictures are associated with the same descriptive information such as the variables FrameNum, FrameNumWrap, PicNum, LongTermFrameIdx, and LongTermPicNum as decoded pictures.

Frames or complementary field pairs marked as "used for short-term reference" or as "used for long-term reference" can be used as a reference for inter prediction when decoding a frame until the frame, the complementary field pair, or one of its constituent fields is marked as "unused for reference". A field marked as "used for short-term reference" or as "used for long-term reference" can be used as a reference for inter prediction when decoding a field until marked as "unused for reference".

A picture can be marked as "unused for reference" by the sliding window reference picture marking process, a first-in, first-out mechanism specified in subclause G.8.2.4.2, or by the adaptive memory control reference picture marking process, a customised adaptive marking operation specified in subclauses G.8.2.4.3 and G.8.2.4.4.

A short-term reference picture is identified for use in the decoding process by its variables FrameNum and FrameNumWrap and its picture number PicNum, and, when it is a reference base picture, by the marking as "reference base picture". A long-term reference picture is identified for use in the decoding process by its variable LongTermFrameIdx, its long-term picture number LongTermPicNum, and, when it is a reference base picture, by the marking as "reference base picture".

When the current picture is not an IDR picture, the variables FrameNum, FrameNumWrap, PicNum, LongTermFrameIdx, and LongTermPicNum are assigned to the reference pictures by invoking the SVC decoding process for picture numbers as specified in subclause G.8.2.2 with currDependencyId and refPicListConstructionFlag set equal to 0 as the inputs.

Decoded reference picture marking proceeds in the following ordered steps:

1. All slices of the current access unit are decoded.
2. Depending on the current picture, the following applies:
   - If the current picture is an IDR picture, the following ordered steps are specified:
     a. All reference pictures are marked as "unused for reference".
     b. Depending on long_term_reference_flag, the following applies:
        - If long_term_reference_flag is equal to 0, the following ordered steps are specified:
          i. The IDR picture is marked as "used for short-term reference" and MaxLongTermFrameIdx is set equal to "no long-term frame indices".
          ii. When store_ref_base_pic_flag is equal to 1, the reference base picture of the IDR picture is marked as "used for short-term reference" and as "reference base picture".
        - Otherwise (long_term_reference_flag is equal to 1), the following ordered steps are specified:
          i. The IDR picture is marked as "used for long-term reference", the LongTermFrameIdx for the IDR picture is set equal to 0, and MaxLongTermFrameIdx is set equal to 0.
          ii. When store_ref_base_pic_flag is equal to 1, the reference base picture of the IDR picture is marked as "used for long-term reference" and as "reference base picture", and the LongTermFrameIdx for the reference base picture of the IDR picture is set equal to 0.
   - Otherwise (the current picture is not an IDR picture), the following ordered steps are specified:

a. When adaptive_ref_base_pic_marking_mode_flag is equal to 1, the SVC adaptive memory control reference base picture marking process as specified in subclause G.8.2.4.3 is invoked.

NOTE 1 – By this invocation of the process specified in subclause G.8.2.4.3, pictures that are marked as "used for reference" and "reference base picture" can be marked as "unused for reference".

With currTId being the value of temporal_id for the current access unit, the bitstream shall not contain data that result in the marking of pictures with temporal_id less currTId as "unused for reference" during this invocation of the process in subclause G.8.2.4.3.

b. Depending on adaptive_ref_pic_marking_mode_flag, the following applies:

– If adaptive_ref_pic_marking_mode_flag is equal to 1, the SVC adaptive memory control decoded reference picture marking process as specified in subclause G.8.2.4.4 is invoked.

With currTId being the value of temporal_id for the current access unit, the bitstream shall not contain data that result in the marking of pictures with temporal_id less currTId as "unused for reference" during this invocation of the process in subclause G.8.2.4.4.

– Otherwise (adaptive_ref_pic_marking_mode_flag is equal to 0), the SVC sliding window decoded reference picture marking process as specified in subclause G.8.2.4.2 is invoked with refBasePicFlag equal to 0 as the input.

c. When the current picture was not marked as "used for long-term reference" by a memory_management_control_operation command equal to 6, the current picture is marked as "used for short-term reference" and, when the current picture is the second field (in decoding order) of a complementary reference field pair and the first field is marked as "used for short-term reference", the complementary field pair is also marked as "used for short-term reference".

d. When store_ref_base_pic_flag is equal to 1 and the reference base picture for the current picture was not marked as "used for long-term reference" by a memory_management_control_operation command equal to 6, the following ordered steps are specified:

i. When adaptive_ref_base_pic_marking_mode_flag is equal to 0, the following ordered steps are specified:

(1) The SVC decoding process for picture numbers as specified in subclause G.8.2.2 is invoked with currDependencyId and refPicListConstructionFlag set equal to 0 as the inputs.

(2) The SVC sliding window decoded reference picture marking process as specified in subclause G.8.2.4.2 is invoked with refBasePicFlag equal to 1 as the input.

ii. The reference base picture of the current picture is marked as "used for short-term reference"and as "reference base picture" and, when the reference base picture of the current picture is the second reference base field (in decoding order) of a complementary reference base field pair and the first reference base field is marked as "used for short-term reference" (and "reference base picture"), the complementary reference base field pair is also marked as "used for short-term reference" and "reference base picture".

NOTE 2 – When both the decoded picture and the reference base picture for an access unit (including the current access unit) are marked as "used for reference", either both pictures are marked as "used for short-term reference" or both pictures are marked as "used for long-term reference" after the completion of the process specified in this subclause. And in the latter case, the same value of LongTermFrameIdx is assigned to both pictures.

It is a requirement of bitstream conformance that, after marking the current decoded reference picture and, when store_ref_base_pic_flag is equal to 1, the current reference base picture, the total number of frames with at least one field marked as "used for reference", plus the number of complementary field pairs with at least one field marked as "used for reference", plus the number of non-paired fields marked as "used for reference" shall not be greater than Max(max_num_ref_frames, 1).

NOTE 3 – For this constraint, the pictures marked as "reference base pictures" and the pictures not marked as "reference base picture" are taken into account.

G.8.2.4.2 SVC sliding window decoded reference picture marking process

Input to this process is a variable refBasePicFlag.

The variable newFrameBufferFlag is derived as follows:

– If one of the following conditions is true, newFrameBufferFlag is set equal to 0:

– refBasePicFlag is equal to 0, the current picture is a coded field that is the second field in decoding order of a complementary reference field pair, and the first field of the complementary reference field pair has been marked as "used for short-term reference",

– refBasePicFlag is equal to 1, the current reference base picture is a reference base field that is the second field in decoding order of a complementary reference base field pair, and the first field has been marked as "used for short-term reference" (and "reference base picture").

– Otherwise, newFrameBufferFlag is set equal to 1.

When newFrameBufferFlag is equal to 1, the following ordered steps are specified:

1. Let numShortTerm be the total number of reference frames, complementary reference field pairs, and non-paired reference fields for which at least one field is marked as "used for short-term reference". Let numLongTerm be the total number of reference frames, complementary reference field pairs, and non-paired reference fields for which at least one field is marked as "used for long-term reference".

   NOTE 1 – For this derivation of numShortTerm and numLongTerm, the pictures marked as "reference base pictures" and the pictures not marked as "reference base picture" are taken into account.

2. When numShortTerm + numLongTerm is equal to Max(max_num_ref_frames, 1), the following ordered steps are specified:

   a. The condition that numShortTerm is greater than 0 shall be fulfilled.

   b. Let frameNumWrapDecPic be the smallest value of FrameNumWrap that is assigned to reference frames, complementary reference field pairs, and non-paired reference fields that are marked as "used for short-term reference" and not marked as "reference base pictures". When there doesn't exist any reference frame, complementary reference field pair, or non-paired reference field that is marked as "used for short-term reference" and not marked as "reference base picture", frameNumWrapDecPic is set equal to MaxFrameNum.

   c. Let frameNumWrapBasePic be the smallest value of FrameNumWrap that is assigned to reference frames, complementary reference field pairs, and non-paired reference fields that are marked as "used for short-term reference" and marked as "reference base pictures". When there doesn't exist any reference frame, complementary reference field pair, or non-paired reference field that is marked as "used for short-term reference" and marked as "reference base picture", frameNumWrapBasePic is set equal to MaxFrameNum.

      NOTE 2 – The value of MaxFrameNum is greater than all values of FrameNumWrap that are assigned to reference frames, complementary reference field pairs, and non-paired reference fields marked as "used for short-term reference.

   d. The short-term reference frame, complementary reference field pair, or non-paired reference field picX is derived as follows:

      – If frameNumWrapDecPic is less than frameNumWrapBasePic, picX is the short-term reference frame, complementary reference field pair, or non-paired reference field that has the value of FrameNumWrap equal to frameNumWrapDecPic (and is not marked as "reference base picture").

      – Otherwise (frameNumWrapDecPic is greater than or equal to frameNumWrapBasePic), picX is the short-term reference frame, complementary reference field pair, or non-paired reference field that has the value of FrameNumWrap equal to frameNumWrapBasePic and is marked as "reference base picture".

   e. It is a requirement of bitstream conformance that the short-term reference frame, complementary reference field pair, or non-paired reference field picX shall not be the current picture or the complementary field pair that contains the current picture.

      NOTE 3 – When refBasePicFlag is equal to 1, the current picture has been marked as "used for short-term reference" in the same invocation of the process specified in subclause G.8.2.4.1.

   f. The short-term reference frame, complementary reference field pair, or non-paired reference field picX is marked as "unused for reference". When it is a frame or a complementary field pair, both of its fields are also marked as "unused for reference".

G.8.2.4.3 SVC adaptive memory control reference base picture marking process

This process is invoked when adaptive_ref_base_pic_marking_mode_flag is equal to 1.

The memory_management_base_control_operation commands with values of 1 and 2 are processed in the order they occur in the dec_ref_base_pic_marking( ) syntax structure after the current picture has been decoded. The memory_management_base_control_operation command with value of 0 specifies the end of the memory_management_base_control_operation commands.

Memory management control base operations are applied to pictures as follows:

– If field_pic_flag is equal to 0, memory_management_base_control_operation commands are applied to the reference base frames or complementary reference base field pairs specified.

– Otherwise (field_pic_flag is equal to 1), memory_management_base_control_operation commands are applied to the individual reference base fields specified.

For each memory_management_base_control_operation command with a value not equal to 0, the following applies:

– If memory_management_base_control_operation is equal to 1, the marking process of a short-term reference picture as "unused for reference" as specified in subclause 8.2.5.4.1, is invoked with substituting difference_of_pic_nums_minus1 with difference_of_base_pic_nums_minus1. For this invocation of the process specified in subclause 8.2.5.4.1, all pictures that are not marked as "reference base picture" are considered as not present.

  NOTE 1 – Short-term reference pictures that are not marked as "reference base pictures" cannot be marked as "unused for reference" by a memory_management_base_control_operation equal to 1.

– Otherwise, if memory_management_base_control_operation is equal to 2, the marking process of a long-term reference picture as "unused for reference" as specified in subclause 8.2.5.4.2 is invoked with substituting long_term_pic_num with long_term_base_pic_num. For this invocation of the process specified in subclause 8.2.5.4.2, all pictures that are not marked as "reference base picture" are considered as not present.

  NOTE 2 – Long-term reference pictures that are not marked as "reference base pictures" cannot be marked as "unused for reference" by a memory_management_base_control_operation equal to 2.

G.8.2.4.4 SVC adaptive memory control decoded reference picture marking process

This process is invoked when adaptive_ref_pic_marking_mode_flag is equal to 1.

The memory_management_control_operation commands with values of 1 to 6 are processed in the order they occur in the dec_ref_pic_marking( ) syntax structure after the current picture has been decoded. The memory_management_control_operation command with value of 0 specifies the end of the memory_management_control_operation commands.

Memory management control operations are applied to pictures as follows:

– If field_pic_flag is equal to 0, memory_management_control_operation commands are applied to the frames or complementary reference field pairs specified.

– Otherwise (field_pic_flag is equal to 1), memory_management_control_operation commands are applied to the individual reference fields specified.

For each memory_management_control_operation command with a value not equal to 0, the following applies:

– If memory_management_control_operation is equal to 1, the marking process of a short-term reference picture as "unused for reference" as specified in subclause 8.2.5.4.1 is invoked. For this invocation of the process specified in subclause 8.2.5.4.1, all pictures that are marked as "reference base picture" are considered as not present.

  NOTE 1 – Short-term reference pictures that are marked as "reference base pictures" cannot be marked as "unused for reference" by a memory_management_control_operation equal to 1.

– Otherwise, if memory_management_control_operation is equal to 2, the marking process of a long-term reference picture as "unused for reference" as specified in subclause 8.2.5.4.2 is invoked. For this invocation of the process specified in subclause 8.2.5.4.2, all pictures that are marked as "reference base picture" are considered as not present.

  NOTE 2 – Long-term reference pictures that are marked as "reference base pictures" cannot be marked as "unused for reference" by a memory_management_control_operation equal to 2.

– Otherwise, if memory_management_control_operation is equal to 3, the following ordered steps are specified:

  1. The assignment process of a LongTermFrameIdx to a short-term reference picture as specified in subclause 8.2.5.4.3 is invoked. For this invocation of the process specified in subclause 8.2.5.4.3, all pictures that are marked as "reference base picture" are considered as not present. The variable picNumX is set equal to the value picNumX that is derived during the invocation of the process specified in subclause 8.2.5.4.3.

2. Depending on whether there exists a picture that is marked as "reference base picture" and "used for short-term reference" and has a value of PicNum equal to picNumX, the following applies:

   – If there exists a picture that is marked as "reference base picture" and "used for short-term reference" and has a value of PicNum equal to picNumX, the assignment process of a LongTermFrameIdx to a short-term reference picture as specified in subclause 8.2.5.4.3 is invoked again. For this second invocation of the process specified in subclause 8.2.5.4.3, all pictures that are not marked as "reference base picture" are considered as not present.

     NOTE 3 – When the marking of a decoded picture (not marked as "reference base picture") is changed from "used for short-term reference" to "used for long-term reference" and there exists a reference base picture (marked as "reference base picture") that has the same value of PicNum as the decoded picture (before the marking is modified), the marking of this reference base picture is also changed from "used for short-term reference" to "used for long-term reference" and the same value of LongTermFrameIdx is assigned to both the decoded picture and the reference base picture.

   – Otherwise, if LongTermFrameIdx equal to long_term_frame_idx is assigned to a long-term reference frame marked as "reference base picture" or a long-term complementary reference field pair marked as "reference base picture", that frame or complementary field pair and both of its fields are marked as "unused for reference".

   – Otherwise, if LongTermFrameIdx equal to long_term_frame_idx is assigned to a long-term reference field marked as "reference base picture", and the associated decoded picture (not marked as "reference base picture") is not part of a complementary field pair that includes the picture specified by picNumX (before invoking the process specified in subclause 8.2.5.4.3) and not marked as "reference base picture", that field is marked as "unused for reference".

     NOTE 4 – When a particular value of LongTermFrameIdx is assigned to a reference base picture (marked as "reference base picture") and a decoded picture (not marked as "reference base picture"), the reference base picture is either associated with the same access unit as the decoded picture or with an access unit that represents a field that is part of a complementary field pair that includes the decoded picture.

   – Otherwise, the reference picture marking is not modified.

– Otherwise, if memory_management_control_operation is equal to 4, the decoding process for MaxLongTermFrameIdx as specified in subclause 8.2.5.4.4 is invoked.

  NOTE 5 – For this invocation of the process specified in subclause 8.2.5.4.4, the pictures marked as "reference base pictures" and the pictures not marked as "reference base picture" are taken into account.

– Otherwise, if memory_management_control_operation is equal to 5, the marking process of all reference pictures as "unused for reference" and setting MaxLongTermFrameIdx to "no long-term frame indices" as specified in subclause 8.2.5.4.5 is invoked.

  NOTE 6 – For this invocation of the process specified in subclause 8.2.5.4.5, the pictures marked as "reference base pictures" and the pictures not marked as "reference base picture" are taken into account.

– Otherwise (memory_management_control_operation is equal to 6), the following ordered steps are specified:

1. The process for assigning a long-term frame index to the current picture as specified in subclause 8.2.5.4.6 in invoked. For this invocation of the process specified in subclause 8.2.5.4.6, all pictures that are marked as "reference base picture" are considered as not present.

2. Depending on store_ref_base_pic_flag, the following applies:

   – If store_ref_base_pic_flag is equal to 1, the reference base picture of the current picture is marked as "reference base picture" and the process for assigning a long-term frame index to the current picture as specified in subclause 8.2.5.4.6 is invoked again. For this second invocation of the process specified in subclause 8.2.5.4.6, the reference base picture is considered as the current picture and all pictures that are not marked as "reference base picture" are considered as not present. When the reference base picture of the current picture is the second reference base field (in decoding order) of a complementary reference base field pair, the complementary reference base field pair is also marked as "reference base picture".

     NOTE 7 – When the current decoded picture is marked as "used for long-term reference" and store_ref_base_pic_flag is equal to 1, the current reference base picture is also marked as "used for long-term reference" and the same value of LongTermFrameIdx is assigned to both the current decoded picture and the current reference base picture. The current reference base picture is additionally marked as "reference base picture".

   – Otherwise, if LongTermFrameIdx equal to long_term_frame_idx is assigned to a long-term reference frame marked as "reference base picture" or a long-term complementary reference field pair marked as "reference base picture", that frame or complementary field pair and both of its fields are marked as "unused for reference".

– Otherwise, if LongTermFrameIdx equal to long_term_frame_idx is assigned to a long-term reference field marked as "reference base picture", and the associated decoded picture (not marked as "reference base picture") is not part of a complementary field pair that includes the current picture, that field is marked as "unused for reference".

NOTE 8 – When a particular value of LongTermFrameIdx is assigned to a reference base picture (marked as "reference base picture") and a decoded picture (not marked as "reference base picture"), the reference base picture is either associated with the same access unit as the decoded picture or with an access unit that represents a field that is part of a complementary field pair that includes the decoded picture.

– Otherwise, the reference picture marking is not modified.

3. It is a requirement of bitstream conformance that, after marking the current decoded reference picture and, when store_ref_base_pic_flag is equal to 1, the current reference base picture, the total number of frames with at least one field marked as "used for reference", plus the number of complementary field pairs with at least one field marked as "used for reference", plus the number of non-paired fields marked as "used for reference" shall not be greater than Max(max_num_ref_frames, 1).

NOTE 9 – For this constraint, the pictures marked as "reference base pictures" and the pictures not marked as "reference base picture" are taken into account.

NOTE 10 – Under some circumstances, the above statement may impose a constraint on the order in which a memory_management_control_operation syntax element equal to 6 can appear in the decoded reference picture marking syntax relative to a memory_management_control_operation syntax element equal to 1, 2, 3, or 4, or it may impose a constraint on the value of adaptive_ref_base_pic_marking_mode_flag.

G.8.2.5 SVC decoding process for gaps in frame_num

Input to this process is a list dqIdList of integer values specifying layer representation identifiers.

Let depRepSet be the set of dependency representations for which (dependency_id << 4) is contained in the list dqIdList.

For all dependency representations of the set depRepSet, the following applies:

– The variable currDependencyId is set equal to the value of dependency_id for the currently considered dependency representation of the set depRepSet.

– The syntax elements gaps_in_frame_num_value_allowed_flag and frame_num and the derived upper-case variables PrevRefFrameNum and MaxFrameNum are the syntax elements and derived upper-case variables for the considered dependency representation.

– When gaps_in_frame_num_value_allowed_flag is equal to 0, the bitstream shall not contain data that result in frame_num not being equal to PrevRefFrameNum or (PrevRefFrameNum + 1) % MaxFrameNum.

NOTE – When gaps_in_frame_num_value_allowed_flag is equal to 0 and frame_num is not equal to PrevRefFrameNum and is not equal to (PrevRefFrameNum + 1) % MaxFrameNum, the decoding process should infer an unintentional loss of pictures.

– When frame_num is not equal to PrevRefFrameNum and is not equal to (PrevRefFrameNum + 1) % MaxFrameNum, the decoding process for gaps in frame_num as specified in subclause 8.2.5.2 is invoked. For this invocation of the process specified in subclause 8.2.5.2, the modifications a) and b) specified in subclause G.8.2 apply, the invocation of the decoding process for picture numbers specified in subclause 8.2.4.1 is substituted with the invocation of the SVC decoding process for picture numbers specified in subclause G.8.2.2 with currDependencyId and refPicListConstructionFlag equal to 0 as the inputs, and the invocation of sliding window picture marking process specified in subclause 8.2.5.3 is substituted with the invocation of the SVC sliding window decoded reference picture marking process specified in subclause G.8.2.4.2 with refBasePicFlag equal to 0 as the input.

G.8.3 SVC intra decoding processes

Subclause G.8.3.1 specifies the SVC derivation process for intra prediction modes.

Subclause G.8.3.2 specifies the SVC intra sample prediction and construction process.

G.8.3.1 SVC derivation process for intra prediction modes

This process is only invoked when base_mode_flag is equal to 0 and mbType[CurrMbAddr] specified as input to this process is equal to I_PCM, I_16x16, I_8x8, or I_4x4.

Inputs to this process are:

– a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,

– a list fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,

– a list baseModeFlag with PicSizeInMbs elements specifying the syntax element base_mode_flag for the macroblocks of the current layer representation,

– a list mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,

– a (PicSizeInMbs)x16 array ipred4x4 specifying Intra_4x4 prediction modes for macroblocks of the current layer representation,

– a (PicSizeInMbs)x4 array ipred8x8 specifying Intra_8x8 prediction modes for macroblocks of the current layer representation,

– a list ipred16x16 with PicSizeInMbs elements specifying Intra_16x16 prediction modes for macroblocks of the current layer representation,

– when ChromaArrayType is equal to 1 or 2, a list ipredChroma with PicSizeInMbs elements specifying intra chroma prediction modes for macroblocks of the current layer representation.

Outputs of this process are:

– a modified version of the array ipred4x4,

– a modified version of the array ipred8x8,

– a modified version of the list ipred16x16,

– when ChromaArrayType is equal to 1 or 2, a modified version of the array ipredChroma.

For all processes specified in clause 6 that are invoked from the process specified in this subclause or a child process of the process specified in this subclause, the following modifications apply:

a) In subclause 6.4.12.2, a macroblock with address mbAddr is treated as field macroblock when fieldMbFlag[mbAddr] is equal to 1, and it is treated as frame macroblock when fieldMbFlag[mbAddr] is equal to 0. In particular, the current macroblock is treated as field macroblock when fieldMbFlag[CurrMbAddr] is equal to 1, and it is treated as frame macroblock when fieldMbFlag[CurrMbAddr] is equal to 0.

b) In subclause 6.4.8, a macroblock with address mbAddr is treated to belong to a different slice than the current macroblock CurrMbAddr, when sliceIdc[mbAddr] is not equal to sliceIdc[CurrMbAddr].

c) In subclause 6.4.12.2, a macroblock mbAddr is treated as top macroblock when (mbAddr % 2) is equal to 0, and it is treated as bottom macroblock when (mbAddr % 2) is equal to 1.

When mbType[CurrMbAddr] is not equal to I_PCM, the following applies:

– If mbType[CurrMbAddr] is equal to I_4x4, the SVC derivation process for Intra_4x4 prediction modes as specified in subclause G.8.3.1.1 is invoked with sliceIdc, fieldMbFlag, baseModeFlag, mbType, ipred4x4, and ipred8x8 as the inputs and the output is a modified version of the array ipred4x4.

– Otherwise, if mbType[CurrMbAddr] is equal to I_8x8, the SVC derivation process for Intra_8x8 prediction modes as specified in subclause G.8.3.1.2 is invoked with sliceIdc, fieldMbFlag, baseModeFlag, mbType, ipred4x4, and ipred8x8 as the inputs and the output is a modified version of the array ipred8x8.

– Otherwise, if mbType[CurrMbAddr] is equal to I_16x16, ipred16x16[CurrMbAddr] is set equal to Intra16x16PredMode.

When ChromaArrayType is equal to 1 or 2 and mbType[CurrMbAddr] is not equal to I_PCM, ipredChroma[CurrMbAddr] is set equal to intra_chroma_pred_mode.

G.8.3.1.1 SVC derivation process for Intra_4x4 prediction modes

Inputs to this process are:

– a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,

– a list fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,

– a list baseModeFlag with PicSizeInMbs elements specifying the syntax element base_mode_flag for the macroblocks of the current layer representation,

– a list mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,

– a (PicSizeInMbs)x16 array ipred4x4 specifying Intra_4x4 prediction modes for macroblocks of the current layer representation,

– a (PicSizeInMbs)x4 array ipred8x8 specifying Intra_8x8 prediction modes for macroblocks of the current layer representation.

Output of this process is a modified version of the array ipred4x4.

The 4x4 blocks indexed by c4x4BlkIdx = 0..15 are processed in increasing order of c4x4BlkIdx, and for each 4x4 block, the following ordered steps are specified:

1. The derivation process for neighbouring 4x4 luma blocks as specified in subclause 6.4.11.4 is invoked with c4x4BlkIdx as the input and the outputs are assigned to mbAddrA, c4x4BlkIdxA, mbAddrB, and c4x4BlkIdxB. For this invocation of the process in subclause 6.4.11.4, the modifications specified in items a) through c) in subclause G.8.3.1 apply.

2. For N being replaced by A and B, the variables availableFlagN are derived as follows:

   – If the macroblock mbAddrN is available and any of the following conditions are true, availableFlagN is set equal to 1:

     – constrained_intra_pred_flag is equal to 0,

     – mbType[mbAddrN] is equal to I_PCM and tcoeff_level_prediction_flag is equal to 1,

     – mbType[mbAddrN] is equal to I_PCM and baseModeFlag[mbAddrN] is equal to 0,

     – mbType[mbAddrN] is equal to I_16x16, I_8x8, or I_4x4.

   – Otherwise, availableFlagN is set equal to 0.

3. The variable dcPredModePredictedFlag is derived as follows:

   – If availableFlagA or availableFlagB is equal to 0, dcPredModePredictedFlag is set equal to 1.

   – Otherwise (availableFlagA is equal to 1 and availableFlagB is equal to 1), dcPredModePredictedFlag is set equal to 0.

4. For N being replaced by A and B, the variables intraMxMPredModeN are derived as follows:

   – If dcPredModePredictedFlag is equal to 0 and mbType[mbAddrN] is equal to I_4x4, intraMxMPredModeN is set equal to ipred4x4[mbAddrN][c4x4BlkIdxN].

   – Otherwise, if dcPredModePredictedFlag is equal to 0 and mbType[mbAddrN] is equal to I_8x8, intraMxMPredModeN is set equal to ipred8x8[mbAddrN][c4x4BlkIdxN >> 2].

   – Otherwise (dcPredModePredictedFlag is equal to 1 or (mbType[mbAddrN] is not equal to I_4x4 and mbType[mbAddrN] is not equal to I_8x8)), intraMxMPredModeN is set equal to 2.

5. The element ipred4x4[CurrMbAddr][c4x4BlkIdx] of the array ipred4x4 is derived by applying the procedure specified in the following pseudo-code:

```
predIntra4x4PredMode = Min( intraMxMPredModeA, intraMxMPredModeB )
if( prev_intra4x4_pred_mode_flag[ c4x4BlkIdx ] )
   ipred4x4[ CurrMbAddr ][ c4x4BlkIdx ] = predIntra4x4PredMode
else if( rem_intra4x4_pred_mode[ c4x4BlkIdx ] < predIntra4x4PredMode )          (G-85)
   ipred4x4[ CurrMbAddr ][ c4x4BlkIdx ] = rem_intra4x4_pred_mode[ c4x4BlkIdx ]
else
   ipred4x4[ CurrMbAddr ][ c4x4BlkIdx ] = rem_intra4x4_pred_mode[ c4x4BlkIdx ] + 1
```

G.8.3.1.2 SVC derivation process for Intra_8x8 prediction modes

Inputs to this process are:

– a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,

– a list fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,

– a list baseModeFlag with PicSizeInMbs elements specifying the syntax element base_mode_flag for the macroblocks of the current layer representation,

– a list mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,

– a (PicSizeInMbs)x16 array ipred4x4 specifying Intra_4x4 prediction modes for macroblocks of the current layer representation,

– a (PicSizeInMbs)x4 array ipred8x8 specifying Intra_8x8 prediction modes for macroblocks of the current layer representation.

Output of this process is a modified version of the array ipred8x8.

The 8x8 blocks indexed by c8x8BlkIdx = 0..3 are processed in increasing order of c8x8BlkIdx, and for each 8x8 block, the following ordered steps are specified:

1. The derivation process for neighbouring 8x8 luma blocks as specified in subclause 6.4.11.2 is invoked with c8x8BlkIdx as the input and the outputs are assigned to mbAddrA, c8x8BlkIdxA, mbAddrB, and c8x8BlkIdxB. For this invocation of the process in subclause 6.4.11.2, the modifications specified in items a) through c) in subclause G.8.3.1 apply.

2. For N being replaced by A and B, the variables availableFlagN are derived as follows:

   – If the macroblock mbAddrN is available and any of the following conditions are true, availableFlagN is set equal to 1:

     – constrained_intra_pred_flag is equal to 0,

     – mbType[mbAddrN] is equal to I_PCM and tcoeff_level_prediction_flag is equal to 1,

     – mbType[mbAddrN] is equal to I_PCM and baseModeFlag[mbAddrN] is equal to 0,

     – mbType[mbAddrN] is equal to I_16x16, I_8x8, or I_4x4.

   – Otherwise, availableFlagN is set equal to 0.

3. The variable dcPredModePredictedFlag is derived as follows:

   – If availableFlagA or availableFlagB is equal to 0, dcPredModePredictedFlag is set equal to 1.

   – Otherwise (availableFlagA is equal to 1 and availableFlagB are equal to 1), dcPredModePredictedFlag is set equal to 0.

4. For N being replaced by A and B, the variables intraMxMPredModeN are derived as follows:

   – If dcPredModePredictedFlag is equal to 0 and mbType[mbAddrN] is equal to I_4x4, intraMxMPredModeN is set equal to ipred4x4[mbAddrN][c8x8BlkIdxN * 4 + c4x4Idx] with the variable c4x4Idx being derived as follows:

     – If N is equal to B, c4x4Idx is set equal to 2.

     – Otherwise, if fieldMbFlag[CurrMbAddr] is equal to 0, fieldMbFlag[mbAddrN] is equal to 1, and c8x8BlkIdx is equal to 2, c4x4Idx is set equal to 3.

     – Otherwise (N is equal to A and (fieldMbFlag[CurrMbAddr] is equal to 1 or fieldMbFlag[mbAddrN] is equal to 0 or c8x8BlkIdx is not equal to 2)), c4x4Idx is set equal to 1.

   – Otherwise, if dcPredModePredictedFlag is equal to 0 and mbType[mbAddrN] is equal to I_8x8, intraMxMPredModeN is set equal to ipred8x8[mbAddrN][c8x8BlkIdxN].

   – Otherwise (dcPredModePredictedFlag is equal to 1 or (mbType[mbAddrN] is not equal to I_4x4 and mbType[mbAddrN] is not equal to I_8x8)), intraMxMPredModeN is set equal to 2.

5. The element ipred8x8[CurrMbAddr][c8x8BlkIdx] of the array ipred8x8 is derived by applying the procedure specified in the following pseudo-code:

```
predIntra8x8PredMode = Min( intraMxMPredModeA, intraMxMPredModeB )
if( prev_intra8x8_pred_mode_flag[ c8x8BlkIdx ] )
   ipred8x8[ CurrMbAddr ][ c8x8BlkIdx ] = predIntra8x8PredMode
else if( rem_intra8x8_pred_mode[ c8x8BlkIdx ] < predIntra8x8PredMode )          (G-86)
   ipred8x8[ CurrMbAddr ][ c8x8BlkIdx ] = rem_intra8x8_pred_mode[ c8x8BlkIdx ]
else
   ipred8x8[ CurrMbAddr ][ c8x8BlkIdx ] = rem_intra8x8_pred_mode[ c8x8BlkIdx ] + 1
```

G.8.3.2 SVC intra sample prediction and construction process

This process is only invoked when mbType specified as input to this process is equal to I_PCM, I_16x16, I_8x8, or I_4x4.

Inputs to this process are:

– a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,

– a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,

– a list baseModeFlag with PicSizeInMbs elements specifying the syntax element base_mode_flag for the macroblocks of the current layer representation,

– a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,

– a list ipred4x4 with 16 elements specifying Intra_4x4 prediction modes for the current macroblock,

– a list ipred8x8 with 4 elements specifying Intra_8x8 prediction modes for the current macroblock,

– a variable ipred16x16 specifying the Intra_16x16 prediction mode for the current macroblock,

– a variable ipredChroma specifying the intra chroma prediction mode for the current macroblock,

– a variable cTrafo specifying the transform type for the current macroblock,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,

– a $(\text{PicWidthInSamples}_L)\text{x}(\text{PicHeightInSamples}_L)$ array picSamples_L containing constructed luma sample values for the current layer representation.

– when ChromaArrayType is not equal to 0, two $(\text{PicWidthInSamples}_C)\text{x}(\text{PicHeightInSamples}_C)$ arrays picSamples_{Cb} and picSamples_{Cr} containing constructed chroma sample values for the current layer representation.

Outputs of this process are:

– a modified version of the array picSamples_L,

– when ChromaArrayType is not equal to 0, modified versions of the arrays picSamples_{Cb} and picSamples_{Cr}.

For all processes specified in clauses 6 or 8 that are invoked from the process specified in this subclause or a child process of the process specified in this subclause, the following modifications apply.

a) In subclause 6.4.12.2, a macroblock with address mbAddr is treated as field macroblock when fieldMbFlag[mbAddr] is equal to 1, and it is treated as frame macroblock when fieldMbFlag[mbAddr] is equal to 0. In particular, the current macroblock is treated as field macroblock when fieldMbFlag[CurrMbAddr] is equal to 1, and it is treated as frame macroblock when fieldMbFlag[CurrMbAddr] is equal to 0.

b) In subclause 6.4.8, a macroblock with address mbAddr is treated to belong to a different slice than the current macroblock CurrMbAddr, when MbToSliceGroupMap[mbAddr] is not equal to MbToSliceGroupMap[CurrMbAddr] or mbAddr is less than ((sliceIdc[CurrMbAddr] >> 7) * (1 + MbaffFrameFlag)), where MbToSliceGroupMap represents the variable that is derived as specified in subclause 8.2.2 for the layer representation with DQId equal to (sliceIdc[CurrMbAddr] & 127).

   NOTE 1 – When MaxTCoeffLevelPredFlag is equal to 0 or when all macroblocks of the current layer picture are covered by slices with the same value of DQId, the above condition can be simplified. In this case, two macroblocks mbAddrA and mbAddrB can be treated to belong to different slices, when sliceIdc[mbAddrA] is not equal to sliceIdc[mbAddrB].

c) In subclause 6.4.12.2, a macroblock mbAddr is treated as top macroblock when (mbAddr % 2) is equal to 0, and it is treated as bottom macroblock when (mbAddr % 2) is equal to 1.

d) In subclauses 8.3.1.2, 8.3.2.2, 8.3.3, and 8.3.4, the variables Intra4x4PredMode, Intra8x8PredMode, Intra16x16PredMode, and intra_chroma_pred_mode are replaced by ipred4x4, ipred8x8, ipred16x16, and ipredChroma, respectively.

e) In subclauses 8.3.1.2, 8.3.2.2, 8.3.3, and 8.3.4, the syntax element mb_type of a macroblock with macroblock address mbAddr is replaced by mbType[mbAddr].

f) The value of constrained_intra_pred_flag that is referred to in subclauses 8.3.1.2, 8.3.2.2, 8.3.3, and 8.3.4 is specified as follows:

  – If (sliceIdc[CurrMbAddr] & 127) is less than DQIdMax, the value of constrained_intra_pred_flag is the value of constrained_intra_pred_flag of the active layer picture parameter set for the layer representation with DQId equal to (sliceIdc[CurrMbAddr] & 127).

  – Otherwise ((sliceIdc[CurrMbAddr] & 127) is equal to DQIdMax), the value of constrained_intra_pred_flag is the value of constrained_intra_pred_flag of the active picture parameter set.

g) In subclauses 8.3.1.2, 8.3.2.2, 8.3.3, and 8.3.4, a macroblock with mbAddrN is treated as coded in an Inter macroblock prediction mode when all of the following conditions are false:

  – mbType[mbAddrN] is equal to I_PCM and tcoeff_level_prediction_flag for the slice with DQId equal to (sliceIdc[mbAddrN] & 127) and first_mb_in_slice equal to (sliceIdc[mbAddrN] >> 7) is equal to 1,

  – mbType[mbAddrN] is equal to I_PCM and baseModeFlag[mbAddrN] is equal to 0,

  – mbType[mbAddrN] is equal to I_16x16, I_8x8, or I_4x4,

  – sliceIdc[mbAddrN] is not equal to sliceIdc[CurrMbAddr].

  NOTE 2 – The latter condition does only have an impact on the decoding process when MaxTCoeffLevelPredFlag is equal to 1 and not all macroblocks of the current layer picture are covered by slices with the same value of DQId.

  NOTE 3 – Encoder designers are encouraged to generate bitstreams for which the removal of zero or more slice data NAL units with quality_id greater than 0 cannot result in a conforming bitstream for which a macroblock with address mbAddr is intra-predicted from a macroblock with address mbAddrN and sliceIdc[mbAddrN] not equal to sliceIdc[mbAddr].

The SVC intra sample prediction and construction process proceeds in the following ordered steps:

1. The construction process for luma residuals or chroma residuals with ChromaArrayType equal to 3 as specified in subclause G.8.5.3.1 is invoked with cTrafo and sTCoeff as the inputs and the outputs are residual luma sample values as a 16x16 array $mbRes_L$ with elements $mbRes_L$[x, y].

2. When ChromaArrayType is not equal to 0, the construction process for chroma residuals as specified in subclause G.8.5.3.2 is invoked with cTrafo and sTCoeff as the inputs and the outputs are residual chroma sample values as two (MbWidthC)x(MbHeightC) arrays $mbRes_{Cb}$ and $mbRes_{Cr}$ with elements $mbRes_{Cb}$[x, y] and $mbRes_{Cr}$[x, y], respectively.

3. The SVC intra prediction and construction process for luma samples or chroma samples with ChromaArrayType equal to 3 as specified in subclause G.8.3.2.1 is invoked with $BitDepth_Y$, sliceIdc, fieldMbFlag, mbType, ipred4x4, ipred8x8, ipred16x16, $mbRes_L$, and $picSamples_L$ as the inputs and the output is a modified version of the array $picSamples_L$.

4. When ChromaArrayType is not equal to 0, the SVC intra prediction and construction process for chroma samples as specified in subclause G.8.3.2.2 is invoked with sliceIdc, fieldMbFlag, mbType, ipred4x4, ipred8x8, ipred16x16, ipredChroma, $mbRes_{Cb}$, $mbRes_{Cr}$, $picSamples_{Cb}$, and $picSamples_{Cr}$ as the inputs and the outputs are modified versions of the arrays $picSamples_{Cb}$ and $picSamples_{Cr}$.

G.8.3.2.1 SVC intra prediction and construction process for luma samples or chroma samples with ChromaArrayType equal to 3

Inputs to this process are:

– a variable bitDepth specifying the bit depth,

– a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,

– a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,

– a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,

– a list ipred4x4 with 16 elements specifying Intra_4x4 prediction modes for the current macroblock,

– a list ipred8x8 with 4 elements specifying Intra_8x8 prediction modes for the current macroblock,

– a variable ipred16x16 specifying the Intra_16x16 prediction mode for the current macroblock,

– a 16x16 array mbRes containing residual sample values for the current macroblock,

– a ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array picSamples containing constructed sample values for the current layer representation.

Outputs of this process is a modified version of the array picSamples.

Depending on mbType[CurrMbAddr], the following applies:

– If mbType[CurrMbAddr] is equal to I_PCM, the SVC construction process for luma samples and chroma samples with ChromaArrayType equal to 3 of I_PCM macroblocks as specified in subclause G.8.3.2.1.1 is invoked with fieldMbFlag, mbRes, and picSamples as the inputs and the output is a modified version of the array picSamples.

– Otherwise, if mbType[CurrMbAddr] is equal to I_4x4, the SVC Intra_4x4 sample prediction and construction process as specified in subclause G.8.3.2.1.2 is invoked with bitDepth, sliceIdc, fieldMbFlag, mbType, ipred4x4, mbRes, and picSamples as the inputs and the output is a modified version of the array picSamples.

– Otherwise, if mbType[CurrMbAddr] is equal to I_8x8, the SVC Intra_8x8 sample prediction and construction process as specified in subclause G.8.3.2.1.3 is invoked with bitDepth, sliceIdc, fieldMbFlag, mbType, ipred8x8, mbRes, and picSamples as the inputs and the output is a modified version of the array picSamples.

– Otherwise (mbType[CurrMbAddr] is equal to I_16x16), the SVC Intra_16x16 sample prediction and construction process as specified in subclause G.8.3.2.1.4 is invoked with bitDepth, sliceIdc, fieldMbFlag, mbType, ipred16x16, mbRes, and picSamples as the inputs and the output is a modified version of the array picSamples.

G.8.3.2.1.1 SVC construction process for luma samples and chroma samples with ChromaArrayType equal to 3 of I_PCM macroblocks

Inputs to this process are:

– a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,

– a 16x16 array mbRes containing residual sample values for the current macroblock,

– a ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array picSamples containing constructed sample values for the current layer representation.

Output of this process is a modified version of the array picSamples.

The picture sample array construction process for a signal component as specified in subclause G.8.5.4.3 is invoked with fieldMbFlag[CurrMbAddr], mbW set equal to 16, mbH set equal to 16, mbRes, and picSamples as the inputs and the output is a modified version of the array picSamples.

G.8.3.2.1.2 SVC Intra_4x4 sample prediction and construction process

Inputs to this process are:

– a variable bitDepth specifying the bit depth,

– a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,

– a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,

– a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,

– a list ipred4x4 with 16 elements specifying Intra_4x4 prediction modes for the current macroblock,

– a 16x16 array mbRes containing residual sample values for the current macroblock,

– a $(PicWidthInSamples_L)x(PicHeightInSamples_L)$ array picSamples containing constructed sample values for the current layer representation.

Output of this process is a modified version of the array picSamples.

Let mbSamples be a 16x16 array containing constructed intra sample values for the current macroblock. All elements of mbSamples are initially set equal to 0.

The 4x4 blocks indexed by c4x4BlkIdx = 0..15 are processed in increasing order of c4x4BlkIdx, and for each 4x4 block, the following ordered steps are specified:

1. The Intra_4x4 sample prediction process as specified in subclause 8.3.1.2 is invoked with c4x4BlkIdx and picSamples as the inputs and the outputs are intra prediction sample values as a 4x4 array pred4x4 with elements pred4x4[x, y]. For this invocation of the process in subclause 8.3.1.2, the modifications specified in items a) through g) of subclause G.8.3.2 apply. Additionally in subclause 8.3.1.2.3, which may be invoked as part of the process specified in subclause 8.3.1.2, the variable $BitDepth_Y$ is replaced by bitDepth.
2. The inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with c4x4BlkIdx as the input and the output is assigned to (xP, yP).
3. For x = xP..(xP + 3) and y = yP..(yP + 3) and with Clip(a) specifying Clip3(0, (1 << bitDepth) − 1, a), the elements mbSamples[x, y] of the 16x16 array mbSamples are derived by

$$mbSamples[x, y] = Clip(pred4x4[x - xP, y - yP] + mbRes[x, y]) \quad (G-87)$$

4. The picture sample array construction process for a signal component as specified in subclause G.8.5.4.3 is invoked with fieldMbFlag[CurrMbAddr], mbW set equal to 16, mbH set equal to 16, mbSamples, and picSamples as the inputs and the output is a modified version of the array picSamples.

NOTE – When c4x4BlkIdx is less than 15, the array mbSamples does only contain constructed intra samples for 4x4 blocks with c4x4BlkIdx less than or equal to the current value of c4x4BlkIdx.

G.8.3.2.1.3 SVC Intra_8x8 sample prediction and construction process

Inputs to this process are:

– a variable bitDepth specifying the bit depth,
– a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,
– a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,
– a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,
– a list ipred8x8 with 4 elements specifying Intra_8x8 prediction modes for the current macroblock,
– a 16x16 array mbRes containing residual sample values for the current macroblock,
– a $(PicWidthInSamples_L)x(PicHeightInSamples_L)$ array picSamples containing constructed sample values for the current layer representation.

Output of this process is a modified version of the array picSamples.

Let mbSamples be a 16x16 array containing constructed intra sample values for the current macroblock. All elements of mbSamples are initially set equal to 0.

The 8x8 blocks indexed by c8x8BlkIdx = 0..3 are processed in increasing order of c8x8BlkIdx, and for each 8x8 block, the following ordered steps are specified:

1. The Intra_8x8 sample prediction process as specified in subclause 8.3.2.2 is invoked with c8x8BlkIdx and picSamples as the inputs and the outputs are intra prediction sample values as an 8x8 array pred8x8 with elements pred8x8[x, y]. For this invocation of the process in subclause 8.3.2.2, the modifications specified in items a) through g) of subclause G.8.3.2 apply. Additionally in subclause 8.3.2.2.4, which may be invoked as part of the process specified in subclause 8.3.2.2, the variable $BitDepth_Y$ is replaced by bitDepth.
2. The inverse 8x8 luma block scanning process as specified in subclause 6.4.5 is invoked with c8x8BlkIdx as the input and the output is assigned to (xP, yP).

3. For x = xP..(xP + 7) and y = yP..(yP + 7) and with Clip(a) specifying Clip3(0, (1 << bitDepth) − 1, a), the elements mbSamples[x, y] of the 16x16 array mbSamples are derived by

$$\text{mbSamples}[\ x, y\ ] = \text{Clip}(\ \text{pred8x8}[\ x - \text{xP}, y - \text{yP}\ ] + \text{mbRes}[\ x, y\ ]\ ) \qquad \text{(G-88)}$$

4. The picture sample array construction process for a signal component as specified in subclause G.8.5.4.3 is invoked with fieldMbFlag[CurrMbAddr], mbW set equal to 16, mbH set equal to 16, mbSamples, and picSamples as the inputs and the output is a modified version of the array picSamples.

   NOTE – When c8x8BlkIdx is less than 3, the array mbSamples does only contain constructed intra samples for 8x8 blocks with c8x8BlkIdx less than or equal to the current value of c8x8BlkIdx.

G.8.3.2.1.4 SVC Intra_16x16 sample prediction and construction process

Inputs to this process are:

– a variable bitDepth specifying the bit depth,

– a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,

– a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,

– a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,

– a variable ipred16x16 specifying the Intra_16x16 prediction mode for the current macroblock,

– a 16x16 array mbRes containing residual sample values for the current macroblock,

– a ($\text{PicWidthInSamples}_L$)x($\text{PicHeightInSamples}_L$) array picSamples containing constructed sample values for the current layer representation.

Output of this process is a modified version of the array picSamples.

The SVC Intra_16x16 sample prediction and construction process proceeds in the following ordered steps:

1. The Intra_16x16 prediction process for luma samples as specified in subclause 8.3.3 is invoked with picSamples as the input and the outputs are intra prediction sample values as a 16x16 array pred16x16 with elements pred16x16[x, y]. For this invocation of the process in subclause 8.3.3, the modifications specified in items a) through g) of subclause G.8.3.2 apply. Additionally in subclause 8.3.3.3, which may be invoked as part of the process specified in subclause 8.3.3, the variable BitDepth_Y is replaced by bitDepth.

2. With Clip(a) specifying Clip3(0, (1 << bitDepth) − 1, a), the 16x16 array mbSamples is derived by

$$\text{mbSamples}[\ x, y\ ] = \text{Clip}(\ \text{pred16x16}[\ x, y\ ] + \text{mbRes}[\ x, y\ ]\ ) \qquad \text{with } x, y = 0..15 \qquad \text{(G-89)}$$

3. The picture sample array construction process for a signal component as specified in subclause G.8.5.4.3 is invoked with fieldMbFlag[CurrMbAddr], mbW set equal to 16, mbH set equal to 16, mbSamples, and picSamples as the inputs and the output is a modified version of the array picSamples.

G.8.3.2.2 SVC intra prediction and construction process for chroma samples

Inputs to this process are:

– a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,

– a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,

– a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,

– a list ipred4x4 with 16 elements specifying Intra_4x4 prediction modes for the current macroblock,

– a list ipred8x8 with 4 elements specifying Intra_8x8 prediction modes for the current macroblock,

– a variable ipred16x16 specifying the Intra_16x16 prediction mode for the current macroblock,

– a variable ipredChroma specifying the intra chroma prediction mode for the current macroblock,

– two (MbWidthC)x(MbHeightC) arrays mbRes_{Cb} and mbRes_{Cr} containing residual chroma sample values for the current macroblock,

– two $(\text{PicWidthInSamples}_C)\text{x}(\text{PicHeightInSamples}_C)$ arrays picSamples_{Cb} and picSamples_{Cr} containing constructed sample values for the current layer representation.

Outputs of this process are modified versions of the arrays picSamples_{Cb} and picSamples_{Cr}.

Depending on ChromaArrayType, the following applies:

– If ChromaArrayType is equal to 1 or 2, the following applies:
  – If mbType[CurrMbAddr] is equal to I_PCM, the SVC construction process for chroma samples of I_PCM macroblock as specified in subclause G.8.3.2.2.1 is invoked with fieldMbFlag, mbRes_{Cb}, mbRes_{Cr}, picSamples_{Cb}, and picSamples_{Cr} as the inputs and the outputs are modified versions of picSamples_{Cb} and picSamples_{Cr}.
  – Otherwise (mbType[CurrMbAddr] is not equal to I_PCM), the SVC intra prediction and construction process for chroma samples with ChromaArrayType equal to 1 or 2 as specified in subclause G.8.3.2.2.2 is invoked with sliceIdc, fieldMbFlag, mbType, ipredChroma, mbRes_{Cb}, mbRes_{Cr}, picSamples_{Cb}, and picSamples_{Cr} as the inputs and the outputs are modified versions of the arrays picSamples_{Cb} and picSamples_{Cr}.
– Otherwise (ChromaArrayType is equal to 3), for CX being replaced by Cb and Cr, the SVC intra prediction and construction process for luma samples or chroma samples with ChromaArrayType equal to 3 as specified in subclause G.8.3.2.1 is invoked with BitDepth_C, sliceIdc, fieldMbFlag, mbType, ipred4x4, ipred8x8, ipred16x16, mbRes_{CX}, and picSamples_{CX} as the inputs and the output is a modified version of the array picSamples_{CX}.

G.8.3.2.2.1 SVC construction process for chroma samples of I_PCM macroblocks

Inputs to this process are:

– a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,
– two (MbWidthC)x(MbHeightC) arrays mbRes_{Cb} and mbRes_{Cr} containing residual chroma sample values for the current macroblock,
– two $(\text{PicWidthInSamples}_C)\text{x}(\text{PicHeightInSamples}_C)$ arrays picSamples_{Cb} and picSamples_{Cr} containing constructed chroma sample values for the current layer representation.

Outputs of this process are modified versions of the arrays picSamples_{Cb} and picSamples_{Cr}.

For CX being replaced by Cb and Cr, the picture sample array construction process for a signal component as specified in subclause G.8.5.4.3 is invoked with fieldMbFlag[CurrMbAddr], mbW set equal to MbWidthC, mbH set equal to MbHeightC, mbRes_{CX}, and picSamples_{CX} as the inputs and the output is a modified version of the array picSamples_{CX}.

G.8.3.2.2.2 SVC intra prediction and construction process for chroma samples with ChromaArrayType equal to 1 or 2

This process is only invoked when ChromaArrayType is equal to 1 or 2.

Inputs to this process are:

– a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,
– a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,
– a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,
– a variable ipredChroma specifying the intra chroma prediction mode for the current macroblock,
– two (MbWidthC)x(MbHeightC) arrays mbRes_{Cb} and mbRes_{Cr} containing residual chroma sample values for the current macroblock,
– two $(\text{PicWidthInSamples}_C)\text{x}(\text{PicHeightInSamples}_C)$ arrays picSamples_{Cb} and picSamples_{Cr} containing constructed chroma sample values for the current layer representation.

Outputs of this process are modified versions of the arrays picSamples_{Cb} and picSamples_{Cr}.

The SVC intra prediction and construction process for chroma samples with ChromaArrayType equal to 1 or 2 proceeds in the following ordered steps:

1. The intra prediction process for chroma samples as specified in subclause 8.3.4 is invoked with $picSamples_{Cb}$ and $picSamples_{Cr}$ as the inputs and the outputs are intra prediction chroma sample values as two (MbWidthC)x(MbHeightC) arrays $pred_{Cb}$ and $pred_{Cr}$ with elements $pred_{Cb}[x, y]$ and $pred_{Cr}[x, y]$, respectively. For this invocation of the process in subclause 8.3.4, the modifications specified in items a) through g) of subclause G.8.3.2 apply.

2. For CX being replaced by Cb and Cr, the (MbWidthC)x(MbHeightC) array $mbSamples_{CX}$ is derived by

$$mbSamples_{CX}[x, y] = Clip1_C(pred_{CX}[x, y] + mbRes_{CX}[x, y]) \quad \text{with } x = 0..(MbWidthC - 1) \text{ and } y = 0..(MbHeightC - 1) \qquad \text{(G-90)}$$

3. For CX being replaced by Cb and Cr, the picture sample array construction process for a signal component as specified in subclause G.8.5.4.3 is invoked with fieldMbFlag[CurrMbAddr], mbW set equal to MbWidthC, mbH set equal to MbHeightC, $mbSamples_{CX}$, and $picSamples_{CX}$ as the inputs and the output is a modified version of the array $picSamples_{CX}$.

G.8.4 SVC Inter prediction process

Subclause G.8.4.1 specifies the SVC derivation process for motion vector components and reference indices.

Subclause G.8.4.2 specifies the SVC decoding process for Inter prediction samples

G.8.4.1 SVC derivation process for motion vector components and reference indices

Inputs to this process are:

– a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,

– a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,

– a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,

– a (PicSizeInMbs)x4 array subMbType specifying sub-macroblock types for the macroblocks of the current layer representation,

– two (PicSizeInMbs)x4 arrays predFlagL0 and predFlagL1 specifying prediction utilization flags for the macroblocks of the current layer representation,

– two (PicSizeInMbs)x4 arrays refIdxL0 and refIdxL1 specifying reference indices for the macroblocks of the current layer representation,

– two (PicSizeInMbs)x4x4x2 arrays mvL0 and mvL1 specifying motion vector components for the macroblocks of the current layer representation,

– a one-dimensional array mvCnt with PicSizeInMbs elements specifying the number of motion vectors for the macroblocks of the current layer representation,

– two 2x2 arrays refIdxILPredL0 and refIdxILPredL1 specifying inter-layer reference index predictors for the current macroblock,

– two 4x4x2 arrays mvILPredL0 and mvILPredL1 specifying inter-layer motion vector predictors for the current macroblock,

– when DQId is equal to 0 and (slice_type % 5) is equal to 1, the reference list refPicList1.

Outputs of this process are:

– modified versions of the arrays predFlagL0 and predFlagL1,

– modified versions of the arrays refIdxL0 and refIdxL1,

– modified versions of the arrays mvL0 and mvL1,

– a modified version of the array mvCnt.

Depending on mbType[CurrMbAddr], the following applies:

– If mbType[CurrMbAddr] is equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL, the arrays predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, and mvCnt are modified by:

$$predFlagLX[CurrMbAddr][m] = 0 \quad \text{with } X = 0..1, m = 0..3 \quad (G-91)$$
$$refIdxLX[CurrMbAddr][m] = -1 \quad \text{with } X = 0..1, m = 0..3 \quad (G-92)$$
$$mvLX[CurrMbAddr][m][s][c] = 0 \quad \text{with } X = 0..1, m = 0..3, s = 0..3, c = 0..1 \quad (G-93)$$
$$mvCnt[CurrMbAddr] = 0 \quad (G-94)$$

– Otherwise (mbType[CurrMbAddr] is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL), the arrays predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, and mvCnt are modified as specified by the following text.

The variable numMbPart is derived as follows:

– If mbType[CurrMbAddr] is equal to B_Skip or B_Direct_16x16 and DQId is equal to 0 (nal_unit_type is not equal to 20), numMbPart is set equal to 4.

– Otherwise, if mbType[CurrMbAddr] is equal to B_Skip or B_Direct_16x16 (and DQId is greater than 0 and nal_unit_type is equal to 20), numMbPart is set equal to 1.

– Otherwise (mbType[CurrMbAddr] is not equal to B_Skip or B_Direct_16x16), numMbPart is set equal to NumMbPart(mbType[CurrMbAddr]).

The macroblock partition index mbPartIdx proceeds over the values 0..(numMbPart − 1), and for each value of mbPartIdx the following ordered steps are specified:

1. The variable isDirectFlag is derived as follows:
   – If any of the following conditions are true, isDirectFlag is set equal to 1:
     – mbType[CurrMbAddr] is equal to B_Skip or B_Direct_16x16,
     – mbType[CurrMbAddr] is equal to B_8x8 and subMbType[CurrMbAddr][mbPartIdx] is equal to B_Direct_8x8.
   – Otherwise, isDirectFlag is set equal to 0.
2. The variable numSubMbPart is derived as follows:
   – If isDirectFlag is equal to 1 and DQId is equal to 0 (nal_unit_type is not equal to 20), numSubMbPart is set equal to 4.
   – Otherwise, if isDirectFlag is equal to 1 (and DQId is greater than 0 and nal_unit_type is equal to 20), numSubMbPart is set equal to 1.
   – Otherwise (isDirectFlag is equal to 0), numSubMbPart is set equal to NumSubMbPart(subMbType[CurrMbAddr][mbPartIdx]).
3. The sub-macroblock partition index subMbPartIdx proceeds over values 0..(numSubMbPart − 1), and for each value of subMbPartIdx the SVC derivation process for luma motion vector components and reference indices of a macroblock or sub-macroblock partition as specified in subclause G.8.4.1.1 is invoked with mbPartIdx, subMbPartIdx, isDirectFlag, sliceIdc, fieldMbFlag, mbType, subMbType, predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, mvCnt, refIdxILPredL0, refIdxILPredL1, mvILPredL0, mvILPredL1, and, when DQId is equal to 0 and (slice_type % 5) is equal to 1, the reference picture list refPicList1 as the inputs and the outputs are modified versions of the arrays predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, and mvCnt.

G.8.4.1.1 SVC derivation process for luma motion vector components and reference indices of a macroblock or sub-macroblock partition

This subclause is only invoked when mbType[CurrMbAddr], which is specified as input to this subclause, is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL.

Inputs to this process are:

– a variable mbPartIdx specifying the current macroblock partition,

– a variable subMbPartIdx specifying the current sub-macroblock partition,

– a variable isDirectFlag specifying whether the current macroblock partition is coded in direct mode,

– a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,

– a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,

– a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,

– a (PicSizeInMbs)x4 array subMbType specifying sub-macroblock types for the macroblocks of the current layer representation,

– two (PicSizeInMbs)x4 arrays predFlagL0 and predFlagL1 specifying prediction utilization flags for the macroblocks of the current layer representation,

– two (PicSizeInMbs)x4 arrays refIdxL0 and refIdxL1 specifying reference indices for the macroblocks of the current layer representation,

– two (PicSizeInMbs)x4x4x2 arrays mvL0 and mvL1 specifying motion vector components for the macroblocks of the current layer representation,

– a one-dimensional array mvCnt with PicSizeInMbs elements specifying the number of motion vectors for the macroblocks of the current layer representation,

– two 2x2 arrays refIdxILPredL0 and refIdxILPredL1 specifying inter-layer reference index predictors for the current macroblock,

– two 4x4x2 arrays mvILPredL0 and mvILPredL1 specifying inter-layer motion vector predictors for the current macroblock,

– when DQId is equal to 0 and (slice_type % 5) is equal to 1, the reference picture list refPicList1.

Outputs of this process are:

– modified versions of the arrays predFlagL0 and predFlagL1,

– modified versions of the arrays refIdxL0 and refIdxL1,

– modified versions of the arrays mvL0 and mvL1,

– a modified version of the array mvCnt.

For all processes specified in clauses 6 or 8 that are invoked from the process specified in this subclause or a child process of the process specified in this subclause, the following modifications apply:

a) In subclauses 6.4.12.2 and 8.4.1.3.2, a macroblock with address mbAddr is treated as field macroblock when fieldMbFlag[mbAddr] is equal to 1, and it is treated as frame macroblock when fieldMbFlag[mbAddr] is equal to 0. In particular, the current macroblock is treated as field macroblock when fieldMbFlag[CurrMbAddr] is equal to 1, and it is treated as frame macroblock when fieldMbFlag[CurrMbAddr] is equal to 0.

b) In subclause 6.4.8, a macroblock with address mbAddr is treated to belong to a different slice than the current macroblock CurrMbAddr, when sliceIdc[mbAddr] is not equal to sliceIdc[CurrMbAddr].

c) In subclause 6.4.12.2, a macroblock mbAddr is treated as top macroblock when (mbAddr % 2) is equal to 0, and it is treated as bottom macroblock when (mbAddr % 2) is equal to 1.

d) In subclauses 6.4.2.1, 6.4.2.2, 6.4.11.7, 8.4.1.1, 8.4.1.3, any occurrence of mb_type is replaced by mbType[CurrMbAddr] with mbType being the array mbType that is input to this subclause.

e) In subclauses 6.4.2.2 and 6.4.11.7, any occurrence of sub_mb_type is replaced by subMbType[CurrMbAddr] with subMbType being the array subMbType that is input to this subclause.

f) In subclause 6.4.11.7, mb_type for a macroblock with macroblock address mbAddrN is replaced by mbType[mbAddrN] with mbType being the array mbType that is input to this subclause and sub_mb_type for a macroblock with macroblock address mbAddrN is replaced by subMbType[mbAddrN] with subMbType being the array subMbType that is input to this subclause.

g) In subclause 6.4.11.7, a macroblock partition or sub-macroblock partition given by mbAddrN, mbPartIdxN, and subMbPartIdxN is treated as not yet decoded when mbAddrN is equal to CurrMbAddr and (4 * mbPartIdxN + subMbPartIdxN) is greater than (4 * mbPartIdx + subMbPartIdx).

h) In subclause 8.4.1.3.2, a macroblock with mbAddrN is treated as coded in an Intra macroblock prediction mode when mbType[mbAddrN] is equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL.

i) In subclause 8.4.1.3.2, the variable predFlagLX of a macroblock or sub-macroblock partition given by mbAddrN\mbPartIdxN\subMbPartIdxN is replaced by predFlagLX[mbAddrN][mbPartIdxN] with predFlagLX being the array predFlagLX that is input to this subclause.

j) In subclause 8.4.1.3.2, the motion vector MvLX[mbPartIdxN][subMbPartIdxN] and the reference index RefIdxLX[mbPartIdxN] of a macroblock or sub-macroblock partition given by mbAddrN\mbPartIdxN\subMbPartIdxN are replaced by mvLX[mbAddrN][mbPartIdxN][subMbPartIdxN] and refIdxLX[mbAddrN][mbPartIdxN], respectively, with mvLX and refIdxLX being the arrays mvLX and refIdxLX, respectively, that are input to this subclause.

k) In subclause 8.4.1.2.1, any occurrence of RefPicList1[0] is replaced by refPicList1[0] with refPicList1[0] being the first layer field (when field_pic_flag is equal to 1) or the first layer frame or layer complementary field pair (when field_pic_flag is equal to 0) in the reference picture list refPicList1 that is specified as input to this subclause. The reference picture list refPicList1 is a reference list of layer pictures that correspond to layer representations with DQId equal to 0 of previously decoded access units.

l) In subclause 8.4.1.2.1, the current picture CurrPic represents the current layer picture with DQId equal to 0 and the variable colPic specifies the layer picture, for the layer representation with DQId equal to 0, that contains the co-located macroblock as specified in Table 8-6.

m) In subclause 8.4.1.2.1, all picture order count values are picture order count value for the dependency representation with dependency_id equal to 0.

n) In subclause 8.4.1.2.1, the modification b) specified in subclause G.8.2 applies with currDependencyId being equal to 0.

o) In subclause 8.4.1.2.1, for deriving the variable fieldDecodingFlagX, the macroblock mbAddrX is treated as field macroblock when fieldMbColPicFlag[mbAddrX] is equal to 1, it is treated as frame macroblock when fieldMbColPicFlag[mbAddrX] is equal to 0. The array fieldMbColPicFlag specifies the array fieldMbFlag that was derived by the process in subclause G.8.1.5.1 for the layer representation with DQId equal to 0.

p) In subclause 8.4.1.2.1, the variables PredFlagL0, PredFlagL1, RefIdxL0, RefIdxL1, MvL0, and MvL1 for the macroblock mbAddrCol inside the picture colPic are replaced with the predFlagL0[mbAddrCol], predFlagL1[mbAddrCol], refIdxL0[mbAddrCol], refIdxL1[mbAddrCol], mvL0[mbAddrCol], and mvL1[mbAddrCol], respectively, that have been derived for the layer picture colPic that is associated with DQId equal to 0.

q) In subclause 8.4.1.2.1, the macroblock mbAddrCol is interpreted as coded in an Intra macroblock prediction mode when mbType[mbAddrCol] that has been derived for the layer picture colPic that is associated with DQId equal to 0 is equal to I_16x16, I_8x8, I_4x4, or I_PCM.

r) In subclause 8.4.1.2.1, the syntax element mb_type of the macroblock with address mbAddrCol inside the picture colPic is replaced with mbType[mbAddrCol] that has been derived for the layer picture colPic that is associated with DQId equal to 0 and the syntax element list sub_mb_type of the macroblock with address mbAddrCol inside the picture colPic is replaced with the list subMbType[mbAddrCol] that has been derived for the layer picture colPic that is associated with DQId equal to 0.

s) In subclause 8.4.1.2.2, the co-located macroblock is treated as field macroblock when fieldMbColPicFlag[mbAddrCol] is equal to 1, it is treated as frame macroblock when fieldMbColPicFlag[mbAddrCol] is equal to 0. The array fieldMbColPicFlag specifies the array fieldMbFlag that was derived by the process in subclause G.8.1.5.1 for the layer representation with DQId equal to 0. The macroblock address mbAddrCol is the macroblock address of the co-located macroblock as derived in subclause 8.4.1.2.1.

The reference index predictors refIdxPredL0 and refIdxPredL1, the motion vector predictors mvPredL0 and mvPredL1, and the variable mvCntInc are derived as follows:

– If mbType[CurrMbAddr] is equal to P_Skip, the reference index predictor refIdxPredL1 is set equal to −1, both components of the motion vector predictor mvPredL1 are set equal to 0, the variable mvCntInc is set equal to 1, and the derivation process for luma motion vectors for skipped macroblocks in P slices as specified in subclause 8.4.1.1 is invoked with the outputs being assigned to the motion vector predictor mvPredL0 and the reference index predictor refIdxPredL0. For this invocation of the process in subclause 8.4.1.1, the modifications specified above in items a) through j) of this subclause apply.

– Otherwise, if isDirectFlag is equal to 1 and DQId is equal to 0 (nal_unit_type is not equal to 20), the derivation process for spatial direct luma motion vector and reference index prediction mode as specified in subclause 8.4.1.2.2 is invoked with mbPartIdx and subMbPartIdx as the inputs and the output variables refIdxL0, refIdxL1, mvL0, mvL1, and subMvCnt are assigned to the reference index predictors refIdxPredL0 and refIdxPredL1, the motion vectors predictors mvPredL0 and mvPredL1, and the variable mvCntInc, respectively. For this invocation of the process in subclause 8.4.1.2.2, the modifications specified above in items a) through s) of this subclause apply.

NOTE – When the current subclause is invoked, direct_spatial_mv_pred_flag is always equal to 1.

– Otherwise, if isDirectFlag is equal to 1 (and DQId is greater than 0 and nal_unit_type is equal to 20), the SVC derivation process for luma motion vectors and reference indices for B_Skip, B_Direct_16x16, and B_Direct_8x8 in NAL units with nal_unit_type equal to 20 as specified in subclause G.8.4.1.2 is invoked with mbPartIdx, fieldMbFlag, mbType, subMbType, predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, and mvL1 as the inputs and the outputs are refIdxPredL0, refIdxPredL1, mvPredL0, mvPredL1, and mvCntInc.

– Otherwise, the variable mvCntInc is initially set equal to 0, and for X being replaced by 0 and 1, the following applies:

  – If any of the following conditions are true, refIdxPredLX is set equal to −1 and both components of mvPredLX are set equal to 0:

    – mbType[CurrMbAddr] is not equal to P_8x8, P_8x8ref0, or B_8x8 and MbPartPredMode(mbType[CurrMbAddr], mbPartIdx) is not equal to Pred_LX or BiPred,

    – mbType[CurrMbAddr] is equal to P_8x8, P_8x8ref0, or B_8x8 and SubMbPartPredMode(subMbType[CurrMbAddr][mbPartIdx]) is not equal to Pred_LX or BiPred.

  – Otherwise, if base_mode_flag is equal to 1 or motion_prediction_flag_lX[mbPartIdx] is equal to 1, the following ordered steps are specified:

    1. The inverse macroblock partition scanning process as specified in subclause 6.4.2.1 is invoked with mbPartIdx as the input and the output is assigned to (xP, yP). For this invocation of the process in subclause 6.4.2.1, the modification specified above in item d) of this subclause applies.

    2. Inverse sub-macroblock partition scanning process as specified in subclause 6.4.2.2 is invoked with mbPartIdx and subMbPartIdx as the inputs and the output is assigned to (xS, yS). For this invocation of the process in subclause 6.4.2.2, the modifications specified above in items d) and e) of this subclause apply.

    3. The reference index predictor refIdxPredLX and the motion vector predictor mvPredLX are derived by

$$\begin{aligned}&\text{refIdxPredLX} = \text{refIdxILPredLX}[\,(\,xP + xS\,) / 8, (\,yP + yS\,) / 8\,]\\&\text{mvPredLX}[\,c\,] = \text{mvILPredLX}[\,(\,xP + xS\,) / 4, (\,yP + yS\,) / 4\,][\,c\,] \quad \text{with } c = 0..1\end{aligned} \tag{G-95}$$

    The bitstream shall not contain data that result in refIdxPredLX less than 0 or refIdxPredLX greater than num_ref_idx_active_lX_minus1.

    The bitstream shall not contain data that result in horizontal motion vector components mvPredLX[0] or vertical motion vector components mvPredLX[1] that exceed the range for motion vector components specified in subclause G.10.2.

    4. mvCntInc is set equal to (mvCntInc + 1).

  – Otherwise, the following ordered steps are specified:

    1. Depending on mbType[CurrMbAddr], the reference index predictor refIdxPredLX is derived as follows:

       – If mbType[CurrMbAddr] is equal to P_8x8ref0, refIdxPredLX is set equal to 0.

       – Otherwise (mbType[CurrMbAddr] is not equal to P_8x8ref0), refIdxPredLX is set equal to ref_idx_lX[mbPartIdx].

    2. The derivation process for luma motion vector prediction as specified in subclause 8.4.1.3 is invoked with mbPartIdx, subMbPartIdx, refIdxPredLX, and currSubMbType set equal to subMbType[CurrMbAddr][mbPartIdx] as the inputs and the output is assigned to mvPredLX. For this invocation of the process in subclause 8.4.1.3, the modifications specified in items a) through j) of this subclause apply

    3. mvCntInc is set equal to (mvCntInc + 1).

For X being replaced by 0 and 1, the arrays refIdxLX, predFlagLX, and mvLX are modified by applying the following ordered steps:

1. When subMbPartIdx is equal to 0, the arrays refIdxLX and predFlagLX are modified by

$$\text{refIdxLX[CurrMbAddr][mbPartIdx]} = \text{refIdxPredLX} \qquad \text{(G-96)}$$
$$\text{predFlagLX[CurrMbAddr][mbPartIdx]} = (\ (\ \text{refIdxPredLX} < 0\ )\ ?\ 0\ :\ 1\ ) \qquad \text{(G-97)}$$

2. The array mvLX is modified by

$$\text{mvLX[CurrMbAddr][mbPartIdx][subMbPartIdx][c]} = \text{mvPredLX[c]} \quad \text{with } c = 0..1 \qquad \text{(G-98)}$$

3. When predFlagLX[CurrMbAddr][mbPartIdx] is equal to 1, base_mode_flag is equal to 0, isDirectFlag is equal to 0, and mbType[CurrMbAddr] is not equal to P_Skip, the array mvLX is modified by

$$\text{mvLX[CurrMbAddr][mbPartIdx][subMbPartIdx][c]} \mathrel{+}= \text{mvd\_lX[mbPartIdx][subMbPartIdx][c]} \quad \text{with } c = 0..1 \qquad \text{(G-99)}$$

The array mvCnt is modified as follows:

– If mbPartIdx is equal to 0 and subMbPartIdx is equal to 0, mvCnt[CurrMbAddr] is set equal to mvCntInc.

– Otherwise (mbPartIdx is greater than 0 or subMbPartIdx is greater than 0), the array mvCnt is modified by

$$\text{mvCnt[CurrMbAddr]} \mathrel{+}= \text{mvCntInc} \qquad \text{(G-100)}$$

G.8.4.1.2 SVC derivation process for luma motion vectors and reference indices for B_Skip, B_Direct_16x16, and B_Direct_8x8 in NAL units with nal_unit_type equal to 20

Inputs to this process are:

– a variable mbPartIdx specifying the current macroblock partition,

– a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,

– a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,

– a (PicSizeInMbs)x4 array subMbType specifying sub-macroblock types for the macroblocks of the current layer representation,

– two (PicSizeInMbs)x4 arrays predFlagL0 and predFlagL1 specifying prediction utilization flags for the macroblocks of the current layer representation,

– two (PicSizeInMbs)x4 arrays refIdxL0 and refIdxL1 specifying reference indices for the macroblocks of the current layer representation,

– two (PicSizeInMbs)x4x4x2 arrays mvL0 and mvL1 specifying motion vector components for the macroblocks of the current layer representation.

Outputs of this process are:

– the reference index predictors refIdxPredL0 and refIdxPredL1,

– the motion vector predictors mvPredL0 and mvPredL1,

– the variable mvCntInc.

The variable currSubMbType is derived as follows:

– If mbType[CurrMbAddr] is equal to B_Skip or B_Direct_16x16, currSubMbType is marked as "unspecified".

– Otherwise (mbType[CurrMbAddr] is equal to B_8x8 and subMbType[CurrMbAddr][mbPartIdx] is equal to B_Direct_8x8), currSubMbType is set equal to B_Bi_8x8.

NOTE – The variable currSubMbType is only used for deriving the variable predPartWidth in subclause 6.4.11.7, which specifies the partition width of the current macroblock or sub-macroblock partition for determining neighbouring partitions that are used for motion vector prediction. Inside subclause 6.4.11.7, the variable predPartWidth is set equal to 16 when the current macroblock is coded with macroblock type equal to B_Skip or B_Direct_16x16 or the current sub-macroblock is coded with sub macroblock type equal B_Direct_8x8. When the current subclause is invoked for a sub-macroblock coded with sub-macroblock type equal to B_Direct_8x8 (the current subclause is only invoked for NAL units with nal_unit_type equal to 20), currSubMbType is set equal to B_Bi_8x8 in order to set the variable predPartWidth equal to 8 in subclause 6.4.11.7.

For X being replaced by 0 and 1, the reference index predictor refIdxPredLX is derived by applying the following ordered steps:

1. The derivation process for motion data of neighbouring partitions as specified in subclause 8.4.1.3.2 is invoked with mbPartIdx, subMbPartIdx set equal to 0, currSubMbType, and listSuffixFlag set equal to X as the inputs and the outputs are the reference indices refIdxLXN with N being replaced by A, B, and C. For this invocation of the process in subclause 8.4.1.3.2, the modifications specified in items a) through j) of subclause G.8.4.1.1 apply.

2. The reference index predictor refIdxPredLX is derived by

$$\text{refIdxPredLX} = \text{MinPositive}(\text{ refIdxLXA, MinPositive}(\text{ refIdxLXB, refIdxLXC })) \quad \text{(G-101)}$$

with

$$\text{MinPositive}(\text{ x, y }) = \begin{cases} \text{Min}(x, y) & \text{if } x >= 0 \text{ and } y >= 0 \\ \text{Max}(x, y) & \text{otherwise} \end{cases} \quad \text{(G-102)}$$

When both reference index predictors refIdxPredL0 and refIdxPredL1 are less than 0, refIdxPredL0 and refIdxPredL1 are set equal to 0.

For X being replaced by 0 and 1, the motion vector predictor mvPredLX is derived as follows:

– If refIdxPredLX is greater than or equal to 0, the derivation process for luma motion vector prediction as specified in subclause 8.4.1.3 is invoked with mbPartIdx, subMbPartIdx set equal to 0, refIdxPredLX, and currSubMbType as the inputs and the output is assigned to mvPredLX. For this invocation of the process in subclause 8.4.1.3, the modifications specified in items a) through j) of subclause G.8.4.1.1 apply.

– Otherwise, both components of the motion vector mvPredLX are set equal to 0.

The variable mvCntInc is derived as specified by the following ordered steps:

1. mvCntInc is set equal to 0

2. When refIdxPredL0 is greater than or equal to 0, mvCntInc is set equal to (mvCntInc + 1).

3. When refIdxPredL1 is greater than or equal to 0, mvCntInc is set equal to (mvCntInc + 1).

G.8.4.2 SVC decoding process for Inter prediction samples

Inputs to this process are:

– a variable targetQId specifying the quality_id value for the target layer representation,

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a variable sliceIdc specifying the slice identification for the current macroblock,

– a variable mbType specifying the macroblock type for the current macroblock,

– a list subMbType with 4 elements specifying the sub-macroblock types for the current macroblock,

– two lists predFlagL0 and predFlagL1 with 4 elements specifying prediction utilization flags for the current macroblock,

– two lists refIdxL0 and refIdxL1 with 4 elements specifying reference indices for the current macroblock,

– two 4x4x2 arrays mvL0 and mvL1 specifying motion vectors components for the current macroblock,

– when present, a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– when present, a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying macroblock types for the macroblocks of the reference layer representation,

– the reference picture lists refPicList0 and refPicList1 (when available),

– a $(\text{PicWidthInSamples}_L)\text{x}(\text{PicHeightInSamples}_L)$ array picSamples_L of luma sample values,

– a $(PicWidthInSamples_L)x(PicHeightInSamples_L)$ array $picRes_L$ of residual luma sample values,

– when ChromaArrayType is not equal to 0, two $(PicWidthInSamples_C)x(PicHeightInSamples_C)$ arrays $picSamples_{Cb}$ and $picSamples_{Cr}$ of chroma sample values,

– when ChromaArrayType is not equal to 0, two $(PicWidthInSamples_C)x(PicHeightInSamples_C)$ arrays $picRes_{Cb}$ and $picRes_{Cr}$ of residual chroma sample values.

Outputs of this process are:

– a modified version of the array of luma sample values $picSamples_L$,

– a modified version of the array of residual luma sample values $picRes_L$,

– when ChromaArrayType is not equal to 0, modified versions of the two arrays of chroma sample values $picSamples_{Cb}$ and $picSamples_{Cr}$,

– when ChromaArrayType is not equal to 0, modified versions of the two arrays of residual chroma sample values $picRes_{Cb}$ and $picRes_{Cr}$.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the current macroblock, which is the macroblock with address CurrMbAddr inside the layer representation with DQId equal to (sliceIdc & 127), the slice header of the current slice, which is the slice that contains the current macroblock, the current picture parameter, which is identified by the syntax element pic_parameter_set_id inside the slice header of the current slice, and the current sequence parameter, which is identified by the syntax element seq_parameter_set_id inside the current picture parameter set.

For all processes specified in clauses 6 or 8 that are invoked from the process specified in this subclause or a child process of the process specified in this subclause, the following modifications apply:

a) In subclauses 8.4.3, 8.4.1.4, and 8.4.2.1, the current macroblock is treated as field macroblock when fieldMbFlag is equal to 1, and it is treated as frame macroblock when fieldMbFlag is equal to 0. When field_pic_flag is equal to 0 and the current macroblock CurrMbAddr is a field macroblock, its parity is equal to top when (CurrMbAddr % 2) is equal to 0 and its parity is equal to bottom when (CurrMbAddr % 2) is equal to 1.

b) In subclauses 8.4.3 and 8.4.2.1, any occurrence of RefPicList0 or RefPicList1 is replaced with refPicList0 or refPicList1, respectively, with refPicList0 and refPicList1 being the reference picture lists specified as inputs to this subclause.

c) In subclause 8.4.1.4, the reference picture referred by refIdxLX is specified by refPicListX[refIdxLX] with refPicList0 and refPicList1 specified as inputs to this subclause.

d) In subclauses 8.4.2.2.1 and 8.4.2.2.2, any occurrence of mb_field_decoding_flag is replaced by fieldMbFlag.

e) Decoded pictures are represented by the sample arrays S_L and, when ChromaArrayType is not equal to 0, S_{Cb} and S_{Cr}, reference base pictures are represented by the sample arrays B_L and, when ChromaArrayType is not equal to 0, B_{Cb} and B_{Cr}. When reference base pictures are referenced in the inter prediction process via subclause 8.4.2.1, the samples arrays B_L, B_{Cb}, and B_{Cr} are referred to as S_L, S_{Cb}, and S_{Cr}, respectively. The sample arrays S_L, S_{Cb}, S_{Cr}, B_L, B_{Cb}, and B_{Cr} that referenced in the inter prediction process via subclause 8.4.2.1 are constructed as specified in subclause G.8.

Let $predMb_L$ be a 16x16 array of luma prediction samples for the macroblock mbAddr.

When ChromaArrayType is not equal to 0, let $predMb_{Cb}$ and $predMb_{Cr}$ be two (MbWidthC)x(MbHeightC) arrays of chroma prediction samples for the macroblock mbAddr.

The variable numMbPart is derived as follows:

– If mbType is equal to B_Skip or B_Direct_16x16 and DQId is equal to 0 (nal_unit_type is not equal to 20), numMbPart is set equal to 4.

– Otherwise, if mbType is equal to B_Skip or B_Direct_16x16 (and DQId is greater than 0 and nal_unit_type is equal to 20), numMbPart is set equal to 1.

– Otherwise (mbType is not equal to B_Skip or B_Direct_16x16), numMbPart is set equal to NumMbPart(mbType).

The macroblock partition index mbPartIdx proceeds over the values 0..(numMbPart − 1), and for each value of mbPartIdx the following ordered steps are specified:

1. The variable isDirectFlag is derived as follows:
   – If any of the following conditions are true, isDirectFlag is set equal to 1:
     – mbType is equal to B_Skip or B_Direct_16x16,
     – mbType is equal to B_8x8 and subMbType[mbPartIdx] is equal to B_Direct_8x8.
   – Otherwise, isDirectFlag is set equal to 0.
2. The variables implicitModeFlag and explicitModeFlag are derived as follows:
   – If weighted_bipred_idc is equal to 2, (slice_type % 5) is equal to 1, predFlagL0[mbPartIdx] is equal to 1, and predFlagL1[mbPartIdx] is equal to 1, implicitModeFlag is set equal to 1 and explicitModeFlag is set equal to 0.
   – Otherwise, if weighted_bipred_idc is equal to 1, (slice_type % 5) is equal to 1, and predFlagL0[mbPartIdx] + predFlagL1[mbPartIdx] is equal to 1 or 2, implicitModeFlag is set equal to 0 and explicitModeFlag is set equal to 1.
   – Otherwise, if weighted_pred_flag is equal to 1, (slice_type % 5) is equal to 0, and predFlagL0[mbPartIdx] is equal to 1, implicitModeFlag is set equal to 0 and explicitModeFlag is set equal to 1.
   – Otherwise, implicitModeFlag is set equal to 0 and explicitModeFlag is set equal to 0.
3. When implicitModeFlag is equal to 1 or explicitModeFlag is equal to 1, the SVC derivation process for prediction weights as specified in subclause G.8.4.2.1 is invoked with fieldMbFlag, refIdxL0[mbPartIdx], refIdxL1[mbPartIdx], predFlagL0[mbPartIdx], predFlagL1[mbPartIdx], refPicList0, and refPicList1 (when available) as inputs and the outputs are assigned to $logWD_L$, w_{0L}, w_{1L}, o_{0L}, o_{1L}, and when ChromaArrayType is not equal to 0, $logWD_C$, w_{0C}, w_{1C}, o_{0C}, o_{1C} with C being replaced by Cb and Cr.
4. The luma location (xP, yP) is derived as follows:
   – If mbType is equal to B_Skip or B_Direct_16x16, xP is set equal to (8 * (mbPartIdx % 2)) and yP is set equal to (8 * (mbPartIdx / 2)).
   – Otherwise (mbType is not equal to B_Skip or B_Direct_16x16), the inverse macroblock partition scanning process as specified in subclause 6.4.2.1 is invoked with mbPartIdx as the input and the output is assigned to (xP, yP). For this invocation of the process in subclause 6.4.2.1, any occurrence of mb_type is replaced by mbType.
5. The variable numSubMbPart is derived as follows:
   – If isDirectFlag is equal to 1 and DQId is equal to 0 (nal_unit_type is not equal to 20), numSubMbPart is set equal to 4.
   – Otherwise, if isDirectFlag is equal to 1 (and DQId is greater than 0 and nal_unit_type is equal to 20), numSubMbPart is set equal to 1.
   – Otherwise (isDirectFlag is equal to 0), numSubMbPart is set equal to NumSubMbPart(subMbType[mbPartIdx])
6. The sub-macroblock partition index proceeds over values 0..(numSubMbPart − 1), and for each value of subMbPartIdx the following ordered steps are specified:
   a. The variables partWidth and partHeight are derived as follows:
      – If isDirectFlag is equal to 1 and DQId is equal to 0 (nal_unit_type is not equal to 20), partWidth and partHeight are set equal to 4.
      – Otherwise, if isDirectFlag is equal to 1 (and DQId is greater than 0 and nal_unit_type is equal to 20), the following applies:
        – If mbType is equal to B_Skip or B_Direct_16x16, partWidth and partHeight are set equal to 16.
        – Otherwise (mbType is equal to B_8x8 and subMbType[mbPartIdx] is equal to B_Direct_8x8), partWidth and partHeight are set equal to 8.

– Otherwise (isDirectFlag is equal to 0), the following applies:

– If mbType is not equal to P_8x8, P_8x8ref0, or B_8x8, partWidth and partHeight are derived by

$$\text{partWidth} = \text{MbPartWidth(mbType)} \tag{G-103}$$
$$\text{partHeight} = \text{MbPartHeight(mbType)} \tag{G-104}$$

– Otherwise (mbType is equal to P_8x8, P_8x8ref0, or B_8x8), partWidth and partHeight are derived by

$$\text{partWidth} = \text{SubMbPartWidth(subMbType[mbPartIdx])} \tag{G-105}$$
$$\text{partHeight} = \text{SubMbPartHeight(subMbType[mbPartIdx])} \tag{G-106}$$

b. When ChromaArrayType is not equal to 0, the variables partWidthC and partHeightC are derived by

$$\text{partWidthC} = \text{partWidth / SubWidthC} \tag{G-107}$$
$$\text{partHeightC} = \text{partHeight / SubWidthC} \tag{G-108}$$

c. For X being replaced by 0 and 1, when ChromaArrayType is not equal to 0 and predFlagLX[mbPartIdx] is equal to 1, the derivation process for chroma motion vectors as specified in subclause 8.4.1.4 is invoked with mvLX[mbPartIdx][subMbPartIdx] and refIdxLX[mbPartIdx] as the inputs and the output is the chroma motion vector mvCLX. For this invocation of the process in subclause 8.4.1.4, the modifications specified above in items a) and c) of this subclause apply.

d. The decoding process for Inter prediction samples as specified in subclause 8.4.2 is invoked with mbPartIdx, subMbPartIdx, partWidth and partHeight, partWidthC and partHeightC (if available), luma motion vectors mvL0[mbPartIdx][subMbPartIdx] and mvL1[mbPartIdx][subMbPartIdx], chroma motion vectors mvCL0 and mvCL1 (if available), reference indices refIdxL0[mbPartIdx] and refIdxL1[mbPartIdx], prediction utilization flags predFlagL0[mbPartIdx] and predFlagL1[mbPartIdx] as well as variables for weighted prediction logWD_L, w_{0L}, w_{1L}, o_{1L}, o_{0L}, and when ChromaArrayType is not equal to 0, logWD_C, w_{0C}, w_{1C}, o_{1C}, and o_{0C} (with C being replaced by Cb and Cr) as the inputs and the outputs are a (partWidth)x(partHeight) array predPart_L of luma prediction samples and, when ChromaArrayType is not equal to 0, two (partWidthC)x(partHeightC) arrays predPart_{Cb} and predPart_{Cr} of chroma prediction samples. For this invocation of the process in subclause 8.4.2, the modifications specified above in items a), b), d), and e) of this subclause apply.

e. The luma location (xS, yS) is derived as follows:

– If mbType is equal to B_8x8 and subMbType[mbPartIdx] is equal to B_Direct_8x8, xS is set equal to (4 * (subMbPartIdx % 2)) and yS is set equal to (4 * (subMbPartIdx / 2)).

– Otherwise (mbType is not equal to B_8x8 or subMbType[mbPartIdx] is not equal to B_Direct_8x8), the inverse sub-macroblock partition scanning process as specified in subclause 6.4.2.2 is invoked with mbPartIdx and subMbPartIdx as the inputs and the output is assigned to (xS, yS). For this invocation of the process in subclause 6.4.2.2, any occurrence of mb_type is replaced by mbType and any occurrence of sub_mb_type is replaced by subMbType.

f. For x = 0..(partWidth − 1) and y = 0..(partHeight − 1), the 16x16 array predMb_L is modified by

$$\text{predMb}_L[\ xP + xS + x,\ yP + yS + y\ ] = \text{predPart}_L[\ x, y\ ] \tag{G-109}$$

g. When ChromaArrayType is not equal to 0, for x = 0..(partWidthC − 1) and y = 0..(partHeightC − 1), the (MbWidthC)x(MbHeightC) arrays predMb_{Cb} and predMb_{Cr} are modified by

$$\text{predMb}_{Cb}[\ (xP + xS) / \text{SubWidthC} + x,\ (yP + yS) / \text{SubHeightC} + y\ ] = \text{predPart}_{Cb}[\ x, y\ ] \tag{G-110}$$
$$\text{predMb}_{Cr}[\ (xP + xS) / \text{SubWidthC} + x,\ (yP + yS) / \text{SubHeightC} + y\ ] = \text{predPart}_{Cr}[\ x, y\ ] \tag{G-111}$$

When targetQId is equal to 0, base_mode_flag is equal to 1, MbaffFrameFlag is equal to 0, RefLayerMbaffFrameFlag is equal to 0, and RestrictedSpatialResolutionChangeFlag is equal to 0, the intra-inter prediction combination process specified in subclause G.8.4.2.2 is invoked with fieldMbFlag, refLayerFieldMbFlag, refLayerMbType, predMb_L, picSamples_L, picRes_L, and, when ChromaArrayType is not equal to 0, predMb_{Cb}, predMb_{Cr}, picSamples_{Cb}, picSamples_{Cr}, picRes_{Cb}, and picRes_{Cr} as the inputs, and the outputs are modified versions of predMb_L and picRes_L, and, when ChromaArrayType is not equal to 0, modified versions of predMb_{Cb}, predMb_{Cr}, picRes_{Cb}, and picRes_{Cr}.

The picture sample array construction process as specified in subclause G.8.5.4.1 is invoked with fieldMbFlag, predMb_L, picSamples_L, and, when ChromaArrayType is not equal to 0, predMb_{Cb}, predMb_{Cr}, picSamples_{Cb}, and picSamples_{Cr} as the inputs and the outputs are a modified version of picSamples_L and, when ChromaArrayType is not equal to 0, modified versions of picSamples_{Cb}, and picSamples_{Cr}.

G.8.4.2.1 SVC derivation process for prediction weights

Inputs to this process are:

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– the reference indices refIdxL0 and refIdxL1 for the current macroblock partition,

– the prediction list utilization flags predFlagL0 and predFlagL1 for the current macroblock partition,

– the reference picture lists refPicList0 and refPicList1 (when available).

Outputs of this process are:

– variables for weighted prediction of luma samples $logWD_L$, w_{0L}, w_{1L}, o_{0L}, o_{1L},

– when ChromaArrayType is not equal to 0 (monochrome), variables for weighted prediction of chroma samples $logWD_C$, w_{0C}, w_{1C}, o_{0C}, o_{1C} with C being replaced by Cb and Cr.

Depending on base_pred_weight_table_flag, the following applies:

– If base_pred_weight_table_flag is equal to 0, the derivation process for prediction weights as specified in subclause 8.4.3 is invoked with refIdxL0, refIdxL1, predFlagL0, and predFlagL1 as inputs and the outputs are assigned to $logWD_L$, w_{0L}, w_{1L}, o_{0L}, o_{1L}, and when ChromaArrayType is not equal to 0, $logWD_C$, w_{0C}, w_{1C}, o_{0C}, o_{1C} with C being replaced by Cb and Cr. For this invocation of the process in subclause 8.4.3, the modifications specified in items a) and b) of subclause G.8.4.2 apply.

– Otherwise (base_pred_weight_table_flag is equal to 1), for X being replaced by 0 and 1, the following ordered steps are specified:

  1. Let dqIdList be the list of DQId values that is derived by invoking the derivation process for the set of layer representations required for decoding as specified in subclause G.8.1.1. Let baseDQId be the largest value dqId, inside the list dqIdList, that has the following properties: a) dqId is less than the current value of DQId; b) the slices of the layer representation with DQId equal to dqId have base_pred_weight_table_flag equal to 0.

  2. Let baseSlice be any slice of the layer representation with DQId equal to baseDQId.

  3. Let refLayerLumaLogWD, aRefLayerLumaWeightLX[], and aRefLayerLumaOffsetLX[] be variables that are set equal to the values of the syntax elements luma_log2_weight_denom, luma_weight_lX[], and luma_offset_lX[], respectively, of baseSlice.

  4. When ChromaArrayType is not equal to 0, let refLayerChromaLogWD, aRefLayerChromaWeightLX[][], and aRefLayerChromaOffsetLX[][] be variables that are set equal to the values of the syntax elements chroma_log2_weight_denom, chroma_weight_lX[], and chroma_offset_lX[], respectively, of baseSlice.

  5. The variable refIdxLXWP is derived as follows:

     – If MbaffFrame is equal to 1 and fieldMbFlag is equal to 1,

     $$\text{refIdxLXWP} = \text{refIdxLX} >> 1 \qquad \text{(G-112)}$$

     – Otherwise (MbaffFrameFlag is equal to 0 or fieldMbFlag is equal to 0),

     $$\text{refIdxLXWP} = \text{refIdxLX} \qquad \text{(G-113)}$$

  6. The variables $logWD_L$, w_{XL}, o_{XL} are derived by:

     $$logWD_L = \text{refLayerLumaLogWD} \qquad \text{(G-114)}$$
     $$w_{XL} = \text{aRefLayerLumaWeightLX[refIdxLXWP]} \qquad \text{(G-115)}$$
     $$o_{XL} = \text{aRefLayerLumaOffsetLX[refIdxLXWP]} * (1 << (BitDepth_Y - 8)) \qquad \text{(G-116)}$$

  7. When ChromaArrayType is not equal to 0, the variables $logWD_C$, w_{XC}, o_{XC} (with C being replaced by Cb and Cr and iCbCr = 0 for Cb and iCbCr = 1 for Cr) are derived by:

     $$logWD_C = \text{refLayerChromaLogWD} \qquad \text{(G-117)}$$
     $$w_{XC} = \text{aRefLayerChromaWeightLX[refIdxLXWP][iCbCr]} \qquad \text{(G-118)}$$
     $$o_{XC} = \text{aRefLayerChromaOffsetLX[refIdxLXWP][iCbCr]} * (1 << (BitDepth_C - 8)) \qquad \text{(G-119)}$$

8. When predFlagL0 and predFlagL1 are equal to 1, the following constraint shall be obeyed for C equal to L and, when ChromaArrayType is not equal to 0, Cb and Cr

$$-128 <= w_{0C} + w_{1C} <= ((logWD_C \; == \; 7) \; ? \; 127 : 128) \quad \text{(G-120)}$$

G.8.4.2.2 Intra-inter prediction combination process

Inputs to this process are:

- a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,
- a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,
- a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying macroblock types for the macroblocks of the reference layer representation,
- a 16x16 array $predMb_L$ of luma inter prediction samples for the current macroblock,
- a ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array $picSamples_L$ of luma sample values,
- a ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array $picRes_L$ of residual luma sample values,
- when ChromaArrayType is not equal to 0, two (MbWidthC)x(MbHeightC) arrays $predMb_{Cb}$ and $predMb_{Cr}$ of chroma prediction samples for the macroblock mbAddr,
- when ChromaArrayType is not equal to 0, two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $picSamples_{Cb}$ and $picSamples_{Cr}$ of chroma sample values,
- when ChromaArrayType is not equal to 0, two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $picRes_{Cb}$ and $picRes_{Cr}$ of residual chroma sample values.

Outputs of this process are:

- a modified version of the array $predMb_L$ of luma prediction samples for the macroblock mbAddr,
- a modified version of the array $picRes_L$ of residual luma sample values,
- when ChromaArrayType is not equal to 0, modified versions of the two arrays $predMb_{Cb}$ and $predMb_{Cr}$ of chroma prediction samples for the macroblock mbAddr,
- when ChromaArrayType is not equal to 0, modified versions of the two arrays $picRes_{Cb}$ and $picRes_{Cr}$ of residual chroma sample values.

Let $predMbTemp_L$ be a 16x16 array and, when ChromaArrayType is not equal to 0, let $predMbTemp_{Cb}$ and $predMbTemp_{Cr}$ be two (MbWidthC)x(MbHeightC) arrays. The macroblock sample array extraction process as specified in subclause G.8.5.4.2 is invoked with fieldMbFlag, $picSamples_L$, and when ChromaArrayType is not equal to 0, $picSamples_{Cb}$ and $picSamples_{Cr}$ as the inputs and the outputs are assigned to $predMbTemp_L$, and when ChromaArrayType is not equal to 0, $predMbTemp_{Cb}$ and $predMbTemp_{Cr}$.

Let $resMb_L$ be a 16x16 array and, when ChromaArrayType is not equal to 0, let $resMb_{Cb}$ and $resMb_{Cr}$ be two (MbWidthC)x(MbHeightC) arrays. The macroblock sample array extraction process as specified in subclause G.8.5.4.2 is invoked with fieldMbFlag, $picRes_L$, and, when ChromaArrayType is not equal to 0, $picRes_{Cb}$ and $picRes_{Cr}$ as the inputs and the outputs are assigned to $resMb_L$ and, when ChromaArrayType is not equal to 0, $resMb_{Cb}$ and $resMb_{Cr}$.

For x proceeding over the values 0..15 and y proceeding over the values 0..15, the following ordered steps are specified:

1. The derivation process for reference layer macroblocks as specified in subclause G.6.1 is invoked with the luma location (x, y), fieldMbFlag, refLayerFieldMbFlag, and refLayerMbType as the inputs and the outputs are assigned to mbAddrRefLayer and (xRef, yRef).
2. When refLayerMbType[mbAddrRefLayer] is equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL, the following applies:
   a. The prediction luma sample $predMb_L$[x, y] is modified by

$$predMb_L[\, x, y \,] = predMbTemp_L[\, x, y \,] \quad \text{(G-121)}$$

b. When ChromaArrayType is not equal to 0, (x % SubWidthC) is equal to 0, and (y % SubHeightC) is equal to 0, the prediction chroma samples $predMb_{Cb}$[x / SubWidthC, y / SubHeightC] and $predMb_{Cr}$[x / SubWidthC, y / SubHeightC] are modified by

$$predMb_{Cb}[\ x / SubWidthC, y / SubHeightC\ ] = predMbTemp_{Cb}[\ x / SubWidthC, y / SubHeightC\ ] \quad \text{(G-122)}$$

$$predMb_{Cr}[\ x / SubWidthC, y / SubHeightC\ ] = predMbTemp_{Cr}[\ x / SubWidthC, y / SubHeightC\ ] \quad \text{(G-123)}$$

c. The residual luma sample $resMb_L$[x, y] is set equal to 0.

d. When ChromaArrayType is not equal to 0, (x % SubWidthC) is equal to 0, and (y % SubHeightC) is equal to 0, the residual chroma samples $resMb_{Cb}$[x / SubWidthC, y / SubHeightC] and $resMb_{Cr}$[x / SubWidthC, y / SubHeightC] are set equal to 0.

The picture sample array construction process as specified in subclause G.8.5.4.1 is invoked with fieldMbFlag, $resMb_L$, $picRes_L$, and, when ChromaArrayType is not equal to 0, $resMb_{Cb}$, $resMb_{Cr}$, $picRes_{Cb}$, and $picRes_{Cr}$ as the inputs and the outputs are a modified version of the array $picRes_L$ and, when ChromaArrayType is not equal to 0, modified versions of the arrays $picRes_{Cb}$ and $picRes_{Cr}$.

G.8.5 SVC transform coefficient decoding and sample array construction processes

Subclause G.8.5.1 specifies the transform coefficient scaling and refinement process.

Subclause G.8.5.2 specifies the transform coefficient level scaling process prior to transform coefficient refinement.

Subclause G.8.5.3 specifies the residual construction and accumulation process.

Subclause G.8.5.4 specifies the sample array accumulation process.

Subclause G.8.5.5 specifies the sample array re-initialisation process.

G.8.5.1 Transform coefficient scaling and refinement process

Inputs to this process are:

– a variable refinementFlag specifying whether the transform coefficients for the current macroblock are combined with the existent transform coefficients for the current macroblock, which were obtained from the reference layer representation,

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a variable mbType specifying the macroblock type for the current macroblock,

– a variable cTrafo specifying the transform type for the current macroblock,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,

– a list of transform coefficient level values tCoeffLevel with (256 + 2 * MbWidthC * MbHeightC) elements,

– the luma quantisation parameter tQP_Y,

– when ChromaArrayType is not equal to 0, the chroma quantisation parameters tQP_{Cb} and tQP_{Cr}.

Outputs of this process are:

– a modified version of the list sTCoeff,

– a modified version of the list tCoeffLevel.

The scaling functions are derived as specified in subclause 8.5.9. For this invocation of subclause 8.5.9, the current macroblock is considered as coded using an Intra macroblock prediction mode when mbType is equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL; otherwise it is considered as coded using an Inter macroblock prediction mode.

The variable qP'_Y is set equal to (tQP_Y + $QpBdOffset_Y$). When ChromaArrayType is not equal to 0, the variables qP'_{Cb} and qP'_{Cr} are set equal to (tQP_{Cb} + $QpBdOffset_C$) and (tQP_{Cr} + $QpBdOffset_C$), respectively.

When refinementFlag is equal to 0, all (256 + 2 * MbWidthC * MbHeightC) elements of the lists sTCoeff and tCoeffLevel are set equal to 0.

The refinement process for luma transform coefficients as specified in subclause G.8.5.1.1 is invoked with iYCbCr set equal to 0, fieldMbFlag, cTrafo, sTCoeff, tCoeffLevel, and qP'_Y as the inputs and the outputs are modified versions of the lists sTCoeff and tCoeffLevel.

When ChromaArrayType is not equal to 0, the refinement process for chroma transform coefficients as specified in subclause G.8.5.1.2 is invoked with fieldMbFlag, cTrafo, sTCoeff, tCoeffLevel, qP'_{Cb}, and qP'_{Cr} as the inputs and the outputs are modified versions of the lists sTCoeff and tCoeffLevel.

G.8.5.1.1 Refinement process for luma transform coefficients or chroma transform coefficients with ChromaArrayType equal to 3

Inputs to this process are:

– a variable iYCbCr specifying the colour component,
– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,
– a variable cTrafo specifying the transform type,
– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,
– a list of transform coefficient level values tCoeffLevel with (256 + 2 * MbWidthC * MbHeightC) elements,
– when iYCbCr is equal to 0, the luma quantisation parameter qP'_Y,
– when iYCbCr is greater than 0, the chroma quantisation parameters qP'_{Cb} and qP'_{Cr}.

Outputs of this process are:

– a modified version of the list sTCoeff,
– a modified version of the list tCoeffLevel.

When iYCbCr is not present as input to this subclause, it is inferred to be equal to 0.

Depending on iYCbCr, the variables bitDepth, qP, cO, coeffLevel4x4, coeffLevel8x8, coeffDCLevel, and coeffACLevel are derived as follows:

– If iYCbCr is equal to 0, bitDepth is set equal to $BitDepth_Y$, qP is set equal to qP'_Y, cO is set equal to 0, coeffLevel4x4 is set equal to LumaLevel4x4, coeffLevel8x8 is set equal to LumaLevel8x8, coeffDCLevel is set equal to Intra16x16DCLevel, and coeffACLevel is set equal to Intra16x16ACLevel.
– Otherwise, if iYCbCr is equal to 1, bitDepth is set equal to $BitDepth_C$, qP is set equal to qP'_{Cb}, cO is set equal to 256, coeffLevel4x4 is set equal to CbLevel4x4, coeffLevel8x8 is set equal to CbLevel8x8, coeffDCLevel is set equal to CbIntra16x16DCLevel, and coeffACLevel is set equal to CbIntra16x16ACLevel.
– Otherwise (iYCbCr is equal to 2), bitDepth is set equal to $BitDepth_C$, qP is set equal to qP'_{Cr}, cO is set equal to (256 + MbWidthC * MbHeightC), coeffLevel4x4 is set equal to CrLevel4x4, coeffLevel8x8 is set equal to CrLevel8x8, coeffDCLevel is set equal to CrIntra16x16DCLevel, and coeffACLevel is set equal to CrIntra16x16ACLevel.

Depending on cTrafo, the following applies:

– If cTrafo is equal to T_PCM, the assignment process for luma transform coefficient values or chroma transform coefficient values with ChromaArrayType equal to 3 for I_PCM macroblocks as specified in subclause G.8.5.1.1.1 is invoked with iYCbCr, sTCoeff, and tCoeffLevel as the inputs and the outputs are modified versions of sTCoeff and tCoeffLevel.
– Otherwise, if cTrafo is equal to T_4x4, the refinement process for transform coefficients of residual 4x4 blocks as specified in subclause G.8.5.1.1.2 is invoked with fieldMbFlag, bitDepth, qP, cO, coeffLevel4x4, sTCoeff, and tCoeffLevel as the inputs and the outputs are modified versions of the lists sTCoeff and tCoeffLevel.
– Otherwise, if cTrafo is equal to T_8x8, the refinement process for transform coefficients of residual 8x8 blocks as specified in subclause G.8.5.1.1.3 is invoked with fieldMbFlag, bitDepth, qP, cO, coeffLevel8x8, sTCoeff, and tCoeffLevel as the inputs and the outputs are modified versions of the lists sTCoeff and tCoeffLevel.
– Otherwise (cTrafo is equal to T_16x16), the refinement process for transform coefficients of Intra_16x16 macroblocks as specified in subclause G.8.5.1.1.4 is invoked with fieldMbFlag, bitDepth, qP, cO, coeffDCLevel, coeffACLevel, coeffLevel4x4, sTCoeff, and tCoeffLevel as the inputs and the outputs are modified versions of the lists sTCoeff and tCoeffLevel.

G.8.5.1.1.1 Assignment process for luma transform coefficient values or chroma transform coefficient values with ChromaArrayType equal to 3 for I_PCM macroblocks

Inputs to this process are:

– a variable iYCbCr specifying the colour component,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,

– a list of transform coefficient level values tCoeffLevel with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are:

– a modified version of the list sTCoeff,

– a modified version of the list tCoeffLevel.

When base_mode_flag is equal to 0, the following ordered steps are specified:

1. Depending on iYCbCr, the variables cO, cListOffset and pcmSample are derived by

$$\text{cO} = \text{iYCbCr} * 256$$
$$\text{cListOffset} = ((\text{iYCbCr} == 0) \ ? \ 0 : (\text{iYCbCr} - 1) * 256) \quad \text{(G-124)}$$
$$\text{pcmSample} = ((\text{iYCbCr} == 0) \ ? \ \text{pcm\_sample\_luma} : \text{pcm\_sample\_chroma}) \quad \text{(G-125)}$$

2. The lists tCoeffLevel and sTCoeff are modified by

$$\text{tCoeffLevel[cO + k]} = 0 \qquad \text{with } k = 0..255 \quad \text{(G-126)}$$

$$\text{sTCoeff[cO + k]} = \text{pcmSample[cListOffset + k]} \qquad \text{with } k = 0..255 \quad \text{(G-127)}$$

G.8.5.1.1.2 Refinement process for transform coefficients of residual 4x4 blocks

Inputs to this process are:

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a variable bitDepth specifying the bit depth,

– a variable qP specifying the quantisation parameter value,

– a variable cO specifying the first coefficient index in the list of scaled transform coefficient values sTCoeff and in the list of transform coefficient values tCoeffLevel,

– a variable coeffLevel4x4 representing LumaLevel4x4, CbLevel4x4, or CrLevel4x4,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,

– a list of transform coefficient level values tCoeffLevel with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are:

– a modified version of the list sTCoeff,

– a modified version of the list tCoeffLevel.

Depending on tcoeff_level_prediction_flag, the following applies:

– If tcoeff_level_prediction_flag is equal to 1, the list sTCoeff is modified by

$$\text{sTCoeff[cO + k]} = 0 \qquad \text{with } k = 0..255 \quad \text{(G-128)}$$

– Otherwise (tcoeff_level_prediction_flag is equal to 0), the list tCoeffLevel is modified by

$$\text{tCoeffLevel[cO + k]} = 0 \qquad \text{with } k = 0..255 \quad \text{(G-129)}$$

For each residual 4x4 block indexed by c4x4BlkIdx = 0..15, the following ordered steps are specified:

1. The inverse scanning process for 4x4 transform coefficients and scaling lists as specified in subclause 8.5.6 is invoked with coeffLevel4x4[c4x4BlkIdx] as the input and the outputs are transform coefficient level values as a 4x4 array c with elements c_{ij}. For this invocation of the process in subclause 8.5.6, the current macroblock is treated as field macroblock when fieldMbFlag is equal to 1, and it is treated as frame macroblock when fieldMbFlag is equal to 0.

2. The list tCoeffLevel and the 4x4 array c are modified by

$$\text{tCoeffLevel[cO + 16 * c4x4BlkIdx + 4 * i + j]} \mathrel{+}= c_{ij} \qquad \text{with } i, j = 0..3 \quad \text{(G-130)}$$

c_{ij} = tCoeffLevel[cO + 16 * c4x4BlkIdx + 4 * i + j]   with i, j = 0..3   (G-131)

3. The scaling process for residual 4x4 blocks as specified in subclause 8.5.12.1 is invoked with bitDepth, qP, and the 4x4 array c as the inputs and the outputs are scaled transform coefficient values as a 4x4 array d with elements d_{ij}. For this invocation of the process in subclause 8.5.12.1, the array c is treated as not relating to a luma residual block coded using the Intra_16x16 macroblock prediction mode and as not relating to a chroma residual block.

4. The list sTCoeff is modified by

   sTCoeff[cO + 16 * c4x4BlkIdx + 4 * i + j] += d_{ij} with i, j = 0..3   (G-132)

The bitstream shall not contain data that result in any element sTCoeff[cO + k] with k = 0..255 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

The bitstream shall not contain data that result in any element tCoeffLevel[cO + k] with k = 0..255 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

G.8.5.1.1.3 Refinement process for transform coefficients of residual 8x8 blocks

Inputs to this process are:

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a variable bitDepth specifying the bit depth,

– a variable qP specifying the quantisation parameter value,

– a variable cO specifying the first coefficient index in the list of scaled transform coefficient values sTCoeff and in the list of transform coefficient values tCoeffLevel,

– a variable coeffLevel8x8 representing LumaLevel8x8, CbLevel8x8, or CrLevel8x8,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,

– a list of transform coefficient level values tCoeffLevel with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are:

– a modified version of the list sTCoeff,

– a modified version of the list tCoeffLevel.

Depending on tcoeff_level_prediction_flag, the following applies:

– If tcoeff_level_prediction_flag is equal to 1, the list sTCoeff is modified by

  sTCoeff[cO + k] = 0   with k = 0..255   (G-133)

– Otherwise (tcoeff_level_prediction_flag is equal to 0), the list tCoeffLevel is modified by

  tCoeffLevel[cO + k] = 0   with k = 0..255   (G-134)

For each residual 8x8 block indexed by c8x8BlkIdx = 0..3, the following ordered steps are specified:

1. The inverse scanning process for 8x8 transform coefficients and scaling lists as specified in subclause 8.5.7 is invoked with coeffLevel8x8[c8x8BlkIdx] as the input and the outputs are transform coefficient level values as an 8x8 array c with elements c_{ij}. For this invocation of the process in subclause 8.5.7, the current macroblock is treated as field macroblock when fieldMbFlag is equal to 1, and it is treated as frame macroblock when fieldMbFlag is equal to 0.

2. The list tCoeffLevel and the 8x8 array c are modified by

   tCoeffLevel[cO + 64 * c8x8BlkIdx + 8 * i + j] += c_{ij} with i, j = 0..7   (G-135)

   c_{ij} = tCoeffLevel[cO + 64 * c8x8BlkIdx + 8 * i + j]   with i, j = 0..7   (G-136)

3. The scaling process for residual 8x8 blocks as specified in subclause 8.5.13.1 is invoked with bitDepth, qP, and the 8x8 array c as the inputs and the outputs are scaled transform coefficient values as an 8x8 array d with elements d_{ij}.

4. The list sTCoeff is modified by

   sTCoeff[cO + 64 * c8x8BlkIdx + 8 * i + j] += d_{ij} with i, j = 0..7   (G-137)

The bitstream shall not contain data that result in any element sTCoeff[cO + k] with k = 0..255 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

The bitstream shall not contain data that result in any element tCoeffLevel[cO + k] with k = 0..255 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

G.8.5.1.1.4 Refinement process for transform coefficients of Intra_16x16 macroblocks

This process is only invoked when base_mode_flag is equal to 0 or tcoeff_level_prediction_flag is equal to 1.

Inputs to this process are:

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a variable bitDepth specifying the bit depth,

– a variable qP specifying the quantisation parameter value,

– a variable cO specifying the first coefficient index in the list of scaled transform coefficient values sTCoeff and in the list of transform coefficient values tCoeffLevel,

– a variable coeffDCLevel representing Intra16x16DCLevel, CbIntra16x16DCLevel, or CrIntra16x16DCLevel,

– a variable coeffACLevel representing Intra16x16ACLevel, CbIntra16x16ACLevel, or CrIntra16x16ACLevel,

– a variable coeffLevel4x4 representing LumaLevel4x4, CbLevel4x4, or CrLevel4x4,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,

– a list of transform coefficient level values tCoeffLevel with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are:

– a modified version of the list sTCoeff,

– a modified version of the list tCoeffLevel.

NOTE 1 – When tcoeff_level_prediction_flag is equal to 0, this subclause is always invoked as part of an invocation of subclause G.8.5.1 with refinementFlag equal to 0, in which case all elements of the list tCoeffLevel are set equal to 0 before invoking this subclause.

For the DC transform coefficients of all residual 4x4 blocks, the following ordered steps are specified:

1. Depending on base_mode_flag, the 4x4 array c with elements c_{ij} is derived as follows:

   – If base_mode_flag is equal to 0, the inverse scanning process for 4x4 transform coefficients and scaling lists as specified in subclause 8.5.6 is invoked with coeffDCLevel as the input and the outputs are DC transform coefficient level values for all residual 4x4 blocks as a 4x4 array c with elements c_{ij}. For this invocation of the process in subclause 8.5.6, the current macroblock is treated as field macroblock when fieldMbFlag is equal to 1, and it is treated as frame macroblock when fieldMbFlag is equal to 0.

   – Otherwise (base_mode_flag is equal to 1), the 4x4 array c with elements c_{ij} containing DC transform coefficient level values is derived by

   $$c_{ij} = \text{coeffLevel4x4}[\ 8 * (\ i / 2\ ) + 4 * (\ j / 2\ ) + 2 * (\ i \% 2\ ) + (\ j \% 2\ )\ ][\ 0\ ] \quad \text{with } i, j = 0..3 \qquad \text{(G-138)}$$

2. The list tCoeffLevel and the 4x4 array c are modified by

   $$\text{tCoeffLevel}[\ cO + 128 * (\ i / 2\ ) + 64 * (\ j / 2\ ) + 32 * (\ i \% 2\ ) + 16 * (\ j \% 2\ )\ ] \mathrel{+}= c_{ij} \quad \text{with } i, j = 0..3 \qquad \text{(G-139)}$$

   $$c_{ij} = \text{tCoeffLevel}[\ cO + 128 * (\ i / 2\ ) + 64 * (\ j / 2\ ) + 32 * (\ i \% 2\ ) + 16 * (\ j \% 2\ )\ ] \quad \text{with } i, j = 0..3 \qquad \text{(G-140)}$$

3. The scaling and transformation process for DC transform coefficients for Intra_16x16 macroblock type as specified in subclause 8.5.10 is invoked with bitDepth, qP, and c as the inputs and the output is the 4x4 array d with elements d_{ij} representing scaled DC transform coefficient values for all residual 4x4 blocks.

4. The list sTCoeff is modified by

   $$\text{sTCoeff}[\ cO + 128 * (\ i / 2\ ) + 64 * (\ j / 2\ ) + 32 * (\ i \% 2\ ) + 16 * (\ j \% 2\ )\ ] = d_{ij} \quad \text{with } i, j = 0..3 \qquad \text{(G-141)}$$

For each residual 4x4 block indexed by c4x4BlkIdx = 0..15, the following ordered steps are specified:

1. Depending on base_mode_flag, the variable c4x4List, which is a list of 16 entries, is derived as follows:
   - If base_mode_flag is equal to 0, the following applies:

     c4x4List[k] = ((k == 0) ? 0 : coeffACLevel[c4x4BlkIdx][k − 1])   with k = 0..15   (G-142)

   - Otherwise (base_mode_flag is equal to 1), the following applies:

     c4x4List[k] = ((k == 0) ? 0 : coeffLevel4x4[c4x4BlkIdx][k])   with k = 0..15   (G-143)

2. The inverse scanning process for 4x4 transform coefficients and scaling lists as specified in subclause 8.5.6 is invoked with c4x4List as the input and the outputs are transform coefficient level values as a 4x4 array e with elements e_{ij}. For this invocation of the process in subclause 8.5.6, the current macroblock is treated as field macroblock when fieldMbFlag is equal to 1, and it is treated as frame macroblock when fieldMbFlag is equal to 0.

3. The list tCoeffLevel and the 4x4 array e are modified by

   tCoeffLevel[cO + 16 * c4x4BlkIdx + 4 * i + j] += e_{ij} with i, j = 0..3 and i + j > 0   (G-144)

   e_{ij} = tCoeffLevel[cO + 16 * c4x4BlkIdx + 4 * i + j]   with i, j = 0..3 and i + j > 0   (G-145)

4. The scaling process for residual 4x4 blocks as specified in subclause 8.5.12.1 is invoked with bitDepth, qP, and the 4x4 array e as the inputs and the outputs are scaled transform coefficient values as a 4x4 array d with elements d_{ij}. During the process in subclause 8.5.12.1, the array e is treated as relating to a luma residual block coded using the Intra_16x16 macroblock prediction mode.

5. The list sTCoeff is modified by

   sTCoeff[cO + 16 * c4x4BlkIdx + 4 * i + j] = d_{ij} with i, j = 0..3 and i + j > 0   (G-146)

   NOTE 2 – The elements tCoeffLevel[cO + 16* c4x4BlkIdx] and sTCoeff[cO + 16* c4x4BlkIdx] are not modified during the process for a residual 4x4 block with index c4x4BlkIdx.

The bitstream shall not contain data that result in any element tCoeffLevel[cO + k] with k = 0..255 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

G.8.5.1.2 Refinement process for chroma transform coefficients

Inputs to this process are:

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a variable cTrafo specifying the transform type,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,

– a list of transform coefficient level values tCoeffLevel with (256 + 2 * MbWidthC * MbHeightC) elements,

– the chroma quantisation parameters qP'_{Cb} and qP'_{Cr}.

Outputs of this process are:

– a modified version of the list sTCoeff,

– a modified version of the list tCoeffLevel.

For both chroma components indexed by iCbCr = 0..1, the following applies:

– If ChromaArrayType is equal to 1 or 2, the following applies:

  – If cTrafo is equal to T_PCM, the assignment process for chroma transform coefficient values for I_PCM macroblocks as specified in subclause G.8.5.1.2.1 is invoked with iCbCr, sTCoeff, and tCoeffLevel as the inputs and the outputs are modified versions of sTCoeff and tCoeffLevel.

  – Otherwise (cTrafo is not equal to T_PCM), the refinement process for chroma transform coefficients with ChromaArrayType equal to 1 or 2 as specified in subclause G.8.5.1.2.2 is invoked with iCbCr, fieldMbFlag, sTCoeff, tCoeffLevel, qP'_{Cb}, and qP'_{Cr} as the inputs and the outputs are modified versions of the lists sTCoeff and tCoeffLevel.

– Otherwise (ChromaArrayType is equal to 3), the refinement process for luma transform coefficients or chroma transform coefficients with ChromaArrayType equal to 3 as specified in subclause G.8.5.1.1 is invoked with iYCbCr set equal to (1 + iCbCr), fieldMbFlag, cTrafo, sTCoeff, tCoeffLevel, qP'_{Cb}, and qP'_{Cr} as the inputs and the outputs are modified versions of the lists sTCoeff and tCoeffLevel.

G.8.5.1.2.1 Assignment process for chroma transform coefficient values for I_PCM macroblocks

Inputs to this process are:

– a variable iCbCr specifying the chroma component,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,

– a list of transform coefficient level values tCoeffLevel with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are:

– a modified version of the list sTCoeff,

– a modified version of the list tCoeffLevel.

The variable numC is set equal to (MbWidthC * MbHeightC) and the variable cCO is set equal to (iCbCr * numC).

When base_mode_flag is equal to 0, the lists tCoeffLevel and sTCoeff are modified by

$$\text{tCoeffLevel}[\ 256 + \text{cCO} + \text{k}\ ] = 0 \qquad \text{with k} = 0..(\text{numC} - 1) \qquad \text{(G-147)}$$

$$\text{sTCoeff}[\ 256 + \text{cCO} + \text{k}\ ] = \text{pcm\_sample\_chroma}[\ \text{cCO} + \text{k}\ ] \qquad \text{with k} = 0..(\text{numC} - 1) \qquad \text{(G-148)}$$

G.8.5.1.2.2 Refinement process for chroma transform coefficients with ChromaArrayType equal to 1 or 2

This process is only invoked when ChromaArrayType is equal to 1 or 2.

Inputs to this process are:

– a variable iCbCr specifying the chroma component,

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,

– a list of transform coefficient level values tCoeffLevel with (256 + 2 * MbWidthC * MbHeightC) elements,

– the chroma quantisation parameters qP'_{Cb} and qP'_{Cr}.

Outputs of this process are:

– a modified version of the list sTCoeff,

– a modified version of the list tCoeffLevel.

The variables nW, nH, numB, cO, and qP are derived by

$$\text{nW} = \text{MbWidthC} / 4 \qquad \text{(G-149)}$$
$$\text{nH} = \text{MbHeightC} / 4 \qquad \text{(G-150)}$$
$$\text{numB} = \text{nW} * \text{nH} \qquad \text{(G-151)}$$
$$\text{cO} = 256 + (\ \text{iCbCr} * \text{MbWidthC} * \text{MbHeightC}\ ) \qquad \text{(G-152)}$$
$$\text{qP} = (\ (\ \text{iCbCr} == 0\ )\ ?\ qP'_{Cb}\ :\ qP'_{Cr}\ ) \qquad \text{(G-153)}$$

Depending on tcoeff_level_prediction_flag, the following applies:

– If tcoeff_level_prediction_flag is equal to 1, the list sTCoeff is modified by

$$\text{sTCoeff}[\ \text{cO} + \text{k}\ ] = 0 \qquad \text{with k} = 0..(\text{MbWidthC} * \text{MbHeightC} - 1) \qquad \text{(G-154)}$$

– Otherwise (tcoeff_level_prediction_flag is equal to 0), the list tCoeffLevel is modified by

$$\text{tCoeffLevel}[\ \text{cO} + \text{k}\ ] = 0 \qquad \text{with k} = 0..(\text{MbWidthC} * \text{MbHeightC} - 1) \qquad \text{(G-155)}$$

For the chroma DC transform coefficients of all residual 4x4 chroma blocks, the following ordered steps are specified:

1. Depending on ChromaArrayType, the (nW)x(nH) array c with elements c_{ij} is derived as follows:

   – If ChromaArrayType is equal to 1,

   $$c_{ij} = \text{ChromaDCLevel}[\ \text{iCbCr}\ ][\ 2 * \text{i} + \text{j}\ ] \qquad \text{with i} = 0..(\text{nH} - 1),\ \text{j} = 0..(\text{nW} - 1) \qquad \text{(G-156)}$$

   – Otherwise (ChromaArrayType is equal to 2),

c_{ij} = ChromaDCLevel[iCbCr][scan422ChromaDC[2 * i + j]]
with i = 0..(nH − 1), j = 0..(nW − 1),
and scan422ChromaDC = { 0, 2, 1, 5, 3, 6, 4, 7 } (G-157)

2. The list tCoeffLevel and the (nW)x(nH) array c are modified by

tCoeffLevel[cO + 32 * i + 16 * j] += c_{ij} with i = 0..(nH − 1), j = 0..(nW − 1) (G-158)

c_{ij} = tCoeffLevel[cO + 32 * i + 16 * j] with i = 0..(nH − 1), j = 0..(nW − 1) (G-159)

3. The variable qP_{DC} is derived by

qP_{DC} = ((ChromaArrayType = = 1) ? qP : (qP + 3)) (G-160)

4. The (nW)x(nH) array d with elements d_{ij} representing scaled chroma DC transform coefficient values for all residual 4x4 chroma blocks is derived by

d_{ij} = c_{ij} * (LevelScale4x4(qP_{DC} % 6, 0, 0) << (qP_{DC} / 6)) with i = 0..(nH − 1), j = 0..(nW − 1) (G-161)

5. The list sTCoeff is modified by

sTCoeff[cO + 32 * i + 16 * j] += d_{ij} with i = 0..(nH − 1), j = 0..(nW − 1) (G-162)

For each residual 4x4 chroma block indexed by c4x4BlkIdx = 0..(numB − 1), the following ordered steps are specified:

1. The variable c4x4List, which is a list of 16 entries, is derived by

c4x4List[k] = ((k = = 0) ? 0 : ChromaACLevel[iCbCr][c4x4BlkIdx][k − 1])
with k = 0..15 (G-163)

2. The inverse scanning process for 4x4 transform coefficients and scaling lists as specified in subclause 8.5.6 is invoked with c4x4List as the input and the outputs are chroma transform coefficient level values as a 4x4 array e with elements e_{ij}. During the process in subclause 8.5.6, the current macroblock is treated as field macroblock when fieldMbFlag is equal to 1, and it is treated as frame macroblock when fieldMbFlag is equal to 0.

3. The list tCoeffLevel and the 4x4 array e are modified by

tCoeffLevel[cO + 16 * c4x4BlkIdx + 4 * i + j] += e_{ij} with i, j = 0..3 and i + j > 0 (G-164)

e_{ij} = tCoeffLevel[cO + 16 * c4x4BlkIdx + 4 * i + j] with i, j = 0..3 and i + j > 0 (G-165)

4. The scaling process for residual 4x4 blocks as specified in subclause 8.5.12.1 is invoked with $BitDepth_C$, qP, and the 4x4 array e as the inputs and the outputs are scaled chroma transform coefficient values as a 4x4 array d of with elements d_{ij}. During the process in subclause 8.5.12.1, the array e is treated as relating to a chroma residual block.

5. The list sTCoeff is modified by

sTCoeff[cO + 16 * c4x4BlkIdx + 4 * i + j] += d_{ij} with i, j = 0..3 and i + j > 0 (G-166)

NOTE 1 – The elements tCoeffLevel[cO + 16* c4x4BlkIdx] and sTCoeff[cO + 16* c4x4BlkIdx] are not modified during the process for a residual 4x4 chroma block with index c4x4BlkIdx.

The bitstream shall not contain data that result in any element sTCoeff[cO + 16* b + k] with b = 0..(numB − 1) and k = 1..15 that exceeds the range of integer values from $-2^{(7 + BitDepth_C)}$ to $2^{(7 + BitDepth_C)} - 1$, inclusive.

The bitstream shall not contain data that result in any element tCoeffLevel[cO + 16* b + k] with b = 0..(numB − 1) and k = 1..15 that exceeds the range of integer values from $-2^{(7 + BitDepth_C)}$ to $2^{(7 + BitDepth_C)} - 1$, inclusive.

NOTE 2 – The elements tCoeffLevel[cO + 16* b] and sTCoeff[cO + 16* b] with b = 0..(numB − 1) can exceed the range of integer values from $-2^{(7 + BitDepth_C)}$ to $2^{(7 + BitDepth_C)} - 1$, inclusive.

G.8.5.2 Transform coefficient level scaling process prior to transform coefficient refinement

Inputs to this process are:

– a variable cTrafo specifying the luma transform type for the current macroblock,

– a list tCoeffLevel with (256 + 2 * MbWidthC * MbHeightC) elements specifying transform coefficient level values for the current macroblock,

– a variable tQP_Y specifying the luma quantisation parameter for the current macroblock,

– a variable $refQP_Y$ specifying the quantisation parameter for the macroblock of the reference layer representation,

– when ChromaArrayType is not equal to 0, two variables tQP_{Cb} and tQP_{Cr} specifying chroma quantisation parameters for the current macroblock,

– when ChromaArrayType is not equal to 0, two variables $refQP_{Cb}$ and $refQP_{Cr}$ specifying chroma quantisation parameters for the macroblock of the reference layer representation,

Output of this process is a modified version of the list tCoeffLevel.

Table G-6 specifies the scale values cS for transform coefficient level scaling.

Table G-6 – Scale values cS for transform coefficient level scaling

(refQP − cQP + 54) % 6	scale value cS
0	8
1	9
2	10
3	11
4	13
5	14

The variable iYCbCr proceeds over the values from 0 to ((ChromaArrayType == 0) ? 0 : 2), inclusive, and for each value of iYCbCr, the following ordered steps are specified:

1. The variables cO, iMax, cQP, and refQP are derived by

$$\text{cO} = ((\text{iYCbCr} == 0) \;?\; 0 : (256 + (\text{iYCbCr} - 1) * \text{MbWidthC} * \text{MbHeightC})) \quad \text{(G-167)}$$
$$\text{iMax} = ((\text{iYCbCr} == 0) \;?\; 255 : (\text{MbWidthC} * \text{MbHeightC} - 1)) \quad \text{(G-168)}$$
$$\text{cQP} = ((\text{iYCbCr} == 0) \;?\; \text{tQP}_Y : (\text{iYCbCr} == 1 \;?\; \text{tQP}_{Cb} : \text{tQP}_{Cr})) \quad \text{(G-169)}$$
$$\text{refQP} = ((\text{iYCbCr} == 0) \;?\; \text{refQP}_Y : (\text{iYCbCr} == 1 \;?\; \text{refQP}_{Cb} : \text{refQP}_{Cr})) \quad \text{(G-170)}$$

2. The variable cS is set as specified in Table G-6 using the values of refQP and cQP.

3. The variable rShift is calculated by

$$\text{rShift} = (\text{refQP} - \text{cQP} + 54) / 6 \quad \text{(G-171)}$$

4. The list tCoeffLevel of transform coefficient level values is modified by

$$\text{tCoeffLevel}[\text{cO} + i] = ((\text{cS} * \text{tCoeffLevel}[\text{cO} + i]) << \text{rShift}) >> 12 \quad \text{with } i = 0..\text{iMax} \quad \text{(G-172)}$$

The following constraints shall be obeyed:

a) The bitstream shall not contain data that result in any element tCoeffLevel[k] with k = 0..255 that exceeds the range of integer values from $-2^{(7 + \text{BitDepth}_Y)}$ to $2^{(7 + \text{BitDepth}_Y)} - 1$, inclusive.

b) When ChromaArrayType is equal to 1 or 2, the bitstream shall not contain data that result in any element tCoeffLevel[256 + 16* b + k] with b = 0..(MbWidthC * MbHeightC / 8 − 1), and k = 1..15 that exceeds the range of integer values from $-2^{(7 + \text{BitDepth}_C)}$ to $2^{(7 + \text{BitDepth}_C)} - 1$, inclusive.

NOTE 1 – When ChromaArrayType is equal to 1 or 2 and cTrafo is not equal to T_PCM, the elements tCoeffLevel[256 + 16* b] with b = 0..(MbWidthC * MbHeightC / 8 − 1) can exceed the range of integer values from $-2^{(7 + \text{BitDepth}_C)}$ to $2^{(7 + \text{BitDepth}_C)} - 1$, inclusive.

c) When ChromaArrayType is equal to 3, the bitstream shall not contain data that result in any element tCoeffLevel[256 + k] with k = 0..511 that exceeds the range of integer values from $-2^{(7 + \text{BitDepth}_C)}$ to $2^{(7 + \text{BitDepth}_C)} - 1$, inclusive.

NOTE 2 – When tQP_Y is less than 10 and cTrafo is equal to T_16x16, the range of values that can be represented by an alternative representation of the bitstream with entropy_coding_mode_flag equal to 0 and profile_idc equal to 66, 77, or 88, may not be sufficient to represent the full range of values of the elements tCoeffLevel[16 * b] with b = 0..15 that could be necessary to form a close approximation of the content of any possible source picture.

NOTE 3 – When ChromaArrayType is equal to 1 or 2 and tQP_{CX} with CX being replaced by Cb and Cr is less than 4, the range of values that can be represented by an alternative representation of the bitstream with entropy_coding_mode_flag equal to 0 and profile_idc equal to 66, 77, or 88, may not be sufficient to represent the full range of values of the elements tCoeffLevel[256 + 16 * b] with b = 0..(MbWidthC * MbHeightC / 8 − 1) that could be necessary to form a close approximation of the content of any possible source picture.

G.8.5.3 Residual construction and accumulation process

Inputs to this process are:

– a variable accumulationFlag specifying whether the constructed residual sample values for the current macroblock are combined with the existent residual sample value for the macroblock,

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a variable cTrafo specifying the transform type,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,

– a $(PicWidthInSamples_L)x(PicHeightInSamples_L)$ array $picRes_L$ containing residual luma sample values for the current layer representation,

– when ChromaArrayType is not equal to 0, two $(PicWidthInSamples_C)x(PicHeightInSamples_C)$ arrays $picRes_{Cb}$ and $picRes_{Cr}$ containing residual chroma sample values for the current layer representation.

Outputs of this process are:

– a modified version of the array $picRes_L$,

– when ChromaArrayType is not equal to 0, modified versions of the arrays $picRes_{Cb}$ and $picRes_{Cr}$.

The construction process for luma residuals as specified in subclause G.8.5.3.1 is invoked with cTrafo and sTCoeff as the inputs and the outputs are residual luma sample values as a 16x16 array $mbRes_L$.

When ChromaArrayType is not equal to 0, the construction process for chroma residuals as specified in subclause G.8.5.3.2 is invoked with cTrafo and sTCoeff as the inputs and the outputs are residual chroma sample values as two (MbWidthC)x(MbHeightC) arrays $mbRes_{Cb}$ and $mbRes_{Cr}$.

When accumulationFlag is equal to 1, the following ordered steps are specified:

1. The macroblock sample array extraction process as specified in subclause G.8.5.4.2 is invoked with fieldMbFlag, $picRes_L$, and, when ChromaArrayType is equal to 0, $picRes_{Cb}$ and $picRes_{Cr}$ as the inputs and the outputs are a 16x16 array $refLayerMbRes_L$ and, when ChromaArrayType is not equal to 0, two (MbWidthC)x(MbHeightC) arrays $refLayerMbRes_{Cb}$ and $refLayerMbRes_{Cr}$.

2. All elements $mbRes_L[x, y]$ of the 16x16 array $mbRes_L$ with x, y = 0..15 are modified by

$$mbRes_L[x, y] = Clip3(yMin, yMax, mbRes_L[x, y] + refLayerMbRes_L[x, y]) \quad (G-173)$$

   with

$$yMin = -(1 << BitDepth_Y) + 1 \quad (G-174)$$
$$yMax = (1 << BitDepth_Y) - 1 \quad (G-175)$$

3. When ChromaArrayType is not equal to 0, for CX being replaced by Cb and Cr, all elements $mbRes_{CX}[x, y]$ of the (MbWidthC)x(MbHeightC) array $mbRes_{CX}$ with x = 0..(MbWidthC − 1) and y = 0..(MbHeightC − 1) are modified by

$$mbRes_{CX}[x, y] = Clip3(cMin, cMax, mbRes_{CX}[x, y] + refLayerMbRes_{CX}[x, y]) \quad (G-176)$$

   with

$$cMin = -(1 << BitDepth_C) + 1 \quad (G-177)$$
$$cMax = (1 << BitDepth_C) - 1 \quad (G-178)$$

The picture sample array construction process as specified in subclause G.8.5.4.1 is invoked with fieldMbFlag, $mbRes_L$, $picRes_L$, and, when ChromaArrayType is not equal to 0, $mbRes_{Cb}$, $mbRes_{Cr}$, $picRes_{Cb}$, and $picRes_{Cr}$ as the inputs and the outputs are a modified version of the array $picRes_L$ and, when ChromaArrayType is not equal to 0, modified versions of the arrays $picRes_{Cb}$ and $picRes_{Cr}$.

G.8.5.3.1 Construction process for luma residuals or chroma residuals with ChromaArrayType equal to 3

Inputs to this process are:

– a variable iYCbCr specifying the colour component (when present),

– a variable cTrafo specifying the transform type,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are residual sample values as a 16x16 array mbRes with elements mbRes[x, y].

When iYCbCr is not present as input to this subclause, it is inferred to be equal to 0.

Depending on iYCbCr, the variables bitDepth and cO are derived as follows:

– If iYCbCr is equal to 0, bitDepth is set equal to BitDepth_Y and cO is set equal to 0.

– Otherwise, if iYCbCr is equal to 1, bitDepth is set equal to BitDepth_C and cO is set equal to 256.

– Otherwise (iYCbCr is equal to 2), bitDepth is set equal to BitDepth_C and cO is set equal to (256 + MbWidthC * MbHeightC).

Depending on cTrafo, the 16x16 array mbRes is derived as follows:

– If cTrafo is equal to T_PCM, the construction process for luma residuals or chroma residuals with ChromaArrayType equal to 3 of I_PCM macroblocks as specified in subclause G.8.5.3.1.1 is invoked with cO and sTCoeff as the inputs and the output is the 16x16 array mbRes of residual sample values.

– Otherwise, if cTrafo is equal to T_4x4, the construction process for residual 4x4 blocks as specified in subclause G.8.5.3.1.2 is invoked with bitDepth, cO, and sTCoeff as the inputs and the output is the 16x16 array mbRes of residual sample values.

– Otherwise, if cTrafo is equal to T_8x8, the construction process for residual 8x8 blocks as specified in subclause G.8.5.3.1.3 is invoked with bitDepth, cO, and sTCoeff as the inputs and the output is the 16x16 array mbRes of residual sample values.

– Otherwise (cTrafo is equal to T_16x16), the construction process for residuals of Intra_16x16 macroblocks as specified in subclause G.8.5.3.1.4 is invoked with bitDepth, cO, and sTCoeff as the inputs and the output is the 16x16 array mbRes of residual sample values.

G.8.5.3.1.1 Construction process for luma residuals or chroma residuals with ChromaArrayType equal to 3 of I_PCM macroblocks

Inputs to this process are:

– a variable cO specifying the first coefficient index in the list of scaled transform coefficient values sTCoeff,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are residual sample values as a 16x16 array mbRes with elements mbRes[x, y].

The 16x16 array mbRes is derived by:

$$\text{mbRes}[\ x, y\ ] = \text{sTCoeff}[\ \text{cO} + y * 16 + x\ ] \qquad \text{with } x, y = 0..15 \tag{G-179}$$

G.8.5.3.1.2 Construction process for residual 4x4 blocks

Inputs to this process are:

– a variable bitDepth specifying the bit depth,

– a variable cO specifying the first coefficient index in the list of scaled transform coefficient values sTCoeff,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are residual sample values as a 16x16 array mbRes with elements mbRes[x, y].

For each residual 4x4 block indexed by c4x4BlkIdx = 0..15, the following ordered steps are specified:

1. The 4x4 array d with elements d_{ij} is derived by:

$$d_{ij} = \text{sTCoeff}[\ \text{cO} + 16 * \text{c4x4BlkIdx} + 4 * i + j\ ] \qquad \text{with } i, j = 0..3 \tag{G-180}$$

2. The transformation process for residual 4x4 blocks as specified in subclause 8.5.12.2 is invoked with bitDepth and the 4x4 array d as the inputs and the outputs are residual sample value as a 4x4 array r with elements r_{ij}.

3. The inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with c4x4BlkIdx as the input and the output is assigned to (xP, yP).

4. The elements mbRes[x, y] of the 16x16 array mbRes with x = xP..(xP + 3) and y = yP..(yP + 3) are derived by

$$\text{mbRes}[\ xP + j, yP + i\ ] = r_{ij} \qquad \text{with } i, j = 0..3 \tag{G-181}$$

G.8.5.3.1.3 Construction process for residual 8x8 blocks

Inputs to this process are:

– a variable bitDepth specifying the bit depth,

– a variable cO specifying the first coefficient index in the list of scaled transform coefficient values sTCoeff,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are residual sample values as a 16x16 array mbRes with elements mbRes[x, y].

For each residual 8x8 block indexed by c8x8BlkIdx = 0..3, the following ordered steps are specified:

1. The 8x8 array d with elements d_{ij} is derived by:

$$d_{ij} = \text{sTCoeff}[\ cO + 64 * \text{c8x8BlkIdx} + 8 * i + j\ ] \qquad \text{with } i, j = 0..7 \tag{G-182}$$

2. The transformation process for residual 8x8 blocks as specified in subclause 8.5.13.2 is invoked with bitDepth and the 8x8 array d as the inputs and the outputs are residual sample values as an 8x8 array r with elements r_{ij}.

3. The inverse 8x8 luma block scanning process as specified in subclause 6.4.5 is invoked with c8x8BlkIdx as the input and the output is assigned to (xP, yP).

4. The elements mbRes[x, y] of the 16x16 array mbRes with x = xP..(xP + 7) and y = yP..(yP + 7) are derived by

$$\text{mbRes}[\ xP + j, yP + i\ ] = r_{ij} \qquad \text{with } i, j = 0..7 \tag{G-183}$$

G.8.5.3.1.4 Construction process for residuals of Intra_16x16 macroblocks

Inputs to this process are:

– a variable bitDepth specifying the bit depth,

– a variable cO specifying the first coefficient index in the list of scaled transform coefficient values sTCoeff,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are residual sample values as a 16x16 array mbRes with elements mbRes[x, y].

For each residual 4x4 block indexed by c4x4BlkIdx = 0..15, the following ordered steps are specified:

1. The 4x4 array d with elements d_{ij} is derived by:

$$d_{ij} = \text{sTCoeff}[\ cO + 16 * \text{c4x4BlkIdx} + 4 * i + j\ ] \qquad \text{with } i, j = 0..3 \tag{G-184}$$

2. The transformation process for residual 4x4 blocks as specified in subclause 8.5.12.2 is invoked with bitDepth and the 4x4 array d as the inputs and the outputs are residual sample values as a 4x4 array r with elements r_{ij}.

3. The inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with c4x4BlkIdx as the input and the output is assigned to (xP, yP).

4. The elements mbRes[x, y] of the 16x16 array mbRes with x = xP..(xP + 3) and y = yP..(yP + 3) are derived by

$$\text{mbRes}[\ xP + j, yP + i\ ] = r_{ij} \qquad \text{with } i, j = 0..3 \tag{G-185}$$

G.8.5.3.2 Construction process for chroma residuals

Inputs to this process are:

– a variable cTrafo specifying the transform type,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are residual chroma sample values as two (MbWidthC)x(MbHeightC) arrays mbRes_{Cb} and mbRes_{Cr} with elements mbRes_{Cb}[x, y] and mbRes_{Cr}[x, y], respectively.

For both chroma components indexed by iCbCr = 0..1 and for CX being replaced by Cb for iCbCr equal to 0 and Cr for iCbCr equal to 1, the following applies:

– If ChromaArrayType is equal to 1 or 2, the following applies:

  – If cTrafo is equal to T_PCM, the construction process for chroma residuals of I_PCM macroblocks as specified in subclause G.8.5.3.2.1 is invoked with iCbCr and sTCoeff as the inputs and the output is the (MbWidthC)x(MbHeightC) array mbRes_{CX} of residual chroma sample values.

  – Otherwise (cTrafo is not equal to T_PCM), the construction process for chroma residuals with ChromaArrayType equal to 1 or 2 as specified in subclause G.8.5.3.2.2 is invoked with iCbCr and sTCoeff as the inputs and the output is the (MbWidthC)x(MbHeightC) array mbRes_{CX} of residual chroma sample values.

– Otherwise (ChromaArrayType is equal to 3), the construction process for luma residuals or chroma residuals with ChromaArrayType equal to 3 as specified in subclause G.8.5.3.1 is invoked with iYCbCr set equal to (1 + iCbCr), cTrafo, and sTCoeff as the inputs and the output is the (MbWidthC)x(MbHeightC) array mbRes_{CX} of residual chroma sample values.

G.8.5.3.2.1 Construction process for chroma residuals of I_PCM macroblocks

Inputs to this process are:

– a variable iCbCr specifying the chroma component,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are residual chroma sample values as a (MbWidthC)x(MbHeightC) array mbRes with elements mbRes[x, y].

The (MbWidthC)x(MbHeightC) array mbRes is derived by

$$\text{mbRes}[\,x, y\,] = \text{sTCoeff}[\,256 + \text{iCbCr} * \text{MbWidthC} * \text{MbHeightC} + y * \text{MbWidthC} + x\,] \quad \text{with } x = 0..(\text{MbWidthC} - 1) \text{ and } y = 0..(\text{MbHeightC} - 1) \tag{G-186}$$

G.8.5.3.2.2 Construction process for chroma residuals with ChromaArrayType equal to 1 or 2

This process is only invoked when ChromaArrayType is equal to 1 or 2.

Inputs to this process are:

– a variable iCbCr specifying the chroma component,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are residual chroma sample values as a (MbWidthC)x(MbHeightC) array mbRes with elements mbRes[x, y].

The variables nW, nH, numB, and cO are derived by

$$\text{nW} = \text{MbWidthC} / 4 \tag{G-187}$$
$$\text{nH} = \text{MbHeightC} / 4 \tag{G-188}$$
$$\text{numB} = \text{nW} * \text{nH} \tag{G-189}$$
$$\text{cO} = 256 + (\,\text{iCbCr} * \text{MbWidthC} * \text{MbHeightC}\,) \tag{G-190}$$

For the chroma DC transform coefficients of all residual 4x4 chroma blocks, the following ordered steps are specified:

1. The (nW)x(nH) array c with the elements c_{ij} is derived by

   $$c_{ij} = \text{sTCoeff}[\,\text{cO} + 32 * i + 16 * j\,] \qquad \text{with } i = 0..(\text{nH} - 1),\ j = 0..(\text{nW} - 1) \tag{G-191}$$

2. The transformation process for chroma DC transform coefficients as specified in subclause 8.5.11.1 is invoked with BitDepth_C and the (nW)x(nH) array c as the inputs and the outputs are DC values for all residual 4x4 chroma blocks as a (nW)x(nH) array f with elements f_{ij}.

3. Depending on ChromaArrayType, the (nW)x(nH) array dcC with elements dcC_{ij} is derived as follows:

   – If ChromaArrayType is equal to 1,

   $$\text{dcC}_{ij} = f_{ij} >> 5 \qquad \text{with } i = 0..(\text{nH} - 1),\ j = 0..(\text{nW} - 1) \tag{G-192}$$

   – Otherwise (ChromaArrayType is equal to 2),

   $$\text{dcC}_{ij} = (\,f_{ij} + (\,1 << 5\,)\,) >> 6 \qquad \text{with } i = 0..(\text{nH} - 1),\ j = 0..(\text{nW} - 1) \tag{G-193}$$

The bitstream shall not contain data that result in any element dcC_{ij} of dcC with $i = 0..(nH - 1)$ and $j = 0..(nW - 1)$ that exceeds the range of integer values from $-2^{(7 + BitDepth_C)}$ to $2^{(7 + BitDepth_C)} - 1$, inclusive.

NOTE – For the layer representation with dependency_id equal to 0 and quality_id equal to 0, successive invocations of subclause G.8.5.1.2 (as part of an invocation of subclause G.8.5.1) and this subclause yield an array dcC that is identical to the array dcC that would be obtained by an invocation of subclause 8.5.11. However, the intermediate values c_{ij} and f_{ij} with $i = 0..(nH - 1)$ and $j = 0..(nW - 1)$ that are derived in this subclause can exceed the range of integer values from $-2^{(7 + BitDepth_C)}$ to $2^{(7 + BitDepth_C)} - 1$, inclusive.

For each residual 4x4 chroma block indexed by c4x4BlkIdx = 0..(numB − 1), the following ordered steps are specified.

1. The 4x4 array d with elements d_{ij} is derived as follows:
   - The element d_{00} is derived by:

     $d_{00} = dcC_{kl}$ with k = c4x4BlkIdx / 2 and l = c4x4BlkIdx % 2 (G-194)

   - The elements d_{ij} with i, j = 0..3 and i + j > 0 are derived by:

     d_{ij} = sTCoeff[cO + 16 * c4x4BlkIdx + 4 * i + j] (G-195)

2. The transformation process for residual 4x4 blocks as specified in subclause 8.5.12.2 is invoked with $BitDepth_C$ and the 4x4 array d as the inputs and the outputs are residual chroma sample values as a 4x4 array r with elements r_{ij}.

3. The chroma location (xP, yP) is derived by:

   xP = 4 * (c4x4BlkIdx % 2) (G-196)
   yP = 4 * (c4x4BlkIdx / 2) (G-197)

4. The elements mbRes[x, y] of the (MbWidthC)x(MbHeightC) array mbRes with x = xP..(xP + 3) and y = yP..(yP + 3) are derived by:

   mbRes[xP + j, yP + i] = r_{ij} with i, j = 0..3 (G-198)

G.8.5.4 Sample array accumulation process

Inputs to this process are:

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a $(PicWidthInSamples_L)x(PicHeightInSamples_L)$ array $picRes_L$ containing residual luma sample values for the current layer representation,

– a $(PicWidthInSamples_L)x(PicHeightInSamples_L)$ array $picSamples_L$ containing constructed luma sample values for the current layer representation,

– when ChromaArrayType is not equal to 0, two $(PicWidthInSamples_C)x(PicHeightInSamples_C)$ arrays $picRes_{Cb}$ and $picRes_{Cr}$ containing residual chroma sample values for the current layer representation,

– when ChromaArrayType is not equal to 0, two $(PicWidthInSamples_C)x(PicHeightInSamples_C)$ arrays $picSamples_{Cb}$ and $picSamples_{Cr}$ containing constructed chroma sample values for the current layer representation.

Outputs of this process are:

– a modified version of the array $picSamples_L$,

– when ChromaArrayType is not equal to 0, modified versions of the arrays $picSamples_{Cb}$ and $picSamples_{Cr}$.

The macroblock sample array extraction process as specified in subclause G.8.5.4.2 is invoked with fieldMbFlag, $picRes_L$, and, when ChromaArrayType is not equal to 0, $picRes_{Cb}$ and $picRes_{Cr}$ as the inputs and the outputs are assigned to $mbRes_L$ and, when ChromaArrayType is not equal to 0, $mbRes_{Cb}$ and $mbRes_{Cr}$.

The macroblock sample array extraction process as specified in subclause G.8.5.4.2 is invoked with fieldMbFlag, $picSamples_L$, and, when ChromaArrayType is not equal to 0, $picSamples_{Cb}$ and $picSamples_{Cr}$ as the inputs and the outputs are assigned to $mbPred_L$ and, when ChromaArrayType is not equal to 0, $mbPred_{Cb}$ and $mbPred_{Cr}$.

The 16x16 array $mbSamples_L$ is derived by:

$mbSamples_L$[x, y] = $Clip1_Y$($mbPred_L$[x, y] + $mbRes_L$[x, y]) with x, y = 0..15 (G-199)

When ChromaArrayType is not equal to 0, for CX being replaced by Cb and Cr, the (MbWidthC)x(MbHeightC) array $mbSamples_{CX}$ is derived by:

$$mbSamples_{CX}[\, x, y\,] = Clip1_C(\, mbPred_{CX}[\, x, y\,] + mbRes_{CX}[\, x, y\,]\,) \quad \text{with } x = 0..(MbWidthC - 1) \text{ and } y = 0..(MbHeightC - 1) \qquad \text{(G-200)}$$

The picture sample array construction process as specified in subclause G.8.5.4.1 is invoked with fieldMbFlag, $mbSamples_L$, $picSamples_L$, and, when ChromaArrayType is not equal to 0, $mbSamples_{Cb}$, $mbSamples_{Cr}$, $picSamples_{Cb}$, and $picSamples_{Cr}$ as inputs and the outputs are a modified version of $picSamples_L$ and, when ChromaArrayType is not equal to 0, modified versions of $picSamples_{Cb}$ and $picSamples_{Cr}$.

G.8.5.4.1 Picture sample array construction process

Inputs to this process are:

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a 16x16 array $mbArray_L$ containing luma sample values for the current macroblock,

– a ($PicWidthInSamples_L$)x($PicWidthInHeight_L$) array $picArray_L$ containing luma sample values for the current layer representation,

– when ChromaArrayType is not equal to 0, two (MbWidthC)x(MbHeightC) arrays $mbArray_{Cb}$ and $mbArray_{Cr}$ containing chroma sample values for the current macroblock,

– when ChromaArrayType is not equal to 0, two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $picArray_{Cb}$ and $picArray_{Cr}$ containing chroma sample values for the current layer representation.

Outputs of this process are:

– a modified version of the array $picArray_L$,

– when ChromaArrayType is not equal to 0, modified versions of the arrays $picArray_{Cb}$ and $picArray_{Cr}$.

The picture sample array construction process for a colour component as specified in subclause G.8.5.4.3 is invoked with fieldMbFlag, mbW set equal to 16, mbH set equal to 16, $mbArray_L$, and $picArray_L$ as the inputs and the output is a modified version of the array $picArray_L$.

When ChromaArrayType is not equal to 0, for CX being replaced with Cr and Cb, the picture sample array construction process for a colour component as specified in subclause G.8.5.4.3 is invoked with fieldMbFlag, mbW set equal to MbWidthC, mbH set equal to MbHeightC, $mbArray_{CX}$, and $picArray_{CX}$ as the inputs and the output is a modified version of the array $picArray_{CX}$.

G.8.5.4.2 Macroblock sample array extraction process

Inputs to this process are:

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a ($PicWidthInSamples_L$)x($PicWidthInHeight_L$) array $picArray_L$ containing luma sample values for the current layer representation,

– when ChromaArrayType is not equal to 0, two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $picArray_{Cb}$ and $picArray_{Cr}$ containing chroma sample values for the current layer representation.

Outputs of this process are:

– a 16x16 array $mbArray_L$ containing luma sample values for the current macroblock,

– when ChromaArrayType is not equal to 0, two (MbWidthC)x(MbHeightC) arrays $mbArray_{Cb}$ and $mbArray_{Cr}$ containing chroma sample values for the current macroblock.

The macroblock sample array extraction process for a colour component as specified in subclause G.8.5.4.4 is invoked with fieldMbFlag, mbW set equal to 16, mbH set equal to 16, and $picArray_L$ as the inputs and the output is assigned to $mbArray_L$.

When ChromaArrayType is not equal to 0, for CX being replaced with Cr and Cb, the macroblock sample array extraction process for a colour component as specified in subclause G.8.5.4.4 is invoked with fieldMbFlag, mbW set equal to MbWidthC, mbH set equal to MbHeightC, and $picArray_{CX}$ as the inputs and the output is assigned to $mbArray_{CX}$.

G.8.5.4.3 Picture sample array construction process for a colour component

Inputs to this process are:

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a variable mbW specifying the width of a macroblock colour component in samples,

– a variable mbH specifying the height of a macroblock colour component in samples,

– an (mbW)x(mbH) array mbArray containing sample values for a colour component of the current macroblock,

– an (mbW * PicWidthInMbs)x(mbH * PicHeightInMbs) array picArray containing sample values for a colour component of the current layer representation.

Output of this process is a modified version of the array picArray.

The inverse macroblock scanning process as specified in subclause 6.4.1 is invoked with CurrMbAddr as the input and the output is assigned to (xO, yO). During the process in subclause 6.4.1, the current macroblock is treated as field macroblock when fieldMbFlag is equal to 1, and it is treated as frame macroblock when fieldMbFlag is equal to 0.

The sample location (xP, yP) is derived by:

$$xP = (xO >> 4) * mbW \qquad (G-201)$$
$$yP = ((yO >> 4) * mbH) + (yO \% 2) \qquad (G-202)$$

Depending on the variables MbaffFrameFlag and fieldMbFlag, the array picArray is modified as follows:

– If MbaffFrameFlag is equal to 1 and fieldMbFlag is equal to 1,

$$picArray[xP + x, yP + 2 * y] = mbArray[x, y] \quad \text{with } x = 0..(mbW - 1),\ y = 0..(mbH - 1) \qquad (G-203)$$

– Otherwise (MbaffFrameFlag is equal to 0 or fieldMbFlag is equal to 0),

$$picArray[xP + x, yP + y] = mbArray[x, y] \quad \text{with } x = 0..(mbW - 1),\ y = 0..(mbH - 1) \qquad (G-204)$$

G.8.5.4.4 Macroblock sample array extraction process for a colour component

Inputs to this process are:

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a variable mbW specifying the width of a macroblock colour component in samples,

– a variable mbH specifying the height of a macroblock colour component in samples,

– an (mbW * PicWidthInMbs)x(mbH * PicHeightInMbs) array picArray containing sample values for a colour component of the current layer representation.

Output of this process is an (mbW)x(mbH) array mbArray containing sample values for a colour component of the current macroblock.

The inverse macroblock scanning process as specified in subclause 6.4.1 is invoked with CurrMbAddr as the input and the output is assigned to (xO, yO). During the process in subclause 6.4.1, the current macroblock is treated as field macroblock when fieldMbFlag is equal to 1, and it is treated as frame macroblock when fieldMbFlag is equal to 0.

The sample location (xP, yP) is derived by:

$$xP = (xO >> 4) * mbW \qquad (G-205)$$
$$yP = ((yO >> 4) * mbH) + (yO \% 2) \qquad (G-206)$$

Depending on the variables MbaffFrameFlag and fieldMbFlag, the samples of the array mbArray are derived as follows:

– If MbaffFrameFlag is equal to 1 and fieldMbFlag is equal to 1,

$$mbArray[x, y] = picArray[xP + x, yP + 2 * y] \quad \text{with } x = 0..(mbW - 1),\ y = 0..(mbH - 1) \qquad (G-207)$$

– Otherwise (MbaffFrameFlag is equal to 0 or fieldMbFlag is equal to 0),

$$mbArray[x, y] = picArray[xP + x, yP + y] \quad \text{with } x = 0..(mbW - 1),\ y = 0..(mbH - 1) \qquad (G-208)$$

G.8.5.5 Sample array re-initialisation process

Inputs to this process are:

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array $picSamples_L$ containing luma sample values for the current layer representation,

– when ChromaArrayType is not equal to 0, two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $picSamples_{Cb}$ and $picSamples_{Cr}$ containing chroma sample values for the current layer representation.

Outputs of this process are:

– a modified version of the array $picSamples_L$,

– when ChromaArrayType is not equal to 0, modified versions of the arrays $picSamples_{Cb}$ and $picSamples_{Cr}$.

The 16x16 array $mbSamples_L$ is derived by:

$$mbSamples_L[\ x, y\ ] = 0 \qquad \text{with } x, y = 0..15 \tag{G-209}$$

When ChromaArrayType is not equal to 0, for CX being replaced by Cb and Cr, the (MbWidthC)x(MbHeightC) array $mbSamples_{CX}$ is derived by:

$$mbSamples_{CX}[\ x, y\ ] = 0 \qquad \text{with } x = 0..(MbWidthC - 1) \text{ and } y = 0..(MbHeightC - 1) \tag{G-210}$$

The picture sample array construction process as specified in subclause G.8.5.4.1 is invoked with fieldMbFlag, $mbSamples_L$, $picSamples_L$, and, when ChromaArrayType is not equal to 0, $mbSamples_{Cb}$, $mbSamples_{Cr}$, $picSamples_{Cb}$, and $picSamples_{Cr}$ as inputs and the outputs are a modified version of $picSamples_L$ and, when ChromaArrayType is not equal to 0, modified versions of $picSamples_{Cb}$ and $picSamples_{Cr}$.

G.8.6 Resampling processes for prediction data, intra samples, and residual samples

Subclause G.8.6.1 specifies the derivation process for inter-layer predictors for macroblock type, sub-macroblock type, references indices, and motion vectors.

Subclause G.8.6.2 specifies the resampling process for intra samples.

Subclause G.8.6.3 specifies the resampling process for residual samples.

G.8.6.1 Derivation process for inter-layer predictors for macroblock type, sub-macroblock type, reference indices, and motion vectors

This process is only invoked when base_mode_flag is equal to 1 or any motion_prediction_flag_lX[mbPartIdx] with X being replaced by 0 and 1 and mbPartIdx = 0..3 is equal to 1.

Inputs to this process are:

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying the macroblock types for the macroblocks of the reference layer representation,

– a (RefLayerPicSizeInMbs)x4 array refLayerSubMbType specifying the sub-macroblock types for the macroblocks of the reference layer representation,

– two (RefLayerPicSizeInMbs)x4 arrays refLayerPredFlagL0 and refLayerPredFlagL1 specifying prediction utilization flags for the macroblocks of the reference layer representation,

– two (RefLayerPicSizeInMbs)x4 arrays refLayerRefIdxL0 and refLayerRefIdxL1 specifying reference indices for the macroblocks of the reference layer representation,

– two (RefLayerPicSizeInMbs)x4x4x2 arrays refLayerMvL0 and refLayerMvL1 specifying motion vector components for the macroblocks of the reference layer representation,

– when CroppingChangeFlag is equal to 1 and (slice_type % 5) is less than 2, the reference picture list refPicList0,

– when CroppingChangeFlag is equal to 1 and (slice_type % 5) is equal to 1, the reference picture list refPicList1.

Outputs of this process are:

– a variable mbTypeILPred specifying a predictor for the macroblock type of the current macroblock,

– a list subMbTypeILPred with 4 elements specifying predictors for sub-macroblock types of the current macroblock,

– two 2x2 arrays refIdxILPredL0 and refIdxILPredL1 specifying inter-layer predictors for the reference indices of the current macroblock,

– two 4x4x2 arrays mvILPredL0 and mvILPredL1 specifying inter-layer predictors for the motion vector components of the current macroblock.

The derivation process for reference layer partition identifications as specified in subclause G.8.6.1.1 is invoked with fieldMbFlag, refLayerFieldMbFlag, refLayerMbType, and refLayerSubMbType as the inputs and the outputs are a variable intraILPredFlag and, when intraILPredFlag is equal to 0, reference layer partition identifications as a 4x4 array refLayerPartIdc with elements refLayerPartIdc[x, y].

When slice_type is equal to EI, the bitstream shall not contain data that result in intraILPredFlag equal to 0.

Depending on intraILPredFlag, the 2x2 arrays refIdxILPredL0 and refIdxILPredL1 and the 4x4x2 array mvILPredL0 and mvILPredL1 are derived as follows:

– If intraILPredFlag is equal to 1, all elements of the 2x2 arrays refIdxILPredL0 and refIdxILPredL1 are set equal to −1 and all elements of the 4x4x2 arrays mvILPredL0 and mvILPredL1 are set equal to 0.

– Otherwise (intraILPredFlag is equal to 0), the derivation process for inter-layer predictors for reference indices and motion vectors as specified in subclause G.8.6.1.2 is invoked with fieldMbFlag, refLayerFieldMbFlag, refLayerPredFlagL0, refLayerPredFlagL1, refLayerRefIdxL0, refLayerRefIdxL1, refLayerMvL0, refLayerMvL1, refLayerPartIdc, refPicList0 (when available), and refPicList1 (when available) as the inputs and the outputs are the arrays refIdxILPredL0, refIdxILPredL1, mvILPredL0, and mvILPredL1.

Depending on intraILPredFlag, the variable mbTypeILPred and the list subMbTypeILPred are derived as follows:

– If intraILPredFlag is equal to 1, all elements subMbTypeILPred[mbPartIdx] of the list subMbTypeILPred with mbPartIdx = 0..3 are marked as unspecified, and the variable mbTypeILPred is derived as follows:

  – If tcoeff_level_prediction_flag is equal to 1, mbTypeILPred is set equal to refLayerMbType[CurrMbAddr].

  – Otherwise (tcoeff_level_prediction_flag is equal to 0), mbTypeILPred is set equal to I_BL.

– Otherwise (intraILPredFlag is equal to 0), the derivation process for inter-layer predictors for P and B macroblock and sub-macroblock types as specified in subclause G.8.6.1.3 is invoked with refIdxILPredL0, refIdxILPredL1, mvILPredL0, and mvILPredL1 as the inputs and the outputs are the variable mbTypeILPred and the list subMbTypeILPred.

G.8.6.1.1 Derivation process for reference layer partition identifications

Inputs to this process are:

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying the macroblock types for the macroblocks of the reference layer representation,

– a (RefLayerPicSizeInMbs)x4 array refLayerSubMbType specifying the sub-macroblock types for the macroblocks of the reference layer representation.

Outputs of this process are:

– a variable intraILPredFlag specifying whether the samples of the current macroblock in the current layer representation can be predicted by inter-layer intra prediction or, in the case of tcoeff_level_prediction_flag equal to 1, by a combination of intra-layer intra prediction and inter-layer prediction,

– when intraILPredFlag is equal to 0, reference layer partition identifications for the current macroblock as a 4x4 array refLayerPartIdc with elements refLayerPartIdc[x, y].

When the 4x4 array refLayerPartIdc is output of this process, each of its elements refLayerPartIdc[x, y] specifies the macroblock address, the macroblock partition index, and the sub-macroblock partition index of the partition in the reference layer representation that can be used for inter-layer motion prediction of the macroblock or sub-macroblock partition of the current macroblock that contains the 4x4 block with coordinates x and y.

For each 4x4 block with block coordinates x, y = 0..3, the element refLayerPartIdc[x, y] of the 4x4 array refLayerPartIdc is derived by applying the following ordered steps:

1. The derivation process for reference layer partitions as specified in subclause G.6.2 is invoked with the luma location (4 * x + 1, 4 * y + 1), fieldMbFlag, refLayerFieldMbFlag, refLayerMbType, and refLayerSubMbType as the inputs and the outputs are a macroblock address refMbAddr, a macroblock partition index refMbPartIdx, and a sub-macroblock partition index refSubMbPartIdx of a partition in the reference layer representation.

2. The element refLayerPartIdc[x, y] of the array refLayerPartIdc is derived as follows:

   – If refLayerMbType[refMbAddr] is equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL, refLayerPartIdc[x, y] is set equal to −1.

   – Otherwise (refLayerMbType[refMbAddr] is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL), refLayerPartIdc[x, y] is derived by

   refLayerPartIdc[x, y] = 16 * refMbAddr + 4 * refMbPartIdx + refSubMbPartIdx (G-211)

The variable intraILPredFlag is derived as follows:

– If all elements refLayerPartIdc[x, y] with x, y = 0..3 are equal to −1, intraILPredFlag is set equal to 1.

– Otherwise (any element refLayerPartIdc[x, y] with x, y = 0..3 is not equal to −1), intraILPredFlag is set equal to 0.

When intraILPredFlag is equal to 0 and RestrictedSpatialResolutionChangeFlag is equal to 0, the 4x4 array refLayerPartIdc is modified by the following ordered steps:

1. For each 8x8 block with block coordinates xP, yP = 0..1, the following ordered steps are specified.

   a. The variables xO and yO are set equal to (2 * xP) and (2 * yP), respectively.

   b. All elements procI4x4Blk[xS, yS] of the 2x2 array procI4x4Blk with xS, yS = 0..1 are set equal to 0.

   c. The 4x4 blocks of the current 8x8 block with block coordinates xS, yS = 0..1 are processed in increasing order of (2 * yS + xS), and when refLayerPartIdc[xO + xS, yO + yS] is equal to −1 for a 4x4 block, the element procI4x4Blk[xS, yS] of the array procI4x4Blk is set equal to 1 and the following applies:

      – If procI4x4Blk[1 − xS, yS] is equal to 0 and refLayerPartIdc[xO + 1 − xS, yO + yS] is not equal to −1, the element refLayerPartIdc[xO + xS, yO + yS] is modified by

      refLayerPartIdc[xO + xS, yO + yS] = refLayerPartIdc[xO + 1 − xS, yO + yS] (G-212)

      – Otherwise, if procI4x4Blk[xS, 1 − yS] is equal to 0 and refLayerPartIdc[xO + xS, yO + 1 − yS] is not equal to −1, the element refLayerPartIdc[xO + xS, yO + yS] is modified by

      refLayerPartIdc[xO + xS, yO + yS] = refLayerPartIdc[xO + xS, yO + 1 − yS] (G-213)

      – Otherwise, if procI4x4Blk[1 − xS, 1 − yS] is equal to 0 and refLayerPartIdc[xO + 1 − xS, yO + 1 − yS] is not equal to −1, the element refLayerPartIdc[xO + xS, yO + yS] is modified by

      refLayerPartIdc[xO + xS, yO + yS] = refLayerPartIdc[xO + 1 − xS, yO + 1 − yS] (G-214)

      – Otherwise, the element refLayerPartIdc[xO + xS, yO + yS] is not modified.

2. All elements procI8x8Blk[xP, yP] of the 2x2 array procI8x8Blk with xP, yP = 0..1 are set equal to 0.

3. The 8x8 blocks with block coordinates xP, yP = 0..1 are processed in increasing order of (2 * yP + xP), and when refLayerPartIdc[2 * xP, 2 * yP] is equal to −1 for an 8x8 block, the element procI8x8Blk[xP, yP] of the array procI8x8Blk is set equal to 1 and the following applies:

   – If procI8x8Blk[1 − xP, yP] is equal to 0 and refLayerPartIdc[2 − xP, 2 * yP] is not equal to −1, the elements refLayerPartIdc[2 * xP + xS, 2 * yP + yS] with xS, yS = 0..1 are modified by

   $$\text{refLayerPartIdc}[\ 2 * \text{xP} + \text{xS},\ 2 * \text{yP} + \text{yS}\ ] = \text{refLayerPartIdc}[\ 2 - \text{xP},\ 2 * \text{yP} + \text{yS}\ ] \quad \text{(G-215)}$$

   – Otherwise, if procI8x8Blk [xP, 1 − yP] is equal to 0 and refLayerPartIdc[2 * xP, 2 − yP] is not equal to −1, the elements refLayerPartIdc[2 * xP + xS, 2 * yP + yS] with xS, yS = 0..1 are modified by

   $$\text{refLayerPartIdc}[\ 2 * \text{xP} + \text{xS},\ 2 * \text{yP} + \text{yS}\ ] = \text{refLayerPartIdc}[\ 2 * \text{xP} + \text{xS},\ 2 - \text{yP}\ ] \quad \text{(G-216)}$$

   – Otherwise, if procI8x8Blk[1 − xP, 1 − yP] is equal to 0 and refLayerPartIdc [2 − xP, 2 − yP] is not equal to −1, the elements refLayerPartIdc[2 * xP + xS, 2 * yP + yS] with xS, yS = 0..1 are modified by

   $$\text{refLayerPartIdc}[\ 2 * \text{xP} + \text{xS},\ 2 * \text{yP} + \text{yS}\ ] = \text{refLayerPartIdc}[\ 2 - \text{xP},\ 2 - \text{yP}\ ] \quad \text{(G-217)}$$

   – Otherwise, the elements refLayerPartIdc[2 * xP + xS, 2 * yP + yS] with xS, yS = 0..1 are not modified.

   NOTE – By the process specified above the elements refLayerPartIdc[x, y] that are equal to −1 are replaced by elements refLayerPartIdc[x, y] that are not equal to −1. This process can also be applied when RestrictedSpatialResolutionChangeFlag is equal to 1 or intraILPredFlag is equal to 1, but in this case, the 4x4 array refLayerPartIdc is not modified.

G.8.6.1.2 Derivation process for inter-layer predictors for reference indices and motion vectors

Inputs to this process are:

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– two (RefLayerPicSizeInMbs)x4 arrays refLayerPredFlagL0 and refLayerPredFlagL1 specifying prediction utilization flags for the macroblocks of the reference layer representation,

– two (RefLayerPicSizeInMbs)x4 arrays refLayerRefIdxL0 and refLayerRefIdxL1 specifying reference indices for the macroblocks of the reference layer representation,

– two (RefLayerPicSizeInMbs)x4x4x2 arrays refLayerMvL0 and refLayerMvL1 specifying motion vector components for the macroblocks of the reference layer representation,

– a 4x4 array refLayerPartIdc specifying reference layer partition identifications for the 4x4 blocks of the current macroblock,

– when CroppingChangeFlag is equal to 1 and (slice_type % 5) is less than 2, the reference picture list refPicList0,

– when CroppingChangeFlag is equal to 1 and (slice_type % 5) is equal to 1, the reference picture list refPicList1.

Outputs of this process are:

– two 2x2 arrays refIdxILPredL0 and refIdxILPredL1 specifying inter-layer predictors for the reference indices of the current macroblock,

– two 4x4x2 arrays mvILPredL0 and mvILPredL1 specifying inter-layer predictors for the motion vector components of the current macroblock.

Let tempRefIdxPredL0 and tempRefIdxPredL1 be two 4x4 arrays with elements tempRefIdxPredL0[x, y] and tempRefIdxPredL1[x, y], respectively, that specify auxiliary inter-layer predictors for reference indices.

For each 4x4 block indexed by x, y = 0..3 and for X being replaced by 0 and 1, the auxiliary reference index predictor tempRefIdxPredLX[x, y] and the motion vector predictor mvILPredLX[x, y] are derived as follows:

– If refLayerPredFlagLX[refLayerPartIdc[x, y] / 16][(refLayerPartIdc[x, y] % 16) / 4] is equal 0, the reference index predictor tempRefIdxPredLX[x, y] and the motion vector predictor mvILPredLX[x, y] are derived by:

$$\text{tempRefIdxPredLX}[\ x, y\ ] = -1 \quad \text{(G-218)}$$
$$\text{mvILPredLX}[\ x, y\ ][\ 0\ ] = 0 \quad \text{(G-219)}$$
$$\text{mvILPredLX}[\ x, y\ ][\ 1\ ] = 0 \quad \text{(G-220)}$$

– Otherwise (refLayerPredFlagLX[refLayerPartIdc[x, y] / 16][(refLayerPartIdc[x, y] % 16) / 4] is equal to 1), the following ordered steps are specified:

1. The variables refMbAddr, refMbPartIdx, and refSubMbPartIdx are derived by

$$\text{refMbAddr} = \text{refLayerPartIdc[x, y] / 16} \quad \text{(G-221)}$$
$$\text{refMbPartIdx} = \text{(refLayerPartIdc[x, y] \% 16) / 4} \quad \text{(G-222)}$$
$$\text{refSubMbPartIdx} = \text{refLayerPartIdc[x, y] \% 4} \quad \text{(G-223)}$$

2. The auxiliary reference index predictor tempRefIdxPredLX[x, y] is derived by:

$$\begin{aligned}\text{tempRefIdxPredLX[x, y]} = {} & \text{refLayerRefIdxLX[refMbAddr][refMbPartIdx]} \\ & \text{* (1 + fieldMbFlag − field\_pic\_flag)} \quad \text{(G-224)} \\ & \text{/ (1 + refLayerFieldMbFlag[refMbAddr] − RefLayerFieldPicFlag)}\end{aligned}$$

3. The motion vector aMv is set equal to refLayerMvLX[refMbAddr][refMbPartIdx][refSubMbPartIdx], and afterwards its vertical component aMv[1] is modified by:

$$\text{aMv[1] = aMv[1] * (1 + refLayerFieldMbFlag[refMbAddr])} \quad \text{(G-225)}$$

4. The variables scaledW, scaledH, refLayerW, and refLayerH are derived by:

$$\text{scaledW} = \text{ScaledRefLayerPicWidthInSamples}_L \quad \text{(G-226)}$$
$$\text{scaledH} = \text{ScaledRefLayerPicHeightInSamples}_L \text{ * (1 + field\_pic\_flag)} \quad \text{(G-227)}$$
$$\text{refLayerW} = \text{RefLayerPicWidthInSamples}_L \quad \text{(G-228)}$$
$$\text{refLayerH} = \text{RefLayerPicHeightInSamples}_L \text{ * (1 + RefLayerFieldPicFlag)} \quad \text{(G-229)}$$

5. The variables dOX, dOY, dSW, and dSH are derived as follows:

   – If CroppingChangeFlag is equal to 0 or the reference picture refPicListX[tempRefIdxPredLX[x, y]] is not available, dOX, dOY, dSW, and dSH are set equal to 0.

   – Otherwise (CroppingChangeFlag is equal to 1 and the reference picture refPicListX[tempRefIdxPredLX[x, y]] is available), the variables refPicScaledRefLayerLeftOffset, refPicScaledRefLayerRightOffset, refPicScaledRefLayerTopOffset, and refPicScaledRefLayerBottomOffset are set equal to the variables ScaledRefLayerLeftOffset, ScaledRefLayerRightOffset, ScaledRefLayerTopOffset, and ScaledRefLayerBottomOffset, respectively, that are associated with the layer representation of the reference picture refPicListX[tempRefIdxPredLX[x, y]] that has the same value of DQId as the current layer representation, and the variables dOX, dOY, dSW, and dSH are derived by:

$$\text{dOX = ScaledRefLayerLeftOffset − refPicScaledRefLayerLeftOffset} \quad \text{(G-230)}$$
$$\text{dOY = ScaledRefLayerTopOffset − refPicScaledRefLayerTopOffset} \quad \text{(G-231)}$$
$$\text{dSW = ScaledRefLayerRightOffset − refPicScaledRefLayerRightOffset + dOX} \quad \text{(G-232)}$$
$$\text{dSH = ScaledRefLayerBottomOffset − refPicScaledRefLayerBottomOffset + dOY} \quad \text{(G-233)}$$

6. The variables scaleX and scaleY are derived by:

$$\text{scaleX = (((scaledW + dSW) << 16) + (refLayerW >> 1)) / refLayerW} \quad \text{(G-234)}$$
$$\text{scaleY = (((scaledH + dSH) << 16) + (refLayerH >> 1)) / refLayerH} \quad \text{(G-235)}$$

7. The motion vector aMv is scaled by:

$$\text{aMv[0] = (aMv[0] * scaleX + 32768) >> 16} \quad \text{(G-236)}$$
$$\text{aMv[1] = (aMv[1] * scaleY + 32768) >> 16} \quad \text{(G-237)}$$

8. When CroppingChangeFlag is equal to 1, the motion vector aMv is modified by applying the following ordered steps:

   a. The inverse macroblock scanning process as specified in subclause 6.4.1 is invoked with CurrMbAddr as the input and the output is a luma location (xMbPic, yMbPic). For this invocation of the process specified in subclause 6.4.1, the current macroblock is treated as field macroblock when fieldMbFlag is equal to 1, and it is treated as frame macroblock when fieldMbFlag is equal to 0.

b. The luma location (xFrm, yFrm) is derived by:

$$xFrm = (xMbPic + (4 * x + 1)) \quad (G-238)$$
$$yFrm = (yMbPic + (4 * y + 1) * (1 + fieldMbFlag - field\_pic\_flag)) * (1 + field\_pic\_flag) \quad (G-239)$$

c. The variables scaleX and scaleY are modified by:

$$scaleX = (((4 * dSW) << 16) + (scaledW >> 1)) / scaledW \quad (G-240)$$
$$scaleY = (((4 * dSH) << 16) + (scaledH >> 1)) / scaledH \quad (G-241)$$

d. The motion vector aMv is modified by:

$$aMv[0] += (((xFrm - ScaledRefLayerLeftOffset) * scaleX + 32768) >> 16) - 4 * dOX \quad (G-242)$$
$$aMv[1] += (((yFrm - ScaledRefLayerTopOffset) * scaleY + 32768) >> 16) - 4 * dOY \quad (G-243)$$

9. The motion vector predictor mvILPredLX[x, y] is derived by:

$$mvILPredLX[x, y][0] = aMv[0] \quad (G-244)$$
$$mvILPredLX[x, y][1] = aMv[1] / (1 + fieldMbFlag) \quad (G-245)$$

For each 8x8 block indexed by xP, yP = 0..1 and for X being replaced by 0 or 1, the reference index predictor refIdxILPredLX[xP, yP] is set equal to tempRefIdxPredLX[2 * xP, 2 * yP], and when RestrictedSpatialResolutionChangeFlag is equal to 0, the following ordered steps are specified:

1. The 4x4 blocks indexed by xS, yS = 0..1 of the current 8x8 block are processed in increasing order of (2 * yS + xS), and for each 4x4 block, the reference index predictor refIdxILPredLX[xP, yP] is modified by:

$$refIdxILPredLX[xP, yP] = MinPositive(refIdxILPredLX[xP, yP], tempRefIdxPredLX[2 * xP + xS, 2 * yP + yS]) \quad (G-246)$$

   with

$$MinPositive(a, b) = \begin{cases} Min(a, b) & \text{if } a >= 0 \text{ and } b >= 0 \\ Max(a, b) & \text{otherwise} \end{cases} \quad (G-247)$$

2. The 4x4 blocks indexed by xS, yS = 0..1 of the current 8x8 block are processed in increasing order of (2 * yS + xS), and for each 4x4 block, when tempRefIdxPredLX[2 * xP + xS, 2 * yP + yS] is not equal to the reference index predictor refIdxILPredLX[xP, yP], the following applies:

   – If tempRefIdxPredLX[2 * xP + 1 − xS, 2 * yP + yS] is equal to refIdxILPredLX[xP, yP], the motion vector predictor mvILPredLX[2 * xP + xS, 2 * yP + yS] is modified by:

$$mvILPredLX[2 * xP + xS, 2 * yP + yS] = mvILPredLX[2 * xP + 1 - xS, 2 * yP + yS] \quad (G-248)$$

   – Otherwise, if tempRefIdxPredLX[2 * xP + xS, 2 * yP + 1 − yS] is equal to refIdxILPredLX[xP, yP], the motion vector predictor mvILPredLX[2 * xP + xS, 2 * yP + yS] is modified by:

$$mvILPredLX[2 * xP + xS, 2 * yP + yS] = mvILPredLX[2 * xP + xS, 2 * yP + 1 - yS] \quad (G-249)$$

   – Otherwise (tempRefIdxPredLX[2 * xP + 1 − xS, 2 * yP + 1 − yS] is equal to refIdxILPredLX[xP, yP]), the motion vector predictor mvILPredLX[2 * xP + xS, 2 * yP + yS] is modified by:

$$mvILPredLX[2 * xP + xS, 2 * yP + yS] = mvILPredLX[2 * xP + 1 - xS, 2 * yP + 1 - yS] \quad (G-250)$$

NOTE – The process specified above can also be applied when RestrictedSpatialResolutionChangeFlag is equal to 1, but in this case, the reference index predictor refIdxILPredLX[xP, yP] and the motion vector predictors mvILPredLX[2 * xP + xS, 2 * yP + yS] with xS, yS = 0..1 will not be modified.

When RestrictedSpatialResolutionChangeFlag is equal to 0, slice_type is equal to EB, and direct_8x8_inference_flag is equal to 1, for each 8x8 block indexed by xP, yP = 0..1 and for X being replaced by 0 or 1, the following ordered steps are specified:

1. The motion vector tempMv with components tempMv[0] and tempMv[1] is derived by:

$$\text{tempMv}[\ c\ ] = \text{mvILPredX}[\ 3 * \text{xP},\ 3 * \text{yP}\ ][\ c\ ] \quad \text{with } c = 0..1 \tag{G-251}$$

2. The array mvILPredLX is modified by:

$$\text{mvILPredLX}[\ 2 * \text{xP} + \text{xS},\ 2 * \text{yP} + \text{yS}\ ][\ c\ ] = \text{tempMv}[\ c\ ] \quad \text{with xS, yS} = 0..1 \text{ and } c = 0..1 \tag{G-252}$$

When RestrictedSpatialResolutionChangeFlag is equal to 0, for each 8x8 block indexed by xP, yP = 0..1, the motion vector predictor arrays mvILPredL0 and mvILPredL1 are modified by applying the following ordered steps:

1. The variable maxX is derived as follows:
   – If slice_type is equal to EB, maxX is set equal to 1.
   – Otherwise (slice_type is equal to EP), maxX is set equal to 0.
2. The variables xO and yO are set equal to (2 * xP) and (2 * yP), respectively.
3. The function mvDiff(mv_1, mv_2) of two motion vectors mv_1 and mv_2 is defined by

$$\text{mvDiff}(\ mv_1,\ mv_2\ ) = \text{Abs}(\ mv_1[\ 0\ ] - mv_2[\ 0\ ]\ ) + \text{Abs}(\ mv_1[\ 1\ ] - mv_2[\ 1\ ]\ ) \tag{G-253}$$

4. The variable subPartSize is derived as follows:
   – If for X = 0..maxX, all of the following conditions are true, subPartSize is set equal to 8x8.
     – mvDiff(mvILPredLX[xO, yO], mvILPredLX[xO + 1, yO]) is less than or equal to 1
     – mvDiff(mvILPredLX[xO, yO], mvILPredLX[xO, yO + 1]) is less than or equal to 1
     – mvDiff(mvILPredLX[xO, yO], mvILPredLX[xO + 1, yO + 1]) is less than or equal to 1
   – Otherwise, if for X = 0..maxX, all of the following conditions are true, subPartSize is set equal to 8x4.
     – mvDiff(mvILPredLX[xO, yO], mvILPredLX[xO + 1, yO]) is less than or equal to 1
     – mvDiff(mvILPredLX[xO, yO + 1], mvILPredLX[xO + 1, yO + 1]) is less than or equal to 1
   – Otherwise, if for X = 0..maxX, all of the following conditions are true, subPartSize is set equal to 4x8.
     – mvDiff(mvILPredLX[xO, yO], mvILPredLX[xO, yO + 1]) is less than or equal to 1
     – mvDiff(mvILPredLX[xO + 1, yO], mvILPredLX[xO + 1, yO + 1]) is less than or equal to 1
   – Otherwise, subPartSize is set equal to 4x4.
5. When subPartSize is not equal to 4x4, for X = 0..maxX, the motion vectors tempMvALX and tempMvBLX (when subPartSize is equal to 8x4 or 4x8) are derived as follows:
   – If subPartSize is equal to 8x8, tempMvALX is derived by

$$\begin{aligned}\text{tempMvALX}[\ c\ ] = (\ &\text{mvILPredLX}[\ \text{xO},\ \text{yO}\ ][\ c\ ] + \\ &\text{mvILPredLX}[\ \text{xO} + 1,\ \text{yO}\ ][\ c\ ] + \\ &\text{mvILPredLX}[\ \text{xO},\ \text{yO} + 1\ ][\ c\ ] + \\ &\text{mvILPredLX}[\ \text{xO} + 1,\ \text{yO} + 1\ ][\ c\ ] + 2\ ) >> 2 \quad \text{with } c = 0..1\end{aligned} \tag{G-254}$$

   – Otherwise, if subPartSize is equal to 8x4, tempMvALX and tempMvBLX are derived by

$$\begin{aligned}\text{tempMvALX}[\ c\ ] = (\ &\text{mvILPredLX}[\ \text{xO},\ \text{yO}\ ][\ c\ ] + \\ &\text{mvILPredLX}[\ \text{xO} + 1,\ \text{yO}\ ][\ c\ ] + 1\ ) >> 1 \quad \text{with } c = 0..1\end{aligned} \tag{G-255}$$

$$\begin{aligned}\text{tempMvBLX}[\ c\ ] = (\ &\text{mvILPredLX}[\ \text{xO},\ \text{yO} + 1\ ][\ c\ ] + \\ &\text{mvILPredLX}[\ \text{xO} + 1,\ \text{yO} + 1\ ][\ c\ ] + 1\ ) >> 1 \quad \text{with } c = 0..1\end{aligned} \tag{G-256}$$

– Otherwise (subPartSize is equal to 4x8), tempMvALX and tempMvBLX are derived by

$$\begin{aligned}\text{tempMvALX}[\ c\ ] = (\ &\text{mvILPredLX}[\ xO,\ yO\ ][\ c\ ] + \\ &\text{mvILPredLX}[\ xO,\ yO + 1\ ][\ c\ ] + 1\ ) >> 1 \qquad \text{with } c = 0..1\end{aligned} \tag{G-257}$$

$$\begin{aligned}\text{tempMvBLX}[\ c\ ] = (\ &\text{mvILPredLX}[\ xO + 1,\ yO\ ][\ c\ ] + \\ &\text{mvILPredLX}[\ xO + 1,\ yO + 1\ ][\ c\ ] + 1\ ) >> 1 \qquad \text{with } c = 0..1\end{aligned} \tag{G-258}$$

6. When subPartSize is not equal to 4x4, for X = 0..maxX, the motion vector predictor array mvILPredLX is modified as follows:

   – If subPartSize is equal to 8x8, the array mvILPredLX is modified by

   $$\text{mvILPredLX}[\ xO + xS,\ yO + yS\ ][\ c\ ] = \text{tempMvALX}[\ c\ ] \quad \text{with } xS,\ yS = 0..1 \text{ and } c = 0..1 \tag{G-259}$$

   – Otherwise, if subPartSize is equal to 8x4, the array mvILPredLX is modified by

   $$\text{mvILPredLX}[\ xO + xS,\ yO\ ][\ c\ ] = \text{tempMvALX}[\ c\ ] \qquad \text{with } xS = 0..1 \text{ and } c = 0..1 \tag{G-260}$$
   $$\text{mvILPredLX}[\ xO + xS,\ yO + 1\ ][\ c\ ] = \text{tempMvBLX}[\ c\ ] \qquad \text{with } xS = 0..1 \text{ and } c = 0..1 \tag{G-261}$$

   – Otherwise (subPartSize is equal to 4x8), the array mvILPredLX is modified by

   $$\text{mvILPredLX}[\ xO,\ yO + yS\ ][\ c\ ] = \text{tempMvALX}[\ c\ ] \qquad \text{with } yS = 0..1 \text{ and } c = 0..1 \tag{G-262}$$
   $$\text{mvILPredLX}[\ xO + 1,\ yO + yS\ ][\ c\ ] = \text{tempMvBLX}[\ c\ ] \qquad \text{with } yS = 0..1 \text{ and } c = 0..1 \tag{G-263}$$

G.8.6.1.3 Derivation process for inter-layer predictors for P and B macroblock and sub-macroblock types

This process is only invoked when slice_type is equal to EP or EB.

Inputs to this process are:

– two 2x2 arrays refIdxILPredL0 and refIdxILPredL1 specifying predictors for the reference indices of the current macroblock,

– two 4x4x2 arrays mvILPredL0 and mvILPredL1 specifying predictors for the motion vectors of the current macroblock.

Outputs of this process are:

– a variable mbTypeILPred specifying a predictor for the macroblock type of the current macroblock,

– a list subMbTypeILPred with 4 elements specifying predictors for sub-macroblock types of the current macroblock.

The variable maxX is derived as follows:

– If slice_type is equal to EB, maxX is set equal to 1.

– Otherwise (slice_type is equal to EP), maxX is set equal to 0.

The macroblock type predictor mbTypeILPred is derived by applying the following ordered steps:

1. The variable partitionSize is derived as follows:

   – If for X = 0..maxX, all of the following conditions are true, partitionSize is set equal to 16x16.

     – all elements refIdxILPredLX[x, y] with x, y = 0..1 are the same

     – all elements mvILPredLX[x, y] with x, y = 0..3 are the same

   – Otherwise, if for X = 0..maxX, all of the following conditions are true, partitionSize is set equal to 16x8.

     – refIdxILPredLX[0, 0] is equal to refIdxILPredLX[1, 0]

     – refIdxILPredLX[0, 1] is equal to refIdxILPredLX[1, 1]

     – all elements mvILPredLX[x, y] with x = 0..3 and y = 0..1 are the same

     – all elements mvILPredLX[x, y] with x = 0..3 and y = 2..3 are the same

   – Otherwise, if for X = 0..maxX, all of the following conditions are true, partitionSize is set equal to 8x16.

     – refIdxILPredLX[0, 0] is equal to refIdxILPredLX[0, 1]

– refIdxILPredLX[1, 0] is equal to refIdxILPredLX[1, 1]

– all elements mvILPredLX[x, y] with x = 0..1 and y = 0..3 are the same

– all elements mvILPredLX[x, y] with x = 2..3 and y = 0..3 are the same

– Otherwise, partitionSize is set equal to 8x8.

2. When slice_type is equal to EB and partitionSize is not equal to 8x8, the variable partPredModeA is derived by

$$\text{partPredModeA} = ((\text{refIdxILPredL1}[\ 0, 0\ ] >= 0)\ ?\ 2 : 0) + ((\text{refIdxILPredL0}[\ 0, 0\ ] >= 0)\ ?\ 1 : 0) \quad \text{(G-264)}$$

3. When slice_type is equal to EB and partitionSize is equal to 16x8 or 8x16, the variable partPredModeB is derived by

$$\text{partPredModeB} = ((\text{refIdxILPredL1}[\ 1, 1\ ] >= 0)\ ?\ 2 : 0) + ((\text{refIdxILPredL0}[\ 1, 1\ ] >= 0)\ ?\ 1 : 0) \quad \text{(G-265)}$$

4. Depending on slice_type, partitionSize, partPredModeA (when applicable), and partPredModeB (when applicable), the macroblock type predictor mbTypeILPred is derived as specified in Table G-7.

All elements subMbTypeILPred[mbPartIdx] of the list subMbTypeILPred with mbPartIdx = 0..3 are marked as "unspecified".

When mbTypeILPred is equal to P_8x8 or B_8x8, each element subMbTypeILPred[mbPartIdx] with mbPartIdx = 0..3 is modified by applying the following ordered steps:

1. The coordinate offset (xO, yO) is set equal to (2 * (mbPartIdx % 2), 2 * (mbPartIdx / 2)).

2. The variable subPartitionSize is derived as follows:

   – If for X = 0..maxX, all elements mvILPredLX[xO + xS, yO + yS] with xS, yS = 0..1 are the same, subPartitionSize is set equal to 8x8.

   – Otherwise, if for X = 0..maxX, mvILPredLX[xO, yO] is equal to mvILPredLX[xO + 1, yO] and mvILPredLX[xO, yO + 1] is equal to mvILPredLX[xO + 1, yO + 1], subPartitionSize is set equal to 8x4.

   – Otherwise, if for X = 0..maxX, mvILPredLX[xO, yO] is equal to mvILPredLX[xO, yO + 1] and mvILPredLX[xO + 1, yO] is equal to mvILPredLX[xO + 1, yO + 1], subPartitionSize is set equal to 4x8.

   – Otherwise, subPartitionSize is set equal to 4x4.

3. When slice_type is equal to EB, the variable partPredMode is derived by

$$\text{partPredMode} = ((\text{refIdxILPredL1}[\ xO / 2, yO / 2\ ] >= 0)\ ?\ 2 : 0) + ((\text{refIdxILPredL0}[\ xO / 2, yO / 2\ ] >= 0)\ ?\ 1 : 0) \quad \text{(G-266)}$$

4. Depending on slice_type, subPartitionSize, and partPredMode (when applicable), the sub-macroblock type predictor subMbTypeILPred[mbPartIdx] is derived as specified in Table G-8.

Table G-7 – Macroblock type predictors mbTypeILPred

slice_type	partitionSize	partPredModeA	partPredModeB	mbTypeILPred	slice_type	partitionSize	partPredModeA	partPredModeB	mbTypeILPred
EB	16x16	1	na	B_L0_16x16	EB	16x8	2	3	B_L1_Bi_16x8
EB	16x16	2	na	B_L1_16x16	EB	8x16	2	3	B_L1_Bi_8x16
EB	16x16	3	na	B_Bi_16x16	EB	16x8	3	1	B_Bi_L0_16x8
EB	16x8	1	1	B_L0_L0_16x8	EB	8x16	3	1	B_Bi_L0_8x16
EB	8x16	1	1	B_L0_L0_8x16	EB	16x8	3	2	B_Bi_L1_16x8
EB	16x8	2	2	B_L1_L1_16x8	EB	8x16	3	2	B_Bi_L1_8x16
EB	8x16	2	2	B_L1_L1_8x16	EB	16x8	3	3	B_Bi_Bi_16x8
EB	16x8	1	2	B_L0_L1_16x8	EB	8x16	3	3	B_Bi_Bi_8x16
EB	8x16	1	2	B_L0_L1_8x16	EB	8x8	na	na	B_8x8
EB	16x8	2	1	B_L1_L0_16x8	EP	16x16	na	na	P_L0_16x16
EB	8x16	2	1	B_L1_L0_8x16	EP	16x8	na	na	P_L0_L0_16x8
EB	16x8	1	3	B_L0_Bi_16x8	EP	8x16	na	na	P_L0_L0_8x16
EB	8x16	1	3	B_L0_Bi_8x16	EP	8x8	na	na	P_8x8

Table G-8 – Sub-macroblock type predictors subMbTypeILPred[mbPartIdx]

slice_type	subPartitionSize	partPredMode	subMbTypeILPred [mbPartIdx]	slice_type	subPartitionSize	partPredMode	subMbTypeILPred [mbPartIdx]
EB	8x8	1	B_L0_8x8	EB	4x8	3	B_Bi_4x8
EB	8x8	2	B_L1_8x8	EB	4x4	1	B_L0_4x4
EB	8x8	3	B_Bi_8x8	EB	4x4	2	B_L1_4x4
EB	8x4	1	B_L0_8x4	EB	4x4	3	B_Bi_4x4
EB	8x4	2	B_L1_8x4	EP	8x8	na	P_L0_8x8
EB	8x4	3	B_Bi_8x4	EP	8x4	na	P_L0_8x4
EB	4x8	1	B_L0_4x8	EP	4x8	na	P_L0_4x8
EB	4x8	2	B_L1_4x8	EP	4x4	na	P_L0_4x4

G.8.6.2 Resampling process for intra samples

Inputs to this process are:

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a one-dimensional array refLayerSliceIdc with RefLayerPicSizeInMbs elements specifying slice identifications for the macroblocks of the reference layer representation,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying macroblock types for the macroblocks of the reference layer representation,

– a $(\text{RefLayerPicWidthInSamples}_L)\text{x}(\text{RefLayerPicHeightInSamples}_L)$ array $\text{refLayerPicSamples}_L$ of luma samples for the reference layer representation,

– a $(\text{PicWidthInSamples}_L)\text{x}(\text{PicHeightInSamples}_L)$ array picSamples_L of luma samples,

– when ChromaArrayType is not equal to 0, two $(\text{RefLayerPicWidthInSamples}_C)\text{x}(\text{RefLayerPicHeightInSamples}_C)$ arrays $\text{refLayerPicSamples}_{Cb}$ and $\text{refLayerPicSamples}_{Cr}$ of chroma samples for the reference layer representation,

– when ChromaArrayType is not equal to 0, two $(\text{PicWidthInSamples}_C)\text{x}(\text{PicHeightInSamples}_C)$ arrays picSamples_{Cb} and picSamples_{Cr} of chroma samples.

Outputs of this process are:

– a modified version of the array picSamples_L of luma samples,

– when ChromaArrayType is not equal to 0, modified versions of the arrays picSamples_{Cb} and picSamples_{Cr} of chroma samples.

The resampling process for intra samples of a macroblock colour component as specified in subclause G.8.6.2.1 is invoked with chromaFlag equal to 0, mbW equal to 16, mbH equal to 16, fieldMbFlag, $\text{refLayerPicSamples}_L$, refLayerSliceIdc, refLayerFieldMbFlag, and refLayerMbType as the inputs and the output is the 16x16 array mbPred_L of Intra_Base prediction samples for the luma component of the current macroblock.

When ChromaArrayType is not equal to 0, for CX being replaced by Cb and Cr, the resampling process for intra samples of a macroblock colour component as specified in subclause G.8.6.2.1 is invoked with chromaFlag equal to 1, mbW equal to MbWidthC, mbH equal to MbHeightC, fieldMbFlag, $\text{refLayerPicSamples}_{CX}$, refLayerSliceIdc, refLayerFieldMbFlag, and refLayerMbType as the inputs and the output is the (MbWidthC)x(MbHeightC) array mbPred_{CX} of Intra_Base prediction samples for the CX component of the current macroblock.

The picture sample array construction process as specified in subclause G.8.5.4.1 is invoked with fieldMbFlag, mbPred_L, picSamples_L and, when ChromaArrayType is not equal to 0, mbPred_{Cb}, mbPred_{Cr}, picSamples_{Cb}, and picSamples_{Cr} as the inputs and the outputs are a modified version of picSamples_L and, when ChromaArrayType is not equal to 0, modified versions of picSamples_{Cb}, and picSamples_{Cr}.

G.8.6.2.1 Resampling process for intra samples of a macroblock colour component

Inputs to this process are:

– a variable chromaFlag specifying whether the luma or a chroma component is subject to the resampling process,

– two variables mbW and mbH specifying the width and height, respectively, of a macroblock for the considered colour component,

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– an array refLayerPicSamples, which is a $(\text{RefLayerPicWidthInSamples}_L)\text{x}(\text{RefLayerPicHeightInSamples}_L)$ array containing constructed intra luma sample values for the reference layer representation when chromaFlag is equal to 0 or a $(\text{RefLayerPicWidthInSamples}_C)\text{x}(\text{RefLayerPicHeightInSamples}_C)$ array containing constructed intra chroma sample values for the reference layer representation when chromaFlag is equal to 1,

– a one-dimensional array refLayerSliceIdc with RefLayerPicSizeInMbs elements specifying slice identifications for the macroblocks of the reference layer representation,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying macroblock types for the macroblocks of the reference layer representation.

Output of this process is an (mbW)x(mbH) array mbPred of Intra_Base prediction samples.

The variable botFieldFlag is derived as follows:

– If RefLayerFrameMbsOnlyFlag is equal to 1, botFieldFlag is set equal to 0.

– Otherwise, if field_pic_flag is equal to 1, botFieldFlag is set equal to bottom_field_flag.

– Otherwise, if RefLayerFieldPicFlag is equal to 1, botFieldFlag is set equal to RefLayerBottomFieldFlag.

– Otherwise, if fieldMbFlag is equal to 1, botFieldFlag is set equal to (CurrMbAddr % 2).

– Otherwise, botFieldFlag is set equal to 0.

The variable frameBasedResamplingFlag is derived as follows:

– If all of the following conditions are true, frameBasedResamplingFlag is set equal to 1:
  – RefLayerFrameMbsOnlyFlag is equal to 1,
  – frame_mbs_only_flag is equal to 1.

– Otherwise, frameBasedResamplingFlag is set equal to 0.

The variable topAndBotResamplingFlag is derived as follows:

– If all of the following conditions are true, topAndBotResamplingFlag is set equal to 1:
  – RefLayerFrameMbsOnlyFlag is equal to 0,
  – RefLayerFieldPicFlag is equal to 0,
  – frame_mbs_only_flag is equal to 0,
  – fieldMbFlag is equal to 0.

– Otherwise, topAndBotResamplingFlag is set equal to 0.

The variable botFieldFrameMbsOnlyRefFlag is derived as follows:

– If RefLayerFrameMbsOnlyFlag is equal to 1, fieldMbFlag is equal to 1, and any of the following conditions are true, botFieldFrameMbsOnlyRefFlag is set equal to 1:
  – field_pic_flag is equal to 1 and bottom_field_flag is equal to 1,
  – field_pic_flag is equal to 0 and (CurrMbAddr % 2) is equal to 1,

– Otherwise, botFieldFrameMbsOnlyRefFlag is set equal to 0.

The variable filteringModeFlag is derived as follows:

– If chromaFlag is equal to 0 or ChromaArrayType is equal to 3, filteringModeFlag is set equal to 0.

– Otherwise (chromaFlag is equal to 1 and ChromaArrayType is not equal to 3), filteringModeFlag is set equal to 1.

The array predArray is derived as specified in the following.

– If botFieldFrameMbsOnlyRefFlag is equal to 1, the following ordered steps are specified:
  1. The reference layer sample array construction process prior to intra resampling as specified in subclause G.8.6.2.2 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag, refLayerPicSamples, refLayerSliceIdc, refLayerFieldMbFlag, and refLayerMbType as the inputs and the outputs are the variables refArrayW, refArrayH, the array refSampleArray of reference layer sample values, and the variables xOffset and yOffset.
  2. The variable yBorder is set equal to (2 − chromaFlag).
  3. The interpolation process for Intra_Base prediction as specified in subclause G.8.6.2.3 is invoked with filteringModeFlag, chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag, fldPrdInFrmMbFlag equal to 0, yBorder, refArrayW, refArrayH, refSampleArray, xOffset, and yOffset as the inputs and the output is the (mbW)x(mbH + 2 * yBorder) array topFldPredArray of top field prediction samples.
  4. The vertical interpolation process for Intra_Base prediction as specified in subclause G.8.6.2.4 is invoked with filteringModeFlag, chromaFlag, mbW, mbH, botFieldFlag, yBorder, frameMbFlag equal to 0, and topFldPredArray as the inputs and the output is the (mbW)x(mbH) array mbPred of Intra_Base prediction samples.

– Otherwise, if frameBasedResamplingFlag is equal to 1 or fieldMbFlag is equal to 1, the following ordered steps are specified:

1. The reference layer sample array construction process prior to intra resampling as specified in subclause G.8.6.2.2 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag, refLayerPicSamples, refLayerSliceIdc, refLayerFieldMbFlag, and refLayerMbType as the inputs and the outputs are the variables refArrayW, refArrayH, the array refSampleArray of reference layer sample values, and the variables xOffset and yOffset.

2. The interpolation process for Intra_Base prediction as specified in subclause G.8.6.2.3 is invoked with filteringModeFlag, chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag, fldPrdInFrmMbFlag equal to 0, yBorder equal to 0, refArrayW, refArrayH, refSampleArray, xOffset, and yOffset as the inputs and the output is the (mbW)x(mbH) array mbPred of Intra_Base prediction samples.

– Otherwise, if topAndBotResamplingFlag is equal to 0, the following ordered steps are specified:

1. The reference layer sample array construction process prior to intra resampling as specified in subclause G.8.6.2.2 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag, refLayerPicSamples, refLayerSliceIdc, refLayerFieldMbFlag, and refLayerMbType as the inputs and the outputs are the variables refArrayW, refArrayH, the array refSampleArray of reference layer sample values, and the variables xOffset and yOffset.

2. The variable yBorder is set equal to (2 − chromaFlag).

3. The interpolation process for Intra_Base prediction as specified in subclause G.8.6.2.3 is invoked with filteringModeFlag, chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag, fldPrdInFrmMbFlag equal to 1, yBorder, refArrayW, refArrayH, refSampleArray, xOffset, and yOffset as the inputs and the output is the (mbW)x(mbH / 2 + 2* yBorder) array fieldPredArray of field prediction samples.

4. The vertical interpolation process for Intra_Base prediction as specified in subclause G.8.6.2.4 is invoked with filteringModeFlag, chromaFlag, mbW, mbH, botFieldFlag, yBorder, frameMbFlag equal to 1, and fieldPredArray as the inputs and the output is the (mbW)x(mbH) array mbPred of Intra_Base prediction samples.

– Otherwise (topAndBotResamplingFlag is equal to 1), the following ordered steps are specified:

1. The reference layer sample array construction process prior to intra resampling as specified in subclause G.8.6.2.2 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag equal to 0, refLayerPicSamples, refLayerSliceIdc, refLayerFieldMbFlag, and refLayerMbType as the inputs and the outputs are the variables refArrayTopW, refArrayTopH, the array refSampleArrayTop of top field reference layer sample values, and the variables xOffsetTop and yOffsetTop.

2. The interpolation process for Intra_Base prediction as specified in subclause G.8.6.2.3 is invoked with filteringModeFlag, chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag equal to 0, fldPrdInFrmMbFlag equal to 1, yBorder equal to 0, refArrayTopW, refArrayTopH, refSampleArrayTop, xOffsetTop, and yOffsetTop as the inputs and the output is the (mbW)x(mbH / 2) array topFieldPredArray of top field prediction samples.

3. The reference layer sample array construction process prior to intra resampling as specified in subclause G.8.6.2.2 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag equal to 1, refLayerPicSamples, refLayerSliceIdc, refLayerFieldMbFlag, and refLayerMbType as the inputs and the outputs are the variables refArrayBotW, refArrayBotH, the array refSampleArrayBot of bottom field reference layer sample values, and the variables xOffsetBot and yOffsetBot.

4. The interpolation process for Intra_Base prediction as specified in subclause G.8.6.2.3 is invoked with filteringModeFlag, chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag equal to 1, fldPrdInFrmMbFlag equal to 1, yBorder equal to 0, refArrayBotW, refArrayBotH, refSampleArrayBot, xOffsetBot, and yOffsetBot as the inputs and the output is the (mbW)x(mbH / 2) array botFieldPredArray of bottom field prediction samples.

5. Each sample predArray[x, y] with x = 0..(mbW − 1) and y = 0..(mbH − 1) of the array mbPred of Intra_Base prediction samples is derived by:

$$\text{mbPred}[\,x, y\,] = (\,(\,(\,y \,\%\, 2\,) == 0\,)\ ?\ \text{topFieldPredArray}[\,x, y >> 1\,] : \text{botFieldPredArray}[\,x, y >> 1\,]\,) \tag{G-267}$$

G.8.6.2.2 Reference layer sample array construction process prior to intra resampling

Inputs to this process are:

– a variable chromaFlag specifying whether the luma or a chroma component is subject to the resampling process,

– two variables mbW and mbH specifying the width and height, respectively, of a macroblock for the considered colour component,

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a variable botFieldFlag specifying whether a top or a bottom field is subject to the resampling process (when RefLayerFrameMbsOnlyFlag is equal to 0 or frame_mbs_only_flag is equal to 0),

– an array refLayerPicSamples, which is a ($\text{RefLayerPicWidthInSamples}_L$)x($\text{RefLayerPicHeightInSamples}_L$) array containing constructed intra luma sample values for the reference layer representation when chromaFlag is equal to 0 or a ($\text{RefLayerPicWidthInSamples}_C$)x($\text{RefLayerPicHeightInSamples}_C$) array containing constructed intra chroma sample values for the reference layer representation when chromaFlag is equal to 1,

– a one-dimensional array refLayerSliceIdc with RefLayerPicSizeInMbs elements specifying slice identifications for the macroblocks of the reference layer representation,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying macroblock types for the macroblocks of the reference layer representation.

Outputs of this process are:

– two variables refArrayW and refArrayH specifying the width and height, respectively, of the constructed array of reference layer sample values,

– a (refArrayW)x(refArrayH) array refSampleArray of reference layer sample values,

– two variables xOffset and yOffset specifying the x and y coordinate, respectively, of the reference layer sample location that corresponds to the sample refSampleArray[0, 0] of the array refSampleArray.

The variables refW, refH, refMbW, refMbH, xOffset, yOffset, refArrayW, refArrayH, xMin, yMin, xMax, yMax, yRefScale, and yRefAdd are derived as specified in the following ordered steps:

1. The derivation process for reference layer sample locations in resampling as specified in subclause G.6.3 is invoked with chromaFlag, the sample location (0, 0), fieldMbFlag, and botFieldFlag as the inputs and the output is the sample location (xRefMin16, yRefMin16) in units of 1/16-th sample.

2. The derivation process for reference layer sample locations in resampling as specified in subclause G.6.3 is invoked with chromaFlag, the sample location (mbW − 1, mbH − 1), fieldMbFlag, and botFieldFlag as the inputs and the output is the sample location (xRefMax16, yRefMax16) in units of 1/16-th sample.

3. With Z being replaced by L for chromaFlag equal to 0 and C for chromaFlag equal to 1, the variables refW, refH, refMbW, and refMbH are derived by:

$$\text{refW} = \text{RefLayerPicWidthInSamples}_Z \quad \text{(G-268)}$$
$$\text{refH} = \text{RefLayerPicHeightInSamples}_Z \quad \text{(G-269)}$$
$$\text{refMbW} = ((\text{chromaFlag} == 0) \ ? \ 16 : \text{RefLayerMbWidthC}) \quad \text{(G-270)}$$
$$\text{refMbH} = ((\text{chromaFlag} == 0) \ ? \ 16 : \text{RefLayerMbHeightC}) \quad \text{(G-271)}$$

4. The variables xOffset, yOffset, refArrayW, and refArrayH are derived by:

$$\text{xOffset} = (((\text{xRefMin16} - 64) >> 8) << 4) - (\text{refMbW} >> 1) \quad \text{(G-272)}$$
$$\text{yOffset} = (((\text{yRefMin16} - 64) >> 8) << 4) - (\text{refMbH} >> 1) \quad \text{(G-273)}$$
$$\text{refArrayW} = (((\text{xRefMax16} + 79) >> 8) << 4) + 3 * (\text{refMbW} >> 1) - \text{xOffset} \quad \text{(G-274)}$$
$$\text{refArrayH} = (((\text{yRefMax16} + 79) >> 8) << 4) + 3 * (\text{refMbH} >> 1) - \text{yOffset} \quad \text{(G-275)}$$

NOTE 1 – The derived array size might be larger than the array size that is actually required by the interpolation process for Intra_Base prediction specified in subclause G.8.6.2.3.

5. The variables xMin, yMin, xMax, and yMax are derived by:

$$\text{xMin} = (\text{xRefMin16} >> 4) - \text{xOffset} \quad \text{(G-276)}$$
$$\text{yMin} = (\text{yRefMin16} >> 4) - \text{yOffset} \quad \text{(G-277)}$$
$$\text{xMax} = ((\text{xRefMax16} + 15) >> 4) - \text{xOffset} \quad \text{(G-278)}$$
$$\text{yMax} = ((\text{yRefMax16} + 15) >> 4) - \text{yOffset} \quad \text{(G-279)}$$

6. The variables yRefScale and yRefAdd are derived as follows:
   - If RefLayerFrameMbsOnlyFlag is equal to 1 or RefLayerFieldPicFlag is equal to 1, yRefScale is set equal to 1 and yRefAdd is set equal to 0.
   - Otherwise (RefLayerFrameMbsOnlyFlag is equal to 0 and RefLayerFieldPicFlag is equal to 0), yRefScale is set equal to 2 and yRefAdd is set equal to botFieldFlag.

The variable refSliceIdcMb is marked as "not available".

When constrained_intra_resampling_flag is equal to 1, the variable y proceeds over the values (yMin + 1)..(yMax − 1) and for each value of y, the variable x proceeds over the values (xMin + 1)..(xMax − 1), and for each pair (x, y), the following ordered steps are specified:

1. A reference layer sample location (xRef, yRef) is derived by:

   xRef = Max(0, Min(refW − 1, x + xOffset)) (G-280)
   yRef = yRefScale * Max(0, Min(refH / yRefScale − 1, y + yOffset)) + yRefAdd (G-281)

2. The derivation process for reference layer slice and intra macroblock identifications as specified in subclause G.8.6.2.2.1 is invoked with the reference layer sample location (xRef, yRef), refMbW, refMbH, refLayerSliceIdc, refLayerFieldMbFlag, and refLayerMbType as the inputs and the outputs are the reference layer slice identification refSliceIdc and the variable refIntraMbFlag.

3. When refIntraMbFlag is equal to 1 and refSliceIdcMb is marked as "not available", the variable refSliceIdcMb is marked as "available" and set equal to refSliceIdc.

When constrained_intra_resampling_flag is equal to 1, the following ordered steps are specified:

1. The variable useIntraPredFlag is set equal to 0.

2. For x proceeding over the values 0..15 and y proceeding over the values 0..15, the following ordered steps are specified:

   a. The derivation process for reference layer macroblocks as specified in subclause G.6.1 is invoked with the luma location (x, y), fieldMbFlag, refLayerFieldMbFlag, and refLayerMbType as the inputs and the outputs are assigned to mbAddrRefLayer and (xRef, yRef).

   b. When refLayerMbType[mbAddrRefLayer] is equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL, the variable useIntraPredFlag is set equal to 1.

3. When useIntraPredFlag is equal to 1, it is a requirement of bitstream conformance that the bitstream shall not contain data that result in refSliceIdcMb being marked as "not available".

Each sample refSampleArray[x, y] with x = 0..(refArrayW − 1) and y = 0..(refArrayH − 1) is derived as specified in the following ordered steps:

1. A reference layer sample location (xRef, yRef) is derived by

   xRef = Max(0, Min(refW − 1, x + xOffset)) (G-282)
   yRef = yRefScale * Max(0, Min(refH / yRefScale − 1, y + yOffset)) + yRefAdd (G-283)

2. The derivation process for reference layer slice and intra macroblock identifications as specified in subclause G.8.6.2.2.1 is invoked with the reference layer sample location (xRef, yRef), refMbW, refMbH, refLayerSliceIdc, refLayerFieldMbFlag, and refLayerMbType as the inputs and the outputs are the reference layer slice identification refSliceIdc and the variable refIntraMbFlag.

3. When constrained_intra_resampling_flag is equal to 1, refIntraMbFlag is equal to 1, x is greater than xMin, x is less than xMax, y is greater than yMin, and y is less than yMax, it is a requirement of bitstream conformance that the bitstream shall not contain data that result in refSliceIdc being not equal to refSliceIdcMb.

   NOTE 2 – This constraint specifies that a macroblock cannot be coded with base_mode_flag equal to 1 when it covers intra-coded macroblocks of more than one slice in the reference layer representation, constrained_intra_resampling_flag is equal to 1, and either the inferred macroblock type is equal to I_BL or the conditions for invoking the intra-inter prediction combination process as specified in subclause G.8.4.2.2 are fulfilled.

4. Depending on refIntraMbFlag, constrained_intra_resampling_flag, and refSliceIdc, the following applies:
   - If any of the following conditions are true, the sample refSampleArray[x, y] is marked as "not available for Intra_Base prediction" and its value is set equal to 0:
     - refIntraMbFlag is equal to 0,
     - constrained_intra_resampling_flag is equal to 1 and refSliceIdcMb is marked as "not available",
     - constrained_intra_resampling_flag is equal to 1 and refSliceIdc is not equal to refSliceIdcMb.
   - Otherwise, the sample refSampleArray[x, y] is marked as "available for Intra_Base prediction" and its value is derived by

$$\text{refSampleArray}[\, x, y \,] = \text{refLayerPicSamples}[\, \text{xRef}, \text{yRef} \,] \quad \text{(G-284)}$$

The construction process for not available sample values prior to intra resampling as specified in subclause G.8.6.2.2.2 is invoked with refMbW, refMbH, refArrayW, refArrayH, refSampleArray, xOffset, and yOffset as the inputs and the output is a modified version of the sample array refSampleArray.

G.8.6.2.2.1 Derivation process for reference layer slice and intra macroblock identifications

Inputs to this process are:

- a reference layer sample location (xRef, yRef) relative to the upper-left sample of the considered colour component of the reference layer picture,
- two variables refMbW and refMbH specifying the width and height, respectively, of a reference layer macroblock for the considered colour component,
- a one-dimensional array refLayerSliceIdc with RefLayerPicSizeInMbs elements specifying slice identifications for the macroblocks of the reference layer representation,
- a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,
- a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying macroblock types for the macroblocks of the reference layer representation.

Outputs of this process are:

- a reference layer slice identification refSliceIdc for the slice that covers the input reference layer sample location,
- a variable refIntraMbFlag specifying whether the reference layer macroblock that covers the input reference layer sample location is intra coded.

The reference layer macroblock address refMbAddr is derived as follows:

- If RefLayerMbaffFrameFlag is equal to 0, the variable refMbAddr is derived by

$$\text{refMbAddr} = (\, \text{yRef} / \text{refMbH} \,) * \text{RefLayerPicWidthInMbs} + (\, \text{xRef} / \text{refMbW} \,) \quad \text{(G-285)}$$

- Otherwise (RefLayerMbaffFrameFlag is equal to 1), the variable refMbAddr is derived as specified in the following ordered steps:
  1. A variable refMbAddrTop is derived by

$$\begin{aligned}\text{refMbAddrTop} = 2 * (\, (\, \text{yRef} / (\, 2 * \text{refMbH} \,) \,) * \text{RefLayerPicWidthInMbs} \\ + (\, \text{xRef} / \text{refMbW} \,) \,)\end{aligned} \quad \text{(G-286)}$$

  2. Depending on refLayerFieldMbFlag[refMbAddrTop], the variable refMbAddr is derived as follows:
     - If refLayerFieldMbFlag[refMbAddrTop] is equal to 0, the variable refMbAddr is derived by

$$\text{refMbAddr} = \text{refMbAddrTop} + (\, \text{yRef} \,\%\, (\, 2 * \text{refMbH} \,) \,) / \text{refMbH} \quad \text{(G-287)}$$

     - Otherwise (refLayerFieldMbFlag[refMbAddrTop] is equal to 1), the variable refMbAddr is derived by

$$\text{refMbAddr} = \text{refMbAddrTop} + (\, \text{yRef} \,\%\, 2 \,) \quad \text{(G-288)}$$

The reference layer slice identification refSliceIdc is set equal to refLayerSliceIdc[refMbAddr].

Depending on refLayerMbType[refMbAddr], the variable refIntraMbFlag is derived as follows:

– If refLayerMbType[refMbAddr] is equal to I_4x4, I_8x8, I_16x16, I_PCM, or I_BL, refIntraMbFlag is set equal to 1.

– Otherwise (refLayerMbType[refMbAddr] is not equal to I_4x4, I_8x8, I_16x16, I_PCM, or I_BL), refIntraMbFlag is set equal to 0.

G.8.6.2.2.2 Construction process for not available sample values prior to intra resampling

Inputs to this process are:

– two variables refMbW and refMbH specifying the width and height, respectively, of a reference layer macroblock for the considered colour component,

– two variables refArrayW and refArrayH specifying the width and height, respectively, of the array of reference layer sample values,

– a (refArrayW)x(refArrayH) array refSampleArray of reference layer sample values,

– two variables xOffset and yOffset specifying the x and y coordinates, respectively, of the reference layer sample location that corresponds to the sample refSampleArray[0, 0] of the array refSampleArray.

Output of this process is a modified version of the array refSampleArray.

For each sample refSampleArray[x, y] with x = (refMbW / 2)..(refArrayW − refMbW / 2 − 1) and y = (refMbH / 2)..(refArrayH − refMbH / 2 − 1) that is marked as "not available for Intra_Base prediction", the following ordered steps are specified:

1. The sample location difference (xD, yD) and the variable yA are derived by

$$xR = (x + xOffset) \% refMbW \quad \text{(G-289)}$$
$$yR = (y + yOffset) \% refMbH \quad \text{(G-290)}$$

$$xD = ((xR >= refMbW / 2) \ ? \ (xR - refMbW) \ : \ (xR + 1)) \quad \text{(G-291)}$$
$$yD = ((yR >= refMbH / 2) \ ? \ (yR - refMbH) \ : \ (yR + 1)) \quad \text{(G-292)}$$

$$yA = yD - (refMbH / 2 + 1) * Sign(yD) \quad \text{(G-293)}$$

2. When any of the following conditions are true, yD is set equal to yA:

   – the sample refSampleArray[x, y − yD] is marked as "not available for Intra_Base prediction", the sample refSampleArray[x, y − yA] is marked as "available for Intra_Base prediction", and the sample refSampleArray[x − xD, y] is marked as "available for Intra_Base prediction",

   – all of the samples refSampleArray[x − xD, y], refSampleArray[x, y − yD], and refSampleArray[x − xD, y − yD] are marked as "not available for Intra_Base prediction" and any of the samples refSampleArray[x, y − yA] and refSampleArray[x − xD, y − yA] is marked as "available for Intra_base prediction",

   – Abs(yA) is less than Abs(yD) and any of the following conditions are true:

     – both samples refSampleArray[x, y − yD] and refSampleArray[x, y − yA] are marked as "available for Intra_Base prediction",

     – any of the samples refSampleArray[x, y − yD] and refSampleArray[x − xD, y − yD] is marked as "available for Intra_Base prediction", any of the samples refSampleArray[x, y − yA] and refSampleArray[x − xD, y − yA] is marked as "available for Intra_Base prediction", and the sample refSampleArray[x − xD, y] is marked as "not available for Intra_Base prediction".

   NOTE – The variable yD is never set equal to yA when RefLayerFrameMbsOnlyFlag is equal to 1 or RefLayerFieldPicFlag is equal to 1.

3. The sample value refSampleArray[x, y] is derived as follows:

   – If the sample refSampleArray[x − xD, y] and the sample refSampleArray[x, y − yD] are marked as "available for Intra_Base prediction", the following ordered steps are specified:

     a. A variable cornerSampleAvailableFlag is derived as follows:

        – If the sample refSampleArray[x − xD, y − yD] is marked as "available for Intra_Base prediction", the variable cornerSampleAvailableFlag is set equal to 1.

– Otherwise (the sample refSampleArray[x − xD, y − yD] is marked as "not available for Intra_Base prediction"), the variable cornerSampleAvailable is set equal to 0.

b. The diagonal construction process for not available sample values as specified in subclause G.8.6.2.2.2.1 is invoked with refArrayW, refArrayH, refSampleArray, the sample location difference (xD, yD), the sample location (x, y), and the variable cornerSampleAvailableFlag as the inputs and the output is the sample array refSampleArray with a modified sample value at sample location (x, y).

– Otherwise (the sample refSampleArray[x − xD, y] or the sample refSampleArray[x, y − yD] is marked as "not available for Intra_Base prediction"), the following applies:

  – If the sample refSampleArray[x − xD, y] is marked as "available for Intra_Base prediction", the sample value refSampleArray[x, y] is set equal to refSampleArray[x − xD, y].

  – Otherwise, if the sample refSampleArray[x, y − yD] is marked as "available for Intra_Base prediction", the sample value refSampleArray[x, y] is set equal to refSampleArray[x, y − yD].

  – Otherwise, if the sample refSampleArray[x − xD, y − yD] is marked as "available for Intra_Base prediction", the sample value refSampleArray[x, y] is set equal to refSampleArray[x − xD, y − yD].

  – Otherwise (the samples refSampleArray[x − xD, y], refSampleArray[x, y − yD], and refSampleArray[x − xD, y − yD] are marked as "not available for Intra_Base prediction"), the sample value refSampleArray[x, y] is not modified.

All samples refSampleArray[x, y] with x = 0..(refArrayW − 1) and y = 0..(refArrayH − 1) are marked as "available for Intra_Base prediction".

G.8.6.2.2.2.1 Diagonal construction process for not available sample values

Inputs to this process are:

– two variables refArrayW and refArrayH specifying the width and height, respectively, of the array of reference layer sample values,

– a (refArrayW)x(refArrayH) array p of reference layer sample values,

– a sample location difference (xD, yD),

– a sample location (x, y) inside the reference layer sample array refSampleArray,

– a variable cornerSampleAvailableFlag.

Output of this process is the sample array p with a modified sample value at sample location (x, y).

The variables diffHorVer and sgnXY are derived by

$$\text{diffHorVer} = \text{Abs}(\ xD\ ) - \text{Abs}(\ yD\ ) \qquad \text{(G-294)}$$
$$\text{sgnXY} = \text{Sign}(\ xD * yD\ ) \qquad \text{(G-295)}$$

When cornerSampleAvailableFlag is equal to 0, the following ordered steps are specified:

1. The variable cornerSample is set equal to p[x − xD, y − yD].

2. The sample location (xC, yC) is set equal to (x − xD + Sign(xD), y − yD + Sign(yD)) and the sample value p[x − xD, y − yD] is modified by

$$p[\ x - xD, y - yD\ ] = (\ p[\ x - xD, yC\ ] + p[\ xC, y - yD\ ] + 1\ ) >> 1 \qquad \text{(G-296)}$$

The sample value p[x, y] is derived as follows:

– If diffHorVer is greater than 0, the sample location (xC, yC) is set equal to (x − sgnXY * yD, y − yD) and the sample value p[x, y] is derived by

$$p[\ x, y\ ] = (\ p[\ xC - 1, yC\ ] + 2 * p[\ xC, yC\ ] + p[\ xC + 1, yC\ ] + 2\ ) >> 2 \qquad \text{(G-297)}$$

– Otherwise, if diffHorVer is less than 0, the sample location (xC, yC) is set equal to (x − xD, y − sgnXY * xD) and the sample value p[x, y] is derived by

$$p[\ x, y\ ] = (\ p[\ xC, yC - 1\ ] + 2 * p[\ xC, yC\ ] + p[\ xC, yC + 1\ ] + 2\ ) >> 2 \qquad \text{(G-298)}$$

– Otherwise (diffVerHor is equal to 0), the sample location (xC, yC) is set equal to (x − xD +Sign(xD), y − yD + Sign(yD)) and the sample value p[x, y] is derived by

$$p[x, y] = (p[xC, y - yD] + 2 * p[x - xD, y - yD] + p[x - xD, yC] + 2) >> 2 \quad \text{(G-299)}$$

When cornerSampleAvailableFlag is equal to 0, the sample value p[x − xD, y − yD] is set equal to cornerSample.

G.8.6.2.3 Interpolation process for Intra_Base prediction

Inputs to this process are:

– a variable filteringModeFlag specifying the interpolation method,

– a variable chromaFlag specifying whether the luma or a chroma component is subject to the resampling process,

– two variables mbW and mbH specifying the width and height, respectively, of a macroblock for the considered colour component,

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a variable botFieldFlag specifying whether a top or a bottom field is subject to the resampling process (when RefLayerFrameMbsOnlyFlag is equal to 0 or frame_mbs_only_flag is equal to 0),

– a variable fldPrdInFrmMbFlag specifying whether field prediction for a frame macroblock is applied,

– a variable yBorder specifying the vertical border for the output sample array predSamples,

– two variables refArrayW and refArrayH specifying the width and height, respectively, of the array of reference layer sample values,

– a (refArrayW)x(refArrayH) array refSampleArray of reference layer sample values,

– two variables xOffset and yOffset specifying the x and y coordinate, respectively, of the reference layer sample location that corresponds to the sample refSampleArray[0, 0] of the array refSampleArray.

Output of this process is an (mbW)x(mbH / (1 + fldPrdInFrmMbFlag) + 2 * yBorder) array predArray of interpolated sample values.

Table G-9 specifies the filter coefficients eF[p, x] with p = 0..15 and x = 0..3 of the luma interpolation filter eF for resampling in Intra_Base prediction.

Table G-9 – 16-phase luma interpolation filter for resampling in Intra_Base prediction

phase p	interpolation filter coefficients			
	eF[p, 0]	eF[p, 1]	eF[p, 2]	eF[p, 3]
0	0	32	0	0
1	−1	32	2	−1
2	−2	31	4	−1
3	−3	30	6	−1
4	−3	28	8	−1
5	−4	26	11	−1
6	−4	24	14	−2
7	−3	22	16	−3
8	−3	19	19	−3
9	−3	16	22	−3
10	−2	14	24	−4
11	−1	11	26	−4
12	−1	8	28	−3
13	−1	6	30	−3
14	−1	4	31	−2
15	−1	2	32	−1

Let tempArray be a (refArrayW)x(mbH / (1 + fldPrdInFrmMbFlag) + 2 * yBorder) array of samples. Each sample tempArray[x, y] with x = 0..(refArrayW − 1) and y = 0..(mbH / (1 + fldPrdInFrmMbFlag) + 2 * yBorder − 1) is derived as specified in the following ordered steps:

1. The variable yP is derived by

$$yP = (y - yBorder) * (1 + fldPrdInFrmMbFlag) + botFieldFlag \quad \text{(G-300)}$$

2. The derivation process for reference layer sample locations in resampling as specified in subclause G.6.3 is invoked with chromaFlag, the sample location (0, yP), fieldMbFlag, and botFieldFlag as the inputs and the output is the sample location (xRef16, yRef16) in units of 1/16-th sample.

   NOTE 1 – In this invocation of the process in subclause G.6.3, only the vertical component yRef16 of the sample location needs to be derived.

3. The variables yRef and yPhase are derived by

$$yRef = (yRef16 >> 4) - yOffset \quad \text{(G-301)}$$
$$yPhase = (yRef16 - 16 * yOffset) \% 16 \quad \text{(G-302)}$$

4. Depending on filteringModeFlag, the sample value tempArray[x, y] is derived as follows:

   – If filteringModeFlag is equal to 0, the sample value tempArray[x, y] is derived by

$$\begin{aligned} tempArray[x, y] = {} & eF[yPhase, 0] * refSampleArray[x, yRef - 1] + \\ & eF[yPhase, 1] * refSampleArray[x, yRef] + \\ & eF[yPhase, 2] * refSampleArray[x, yRef + 1] + \\ & eF[yPhase, 3] * refSampleArray[x, yRef + 2] \end{aligned} \quad \text{(G-303)}$$

   – Otherwise (filteringModeFlag is equal to 1), the sample value tempArray[x, y] is derived by

$$\begin{aligned} tempArray[x, y] = {} & (16 - yPhase) * refSampleArray[x, yRef] + \\ & yPhase * refSampleArray[x, yRef + 1] \end{aligned} \quad \text{(G-304)}$$

Each sample predArray[x, y] with x = 0..(mbW − 1) and y = 0..(mbH / (1 + fldPrdInFrmMbFlag) + 2 * yBorder − 1) is derived as specified in the following ordered steps:

1. The derivation process for reference layer sample locations in resampling as specified in subclause G.6.3 is invoked with chromaFlag, the sample location (x, 0), fieldMbFlag, and botFieldFlag as the inputs and the output is the sample location (xRef16, yRef16) in units of 1/16-th sample.

   NOTE 2 – In this invocation of the process in subclause G.6.3, only the horizontal component xRef16 of the sample location needs to be derived.

2. The variables xRef and xPhase are derived by

$$xRef = (xRef16 >> 4) - xOffset \quad \text{(G-305)}$$
$$xPhase = (xRef16 - 16 * xOffset) \% 16 \quad \text{(G-306)}$$

3. Depending on filteringModeFlag, and with Clip1 being replaced by $Clip1_Y$ for chromaFlag equal to 0 and $Clip1_C$ for chromaFlag equal to 1, the sample value predArray[x, y] is derived as follows:

   – If filteringModeFlag is equal to 0, the sample value tempArray[x, y] is derived by

$$\begin{aligned} predArray[x, y] = Clip1((& eF[xPhase, 0] * tempArray[xRef - 1, y] + \\ & eF[xPhase, 1] * tempArray[xRef , y] + \\ & eF[xPhase, 2] * tempArray[xRef + 1, y] + \\ & eF[xPhase, 3] * tempArray[xRef + 2, y] + 512) >> 10) \end{aligned} \quad \text{(G-307)}$$

   – Otherwise (filteringModeFlag is equal to 1), the sample value tempArray[x, y] is derived by

$$\begin{aligned} predArray[x, y] = ((& 16 - xPhase) * tempArray[xRef, y] + \\ & xPhase * tempArray[xRef + 1, y] + 128) >> 8 \end{aligned} \quad \text{(G-308)}$$

G.8.6.2.4 Vertical interpolation process for Intra_Base prediction

Inputs to this process are:

– a variable filteringModeFlag specifying the interpolation method,

– a variable chromaFlag specifying whether the luma or a chroma component is subject to the resampling process,

– two variables mbW and mbH specifying the width and height, respectively, of a macroblock for the considered colour component,

– a variable botFieldFlag specifying whether the sample array fieldPredArray contains interpolated samples for the top or bottom field,

– a variable yBorder specifying the vertical border for the sample array fieldPredArray,

– a variable frameMbFlag specifying whether the current macroblock is a frame or a field macroblock,

– an (mbW)x(mbH / (1 + frameMbFlag) + 2 * yBorder) array fieldPredArray of sample values.

Output of this process is an (mbW)x(mbH) array predArray of interpolated sample values.

Each sample predArray[x, y] with x = 0..(mbW − 1) and y = 0..(mbH − 1) is derived as follows:

– If frameMbFlag is equal to 1 and (y % 2) is equal to botFieldFlag, the sample value predArray[x, y] is derived by

$$\text{predArray}[\,x, y\,] = \text{fieldPredArray}[\,x, (\,y >> 1\,) + \text{yBorder}\,] \tag{G-309}$$

– Otherwise (frameMbFlag is equal to 0 or (y % 2) is not equal to botFieldFlag), the following ordered steps are specified:

1. The variable yFld is derived by

$$\text{yFld} = (\,y >> \text{frameMbFlag}\,) + \text{yBorder} - \text{botFieldFlag} \tag{G-310}$$

2. Depending on filteringModeFlag, and with Clip1 being replaced by Clip1_Y for chromaFlag equal to 0 and Clip1_C for chromaFlag equal to 1, the sample value predArray[x, y] is derived as follows:

   – If filteringModeFlag is equal to 0, the sample value predArray[x, y] is derived by

$$\begin{aligned}\text{predArray}[\,x, y\,] = \text{Clip1}(\,(\,19 * (\,&\text{fieldPredArray}[\,x, \text{yFld}\,] + \\ &\text{fieldPredArray}[\,x, \text{yFld} + 1\,]\,) - \\ 3 * (\,&\text{fieldPredArray}[\,x, \text{yFld} - 1\,] + \\ &\text{fieldPredArray}[\,x, \text{yFld} + 2\,]\,) + 16\,) >> 5\,)\end{aligned} \tag{G-311}$$

   – Otherwise (filteringModeFlag is equal to 1), the sample value predArray[x, y] is derived by

$$\begin{aligned}\text{predArray}[\,x, y\,] = (\,&\text{fieldPredArray}[\,x, \text{yFld}\,] + \\ &\text{fieldPredArray}[\,x, \text{yFld} + 1\,] + 1\,) >> 1\end{aligned} \tag{G-312}$$

G.8.6.2.5 Derivation process for variables related to inter-layer intra prediction

This subclause is only invoked when MinNoInterLayerPredFlag is equal to 0.

Input to this process is a variable currDQId.

Outputs of this process are:

– a variable numILIntraPredSamples,

– a variable numRefLayerILIntraPredMbs.

Unless stated otherwise, all syntax elements and derived upper-case variables that are referred to inside this subclause are syntax elements and derived upper case variables for the layer representation with DQId equal to currDQId.

Inside this subclause, the collective terms currentVars and refLayerVars are specified as follows:

– If SpatialResolutionChangeFlag is equal to 1, the following applies:

  – currentVars is the collective term currentVars after completion of the base decoding process for layer representations with resolution change as specified in subclause G.8.1.3.2 for the layer representation with DQId equal to currDQId,

  – refLayerVars is the collective term refLayerVars after completion of the base decoding process for layer representations with resolution change as specified in subclause G.8.1.3.2 for the layer representation with DQId equal to currDQId.

– Otherwise (SpatialResolutionChangeFlag is equal to 0), the following applies:

  – currentVars is the collective term currentVars after completion of the base decoding process for layer representations without resolution change as specified in subclause G.8.1.3.1 for the layer representation with DQId equal to currDQId,

– refLayerVars is of the collective term currentVars before invoking the base decoding process for layer representations without resolution change as specified in subclause G.8.1.3.1 for the layer representation with DQId equal to currDQId.

Inside this subclause, the arrays of the collective term currentVars are referred to by their names as specified in subclause G.8.1.2.1.

Inside this subclause, the arrays fieldMbFlag and mbType of the collective term refLayerVars are referred to as refLayerFieldMbFlag and refLayerMbType, respectively.

Let currILIntraPredFlag be a ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array and let refILIntraPredFlag be a one-dimensional array with RefLayerPicSizeInMbs elements. All elements of the arrays currILIntraPredFlag and refILIntraPredFlag are initially set equal to 0.

The variable yC proceeds over the values 0..($PicHeightInSamples_L$ − 1). For each value of yC, the variable xC proceeds over the values 0..($PicWidthInSamples_L$ − 1). For each combination of the values yC and xC, the following ordered steps are specified:

1. The variable mbAddr is set equal to the address of the macroblock that contains the luma sample at location (xC, yC) relative to the upper-left sample of the layer picture.
2. Depending on SpatialResolutionChangeFlag, the following applies:
   - If SpatialResolutionChangeFlag is equal to 0, the following ordered steps are specified:
     a. The array element currILIntraPredFlag[xC, yC] is derived as follows:
        - If mbType[mbAddr] is equal to I_BL, currILIntraPredFlag[xC, yC] is set equal to 1.
        - Otherwise (mbType[mbAddr] is not equal to I_BL), currILIntraPredFlag[xC, yC] is set equal to 0.

     b. When currILIntraPredFlag[xC, yC] is equal to 1, the following ordered steps are specified:
        i. The variable refMbAddr is derived as specified in subclause G.8.1.2.2 with mbAddr being the value of mbAddr derived in step 1 of this subclause.
        ii. The array element refILIntraPredFlag[refMbAddr] is set equal to 1.
        iii. When refLayerMbType[refMbAddr] is equal to I_16x16, I_8x8, or I_4x4, let setRefIntraMbs be the set of macroblocks that contain luma or chroma samples that are directly (by the invocation of subclause G.8.3.2 for the macroblock with address refMbAddr) or indirectly (by multiple invocations of subclause G.8.3.2 for macroblocks with mbAddr less than or equal to refMbAddr) used for construction of the intra prediction signal of the macroblock with address refMbAddr in the layer representation with DQId equal to MaxRefLayerDQId.
        iv. For refIntraMbAddr proceeding over the macroblock addresses for the macroblocks of the set setRefIntraMbs, refILIntraPredFlag[refIntraMbAddr] is set equal to 1.
   - Otherwise (SpatialResolutionChangeFlag is equal to 1), the following ordered steps are specified:
     a. When RestrictedSpatialResolutionFlag is equal to 0, MbaffFrameFlag is equal to 0, RefLayerMbaffFrameFlag is equal to 0, and base_mode_flag for the macroblock with address mbAddr is equal to 1, the derivation process for reference layer macroblocks as specified in subclause G.6.1 is invoked with the luma location (xC % 16, yC % 16), fieldMbFlag, refLayerFieldMbFlag, and refLayerMbType as the inputs and the outputs are assigned to mbAddrRefLayer and (xRef, yRef). For this invocation of subclause G.6.1, CurrMbAddr is set equal to mbAddr.
     b. The element currILIntraPredFlag[xC, yC] is derived as follows:
        - If any of the following conditions are true, currILIntraPredFlag[xC, yC] is set equal to 1:
          - mbType[mbAddr] is equal to I_BL,
          - RestrictedSpatialResolutionFlag is equal to 0, MbaffFrameFlag is equal to 0, RefLayerMbaffFrameFlag is equal to 0, base_mode_flag for the macroblock with address mbAddr is equal to 1, and refLayerMbType[mbAddrRefLayer] is equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL.
        - Otherwise, currILIntraPredFlag[xC, yC] is set equal to 0.

c. When currILIntraPredFlag[xC, yC] is equal to 1, the following ordered steps are specified:

i. Let setOfRefSamples be the set of reference layer luma sample locations (xR, yR) of the luma sample values that are used in the filtering processes specified in subclause G.8.6.2.3 and, when applicable, subclause G.8.6.2.4 for deriving the inter-layer intra prediction sample for the luma sample at location (xC, yC) relative to the upper-left luma sample of the layer picture.

ii. For each of the reference layer luma sample locations (xR, yR) of the set setOfRefSamples that correspond to luma samples marked "available for Intra_Base prediction" in the invocation of subclause G.8.6.2.2 for the macroblock with address mbAddr of the layer representation with DQId equal to currDQId, the following ordered steps are specified:

(1) Let refMbAddr be the macroblock address of the macroblock in the layer representation with DQId equal to MaxRefLayerDQId that contains the luma sample at location (xR, yR).

(2) The array element refILIntraPredFlag[refMbAddr] is set equal to 1.

(3) When refLayerMbType[refMbAddr] is equal to I_16x16, I_8x8, or I_4x4, let setRefIntraMbs be the set of macroblocks that contain luma or chroma samples that are directly (by the invocation of subclause G.8.3.2 for the macroblock with address refMbAddr) or indirectly (by multiple invocations of subclause G.8.3.2 for macroblocks with mbAddr less than or equal to refMbAddr) used for construction of the intra prediction signal of the macroblock with address refMbAddr in the layer representation with DQId equal to MaxRefLayerDQId.

(4) For refIntraMbAddr proceeding over the macroblock addresses for the macroblocks of the set setRefIntraMbs, refILIntraPredFlag[refIntraMbAddr] is set equal to 1.

The variable numILIntraPredSamples is set equal to the number of elements of the $(\text{PicWidthInSamples}_L)\text{x}(\text{PicHeightInSamples}_L)$ array currILIntraPredFlag that are equal to 1.

NOTE 1 – The variable numILIntraPredSamples is a measure for the number of luma samples in the layer representation with DQId equal to currDQId that are predicted by inter-layer intra prediction.

The variable numRefLayerILIntraPredMbs is set equal to the number of elements of the array refILIntraPredFlag that are equal to 1.

NOTE 2 – The variable numRefLayerILIntraPredMbs is a measure for the number of intra-coded macroblocks in the reference layer representation that need to be decoded for constructing the inter-layer intra prediction samples of the layer representation with DQId equal to currDQId.

G.8.6.3 Resampling process for residual samples

Inputs to this process are:

- a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,
- a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,
- a one-dimensional array refLayerCTrafo with RefLayerPicSizeInMbs elements specifying the luma transform types for the macroblocks of the reference layer representation,
- a $(\text{RefLayerPicWidthInSamples}_L)\text{x}(\text{RefLayerPicHeightInSamples}_L)$ array $\text{refLayerPicSamples}_L$ of luma samples for the reference layer representation,
- a $(\text{PicWidthInSamples}_L)\text{x}(\text{PicHeightInSamples}_L)$ array picSamples_L of luma samples,
- when ChromaArrayType is not equal to 0, two $(\text{RefLayerPicWidthInSamples}_C)\text{x}(\text{RefLayerPicHeightInSamples}_C)$ arrays $\text{refLayerPicSamples}_{Cb}$ and $\text{refLayerPicSamples}_{Cr}$ of chroma samples for the reference layer representation,
- when ChromaArrayType is not equal to 0, two $(\text{PicWidthInSamples}_C)\text{x}(\text{PicHeightInSamples}_C)$ arrays picSamples_{Cb} and picSamples_{Cr} of chroma samples.

Outputs of this process are:

- a modified version of the array picSamples_L of luma samples,
- when ChromaArrayType is not equal to 0, modified versions of the arrays picSamples_{Cb} and picSamples_{Cr} of chroma samples.

The resampling process for residual samples of a macroblock colour component as specified in subclause G.8.6.3.1 is invoked with chromaFlag equal to 0, mbW equal to 16, mbH equal to 16, fieldMbFlag, $refLayerPicSamples_L$, refLayerFieldMbFlag, and refLayerCTrafo as the inputs and the output is the 16x16 array $mbPred_L$ of residual prediction samples for the luma component of the current macroblock.

When ChromaArrayType is not equal to 0, for CX being replaced by Cb and Cr, the resampling process for residual samples of a macroblock colour component as specified in subclause G.8.6.3.1 is invoked with chromaFlag equal to 1, mbW equal to MbWidthC, mbH equal to MbHeightC, fieldMbFlag, $refLayerPicSamples_{CX}$, refLayerFieldMbFlag, and refLayerCTrafo as the inputs and the output is the (MbWidthC)x(MbHeightC) array $mbPred_{CX}$ of residual prediction samples for the CX component of the current macroblock.

The picture sample array construction process as specified in subclause G.8.5.4.1 is invoked with fieldMbFlag, $mbPred_L$, $picSamples_L$ and, when ChromaArrayType is not equal to 0, $mbPred_{Cb}$, $mbPred_{Cr}$, $picSamples_{Cb}$, and $picSamples_{Cr}$ as the inputs and the outputs are a modified version of $picSamples_L$ and, when ChromaArrayType is not equal to 0, modified versions of $picSamples_{Cb}$, and $picSamples_{Cr}$.

G.8.6.3.1 Resampling process for residual samples of a macroblock colour component

Inputs to this process are:

– a variable chromaFlag specifying whether the luma or a chroma component is subject to the resampling process,

– two variables mbW and mbH specifying the width and height, respectively, of a macroblock for the considered colour component,

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– an array refLayerPicSamples, which is a $(RefLayerPicWidthInSamples_L)x(RefLayerPicHeightInSamples_L)$ array containing constructed residual luma sample values for the reference layer representation when chromaFlag is equal to 0 or a $(RefLayerPicWidthInSamples_C)x(RefLayerPicHeightInSamples_C)$ array containing constructed residual chroma sample values for the reference layer representation when chromaFlag is equal to 1,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– a one-dimensional array refLayerCTrafo with RefLayerPicSizeInMbs elements specifying the luma transform types for the macroblocks of the reference layer representation.

Output of this process is an (mbW)x(mbH) array mbPred of residual prediction samples.

The variable botFieldFlag is derived as follows:

– If RefLayerFrameMbsOnlyFlag is equal to 1, botFieldFlag is set equal to 0.

– Otherwise, if field_pic_flag is equal to 1, botFieldFlag is set equal to bottom_field_flag.

– Otherwise, if RefLayerFieldPicFlag is equal to 1, botFieldFlag is set equal to RefLayerBottomFieldFlag.

– Otherwise, if fieldMbFlag is equal to 1, botFieldFlag is set equal to (CurrMbAddr % 2).

– Otherwise, botFieldFlag is set equal to 0.

The variable frameBasedResamplingFlag is derived as follows:

– If all of the following conditions are true, frameBasedResamplingFlag is set equal to 1:

  – RefLayerFrameMbsOnlyFlag is equal to 1,

  – frame_mbs_only_flag is equal to 1.

– Otherwise, frameBasedResamplingFlag is set equal to 0.

The variable topAndBotResamplingFlag is derived as follows:

– If all of the following conditions are true, topAndBotResamplingFlag is set equal to 1:

  – RefLayerFrameMbsOnlyFlag is equal to 0,

  – RefLayerFieldPicFlag is equal to 0,

  – frame_mbs_only_flag is equal to 0,

  – fieldMbFlag is equal to 0.

– Otherwise, topAndBotResamplingFlag is set equal to 0.

The variable botFieldFrameMbsOnlyRefFlag is derived as follows:

– If RefLayerFrameMbsOnlyFlag is equal to 1, fieldMbFlag is equal to 1, and any of the following conditions are true, botFieldFrameMbsOnlyRefFlag is set equal to 1:
  – field_pic_flag is equal to 1 and bottom_field_flag is equal to 1,
  – field_pic_flag is equal to 0 and (CurrMbAddr % 2) is equal to 1.
– Otherwise, botFieldFrameMbsOnlyRefFlag is set equal to 0.

The array predArray is derived as specified in the following.

– If botFieldFrameMbsOnlyRefFlag is equal to 1, the following ordered steps are specified:
  1. The reference layer sample array construction process prior to residual resampling as specified in subclause G.8.6.3.2 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag, yBorder equal to 1, refLayerPicSamples, refLayerFieldMbFlag, and refLayerCTrafo as the inputs and the outputs are the variables refArrayW, refArrayH, the array refSampleArray of reference layer sample values, the array refTransBlkIdc of reference layer transform block identifications, and the variables xOffset and yOffset.
  2. The interpolation process for residual prediction as specified in subclause G.8.6.3.3 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag, fldPrdInFrmMbFlag equal to 0, yBorder equal to 1, refArrayW, refArrayH, refSampleArray, refTransBlkIdc, xOffset, and yOffset as the inputs and the output is the (mbW)x(mbH + 2) array topFldPredArray of top field prediction samples.
  3. The vertical interpolation process for residual prediction as specified in subclause G.8.6.3.4 is invoked with mbW, mbH, botFieldFlag, yBorder equal to 1, frameMbFlag equal to 0, and topFldPredArray as the inputs and the output is the (mbW)x(mbH) array mbPred of residual prediction samples.
– Otherwise, if frameBasedResamplingFlag is equal to 1 or fieldMbFlag is equal to 1, the following ordered steps are specified:
  1. The reference layer sample array construction process prior to residual resampling as specified in subclause G.8.6.3.2 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag, yBorder equal to 0, refLayerPicSamples, refLayerFieldMbFlag, and refLayerCTrafo as the inputs and the outputs are the variables refArrayW, refArrayH, the array refSampleArray of reference layer sample values, the array refTransBlkIdc of reference layer transform block identifications, and the variables xOffset and yOffset.
  2. The interpolation process for residual prediction as specified in subclause G.8.6.3.3 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag, fldPrdInFrmMbFlag equal to 0, yBorder equal to 0, refArrayW, refArrayH, refSampleArray, refTransBlkIdc, xOffset, and yOffset as the inputs and the output is the (mbW)x(mbH) array mbPred of residual prediction samples.
– Otherwise, if topAndBotResamplingFlag is equal to 0, the following ordered steps are specified:
  1. The reference layer sample array construction process prior to residual resampling as specified in subclause G.8.6.3.2 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag, yBorder equal to 1, refLayerPicSamples, refLayerFieldMbFlag, and refLayerCTrafo as the inputs and the outputs are the variables refArrayW, refArrayH, the array refSampleArray of reference layer sample values, the array refTransBlkIdc of reference layer transform block identifications, and the variables xOffset and yOffset.
  2. The interpolation process for residual prediction as specified in subclause G.8.6.3.3 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag, fldPrdInFrmMbFlag equal to 1, yBorder equal to 1, refArrayW, refArrayH, refSampleArray, refTransBlkIdc, xOffset, and yOffset as the inputs and the output is the (mbW)x(mbH / 2 + 2) array fieldPredArray of field prediction samples.
  3. The vertical interpolation process for residual prediction as specified in subclause G.8.6.3.4 is invoked with mbW, mbH, botFieldFlag, yBorder equal to 1, frameMbFlag equal to 1, and fieldPredArray as the inputs and the output is the (mbW)x(mbH) array mbPred of residual prediction samples.
– Otherwise (topAndBotResamplingFlag is equal to 1), the following ordered steps are specified:
  1. The reference layer sample array construction process prior to residual resampling as specified in subclause G.8.6.3.2 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag equal to 0, yBorder equal to 0, refLayerPicSamples, refLayerFieldMbFlag, and refLayerCTrafo as the inputs and the outputs are the variables refArrayTopW, refArrayTopH, the array refSampleArrayTop of reference layer sample values, the array refTransBlkIdcTop of reference layer transform block identifications, and the variables xOffsetTop and yOffsetTop.

2. The interpolation process for residual prediction as specified in subclause G.8.6.3.3 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag equal to 0, fldPrdInFrmMbFlag equal to 1, yBorder equal to 0, refArrayTopW, refArrayTopH, refSampleArrayTop, refTransBlkIdcTop, xOffsetTop, and yOffsetTop as the inputs and the output is the (mbW)x(mbH / 2) array topFieldPredArray of top field prediction samples.
3. The reference layer sample array construction process prior to residual resampling as specified in subclause G.8.6.3.2 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag equal to 1, yBorder equal to 0, refLayerPicSamples, refLayerFieldMbFlag, and refLayerCTrafo as the inputs and the outputs are the variables refArrayBotW, refArrayBotH, the array refSampleArrayBot of reference layer sample values, the array refTransBlkIdcBot of reference layer transform block identifications, and the variables xOffsetBot and yOffsetBot.
4. The interpolation process for residual prediction as specified in subclause G.8.6.3.3 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag equal to 1, fldPrdInFrmMbFlag equal to 1, yBorder equal to 0, refArrayBotW, refArrayBotH, refSampleArrayBot, refTransBlkIdcBot, xOffsetBot, and yOffsetBot as the inputs and the output is the (mbW)x(mbH / 2) array botFieldPredArray of bottom field prediction samples.
5. Each sample predArray[x, y] with x = 0..(mbW − 1) and y = 0..(mbH − 1) of the array mbPred of residual prediction samples is derived by

$$\begin{aligned}\text{mbPred[x, y]} = (((y \% 2) == 0) \; ? \; & \text{topFieldPredArray[x, y >> 1]} \\ : \; & \text{botFieldPredArray[x, y >> 1])}\end{aligned} \qquad \text{(G-313)}$$

G.8.6.3.2 Reference layer sample array construction process prior to residual resampling

Inputs to this process are:

– a variable chromaFlag specifying whether the luma or a chroma component is subject to the resampling process,

– two variables mbW and mbH specifying the width and height, respectively, of a macroblock for the considered colour component,

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a variable botFieldFlag specifying whether a top or a bottom field is subject to the resampling process (when RefLayerFrameMbsOnlyFlag is equal to 0 or frame_mbs_only_flag is equal to 0),

– a variable yBorder specifying the vertical border for determining the vertical size of the output arrays,

– an array refLayerPicSamples, which is a $(\text{RefLayerPicWidthInSamples}_L)\text{x}(\text{RefLayerPicHeightInSamples}_L)$ array containing constructed residual luma sample values for the reference layer representation when chromaFlag is equal to 0 or a $(\text{RefLayerPicWidthInSamples}_C)\text{x}(\text{RefLayerPicHeightInSamples}_C)$ array containing constructed residual chroma sample values for the reference layer representation when chromaFlag is equal to 1,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– a one-dimensional array refLayerCTrafo with RefLayerPicSizeInMbs elements specifying the luma transform types for the macroblocks of the reference layer representation.

Outputs of this process are:

– two variables refArrayW and refArrayH specifying the width and height, respectively, of the constructed arrays of reference layer sample values and reference layer transform block identification,

– a (refArrayW)x(refArrayH) array refSampleArray of reference layer sample values,

– a (refArrayW)x(refArrayH) array refTransBlkIdc of reference layer transform block identifications,

– two variables xOffset and yOffset specifying the x and y coordinate, respectively, of the reference layer sample location that corresponds to the sample refSampleArray[0, 0] of the array refSampleArray and the transform block identification refTransBlkIdc[0, 0] of the array refTransBlkIdc.

The variables refW, refH, refMbW, refMbH, xOffset, yOffset, refArrayW, refArrayH, yRefScale, and yRefAdd are derived as specified in the following ordered steps:

1. The derivation process for reference layer sample locations in resampling as specified in subclause G.6.3 is invoked with chromaFlag, the sample location (0, −yBorder), fieldMbFlag, and botFieldFlag as the inputs and the output is the sample location (xRefMin16, yRefMin16) in units of 1/16-th sample.

2. The derivation process for reference layer sample locations in resampling as specified in subclause G.6.3 is invoked with chromaFlag, the sample location (mbW − 1, mbH − 1 + yBorder), fieldMbFlag, and botFieldFlag as the inputs and the output is the sample location (xRefMax16, yRefMax16) in units of 1/16-th sample.

3. With Z being replaced by L for chromaFlag equal to 0 and C for chromaFlag equal to 1, the variables refW, refH, refMbW, and refMbH are derived by

$$\text{refW} = \text{RefLayerPicWidthInSamples}_Z \tag{G-314}$$
$$\text{refH} = \text{RefLayerPicHeightInSamples}_Z \tag{G-315}$$
$$\text{refMbW} = ((\text{chromaFlag} == 0) ? 16 : \text{RefLayerMbWidthC}) \tag{G-316}$$
$$\text{refMbH} = ((\text{chromaFlag} == 0) ? 16 : \text{RefLayerMbHeightC}) \tag{G-317}$$

4. The variables xOffset, yOffset, refArrayW, and refArrayH are derived by

$$\text{xOffset} = (\text{xRefMin16} >> 4) \tag{G-318}$$
$$\text{yOffset} = (\text{yRefMin16} >> 4) \tag{G-319}$$
$$\text{refArrayW} = (\text{xRefMax16} >> 4) - \text{xOffset} + 2 \tag{G-320}$$
$$\text{refArrayH} = (\text{yRefMax16} >> 4) - \text{yOffset} + 2 \tag{G-321}$$

5. The variables yRefScale and yRefAdd are derived as follows:

   – If RefLayerFrameMbsOnlyFlag is equal to 1 or RefLayerFieldPicFlag is equal to 1, yRefScale is set equal to 1 and yRefAdd is set equal to 0.

   – Otherwise (RefLayerFrameMbsOnlyFlag is equal to 0 and RefLayerFieldPicFlag is equal to 0), yRefScale is set equal to 2 and yRefAdd is set equal to botFieldFlag.

Each sample refSampleArray[x, y] and each transform block identification refTransBlkIdc[x, y] with x = 0..(refArrayW − 1) and y = 0..(refArrayH − 1) are derived as specified in the following ordered steps:

1. A reference layer sample location (xRef, yRef) is derived by:

$$\text{xRef} = \text{Max}(0, \text{Min}(\text{refW} - 1, x + \text{xOffset})) \tag{G-322}$$
$$\text{yRef} = \text{yRefScale} * \text{Max}(0, \text{Min}(\text{refH} / \text{yRefScale} - 1, y + \text{yOffset})) + \text{yRefAdd} \tag{G-323}$$

2. The sample refSampleArray[x, y] is derived by

$$\text{refSampleArray}[x, y] = \text{refLayerPicSamples}[\text{xRef}, \text{yRef}] \tag{G-324}$$

3. The transform block identification refTransBlkIdc[x, y] is derived by invoking the derivation process for reference layer transform block identifications as specified in subclause G.8.6.3.2.1 with the reference layer sample location (xRef, yRef), chromaFlag, refMbW, refMbH, refLayerFieldMbFlag, and refLayerCTrafo as the inputs and assigning the output to refTransBlkIdc[x, y].

G.8.6.3.2.1 Derivation process for reference layer transform block identifications

Inputs to this process are:

– a reference layer sample location (xRef, yRef) relative to the upper-left sample of the considered colour component of the reference layer picture,

– a variable chromaFlag specifying whether the luma or a chroma component is subject to the resampling process,

– two variables refMbW and refMbH specifying the width and height, respectively, of a reference layer macroblock for the considered colour component,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– a one-dimensional array refLayerCTrafo with RefLayerPicSizeInMbs elements specifying the luma transform types for the macroblocks of the reference layer representation.

Output of this process is a variable refTransBlkIdc specifying an identification for the reference layer transform block that contains the sample at location (xRef, yRef).

The reference layer macroblock address refMbAddr and the reference layer sample location (xM, yM) inside the reference layer macroblock are derived as follows:

– If RefLayerMbaffFrameFlag is equal to 0, the variable refMbAddr and the sample location (xM, yM) are derived by

$$\text{refMbAddr} = (\text{yRef} / \text{refMbH}) * \text{RefLayerPicWidthInMbs} + (\text{xRef} / \text{refMbW}) \quad \text{(G-325)}$$
$$\text{xM} = \text{xRef} \,\%\, \text{refMbW} \quad \text{(G-326)}$$
$$\text{yM} = \text{yRef} \,\%\, \text{refMbH} \quad \text{(G-327)}$$

– Otherwise (RefLayerMbaffFrameFlag is equal to 1), the variable refMbAddr is derived as specified in the following ordered steps:

1. A variable refMbAddrTop and the horizontal sample location xM are derived by

$$\text{refMbAddrTop} = 2 * ((\text{yRef} / (2 * \text{refMbH})) * \text{RefLayerPicWidthInMbs} + (\text{xRef} / \text{refMbW})) \quad \text{(G-328)}$$
$$\text{xM} = \text{xRef} \,\%\, \text{refMbW} \quad \text{(G-329)}$$

2. Depending on refLayerFieldMbFlag[refMbAddrTop], the variable refMbAddr and the vertical sample location yM are derived as follows:

   – If refLayerFieldMbFlag[refMbAddrTop] is equal to 0, the variables refMbAddr and yM are derived by

$$\text{refMbAddr} = \text{refMbAddrTop} + (\text{yRef} \,\%\, (2 * \text{refMbH})) / \text{refMbH} \quad \text{(G-330)}$$
$$\text{yM} = \text{yRef} \,\%\, \text{refMbH} \quad \text{(G-331)}$$

   – Otherwise (refLayerFieldMbFlag[refMbAddrTop] is equal to 1), the variables refMbAddr and yM are derived by

$$\text{refMbAddr} = \text{refMbAddrTop} + (\text{yRef} \,\%\, 2) \quad \text{(G-332)}$$
$$\text{yM} = (\text{yRef} \,\%\, (2 * \text{refMbH})) >> 1 \quad \text{(G-333)}$$

Depending on chromaFlag, RefLayerChromaArrayType, and refLayerCTrafo[refMbAddr], the following applies:

– If (chromaFlag is equal to 0 or RefLayerChromaArrayType is equal to 3) and refLayerCTrafo[refMbAddr] is equal to T_8x8, the variable refTransBlkIdc is derived by:

$$\text{refTransBlkIdc} = 1 + 2 * (4 * \text{refMbAddr} + 2 * (\text{yM} / 8) + (\text{xM} / 8)) \quad \text{(G-334)}$$

– Otherwise ((chromaFlag is equal to 1 and RefLayerChromaArrayType is not equal to 3) or refLayerCTrafo[refMbAddr] is not equal to T_8x8), the variable refTransBlkIdc is derived by

$$\text{refTransBlkIdc} = 2 * (16 * \text{refMbAddr} + 4 * (\text{yM} / 4) + (\text{xM} / 4)) \quad \text{(G-335)}$$

G.8.6.3.3 Interpolation process for residual prediction

Inputs to this process are:

– a variable chromaFlag specifying whether the luma or a chroma component is subject to the resampling process,

– two variables mbW and mbH specifying the width and height, respectively, of a macroblock for the considered colour component,

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a variable botFieldFlag specifying whether a top or a bottom field is subject to the resampling process (when RefLayerFrameMbsOnlyFlag is equal to 0 or frame_mbs_only_flag is equal to 0),

– a variable fldPrdInFrmMbFlag specifying whether field prediction for a frame macroblock is applied,

– a variable yBorder specifying the vertical border for the output sample array predSamples,

– two variables refArrayW and refArrayH specifying the width and height, respectively, of the array of reference layer sample values and the array of transform block identifications,

– a (refArrayW)x(refArrayH) array refSampleArray of reference layer sample values,

– a (refArrayW)x(refArrayH) array refTransBlkIdc of transform block identifications,

– two variables xOffset and yOffset specifying the x and y coordinate, respectively, of the reference layer sample location that corresponds to the sample refSampleArray[0, 0] of the array refSampleArray and the transform block identification refTransBlkIdc[0, 0] of the array refTransBlkIdc.

Output of this process is an (mbW)x(mbH / (1 + fldPrdInFrmMbFlag) + 2 * yBorder) array predArray of interpolated sample values.

Each sample predArray[x, y] with x = 0..(mbW − 1) and y = 0..(mbH / (1 + fldPrdInFrmMbFlag) + 2 *yBorder − 1) is derived as specified in the following ordered steps:

1. The variable yP is derived by:

$$yP = (y - yBorder) * (1 + fldPrdInFrmMbFlag) + botFieldFlag \quad \text{(G-336)}$$

2. The derivation process for reference layer sample locations in resampling as specified in subclause G.6.3 is invoked with chromaFlag, the sample location (x, yP), fieldMbFlag, and botFieldFlag as the inputs and the output is the sample location (xRef16, yRef16) in units of 1/16-th sample.

3. The variables xRef, yRef, xPhase, and yPhase are derived by:

$$xRef = (xRef16 >> 4) - xOffset \quad \text{(G-337)}$$
$$yRef = (yRef16 >> 4) - yOffset \quad \text{(G-338)}$$
$$xPhase = (xRef16 - 16 * xOffset) \% 16 \quad \text{(G-339)}$$
$$yPhase = (yRef16 - 16 * yOffset) \% 16 \quad \text{(G-340)}$$

4. Let tempPred be a one-dimensional array with 2 elements. Each sample value tempPred[dY] with dY = 0..1 is derived as follows:

   – If refTransBlkIdc[xRef, yRef + dY] is equal to refTransBlkIdc[xRef + 1, yRef + dY], the sample value tempPred[dY] is derived by:

$$\begin{aligned} tempPred[dY] = {} & (16 - xPhase) * refSampleArray[xRef, yRef + dY] + \\ & xPhase * refSampleArray[xRef + 1, yRef + dY] \end{aligned} \quad \text{(G-341)}$$

   – Otherwise (refTransBlkIdc[xRef, yRef + dY] is not equal to refTransBlkIdc[xRef + 1, yRef + dY]), the sample value tempPred[dY] is derived by:

$$\begin{aligned} tempPred[dY] = ((xPhase < 8) \; ? \; & refSampleArray[xRef, yRef + dY] \\ : \; & refSampleArray[xRef + 1, yRef + dY]) << 4 \end{aligned} \quad \text{(G-342)}$$

5. With xRefRound set equal to (xRef + (xPhase / 8)), the sample value predArray[x, y] is derived as follows:

   – If refTransBlkIdc[xRefRound, yRef] is equal to refTransBlkIdc[xRefRound, yRef + 1], the sample value predArray[x, y] is derived by:

$$\begin{aligned} predArray[x, y] = ({} & (16 - yPhase) * tempPred[0] + \\ & yPhase * tempPred[1] + 128) >> 8 \end{aligned} \quad \text{(G-343)}$$

   – Otherwise (refTransBlkIdc[xRefRound, yRef] is not equal to refTransBlkIdc[xRefRound, yRef + 1]), the sample value predArray[x, y] is derived by:

$$predArray[x, y] = (((yPhase < 8) \; ? \; tempPred[0] \; : \; tempPred[1]) + 8) >> 4 \quad \text{(G-344)}$$

G.8.6.3.4 Vertical interpolation process for residual prediction

Inputs to this process are:

– two variables mbW and mbH specifying the width and height, respectively, of a macroblock for the considered colour component,

– a variable botFieldFlag specifying whether the sample array fieldPredArray contains interpolated samples for the top or bottom field,

– a variable yBorder specifying the vertical border for the sample array fieldPredArray,

– a variable frameMbFlag specifying whether the current macroblock is a frame or a field macroblock,

– an (mbW)x(mbH / (1 + frameMbFlag) + 2 * yBorder) array fieldPredArray of sample values.

Output of this process is an (mbW)x(mbH) array predArray of interpolated sample values.

Each sample predArray[x, y] with x = 0..(mbW − 1) and y = 0..(mbH − 1) is derived as follows:

– If frameMbFlag is equal to 1 and (y % 2) is equal to botFieldFlag, the sample value predArray[x, y] is derived by

$$predArray[x, y] = fieldPredArray[x, (y >> 1) + yBorder] \quad \text{(G-345)}$$

– Otherwise (frameMbFlag is equal to 0 or (y % 2) is not equal to botFieldFlag), the sample value predArray[x, y] is derived by

$$\text{predArray}[\,x, y\,] = (\,\text{fieldPredArray}[\,x, (\,y >> \text{frameMbFlag}\,) + \text{yBorder} - \text{botFieldFlag}\,] + \text{fieldPredArray}[\,x, (\,y >> \text{frameMbFlag}\,) + \text{yBorder} - \text{botFieldFlag} + 1\,] + 1\,) >> 1 \quad \text{(G-346)}$$

G.8.7 SVC deblocking filter processes

Subclause G.8.7.1 specifies the deblocking filter process for Intra_Base prediction.

Subclause G.8.7.2 specifies the deblocking filter process for target representations.

G.8.7.1 Deblocking filter process for Intra_Base prediction

Inputs to the process are:

– the variable currDQId,

– the collective term currentVars.

Output of this process is a modified version of currentVars.

Let the variable refLayerDQId be equal to the value of the variable MaxRefLayerDQId of the layer representation with DQId equal to currDQId.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the layer representation with DQId equal to refLayerDQId.

Inside this subclause, the arrays that are collectively referred to as currentVars are referred to by their names as specified in subclause G.8.1.2.1.

The derivation process for quantisation parameters used in the deblocking filter process as specified in subclause G.8.7.3 is invoked with deblockingDQId set equal to refLayerDQId, mbType, tQP_Y, and tCoeffLevel as the inputs and the outputs are a list $qpDB_Y$ specifying luma quantisation parameter that are used in the deblocking filter process and, when ChromaArrayType is not equal to 0, two lists $qpDB_{Cb}$ and $qpDB_{Cr}$ specifying chroma quantisation parameters that are used in the deblocking filter process.

Let disableDeblockingFilterIdc, filterOffsetA, and filterOffsetB be equal to the values of disable_inter_layer_deblocking_filter_idc, InterlayerFilterOffsetA, and InterlayerFilterOffsetB, respectively, for any slice of the layer representation with DQId equal to currDQId, that has no_inter_layer_pred_flag equal to 0.

For the current macroblock address CurrMbAddr proceeding over values 0..(PicSizeInMbs − 1), the macroblock deblocking filter process as specified in subclause G.8.7.4 is invoked with interLayerDeblockingFlag = 1, disableDeblockingFilterIdc, filterOffsetA, filterOffsetB, sliceBoundariesOnlyFlag = 0, currentVars, $qpDB_Y$ and, when ChromaArrayType is not equal to 0, $qpDB_{Cb}$ and $qpDB_{Cr}$ as the inputs and the output is a modified version of currentVars.

When disableDeblockingFilterIdc is equal to 3 or 6, for the current macroblock address CurrMbAddr proceeding over values 0..(PicSizeInMbs − 1), the macroblock deblocking filter process as specified in subclause G.8.7.4 is invoked with interLayerDeblockingFlag = 1, disableDeblockingFilterIdc, filterOffsetA, filterOffsetB, sliceBoundariesOnlyFlag = 1, currentVars, $qpDB_Y$ and, when ChromaArrayType is not equal to 0, $qpDB_{Cb}$ and $qpDB_{Cr}$ as the inputs and the output is a modified version of currentVars.

G.8.7.2 Deblocking filter process for target representations

Inputs to the process are:

– the variable currDQId,

– the collective term currentVars.

Output of this process is a modified version of currentVars.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the layer representation with DQId equal to currDQId.

Inside this subclause, the arrays that are collectively referred to as currentVars are referred to by their names as specified in subclause G.8.1.2.1.

The derivation process for quantisation parameters used in the deblocking filter process as specified in subclause G.8.7.3 is invoked with deblockingDQId set equal to currDQId, mbType, tQP_Y, and tCoeffLevel as the inputs and the outputs are a list $qpDB_Y$ specifying luma quantisation parameter that are used in the deblocking filter process and, when ChromaArrayType is not equal to 0, two lists $qpDB_{Cb}$ and $qpDB_{Cr}$ specifying chroma quantisation parameters that are used in the deblocking filter process.

For the current macroblock address CurrMbAddr proceeding over values 0..(PicSizeInMbs − 1), the following ordered steps are specified:

1. Let disableDeblockingFilterIdc, filterOffsetA, and filterOffsetB be equal to the value of disable_deblocking_filter_idc, FilterOffsetA, and FilterOffsetB, respectively, for the slice with DQId equal to (sliceIdc[CurrMbAddr] & 127) and first_mb_in_slice equal to (sliceIdc[CurrMbAddr] >> 7).
2. The macroblock deblocking filter process as specified in subclause G.8.7.4 is invoked with interLayerDeblockingFlag = 0, disableDeblockingFilterIdc, filterOffsetA, filterOffsetB, sliceBoundariesOnlyFlag = 0, currentVars, $qpDB_Y$ and, when ChromaArrayType is not equal to 0, $qpDB_{Cb}$ and $qpDB_{Cr}$ as the inputs and the output is a modified version of currentVars.

For the current macroblock address CurrMbAddr proceeding over values 0..(PicSizeInMbs − 1), the following ordered steps are specified:

1. Let disableDeblockingFilterIdc, filterOffsetA, and filterOffsetB be equal to the value of disable_deblocking_filter_idc, FilterOffsetA, and FilterOffsetB, respectively, for the slice with DQId equal to (sliceIdc[CurrMbAddr] & 127) and first_mb_in_slice equal to (sliceIdc[CurrMbAddr] >> 7).
2. When disableDeblockingFilterIdc is equal to 3 or 6, the macroblock deblocking filter process as specified in subclause G.8.7.4 is invoked with interLayerDeblockingFlag = 0, disableDeblockingFilterIdc, filterOffsetA, filterOffsetB, sliceBoundariesOnlyFlag = 1, currentVars, $qpDB_Y$ and, when ChromaArrayType is not equal to 0, $qpDB_{Cb}$ and $qpDB_{Cr}$ as inputs and the output is a modified version of currentVars.

G.8.7.3 Derivation process for quantisation parameters used in the deblocking filter process

Inputs to this process are:

– a variable deblockingDQId,

– a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current decoded or partly decoded dependency representation,

– a one-dimensional array tQP_Y with PicSizeInMbs elements specifying luma quantisation parameters for the macroblocks of the current decoded or partly decoded dependency representation,

– a (PicSizeInMbs)x(256 + 2 * MbWidthC * MbHeightC) array tCoeffLevel specifying transform coefficient level values for the macroblocks of the current decoded or partly decoded dependency representation.

Outputs of this process are:

– a one-dimensional array $qpDB_Y$ with PicSizeInMbs elements specifying luma quantisation parameters used in the deblocking filter process for the macroblocks of the current decoded or partly decoded dependency representation,

– when ChromaArrayType is not equal to 0, two one-dimensional arrays $qpDB_{Cb}$ and $qpDB_{Cr}$ with PicSizeInMbs elements specifying chroma quantisation parameters used in the deblocking filter process for the macroblocks of the current decoded or partly decoded dependency representation.

The syntax elements and derived upper-case variables that are referred to by the process specified in this subclause are the syntax elements and derived upper-case variables for the layer representation with DQId equal to deblockingDQId.

Let tempQP be a one-dimensional array with PicSizeInMbs elements. All elements tempQP[i] with i = 0..(PicSizeInMbs − 1) are set equal to tQP_Y[i].

When MaxTCoeffLevelPredFlag is equal to 1, the following ordered steps are specified:

1. Let firstMbInSliceGroup and numMbsInSliceGroup be two one-dimensional arrays with (num_slice_groups_minus1 + 1) elements. The array elements are derived as specified by the following pseudo code.

```
for( iGroup = 0; iGroup <= num_slice_groups_minus1; iGroup++ )  {
   firstMbInSliceGroup[ iGroup ] = −1
   numMbsInSliceGroup[ iGroup ] = 0
}
```

```
for( i = 0; i < PicSizeInMbs; i++ ) {                                    (G-347)
   if( firstMbInSliceGroup[ MbToSliceGroupMap[ i ] ] = = −1 )
      firstMbInSliceGroup[ MbToSliceGroupMap[ i ] ] = i
   numMbsInSliceGroup[ MbToSliceGroupMap[ i ] ]++
}
```

2. The variable iGroup proceeds over the values 0..num_slice_groups_minus1. For each value of iGroup, the variable lastMbAddr is set equal to firstMbInSliceGroup[iGroup] and the variable mbIdx proceeds over the values 1..(numMbsInSliceGroup[iGroup] − 1). For each value of mbIdx, the following ordered steps are specified.

   a. The variable mbAddr is derived as specified by the following pseudo-code:

```
mbAddr = lastMbAddr + 1
while( MbToSliceGroupMap[ mbAddr ] != MbToSliceGroupMap[ lastMbAddr ] )          (G-348)
   mbAddr++
```

   b. When mbType[mbAddr] is not equal to I_16x16 and all elements tCoeffLevel[mbAddr][i] with i = 0..(255 + 2 * MbWidthC * MbHeightC) are equal to 0, tempQP[mbAddr] is set equal to tempQP[lastMbAddr].

   c. The variable lastMbAddr is set equal to mbAddr.

The macroblock address mbAddr proceeds over the values 0..(PicSizeInMbs − 1), and for each value of mbAddr, the following ordered steps are specified:

1. The variable $qpDB_Y$[mbAddr] is derived as follows:

   – If mbType[mbAddr] is equal to I_PCM, $qpDB_Y$[mbAddr] is set equal to 0.

   – Otherwise (mbType[mbAddr] is not equal to I_PCM), $qpDB_Y$[mbAddr] is set equal to tempQP[mbAddr].

2. When ChromaArrayType is not equal to 0, for C being replaced by Cb and Cr, the variable $qpDB_C$[mbAddr] is set equal to the value of QP_C that corresponds to a value of $qpDB_Y$[mbAddr] for QP_Y as specified in subclause 8.5.8. During this invocation of the process in subclause 8.5.8, the syntax elements chroma_qp_index_offset and second_chroma_qp_index_offset of the layer representation with DQId equal to deblockingFilterDQId are used.

G.8.7.4 Macroblock deblocking filter process

Inputs to this process are:

– the variables interLayerDeblockingFlag, disableDeblockingFilterIdc, filterOffsetA, filterOffsetB, and sliceBoundariesOnlyFlag,

– the collective term currentVars,

– a one-dimensional array $qpDB_Y$ with PicSizeInMbs elements specifying luma quantisation parameters used in the deblocking filter process for the macroblocks of the current decoded or partly decoded dependency representation,

– when ChromaArrayType is not equal to 0, two one-dimensional arrays $qpDB_{Cb}$ and $qpDB_{Cr}$ with PicSizeInMbs elements specifying chroma quantisation parameters used in the deblocking filter process for the macroblocks of the current decoded or partly decoded dependency representation.

Output of this process is a modified version of currentVars.

In the following of this subclause, the arrays that are collectively referred to as currentVars are referred to by their names as specified in subclause G.8.1.2.1.

The derivation process for neighbouring macroblocks specified in subclause 6.4.11.1 is invoked and the output is assigned to mbAddrA and mbAddrB. For this invocation of the process in subclause 6.4.11.1, the current macroblock is treated as field macroblock when fieldMbFlag[CurrMbAddr] is equal to 1, and it is treated as frame macroblock when fieldMbFlag[CurrMbAddr] is equal to 0.

NOTE 1 – The availability status of the macroblocks mbAddrA and mbAddrB is not used inside this subclause. Slice boundaries are detected using the array sliceIdc.

The variable filterLeftLumaMbEdgeFlag is derived as follows:

– If any of the following conditions are true, the variable filterLeftLumaMbEdgeFlag is set equal to 0:
  – MbaffFrameFlag is equal to 0 and CurrMbAddr % PicWidthInMbs is equal to 0,
  – MbaffFrameFlag is equal to 1 and (CurrMbAddr >> 1) % PicWidthInMbs is equal to 0,
  – disableDeblockingFilterIdc is equal to 1,
  – disableDeblockingFilterIdc is equal to 2 or 5 and sliceIdc[mbAddrA] is different than sliceIdc[CurrMbAddr],
  – disableDeblockingFilterIdc is equal to 3 or 6, sliceBoundariesOnlyFlag is equal to 0, and sliceIdc[mbAddrA] is different than sliceIdc[CurrMbAddr],
  – disableDeblockingFilterIdc is equal to 3 or 6, sliceBoundariesOnlyFlag is equal to 1, and sliceIdc[mbAddrA] is equal to sliceIdc[CurrMbAddr],
  – interLayerDeblockingFlag is equal to 1 and mbType[CurrMbAddr] specifies an Inter macroblock prediction mode.
– Otherwise, the variable filterLeftLumaMbEdgeFlag is set equal to 1.

The variable filterTopLumaMbEdgeFlag is derived as follows:

– If any of the following conditions are true, the variable filterTopLumaMbEdgeFlag is set equal to 0:
  – MbaffFrameFlag is equal to 0 and CurrMbAddr is less than PicWidthInMbs,
  – MbaffFrameFlag is equal to 1, (CurrMbAddr >> 1) is less than PicWidthInMbs, and fieldMbFlag[CurrMbAddr] is equal to 1,
  – MbaffFrameFlag is equal to 1, (CurrMbAddr >> 1) is less than PicWidthInMbs, fieldMbFlag[CurrMbAddr] is equal to 0, and CurrMbAddr % 2 is equal to 0,
  – disableDeblockingFilterIdc is equal to 1,
  – disableDeblockingFilterIdc is equal to 2 or 5 and sliceIdc[mbAddrB] is different than sliceIdc[CurrMbAddr],
  – disableDeblockingFilterIdc is equal to 3 or 6, sliceBoundariesOnlyFlag is equal to 0, and sliceIdc[mbAddrB] is different than sliceIdc[CurrMbAddr],
  – disableDeblockingFilterIdc is equal to 3 or 6, sliceBoundariesOnlyFlag is equal to 1, and sliceIdc[mbAddrB] is equal to sliceIdc[CurrMbAddr],
  – interLayerDeblockingFlag is equal to 1 and mbType[CurrMbAddr] specifies an Inter macroblock prediction mode.
– Otherwise, the variable filterTopLumaMbEdgeFlag is set equal to 1.

The variable filterInternalLumaEdgesFlag is derived as follows:

– If any of the following conditions are true, the variable filterInternalLumaEdgesFlag is set equal to 0:
  – disableDeblockingFilterIdc is equal to 1,
  – disableDeblockingFilterIdc is equal to 3 or 6 and sliceBoundariesOnlyFlag is equal to 1,
  – interLayerDeblockingFlag is equal to 1 mbType[CurrMbAddr] specifies an Inter macroblock prediction mode.
– Otherwise the variable filterInternalLumaEdgesFlag is set equal to 1.

The variables filterLeftChromaMbEdgeFlag, filterTopChromaMbEdgeFlag, and filterInternalChromaEdgesFlag are derived as follows:

– If disableDeblockingFilterIdc is greater than 3, filterLeftChromaMbEdgeFlag, filterTopChromaMbEdgeFlag, and filterInternalChromaEdgesFlag are set equal to 0.
– Otherwise (disableDeblockingFilterIdc is less than 4), filterLeftChromaMbEdgeFlag, filterTopChromaMbEdgeFlag, and filterInternalChromaEdgesFlag are set equal to filterLeftLumaMbEdgeFlag, filterTopLumaMbEdgeFlag, and filterInternalLumaEdgesFlag, respectively.

The variable fieldMbInFrameFlag is derived as follows:

– If MbaffFrameFlag is equal to 1 and fieldMbFlag[CurrMbAddr] is equal to 1, fieldMbInFrameFlag is set equal to 1.

– Otherwise (MbaffFrameFlag is equal to 0 or fieldMbFlag[CurrMbAddr] is equal to 0), fieldMbInFrameFlag is set equal to 0.

When filterLeftLumaMbEdgeFlag is equal to 1, the left vertical luma edge is filtered by invoking the process specified in subclause G.8.7.4.1 with interLayerDeblockingFlag, chromaEdgeFlag set equal to 0, verticalEdgeFlag set equal to 1, fieldModeInFrameFilteringFlag set equal to fieldMbInFrameFlag, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_Y$, currentVars, (xE_k, yE_k) set equal to (0, k) with k = 0..15, and cS_L as the inputs and cS_L as the output.

When filterInternalLumaEdgesFlag is equal to 1, the filtering of the internal vertical luma edges is specified by the following ordered steps:

1. When cTrafo[CurrMbAddr] is not equal to T_8x8, the process specified in subclause G.8.7.4.1 is invoked with interLayerDeblockingFlag, chromaEdgeFlag set equal to 0, verticalEdgeFlag set equal to 1, fieldModeInFrameFilteringFlag set equal to fieldMbInFrameFlag, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_Y$, currentVars, (xE_k, yE_k) set equal to (4, k) with k = 0..15, and cS_L as the inputs and cS_L as the output.

2. The process specified in subclause G.8.7.4.1 is invoked with interLayerDeblockingFlag, chromaEdgeFlag set equal to 0, verticalEdgeFlag set equal to 1, fieldModeInFrameFilteringFlag set equal to fieldMbInFrameFlag, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_Y$, currentVars, (xE_k, yE_k) set equal to (8, k) with k = 0..15, and cS_L as the inputs and cS_L as the output.

3. When cTrafo[CurrMbAddr] is not equal to T_8x8, the process specified in subclause G.8.7.4.1 is invoked with interLayerDeblockingFlag, chromaEdgeFlag set equal to 0, verticalEdgeFlag set equal to 1, fieldModeInFrameFilteringFlag set equal to fieldMbInFrameFlag, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_Y$, currentVars, (xE_k, yE_k) set equal to (12, k) with k = 0..15, and cS_L as the inputs and cS_L as the output.

When filterTopLumaMbEdgeFlag is equal to 1, the filtering of the top horizontal luma edge is specified as follows:

– If MbaffFrameFlag is equal to 1, (CurrMbAddr % 2) is equal to 0, CurrMbAddr is greater than or equal to (2 * PicWidthInMbs), fieldMbFlag[CurrMbAddr] is equal to 0, and fieldMbFlag[CurrMbAddr − 2 * PicWidthInMbs + 1] is equal to 1, the following ordered steps are specified:

  1. The process specified in subclause G.8.7.4.1 is invoked with interLayerDeblockingFlag, chromaEdgeFlag set equal to 0, verticalEdgeFlag set equal to 0, fieldModeInFrameFilteringFlag set equal to 1, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_Y$, currentVars, (xE_k, yE_k) set equal to (k, 0) with k = 0..15, and cS_L as the inputs and cS_L as the output.

  2. The process specified in subclause G.8.7.4.1 is invoked with interLayerDeblockingFlag, chromaEdgeFlag set equal to 0, verticalEdgeFlag set equal to 0, fieldModeInFrameFilteringFlag set equal to 1, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_Y$, currentVars, (xE_k, yE_k) set equal to (k, 1) with k = 0..15, and cS_L as the inputs and cS_L as the output.

– Otherwise, the process specified in subclause G.8.7.4.1 is invoked with interLayerDeblockingFlag, chromaEdgeFlag set equal to 0, verticalEdgeFlag set equal to 0, fieldModeInFrameFilteringFlag set equal to fieldMbInFrameFlag, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_Y$, currentVars, (xE_k, yE_k) set equal to (k, 0) with k = 0..15, and cS_L as the inputs and cS_L as the output.

When filterInternalLumaEdgesFlag is equal to 1, the filtering of the internal horizontal luma edges is specified by the following ordered steps:

1. When cTrafo[CurrMbAddr] is not equal to T_8x8, the process specified in subclause G.8.7.4.1 is invoked with interLayerDeblockingFlag, chromaEdgeFlag set equal to 0, verticalEdgeFlag set equal to 0, fieldModeInFrameFilteringFlag set equal to fieldMbInFrameFlag, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_Y$, currentVars, (xE_k, yE_k) set equal to (k, 4) with k = 0..15, and cS_L as the inputs and cS_L as the output.

2. The process specified in subclause G.8.7.4.1 is invoked with interLayerDeblockingFlag, chromaEdgeFlag set equal to 0, verticalEdgeFlag set equal to 0, fieldModeInFrameFilteringFlag set equal to fieldMbInFrameFlag, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_Y$, currentVars, (xE_k, yE_k) set equal to (k, 8) with k = 0..15, and cS_L as the inputs and cS_L as the output.

3. When cTrafo[CurrMbAddr] is not equal to T_8x8, the process specified in subclause G.8.7.4.1 is invoked with interLayerDeblockingFlag, chromaEdgeFlag set equal to 0, verticalEdgeFlag set equal to 0, fieldModeInFrameFilteringFlag set equal to fieldMbInFrameFlag, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_Y$, currentVars, (xE_k, yE_k) set equal to (k, 12) with k = 0..15, and cS_L as the inputs and cS_L as the output.

When ChromaArrayType is not equal to 0, for the filtering of both chroma components with C being replaced by Cb and Cr in $qpDB_C$ and cS_C, the following ordered steps are specified:

1. When filterLeftChromaMbEdgeFlag is equal to 1, the left vertical chroma edge is filtered by invoking the process specified in subclause G.8.7.4.1 with interLayerDeblockingFlag, chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 1, fieldModeInFrameFilteringFlag set equal to fieldMbInFrameFlag, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_C$, currentVars, (xE_k, yE_k) set equal to (0, k) with k = 0..(MbHeightC − 1), and cS_C as the inputs and cS_C as the output.

2. When filterInternalChromaEdgesFlag is equal to 1, the filtering of the internal vertical chroma edge is specified by the following ordered steps:

   a. When ChromaArrayType is not equal to 3 or cTrafo[CurrMbAddr] is not equal to T_8x8, the process specified in subclause G.8.7.4.1 is invoked with interLayerDeblockingFlag, chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 1, fieldModeInFrameFilteringFlag set equal to fieldMbInFrameFlag, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_C$, currentVars, (xE_k, yE_k) set equal to (4, k) with k = 0..(MbHeightC − 1), and cS_C as the inputs and cS_C as the output.

   b. When ChromaArrayType is equal to 3, the process specified in subclause G.8.7.4.1 is invoked with interLayerDeblockingFlag, chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 1, fieldModeInFrameFilteringFlag set equal to fieldMbInFrameFlag, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_C$, currentVars, (xE_k, yE_k) set equal to (8, k) with k = 0..(MbHeightC − 1), and cS_C as the inputs and cS_C as the output.

   c. When ChromaArrayType is equal to 3 and cTrafo[CurrMbAddr] is not equal to T_8x8, the process specified in subclause G.8.7.4.1 is invoked with interLayerDeblockingFlag, chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 1, fieldModeInFrameFilteringFlag set equal to fieldMbInFrameFlag, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_C$, currentVars, (xE_k, yE_k) set equal to (12, k) with k = 0..(MbHeightC − 1), and cS_C as the inputs and cS_C as the output.

3. When filterTopChromaMbEdgeFlag is equal to 1, the filtering of the top horizontal chroma edge is specified as follows:

   – If MbaffFrameFlag is equal to 1, (CurrMbAddr % 2) is equal to 0, CurrMbAddr is greater than or equal to (2 * PicWidthInMbs), fieldMbFlag[CurrMbAddr] is equal to 0, fieldMbFlag[CurrMbAddr − 2 * PicWidthInMbs + 1] is equal to 1, the following ordered steps are specified:

     a. The process specified in subclause G.8.7.4.1 is invoked with interLayerDeblockingFlag, chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 0, fieldModeInFrameFilteringFlag set equal to 1, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_C$, currentVars, (xE_k, yE_k) set equal to (k, 0) with k = 0..(MbWidthC − 1), and cS_C as the inputs and cS_C as the output.

     b. The process specified in subclause G.8.7.4.1 is invoked with interLayerDeblockingFlag, chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 0, fieldModeInFrameFilteringFlag set equal to 1, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_C$, currentVars, (xE_k, yE_k) set equal to (k, 1) with k = 0..(MbWidthC − 1), and cS_C as the inputs and cS_C as the output.

   – Otherwise, the process specified in subclause G.8.7.4.1 is invoked with interLayerDeblockingFlag, chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 0, fieldModeInFrameFilteringFlag set equal to fieldMbInFrameFlag, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_C$, currentVars, (xE_k, yE_k) set equal to (k, 0) with k = 0..(MbWidthC − 1), and cS_C as the inputs and cS_C as the output.

4. When filterInternalChromaEdgesFlag is equal to 1, the filtering of the internal horizontal chroma edge is specified by the following ordered steps:

   a. When ChromaArrayType is not equal to 3 or cTrafo[CurrMbAddr] is not equal to T_8x8, the process specified in subclause G.8.7.4.1 is invoked with interLayerDeblockingFlag, chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 0, fieldModeInFrameFilteringFlag set equal to fieldMbInFrameFlag, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_C$, currentVars, (xE_k, yE_k) set equal to (k, 4) with k = 0..(MbWidthC − 1), and cS_C as the inputs and cS_C as the output.

b. When ChromaArrayType is not equal to 1, the process specified in subclause G.8.7.4.1 is invoked with interLayerDeblockingFlag, chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 0, fieldModeInFrameFilteringFlag set equal to fieldMbInFrameFlag, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_C$, currentVars, (xE_k, yE_k) set equal to (k, 8) with k = 0..(MbWidthC − 1), and cS_C as the inputs and cS_C as the output.

c. When ChromaArrayType is equal to 2, the process specified in subclause G.8.7.4.1 is invoked with interLayerDeblockingFlag, chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 0, fieldModeInFrameFilteringFlag set equal to fieldMbInFrameFlag, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_C$, currentVars, (xE_k, yE_k) set equal to (k, 12) with k = 0..(MbWidthC − 1), and cS_C as the inputs and cS_C as the output.

d. When ChromaArrayType is equal to 3 and cTrafo[CurrMbAddr] is not equal to T_8x8, the process specified in subclause G.8.7.4.1 is invoked with interLayerDeblockingFlag, chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 0, fieldModeInFrameFilteringFlag set equal to fieldMbInFrameFlag, filterOffsetA, filterOffsetB, qpDB set equal to $qpDB_C$, currentVars, (xE_k, yE_k) set equal to (k, 12) with k = 0..(MbWidthC − 1), and cS_C as the inputs and cS_C as the output.

NOTE 2 – When field mode filtering (fieldModeInFrameFilteringFlag is equal to 1) is applied across the top horizontal edges of a frame macroblock, this vertical filtering across the top or bottom macroblock boundary may involve some samples that extend across an internal block edge that is also filtered internally in frame mode.

NOTE 3 – For example, in 4:2:0 chroma format when cTrafo[CurrMbAddr] is not equal to T_8x8, the following applies. 3 horizontal luma edges, 1 horizontal chroma edge for Cb, and 1 horizontal chroma edge for Cr are filtered that are internal to a macroblock. When field mode filtering (fieldModeInFrameFilteringFlag is equal to 1) is applied to the top edges of a frame macroblock, 2 horizontal luma, 2 horizontal chroma edges for Cb, and 2 horizontal chroma edges for Cr between the frame macroblock and the above macroblock pair are filtered using field mode filtering, for a total of up to 5 horizontal luma edges, 3 horizontal chroma edges for Cb, and 3 horizontal chroma edges for Cr filtered that are considered to be controlled by the frame macroblock. In all other cases, at most 4 horizontal luma, 2 horizontal chroma edges for Cb, and 2 horizontal chroma edges for Cr are filtered that are considered to be controlled by a particular macroblock.

G.8.7.4.1 SVC filtering process for block edges

Inputs to this process are:

– the variable interLayerDeblockingFlag,

– the variable chromaEdgeFlag,

– the variable verticalEdgeFlag,

– the variable fieldModeInFrameFilteringFlag,

– the variables filterOffsetA and filterOffsetB,

– the one-dimensional array qpDB with PicSizeInMbs elements specifying quantisation parameters,

– the collective term currentVars,

– a set of nE sample locations (xE_k, yE_k), with k = 0..(nE − 1), expressed relative to the upper left corner of the macroblock CurrMbAddr. The set of sample locations (xE_k, yE_k) represent the sample locations immediately to the right of a vertical edge (when verticalEdgeFlag is equal to 1) or immediately below a horizontal edge (when verticalEdgeFlag is equal to 0),

– an array of samples s′.

Output of this process is a modified version of the array s′.

The variable nE is derived as follows:

– If chromaEdgeFlag is equal to 0, nE is set equal to 16.

– Otherwise (chromaEdgeFlag is equal to 1), nE is set equal to ((verticalEdgeFlag = = 1) ? MbHeightC : MbWidthC).

Inside this subclause, the arrays that are collectively referred to as currentVars are referred to by their names as specified in subclause G.8.1.2.1.

The variable dy is set equal to (1 + fieldModeInFrameFilteringFlag).

The position of the upper-left luma sample of the macroblock CurrMbAddr is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddr = CurrMbAddr as input and the output being assigned to (xI, yI). During the process in subclause 6.4.1, the current macroblock is treated as field macroblock when fieldMbFlag[CurrMbAddr] is equal to 1, and it is treated as frame macroblock when fieldMbFlag[CurrMbAddr] is equal to 0.

The variables xP and yP are derived as follows:

– If chromaEdgeFlag is equal to 0, xP is set equal to xI and yP is set equal to yI.

– Otherwise (chromaEdgeFlag is equal to 1), xP is set equal to (xI / SubWidthC) and yP is set equal to ((yI + SubHeightC − 1) / SubHeightC).

For each sample location (xE_k, yE_k), k = 0..(nE − 1), the following ordered steps are specified:

1. The filtering process is applied to a set of eight samples across a 4x4 block horizontal or vertical edge denoted as p_i and q_i with i = 0..3 as shown in Figure 8-11 with the edge lying between p_0 and q_0. p_i and q_i with i = 0..3 are specified as follows:

   – If verticalEdgeFlag is equal to 1,

   $$q_i = s'[\ xP + xE_k + i,\ yP + dy * yE_k\ ] \quad (G\text{-}349)$$
   $$p_i = s'[\ xP + xE_k - i - 1,\ yP + dy * yE_k\ ] \quad (G\text{-}350)$$

   – Otherwise (verticalEdgeFlag is equal to 0),

   $$q_i = s'[\ xP + xE_k,\ yP + dy * (\ yE_k + i\ ) - (yE_k\ \%\ 2\ )\ ] \quad (G\text{-}351)$$
   $$p_i = s'[\ xP + xE_k,\ yP + dy * (\ yE_k - i - 1\ ) - (yE_k\ \%\ 2\ )\ ] \quad (G\text{-}352)$$

2. Let mbAddrP and mbAddrQ specify the addresses of the macroblocks that contain the samples p_0 and q_0, respectively.

3. The process specified in subclause G.8.7.4.2 is invoked with the sample values p_i and q_i (i = 0..3), interLayerDeblockingFlag, chromaEdgeFlag, verticalEdgeFlag, filterOffsetA, filterOffsetB, qP_p set equal to qpDB[mbAddrP], qP_q set equal to qpDB[mbAddrQ], sliceIdc, fieldMbFlag, mbType, cTrafo, predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, and rS_L as inputs, and the output is assigned to the filtered result sample values p'_i and q'_i with i = 0..2.

4. The input sample values p_i and q_i with i = 0..2 are replaced by the corresponding filtered result sample values p'_i and q'_i with i = 0..2 inside the sample array s′ as follows:

   – If verticalEdgeFlag is equal to 1,

   $$s'[\ xP + xE_k + i,\ yP + dy * yE_k\ ] = q'_i \quad (G\text{-}353)$$
   $$s'[\ xP + xE_k - i - 1,\ yP + dy * yE_k\ ] = p'_i \quad (G\text{-}354)$$

   – Otherwise (verticalEdgeFlag is equal to 0),

   $$s'[\ xP + xE_k,\ yP + dy * (\ yE_k + i\ ) - (\ yE_k\ \%\ 2\ )\ ] = q'_i \quad (G\text{-}355)$$
   $$s'[\ xP + xE_k,\ yP + dy * (\ yE_k - i - 1\ ) - (\ yE_k\ \%\ 2\ )\ ] = p'_i \quad (G\text{-}356)$$

G.8.7.4.2 SVC filtering process for a set of samples across a horizontal or vertical block edge

Inputs to this process are:

– the input sample values p_i and q_i with i = 0..3 of a single set of samples across an edge that is to be filtered,

– the variable interLayerDeblockingFlag,

– the variable chromaEdgeFlag,

– the variable verticalEdgeFlag,

– the variables filterOffsetA and filterOffsetB,

– the variables qP_p and qP_q,

– the arrays sliceIdc, fieldMbFlag, mbType, cTrafo, predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, and mvL1,

– an array rS_L containing residual sample values.

Outputs of this process are the filtered result sample values p'_i and q'_i with i in the range of 0..2.

The content dependent boundary filtering strength variable bS is derived as follows:

– If chromaEdgeFlag is equal to 0, the SVC derivation process for the luma content dependent boundary filtering strength specified in subclause G.8.7.4.3 is invoked with p_0, q_0, interLayerDeblockingFlag, verticalEdgeFlag, sliceIdc, fieldMbFlag, mbType, cTrafo, predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, and rS_L as inputs, and the output is assigned to bS.

– Otherwise (chromaEdgeFlag is equal to 1), the bS used for filtering a set of samples of a horizontal or vertical chroma edge is set equal to the value of bS for filtering the set of samples of a horizontal or vertical luma edge, respectively, that contains the luma sample at location (SubWidthC * x, SubHeightC * y) inside the luma array of the same field, where (x, y) is the location of the chroma sample q_0 inside the chroma array for that field.

The process specified in subclause 8.7.2.2 is invoked with p_0, q_0, p_1, q_1, chromaEdgeFlag, bS, filterOffsetA, filterOffsetB, qP_p, and qP_q as inputs, and the output is assigned to filterSamplesFlag, indexA, α, and β.

Depending on the variable filterSamplesFlag, the following applies:

– If filterSamplesFlag is equal to 1, the following applies:

  – If bS is less than 4, the process specified in subclause 8.7.2.3 is invoked with p_i and q_i (i = 0..2), chromaEdgeFlag, bS, β, and indexA given as input, and the output is assigned to p'_i and q'_i (i = 0..2).

  – Otherwise (bS is equal to 4), the process specified in subclause 8.7.2.4 is invoked with p_i and q_i (i = 0..3), chromaEdgeFlag, α, and β given as input, and the output is assigned to p'_i and q'_i (i = 0..2).

– Otherwise (filterSamplesFlag is equal to 0), the filtered result samples p'_i and q'_i (i = 0..2) are replaced by the corresponding input samples p_i and q_i:

for i = 0..2, $p'_i = p_i$ (G-357)
for i = 0..2, $q'_i = q_i$ (G-358)

G.8.7.4.3 SVC derivation process for the luma content dependent boundary filtering strength

Inputs to this process are:

– the input sample values p_0 and q_0 of a single set of samples across an edge that is to be filtered,

– the variable interLayerDeblockingFlag,

– the variable verticalEdgeFlag,

– the arrays sliceIdc, fieldMbFlag, mbType, cTrafo, predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, and mvL1,

– the array rS_L containing residual sample values.

Output of this process is the variable bS.

The following variables are derived as specified in the following:

– mbAddrP and mbAddrQ specify the macroblocks containing the samples p_0 and q_0, respectively.

– mbPartIdxP and mbPartIdxQ specify the macroblock partitions containing the samples p_0 and q_0, respectively.

– subMbPartIdxP and subMbPartIdxQ specify the sub-macroblock partitions containing the samples p_0 and q_0, respectively.

– pFLXP and pFLXQ with X being replaced by 0 and 1 are equal to predFlagLX[mbAddrP][mbPartIdxP] and predFlagLX[mbAddrQ][mbPartIdxQ], respectively.

– refLXP and refLXQ with X being replaced by 0 and 1 are equal to refIdxLX[mbAddrP][mbPartIdxP] and refIdxLX[mbAddrQ][mbPartIdxQ], respectively.

– mvLXP and mvLXQ with X being replaced by 0 and 1 are equal to mvLX[mbAddrP][mbPartIdxP][subMbPartP] and mvLX[mbAddrQ][mbPartIdxQ][subMbPartQ], respectively.

– numMvP and numMvQ are equal to (pFL0P + pFL1P) and (pFL0Q + pFL1Q), respectively.

– When numMvP and numMvQ are both equal to 1, the variables refX and mvX with X being replaced by P and Q are derived as follows:

  – If pFL0X is equal to 1, refX is set equal to refL0X and mvX is set equal to mvL0X.

– Otherwise (pFL1X is equal to 1), refX is set equal to refL1X and mvX is set equal to mvL1X.

– sliceX with X being replaced by P and Q is the slice with DQId equal to (sliceIdc[mbAddrX] & 127) and first_mb_in_slice equal to (sliceIdc[mbAddrX] >> 7)

Let the variable mixedModeEdgeFlag be derived as follows:

– If MbaffFrameFlag is equal to 1 and fieldMbFlag[mbAddrP] is not equal to fieldMbFlag[mbAddrQ], mixedModeEdgeFlag is set equal to 1.

– Otherwise, mixedModeEdgeFlag is set equal to 0.

The variable bS is derived as follows:

– If interLayerDeblockingFlag is equal to 1 and mbType[mbAddrP] specifies an Inter macroblock prediction mode, bS is set equal to 0.

  NOTE 1 – This subclause is not invoked when interLayerDeblockingFlag is equal to 1 and mbType[mbAddrQ] specifies an Inter macroblock prediction mode.

– Otherwise, if SpatialResolutionChangeFlag is equal to 1 and either or both mbType[mbAddrP] or mbType[mbAddrQ] is equal to I_BL, the following applies:

  – If either mbType[mbAddrP] or mbType[mbAddrQ] specifies an Intra macroblock prediction mode other than I_BL, the following applies:

    – If verticalEdgeFlag is equal to 1 or both fieldMbFlag[mbAddrP] and fieldMbFlag[mbAddrQ] are equal to 0, bS is set equal to 4.

    – Otherwise (verticalEdgeFlag is equal to 0 and either or both fieldMbFlag[mbAddrP] or fieldMbFlag[mbAddrQ] is equal to 1), bS is set equal to 3.

  – Otherwise, if mbType[mbAddrP] is equal to I_BL and mbType[mbAddrQ] is equal to I_BL, the following applies:

    – If any of the following conditions are true, bS is set equal to 1:

      – cTrafo[mbAddrP] is equal to T_8x8 and the 8x8 luma transform block coded in sliceP and associated with the 8x8 luma block containing sample p_0 contains non-zero transform coefficient levels,

      – cTrafo[mbAddrP] is equal to T_4x4 and the 4x4 luma transform block coded in sliceP and associated with the 4x4 luma block containing sample p_0 contains non-zero transform coefficient levels,

      – cTrafo[mbAddrQ] is equal to T_8x8 and the 8x8 luma transform block coded in sliceQ and associated with the 8x8 luma block containing sample q_0 contains non-zero transform coefficient levels,

      – cTrafo[mbAddrQ] is equal to T_4x4 and the 4x4 luma transform block coded in sliceQ and associated with the 4x4 luma block containing sample q_0 contains non-zero transform coefficient levels.

      NOTE 2 – A luma transform block coded in a particular slice is considered to contain non-zero transform coefficient levels, if non-zero transform coefficients are transmitted in the macroblock layer of the slice for the considered luma transform block. Transform coefficient levels that are transmitted in layers that are used for inter-layer prediction are not taken into account.

    – Otherwise, bS is set equal to 0.

  – Otherwise (either mbType[mbAddrP] or mbType[mbAddrQ] specifies an Inter macroblock prediction mode), the following applies:

    – If any of the following conditions are true, bS is set equal to 2:

      – mbType[mbAddrP] specifies an Inter macroblock prediction type, cTrafo[mbAddrP] is equal to T_8x8, and the array rS_L contains non-zero samples for the 8x8 luma block containing sample p_0,

      – mbType[mbAddrP] specifies an Inter macroblock prediction type, cTrafo[mbAddrP] is equal to T_4x4, and the array rS_L contains non-zero samples for the 4x4 luma block containing sample p_0,

      – mbType[mbAddrQ] specifies an Inter macroblock prediction type, cTrafo[mbAddrQ] is equal to T_8x8, and the array rS_L contains non-zero samples for the 8x8 luma block containing sample q_0,

– mbType[mbAddrQ] specifies an Inter macroblock prediction type, cTrafo[mbAddrQ] is equal to T_4x4, and the array rS_L contains non-zero samples for the 4x4 luma block containing sample q_0.

NOTE 3 – The array rS_L contains samples for the accumulated residual signal. Transform coefficient values of layer representations that are used for inter-layer prediction are taken into account.

– Otherwise, bS is set equal to 1.

– Otherwise, if the block edge is also a macroblock edge and any of the following conditions are true, bS is set equal to 4:

– fieldMbFlag[mbAddrP] is equal to 0 and fieldMbFlag[mbAddrQ] is equal to 0 and either or both mbType[mbAddrP] or mbType[mbAddrQ] specify an Intra macroblock prediction mode,

– MbaffFrameFlag is equal to 1 or field_pic_flag is equal to 1, and verticalEdgeFlag is equal to 1, and either or both mbType[mbAddrP] or mbType[mbAddrQ] specify an Intra macroblock prediction mode.

– Otherwise, if any of the following conditions are true, bS is set equal to 3:

– mixedModeEdgeFlag is equal to 0 and either or both mbType[mbAddrP] or mbType[mbAddrQ] specify an Intra macroblock prediction mode,

– mixedModeEdgeFlag is equal to 1, verticalEdgeFlag is equal to 0, and either or both mbType[mbAddrP] or mbType[mbAddrQ] specify an Intra macroblock prediction mode.

– Otherwise, if any of the following conditions are true, bS is set equal to 2:

– cTrafo[mbAddrP] is equal to T_8x8 and either the array rS_L contains non-zero samples for the 8x8 luma block containing sample p_0 or ((sliceIdc[mbAddrP] & 127) is equal to 0 and the 8x8 luma transform block coded in sliceP and associated with the 8x8 luma block containing sample p_0 contains non-zero transform coefficient levels),

– cTrafo[mbAddrP] is equal to T_4x4 and either the array rS_L contains non-zero samples for the 4x4 luma block containing sample p_0 or ((sliceIdc[mbAddrP] & 127) is equal to 0 and the 4x4 luma transform block coded in sliceP and associated with the 4x4 luma block containing sample p_0 contains non-zero transform coefficient levels),

– cTrafo[mbAddrQ] is equal to T_8x8 and either the array rS_L contains non-zero samples for the 8x8 luma block containing sample q_0 or ((sliceIdc[mbAddrQ] & 127) is equal to 0 and the 8x8 luma transform block coded in sliceQ and associated with the 8x8 luma block containing sample q_0 contains non-zero transform coefficient levels),

– cTrafo[mbAddrQ] is equal to T_4x4 and either the array rS_L contains non-zero samples for the 4x4 luma block containing sample q_0 or ((sliceIdc[mbAddrQ] & 127) is equal to 0 and the 4x4 luma transform block coded in sliceQ and associated with the 4x4 luma block containing sample q_0 contains non-zero transform coefficient levels).

NOTE 4 – The array rS_L contains samples for the accumulated residual signal. Transform coefficient values of layer representations that are used for inter-layer prediction are taken into account.

– Otherwise, if profile_idc is equal to 83 and any of the following conditions are true, bS is set equal to 2:

– cTrafo[mbAddrP] is equal to T_8x8 and the array sTCoeff[mbAddrP] contains non-zero scaled transform coefficient values for the 8x8 luma transform block associated with the 8x8 luma block containing sample p_0,

– cTrafo[mbAddrP] is equal to T_4x4 and the array sTCoeff[mbAddrP] contains non-zero scaled transform coefficient values for the 4x4 luma transform block associated with the 4x4 luma block containing sample p_0,

– cTrafo[mbAddrQ] is equal to T_8x8 and the array sTCoeff[mbAddrQ] contains non-zero scaled transform coefficient values for the 8x8 luma transform block associated with the 8x8 luma block containing sample q_0,

– cTrafo[mbAddrQ] is equal to T_4x4 and the array sTCoeff[mbAddrQ] contains non-zero scaled transform coefficient values for the 4x4 luma transform block associated with the 4x4 luma block containing sample q_0.

– Otherwise, if mixedModeEdgeFlag is equal to 1 or any of the following conditions are true, bS is set equal to 1:

1. numMvP is not equal to numMvQ.

2. numMvP and numMvQ are both equal to 1 and any of the following conditions are true:

– refP and refQ specify different reference pictures,

– the absolute difference between the horizontal or vertical components of the motion vectors mvP and mvQ is greater than or equal to 4 in units of quarter luma frame samples.

3. numRefP and numRefQ are both equal to 2 and any of the following conditions are true:

   a. refL0P and refL1P specify different reference pictures and any of the following conditions are true:

      i. both of the following conditions are true:

         – refL0P and refL0Q specify different reference pictures or refL1P and refL1Q specify different reference pictures,

         – refL0P and refL1Q specify different reference pictures or refL1P and refL0Q specify different reference pictures.

      ii. refL0P and refL0Q specifies the same reference picture, refL1P and refL1Q specify the same reference picture, and any of the following conditions are true:

         – the absolute difference between the horizontal or vertical components of the motion vectors mvL0P and mvL0Q is greater than or equal to 4 in units of quarter luma frame samples,

         – the absolute difference between the horizontal or vertical components of the motion vectors mvL1P and mvL1Q is greater than or equal to 4 in units of quarter luma frame samples.

      iii. refL0P and refL1Q specifies the same reference picture, refL1P and refL0Q specify the same reference picture, and any of the following conditions are true:

         – the absolute difference between the horizontal or vertical components of the motion vectors mvL0P and mvL1Q is greater than or equal to 4 in units of quarter luma frame samples,

         – the absolute difference between the horizontal or vertical components of the motion vectors mvL1P and mvL0Q is greater than or equal to 4 in units of quarter luma frame samples.

   b. refL0P and refL1P specify the same reference picture and any of the following conditions are true:

      i. refL0Q or refL1Q specify a different reference picture than refL0P (or refL1P).

      ii. refL0Q and refL1Q specify the same reference picture as refL0P (and refL1P) and both of the following conditions are true:

         – the absolute difference between the horizontal or vertical components of the motion vectors mvL0P and mvL0Q is greater than or equal to 4 in units of quarter luma frame samples or the absolute difference between the horizontal or vertical components of the motion vectors mvL1P and mvL1Q is greater than or equal to 4 in units of quarter luma frame samples,

         – the absolute difference between the horizontal or vertical components of the motion vectors mvL0P and mvL1Q is greater than or equal to 4 in units of quarter luma frame samples or the absolute difference between the horizontal or vertical components of the motion vectors mvL1P and mvL0Q is greater than or equal to 4 in units of quarter luma frame samples.

   NOTE 5 – The determination of whether the reference pictures used for the two macroblock/sub-macroblock partitions are the same or different is based only on which pictures are referenced, without regard to whether a prediction is formed using an index into reference picture list 0 or an index into reference picture list 1, and also without regard to whether the index position within a reference picture list is different.

   NOTE 6 – A vertical difference of 4 in units of quarter luma frame samples is a difference of 2 in units of quarter luma field samples

– Otherwise, bS is set equal to 0.

The variable interProfileConformanceFlag is derived as follows:

– If DQId is greater than 0, interLayerDeblockingFlag is equal to 0, and any of the following conditions are true, interProfileConformanceFlag is set equal to 1:

  – profile_idc is equal to 83 and constraint_set1_flag is equal to 1,

  – profile_idc is equal to 86 and constraint_set0_flag is equal to 1.

– Otherwise, interProfileConformanceFlag is set equal to 0.

When interProfileConformanceFlag is equal to 1 and both mbType[mbAddrP] and mbType[mbAddrQ] specify an Inter macroblock prediction mode, it is a requirement of bitstream conformance that the following constraints are obeyed:

– When cTrafo[mbAddrP] is equal to T_8x8 and the array sTCoeff[mbAddrP] contains at least one non-zero scaled transform coefficient value for the 8x8 luma transform block associated with the 8x8 luma block containing sample p_0, the bitstream shall not contain data that result in an array rS_L for which all sample values are equal to 0 for the 8x8 luma block containing sample p_0.

– When cTrafo[mbAddrP] is equal to T_4x4 and the array sTCoeff[mbAddrP] contains at least one non-zero scaled transform coefficient value for the 4x4 luma transform block associated with the 4x4 luma block containing sample p_0, the bitstream shall not contain data that result in an array rS_L for which all sample values are equal to 0 for the 4x4 luma block containing sample p_0.

– When cTrafo[mbAddrQ] is equal to T_8x8 and the array sTCoeff[mbAddrQ] contains at least one non-zero scaled transform coefficient value for the 8x8 luma transform block associated with the 8x8 luma block containing sample q_0, the bitstream shall not contain data that result in an array rS_L for which all sample values are equal to 0 for the 8x8 luma block containing sample q_0.

– When cTrafo[mbAddrQ] is equal to T_4x4 and the array sTCoeff[mbAddrQ] contains at least one non-zero scaled transform coefficient value for the 4x4 luma transform block associated with the 4x4 luma block containing sample q_0, the bitstream shall not contain data that result in an array rS_L for which all sample values are equal to 0 for the 4x4 luma block containing sample q_0.

G.8.8 Specification of bitstream subsets

Subclause G.8.8.1 specifies the sub-bitstream extraction process.

Subclause G.8.8.2 specifies the base layer bitstream.

G.8.8.1 Sub-bitstream extraction process

It is requirement of bitstream conformance that any sub-bitstream that is the output of the process specified in this subclause with pIdTarget equal to any value in the range of 0 to 63, inclusive, tIdTarget equal to any value in the range of 0 to 7, inclusive, dIdTarget equal to any value in the range of 0 to 7, inclusive, and qIdTarget equal to any value in the range of 0 to 15, inclusive, shall be conforming to this Recommendation | International Standard.

NOTE – A conforming bitstream contains one or more coded slice NAL units with priority_id equal to 0, dependency_id equal to 0, quality_id equal to 0, and temporal_id equal to 0.

Inputs to this process are:

– a variable pIdTarget (when present),

– a variable tIdTarget (when present),

– a variable dIdTarget (when present),

– a variable qIdTarget (when present).

Output of this process is a sub-bitstream.

When pIdTarget is not present as input to this subclause, pIdTarget is inferred to be equal to 63.

When tIdTarget is not present as input to this subclause, tIdTarget is inferred to be equal to 7.

When dIdTarget is not present as input to this subclause, dIdTarget is inferred to be equal to 7.

When qIdTarget is not present as input to this subclause, qIdTarget is inferred to be equal to 15.

The sub-bitstream is derived by applying the following operations in sequential order:

1. Mark all VCL NAL units and filler data NAL units for which any of the following conditions are true as "to be removed from the bitstream":
   – priority_id is greater than pIdTarget,
   – temporal_id is greater than tIdTarget,
   – dependency_id is greater than dIdTarget,
   – dependency_id is equal to dIdTarget and quality_id is greater than qIdTarget.
2. Remove all access units for which all VCL NAL units are marked as "to be removed from the bitstream".
3. Remove all VCL NAL units and filler data NAL units that are marked as "to be removed from the bitstream".
4. When dIdTarget is equal to 0 and qIdTarget is equal to 0, remove the following NAL units:
   – all NAL units with nal_unit_type equal to 14 or 15,

– all NAL units with nal_unit_type equal to 6 in which the first SEI message has payloadType in the range of 24 to 35, inclusive.

5. Remove all NAL units with nal_unit_type equal to 6 that only contain SEI messages that are part of a scalable nesting SEI message with any of the following properties:

   – sei_temporal_id is greater than tIdTarget,

   – the minimum value of (sei_dependency_id[i] << 4) + sei_quality_id[i] for all i in the range of 0 to num_layer_representations_minus1, inclusive, is greater than (dIdTarget << 4) + qIdTarget.

6. Remove all NAL units with nal_unit_type equal to 6 that contain SEI messages with payloadType equal to 24, 28, or 29.

G.8.8.2 Specification of the base layer bitstream

Each scalable bitstream that conforms to this specification shall contain a base layer bitstream that conforms to one or more of the profiles specified in Annex A. This base layer bitstream is derived by invoking the sub-bitstream extraction process as specified in subclause G.8.8.1 with dIdTarget being equal to 0 and qIdTarget being equal to 0 and the base layer bitstream being the output.

NOTE – Although all scalable bitstreams that conform to one or more of the profiles specified in this annex contain a base layer bitstream that conforms to one or more of the profiles specified in Annex A, the complete scalable bitstream (prior to operation of the base layer extraction process specified in this subclause) may not conform to any profile specified in Annex A.

G.9 Parsing process

Inputs to this process are bits from the RBSP, a request for a value of a syntax element, and values of prior parsed syntax elements (if applicable).

Output of this process is the value of the syntax element.

This process is invoked for all syntax elements in the syntax tables in subclause G.7.3 with descriptors equal to u(v), ue(v), me(v), se(v), te(v), ce(v), and ae(v).

When the parsing process is invoked for the first request for a value of a syntax element in the slice data and entropy_coding_mode_flag is equal to 1, the following ordered steps apply:

1. The initialisation process as specified in subclause 9.3.1 is invoked, where a slice_type equal to EI is interpreted as I, a slice_type equal to EP is interpreted as P, and a slice_type equal to EB is interpreted as B.
2. The initialisation process as specified in subclause G.9.3.1 is invoked.

Depending on entropy_coding_mode_flag and the descriptor, the value of a syntax element is derived as follows:

– If entropy_coding_mode_flag is equal to 0, the following applies:

  1. The parsing process for syntax elements coded as coded as ue(v), se(v), or te(v) is specified in subclause 9.1.
  2. The parsing process for the syntax element coded_block_pattern is specified in subclause G.9.1.
  3. The parsing process for syntax elements of the residual_block_cavlc( ) syntax structure is specified in subclause G.9.2.

– Otherwise (entropy_coding_mode_flag is equal to 1), the value of the syntax element is derived as follows:

  – If the syntax element is equal to base_mode_flag, motion_prediction_flag_l0, motion_prediction_flag_l1, or residual_prediction_flag, the following applies:

    1. The binarization process as specified in subclause G.9.3.2 is invoked.
    2. The decoding process flow as specified in subclause G.9.3.3 is invoked.

  – Otherwise (the syntax element is not equal to base_mode_flag, motion_prediction_flag_l0, motion_prediction_flag_l1, or residual_prediction_flag), the following applies:

    1. The binarization process as specified in subclause 9.3.2 is invoked, where a slice_type equal to EI is interpreted as I, a slice_type equal to EP is interpreted as P, and a slice_type equal to EB is interpreted as B.
    2. The decoding process flow as specified in subclause 9.3.3 is invoked.

       NOTE – For macroblocks with base_mode_flag equal to 1, mb_type is inferred to be equal to Mb_Inferred and the specifications in subclause G.7.4.6 apply.

3. When the syntax element is equal to mb_type and the decoded value of mb_type is equal to I_PCM, the arithmetic decoding engine is initialised after decoding of any pcm_alignment_zero_bit and all pcm_sample_luma and pcm_sample_chroma data as specified in subclause 9.3.1.2.

G.9.1 Alternative parsing process for coded block pattern

This process is invoked for the parsing syntax elements with descriptor equal to me(v) when entropy_coding_mode_flag is equal to 0.

Inputs to this process are bits from the RBSP.

Outputs of this process is a value of the syntax element coded_block_pattern.

The parsing process for the syntax elements begins with reading the bits starting at the current location in the bitstream up to and including the first non-zero bit. By counting the number of leading bits that are equal to 0 and assigning this value to the variable leadingZeroBits, the variable codeNum is then derived as

$$\text{codeNum} = 2^{\text{leadingZeroBits}} - 1 + \text{read\_bits(leadingZeroBits)}$$

where the value returned from read_bits(leadingZeroBits) is interpreted as a binary representation of an unsigned integer with most significant bit written first.

When ref_layer_dq_id is greater than or equal to 0 and (scan_idx_end − scan_idx_start) is less than 15, codeNum is set equal to (codeNum − 1).

Depending on codeNum, the following applies:

– If codeNum is equal to −1, the following ordered steps are specified:

  1. The derivation process for neighbouring macroblocks specified in subclause 6.4.11.1 is invoked and the output is assigned to mbAddrA and mbAddrB.
  2. When mbAddrN is available, the variable codedBlockPatternN (with N being either A or B) is derived as follows:
     – If mb_type for the macroblock mbAddrN is equal to P_Skip, B_Skip, or I_PCM, codedBlockPatternN is set equal to 0.
     – Otherwise (mb_type for the macroblock mbAddrN is not equal to P_Skip, B_Skip, or I_PCM), codedBlockPatternN is set equal to (16 * cbpChromaN + cbpLumaN) with cbpChromaN and cbpLumaN representing the values of CodedBlockPatternLuma and CodedBlockPatternChroma for the macroblock mbAddrN.
  3. Depending on mbAddrA and mbAddrB, the following applies:
     – If mbAddrA is available, coded_block_pattern is set equal to codedBlockPatternA.
     – Otherwise, if mbAddrB is available, coded_block_pattern is set equal to codedBlockPatternB.
     – Otherwise (mbAddrA and mbAddrB are not available), coded_block_pattern is set equal to 0.

– Otherwise (codeNum is greater than or equal to 0), the mapping process for coded block pattern as specified in subclause 9.1.2 is invoked with codeNum as input and the output is assigned to the syntax element coded_block_pattern.

G.9.2 Alternative CAVLC parsing process for transform coefficient levels

This process is invoked for the parsing syntax elements with descriptor equal to ce(v) when entropy_coding_mode_flag is equal to 0.

Inputs to this process are a request for a value of a syntax element, bits from slice data, a maximum number of non-zero transform coefficient levels maxNumCoeff, the luma block index luma4x4BlkIdx or the chroma block index chroma4x4BlkIdx, cb4x4BlkIdx or cr4x4BlkIdx of the current block of transform coefficient levels.

Output of this process is the list coeffLevel containing transform coefficient levels of the luma block with block index luma4x4BlkIdx or the chroma block with block index chroma4x4BlkIdx, cb4x4BlkIdx or cr4x4BlkIdx.

The process is specified in the following ordered steps:

1. All transform coefficient levels, with indices from 0 to maxNumCoeff − 1, in the list coeffLevel are set equal to 0.

2. The total number of non-zero transform coefficient levels TotalCoeff(coeff_token) and the number of trailing one transform coefficient levels TrailingOnes(coeff_token) are derived by parsing coeff_token as specified by the following ordered steps:

   a. The parsing process of coeff_token as specified in subclause 9.2.1 is invoked and the outputs are TotalCoeff(coeff_token), TrailingOnes(coeff_token), and nC.

      NOTE – For macroblocks with base_mode_flag equal to 1, mb_type is inferred to be equal to Mb_Inferred and the specifications in subclause G.7.4.6 apply.

   b. When the CAVLC parsing process is invoked for LumaLevel4x4, LumaLevel8x8, Intra16x16ACLevel, ChromaACLevel, CbIntra16x16ACLevel, or CrIntra16x16ACLevel and (scan_idx_end − scan_idx_start) is less than 15, nC is modified by setting it equal to Min(7, nC), and the additional parsing process for total number of non-zero transform coefficient levels and number of trailing ones as specified in subclause G.9.2.1 is invoked with nC, totalCoeffStart set equal to TotalCoeff(coeff_token), and trailingOnesStart set equal to TrailingOnes(coeff_token) as the inputs and the outputs are assigned to TotalCoeff(coeff_token) and TrailingOnes(coeff_token).

3. When TotalCoeff(coeff_token) is greater than 0, the following ordered steps are specified:

   a. The non-zero transform coefficient levels are derived by parsing trailing_ones_sign_flag, level_prefix, and level_suffix as specified in subclause 9.2.2.

   b. The runs of zero transform coefficient levels before each non-zero transform coefficient level are derived by parsing total_zeros and run_before as specified in subclause G.9.2.2.

   c. The level and run information are combined into the list coeffLevel as specified in subclause 9.2.4.

G.9.2.1 Additional parsing process for total number of non-zero transform coefficient levels and number of trailing ones

Inputs to this process are variables nC, totalCoeffStart, and trailingOnesStart.

Outputs of this process are variables totalCoeff and trailingOnes.

Let invTotalCoeff(coeffTokenIdx) and invTrailingOnes(coeffTokenIdx) be functions that map the variable coeffTokenIdx to the variables nX and nY, respectively, as specified in Table G-10 for each value of the variable nC.

A variable dX is set equal to (scan_idx_end − scan_idx_start + 2). A variable dY is set equal to Min(4, scan_idx_end − scan_idx_start + 2). A variable targetCoeffTokenIdx is derived as specified by Table G-10 given the variables nC, nX = totalCoeffStart, and nY = trailingOnesStart.

The bitstream shall not contain data that result in a value of targetCoeffTokenIdx that exceeds the range of values from 0 to (dX * dY – Min(7, (1 << (dY – 1)))), inclusive.

A variable coeffTokenIdx is derived as specified by the following pseudo code:

```
for( coeffTokenIdx = 0, i = 0; i <= targetCoeffTokenIdx; coeffTokenIdx++ )
  if( invTotalCoeff( coeffTokenIdx ) < dX  &&  invTrailingOnes( coeffTokenIdx ) < dY )        (G-359)
    i++
```

The variable totalCoeff is set equal to invTotalCoeff(coeffTokenIdx − 1) and the variable trailingOnes is set equal to invTrailingOnes(coeffTokenIdx − 1).

When the CAVLC parsing process is invoked for Intra16x16ACLevel, CbIntra16x16ACLevel, CrIntra16x16ACLevel, or ChromaACLevel, it is a requirement of bitstream conformance that the bitstream shall not contain data that result in totalCoeff being greater than (scan_idx_end − Max(1, scan_idx_start) + 1).

Table G-10 – Mapping of (nX, nY) to coeffTokenIdx and vice versa

nY	nX	0 <= nC < 2	2 <= nC < 4	4 <= nC < 8
0	0	0	0	0
0	1	4	7	16
1	1	1	1	1
0	2	9	11	20
1	2	5	5	8
2	2	2	2	2

Table G-10 – Mapping of (nX, nY) to coeffTokenIdx and vice versa

nY	nX	0 <= nC < 2	2 <= nC < 4	4 <= nC < 8
0	3	13	15	23
1	3	10	8	11
2	3	7	9	9
3	3	3	3	3
0	4	17	19	24
1	4	14	12	13
2	4	11	13	12
3	4	6	4	4
0	5	21	22	28
1	5	18	16	15
2	5	15	17	14
3	5	8	6	5
0	6	25	23	30
1	6	22	20	17
2	6	19	21	18
3	6	12	10	6
0	7	29	27	31
1	7	26	24	21
2	7	23	25	22
3	7	16	14	7
0	8	32	31	32
1	8	30	28	25
2	8	27	29	26
3	8	20	18	10
0	9	33	35	36
1	9	34	32	33
2	9	31	33	29
3	9	24	26	19
0	10	37	39	40
1	10	38	36	37
2	10	35	37	34
3	10	28	30	27
0	11	41	42	44
1	11	42	40	41
2	11	39	41	38
3	11	36	34	35
0	12	45	43	47

Table G-10 – Mapping of (nX, nY) to coeffTokenIdx and vice versa

nY	nX	0 <= nC < 2	2 <= nC < 4	4 <= nC < 8
1	12	46	44	45
2	12	43	45	42
3	12	40	38	39
0	13	50	47	49
1	13	49	48	48
2	13	47	49	46
3	13	44	46	43
0	14	54	51	53
1	14	51	54	50
2	14	52	52	51
3	14	48	50	52
0	15	58	55	57
1	15	55	56	54
2	15	56	57	55
3	15	53	53	56
0	16	61	59	61
1	16	59	60	58
2	16	60	61	59
3	16	57	58	60

G.9.2.2 Alternative parsing process for run information

Inputs to this process are bits from slice data and the number of non-zero transform coefficient levels TotalCoeff(coeff_token).

Output of this process is a list of runs of zero transform coefficient levels preceding non-zero transform coefficient levels called runVal.

The variable maxCoeff is derived as follows:

– If the CAVLC parsing process is invoked for Intra16x16DCLevel, CbIntra16x16DCLevel, or CrIntra16x16DCLevel, maxCoeff is set equal to 16.

– Otherwise, if the CAVLC parsing process is invoked for ChromaDCLevel, maxCoeff is set equal to 4 * chroma_format_idc.

– Otherwise, if the CAVLC parsing process is invoked for LumaLevel4x4 or LumaLevel8x8, maxCoeff is set equal to (scan_idx_end − scan_idx_start + 1).

– Otherwise (the CAVLC parsing process is invoked for Intra16x16ACLevel, CbIntra16x16ACLevel, CrIntra16x16ACLevel, or ChromaACLevel), maxCoeff is set equal to (scan_idx_end − Max(1, scan_idx_start) + 1).

Initially, an index i is set equal to 0.

The variable zerosLeft is derived as follows:

– If the number of non-zero transform coefficient levels TotalCoeff(coeff_token) is equal to the maximum number of non-zero transform coefficient levels maxCoeff, a variable zerosLeft is set equal to 0.

– Otherwise (the number of non-zero transform coefficient levels TotalCoeff(coeff_token) is less than the maximum number of non-zero transform coefficient levels maxCoeff), total_zeros is decoded and zerosLeft is set equal to its value.

The VLC used to decode total_zeros is derived as follows:

– If maxCoeff is less than or equal to 4, one of the VLCs specified in Table 9-9(a) is used with tzVlcIndex being derived by

$$\text{tzVlcIndex} = \text{TotalCoeff(coeff\_token)} + 4 - \text{maxCoeff} \quad \text{(G-360)}$$

– Otherwise, if maxCoeff is greater than 4 and less than or equal to 8, one of the VLCs specified in Table 9-9(b) is used with tzVlcIndex being derived by

$$\text{tzVlcIndex} = \text{TotalCoeff(coeff\_token)} + 8 - \text{maxCoeff} \quad \text{(G-361)}$$

– Otherwise, if maxCoeff is greater than 8 and less than 15, VLCs from Tables 9-7 and 9-8 are used with tzVlcIndex being derived by

$$\text{tzVlcIndex} = \text{TotalCoeff(coeff\_token)} + 16 - \text{maxCoeff} \quad \text{(G-362)}$$

– Otherwise (maxCoeff is greater than or equal to 15), VLCs from Tables 9-7 and 9-8 are used with tzVlcIndex equal to TotalCoeff(coeff_token).

The following procedure is then applied iteratively (TotalCoeff(coeff_token) − 1) times:

1. The variable runVal[i] is derived as follows:
   – If zerosLeft is greater than zero, a value run_before is decoded based on Table 9-10 and zerosLeft. runVal[i] is set equal to run_before.
   – Otherwise (zerosLeft is equal to 0), runVal[i] is set equal to 0.
2. The value of runVal[i] is subtracted from zerosLeft and the result assigned to zerosLeft. The result of the subtraction shall be greater than or equal to 0.
3. The index i is incremented by 1.

Finally the value of zerosLeft is assigned to runVal[i].

G.9.3 Alternative CABAC parsing process for slice data in scalable extension

Subclause G.9.3.1 specifies the initialisation process for the alternative CABAC parsing process for slice data in scalable extension.

Subclause G.9.3.2 specifies the binarization process for the alternative CABAC parsing process for slice data in scalable extension.

Subclause G.9.3.3 specifies the decoding process flow for the alternative CABAC parsing process for slice data in scalable extension.

G.9.3.1 Initialisation process

Outputs of this process are the initialised CABAC context variables indexed by ctxIdx.

Tables G-12 and G-13 contain the values of the variables n and m used in the initialisation of context variables that are assigned to syntax element base_mode_flag, motion_prediction_flag_l0, motion_prediction_flag_l1, and residual_prediction_flag in subclause G.7.3.4.1 and G.7.3.6. For all other syntax elements in subclauses G.7.3.4.1 and G.7.3.6 the initialisation process of context variables as specified in subclause 9.3.1 applies.

For each context variable, the two variables pStateIdx and valMPS are initialised. The two values assigned to pStateIdx and valMPS for the initialisation are derived from SliceQP_Y, which is derived in Equation 7-29. Given the two table entries (m, n), the initialisation is specified by the following pseudo-code process:

```
preCtxState = Clip3( 1, 126, ( ( m * Clip3( 0, 51, SliceQPY ) ) >> 4 ) + n )
if( preCtxState <= 63 ) {
    pStateIdx = 63 − preCtxState
    valMPS = 0
} else {                                                          (G-363)
    pStateIdx = preCtxState − 64
```

valMPS = 1
}

In Table G-11, the ctxIdx for which initialisation is needed for each of the slice types EI, EP, and EB are listed. Also listed is the table number that includes the values of m and n needed for the initialisation. For EP and EB slices, the initialisation depends also on the value of the cabac_init_idc syntax element. Note that the syntax element names do not affect the initialisation process.

Table G-11 – Association of ctxIdx and syntax elements for each slice type in the initialisation process

	Syntax element	Table	Slice type		
			EI	EP	EB
macroblock_layer_in_scalable_extension( )	base_mode_flag	Table G-12	1024..1026	1024..1026	1024..1026
mb_pred_in_scalable_extension( ) and sub_mb_pred_in_scalable_extension( )	motion_prediction_flag_l0	Table G-13		1027	1027
	motion_prediction_flag_l1	Table G-13		1028	1028
macroblock_layer_in_scalable_extension( )	residual_prediction_flag	Table G-13		1029..1030	1029..1030

Table G-12 – Values of variables m and n for ctxIdx from 1024 to 1026

ctxIdx	EI slices		Value of cabac_init_idc (EP, EB slices)					
			0		1		2	
	m	n	m	n	m	n	m	n
1024	−14	138	0	75	0	75	0	75
1025	−22	140	2	65	2	65	2	65
1026	−11	99	2	59	2	59	2	59

Table G-13 – Values of variables m and n for ctxIdx from 1027 to 1030

ctxIdx	Value of cabac_init_idc					
	0		1		2	
	m	n	m	n	m	n
1027	−6	67	−6	67	−6	67
1028	−6	67	−6	67	−6	67
1029	−23	104	−23	104	−23	104
1030	−35	106	−35	106	−35	106

G.9.3.2 Binarization process

Input to this process is a request for a syntax element.

Output of this process is the binarization of the syntax element, maxBinIdxCtx, ctxIdxOffset, and bypassFlag.

Associated with each binarization or binarization part of a syntax element is a specific value of the context index offset (ctxIdxOffset) variable and a specific value of the maxBinIdxCtx variable as given in Table G-14.

The variable bypassFlag is set equal to 0.

The possible values of the context index ctxIdx are in the range 1024 to 1030, inclusive. The value assigned to ctxIdxOffset specifies the lower value of the range of ctxIdx assigned to the corresponding binarization or binarization part of a syntax element.

Table G-14 – Syntax elements and associated types of binarization, maxBinIdxCtx, and ctxIdxOffset

Syntax element	Type of binarization	maxBinIdxCtx	ctxIdxOffset
base_mode_flag	FL, cMax=1	0	1024
motion_prediction_flag_l0	FL, cMax=1	0	1027
motion_prediction_flag_l1	FL, cMax=1	0	1028
residual_prediction_flag	FL, cMax=1	0	1029

G.9.3.3 Decoding process flow

Input to this process is a binarization of the requested syntax element, maxBinIdxCtx, bypassFlag and ctxIdxOffset as specified in subclause G.9.3.2.

Output of this process is the value of the syntax element.

This process specifies how each bit of a bit string is parsed for each syntax element.

After parsing each bit, the resulting bit string is compared to all bin strings of the binarization of the syntax element and the following applies:

– If the bit string is equal to one of the bin strings, the corresponding value of the syntax element is the output.

– Otherwise (the bit string is not equal to one of the bin strings), the next bit is parsed.

While parsing each bin, the variable binIdx is incremented by 1 starting with binIdx being set equal to 0 for the first bin.

The parsing of each bin is specified by the following two ordered steps:

1. Given binIdx, maxBinIdxCtx and ctxIdxOffset, ctxIdx is derived as specified in subclause G.9.3.3.1.
2. Given ctxIdx, the value of the bin from the bitstream as specified in subclause 9.3.3.2 is decoded.

G.9.3.3.1 Derivation process for ctxIdx

Inputs to this process are binIdx, maxBinIdxCtx and ctxIdxOffset.

Output of this process is ctxIdx.

Table G-15 shows the assignment of ctxIdx increments (ctxIdxInc) to binIdx for all ctxIdxOffset values for the syntax elements base_mode_flag, motion_prediction_flag_l0, motion_prediction_flag_l1, and residual_prediction_flag.

The ctxIdx to be used with a specific binIdx is the sum of ctxIdxOffset and ctxIdxInc, which is found in Table G-15. When more than one value is listed in Table G-15 or 9-39 for a binIdx, the assignment process for ctxIdxInc for that binIdx is further specified in the subclauses given in parenthesis of the corresponding table entry.

All entries in Table G-15 labelled with "na" correspond to values of binIdx that do not occur for the corresponding ctxIdxOffset.

Table G-15 – Assignment of ctxIdxInc to binIdx for the ctxIdxOffset values related to the syntax elements base_mode_flag and residual_prediction_flag

ctxIdxOffset	binIdx						
	0	1	2	3	4	5	>= 6
1024	0,1,2 (subclause G.9.3.3.2.1)	na	na	na	na	na	na
1027	0	na	na	na	na	na	na
1028	0	na	na	na	na	na	na
1029	0,1 (subclause G.9.3.3.2.2)	na	na	na	na	na	na

G.9.3.3.2 Assignment process of ctxIdxInc using neighbouring syntax elements

Subclause G.9.3.3.2.1 specifies the derivation process of ctxIdxInc for the syntax element base_mode_flag.

Subclause G.9.3.3.2.2 specifies the derivation process of ctxIdxInc for the syntax element residual_prediction_flag.

G.9.3.3.2.1 Derivation process of ctxIdxInc for the syntax element base_mode_flag

Output of this process is ctxIdxInc.

The derivation process for neighbouring macroblocks specified in subclause 6.4.11.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

Let the variable condTermFlagN (with N being either A or B) be derived as follows:

– If mbAddrN is available and base_mode_flag for the macroblock mbAddrN is equal to 1, condTermFlagN is set equal to 0.

– Otherwise (mbAddrN is not available or base_mode_flag for the macroblock mbAddrN is equal to 0), condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived by

$$\text{ctxIdxInc} = \text{condTermFlagA} + \text{condTermFlagB} \quad \text{(G-364)}$$

G.9.3.3.2.2 Derivation process of ctxIdxInc for the syntax element residual_prediction_flag

Output of this process is ctxIdxInc.

Depending on base_mode_flag, the following applies:

– If base_mode_flag is equal to 1, ctxIdxInc is set equal to 0.

– Otherwise (base_mode_flag is equal to 0), ctxIdxInc is set equal to 1.

G.10 Profiles and levels

The specifications in Annex A apply. Additional profiles and specific values of profile_idc are specified in the following.

The profiles that are specified in subclause G.10.1 are also referred to as the profiles specified in Annex G.

G.10.1 Profiles

All constraints for picture parameter sets that are specified in subclauses G.10.1.1 to G.10.1.3 are constraints for picture parameter sets that become the active picture parameter set or an active layer picture parameter set inside the bitstream. All constraints for SVC sequence parameter sets that are specified in subclauses G.10.1.1 to G.10.1.3 are constraints for SVC sequence parameter sets that become the active SVC sequence parameter set or an active layer SVC sequence parameter set inside the bitstream. All constraints for sequence parameter sets of the base layer bitstream that are specified in subclauses G.10.1.1 to G.10.1.3 are constraints for sequence parameter sets that are activated in the base layer bitstream.

G.10.1.1 Scalable Baseline profile

Bitstreams conforming to the Scalable Baseline profile shall obey the following constraints:

a) The base layer bitstream as specified in subclause G.8.8.2 shall obey the following constraints:

   i) All constraints of the Baseline and Constrained Baseline profiles specified in subclauses A.2.1 and A.2.1.1 shall be obeyed.

   ii) Sequence parameter sets should have profile_idc equal to 66. Sequence parameter sets may have profile_idc equal to 77 or 88. Sequence parameter sets shall not have profile_idc equal to a value other than 66, 77, or 88.

   iii) Sequence parameter sets shall have constraint_set0_flag, constraint_set1_flag, and constraint_set2_flag equal to 1.

   NOTE 1 – The above constraint implies that picture parameter sets must have num_slice_groups_minus1 equal to 0 and redundant_pic_cnt_present_flag equal to 0 and that arbitrary slice order is not allowed.

   NOTE 2 – In addition to the base layer constraints specified above in items i) through iii), the value of the syntax element constrained_intra_pred_flag for picture parameter sets of the base layer stream is constrained as specified below in item l).

b) A list of integer values specifying layer representation identifiers is derived by invoking the process specified in subclause G.8.1.1 with the output being the list dqIdList. The SVC sequence parameter sets that are referred to by coded slice NAL units with DQId greater than 0 and DQId in the list dqIdList shall have profile_idc equal to 83 or (profile_idc equal to 86 and constraint_set0_flag equal to 1).

c) Only I, P, EI, EP, and EB slices shall be present.

d) SVC sequence parameter sets shall have chroma_format_idc equal to 1.

e) SVC sequence parameter sets shall have bit_depth_luma_minus8 equal to 0.

f) SVC sequence parameter sets shall have bit_depth_chroma_minus8 equal to 0.

g) SVC sequence parameter sets shall have separate_colour_plane_flag equal to 0.

h) SVC sequence parameter sets shall have qpprime_y_zero_transform_bypass_flag equal to 0.

i) SVC sequence parameter sets shall have frame_mbs_only_flag equal to 1.

j) Picture parameter sets shall have num_slice_groups_minus1 in the range of 0 to 7, inclusive.

k) The value of slice_group_map_type, when present in picture parameter sets, shall be equal to 2.

l) A list of integer values specifying layer representation identifiers is derived by invoking the process specified in subclause G.8.1.1 with the output being the list dqIdList. The variable numDQEntries is set equal to the number of elements in the list dqIdList. When numDQEntries is greater than 1, for any element dqIdList[i] with i = 1..(numDQEntries − 1), when MaxTCoeffLevelPredFlag is equal to 0 for any layer representation with DQId in the set specified by dqIdList[k] with k = 0..i, the picture parameter set that is referenced by the coded slice NAL units of the layer representation with DQId equal to dqIdList[i] shall have constrained_intra_pred_flag equal to 1.

m) For each present layer representation with dependency_id greater than 0, quality_id equal to 0, and MinNoInterLayerPredFlag equal to 0, one of the following constraints shall be obeyed.

  – $\text{ScaledRefLayerPicWidthInSamples}_L$ is equal to $\text{RefLayerPicWidthInSamples}_L$ and $\text{ScaledRefLayerPicHeightInSamples}_L$ is equal to $\text{RefLayerPicHeightInSamples}_L$

  – $\text{ScaledRefLayerPicWidthInSamples}_L$ is equal to ($1.5 * \text{RefLayerPicWidthInSamples}_L$) and $\text{ScaledRefLayerPicHeightInSamples}_L$ is equal to ($1.5 * \text{RefLayerPicHeightInSamples}_L$)

  – $\text{ScaledRefLayerPicWidthInSamples}_L$ is equal to ($2 * \text{RefLayerPicWidthInSamples}_L$) and $\text{ScaledRefLayerPicHeightInSamples}_L$ is equal to ($2 * \text{RefLayerPicHeightInSamples}_L$)

n) For each present layer representation with dependency_id greater than 0, quality_id equal to 0, and MinNoInterLayerPredFlag equal to 0, all of the following constraints shall be obeyed.

  – (ScaledRefLayerLeftOffset % 16) is equal to 0

  – (ScaledRefLayerTopOffset % 16) is equal to 0

o) The level constraints specified in subclause G.10.2 shall be fulfilled.

Conformance of a bitstream to Scalable Baseline profile is indicated by profile_idc equal to 83.

Decoders conforming to the Scalable Baseline profile at a specific level shall be capable of decoding all bitstreams in which both of the following conditions are true:

a) All active SVC sequence parameter sets have one or more of the following conditions fulfilled:

  – profile_idc is equal to 83,

  – profile_idc is equal to 86 and constraint_set0_flag is equal to 1,

  – profile_idc is equal to 66 and constraint_set1_flag is equal to 1,

  – profile_idc is equal to 77 and constraint_set0_flag is equal to 1,

  – profile_idc is equal to 88, constraint_set0_flag is equal to 1, and constraint_set1_flag is equal to 1.

b) level_idc or (level_idc and constraint_set3_flag) for all active SVC sequence parameter sets represent a level less than or equal to the specified level.

G.10.1.2 Scalable High profile

Bitstreams conforming to the Scalable High profile shall obey the following constraints:

a) The base layer bitstream as specified in subclause G.8.8.2 shall obey the following constraints:

   i) All constraints of the High profile specified in subclause A.2.4 shall be obeyed.

   ii) Sequence parameter sets should have profile_idc equal to 100. Sequence parameter sets may have profile_idc equal to 66, 77, or 88 and constraint_set1_flag equal to 1. Sequence parameter sets shall not have profile_idc equal to a value other than 66, 77, 88, or 100.

   iii) The syntax element direct_spatial_mv_pred_flag shall be equal to 1.

   NOTE – In addition to the base layer constraints specified above in items i) through iii), the value of the syntax element constrained_intra_pred_flag for picture parameter sets of the base layer stream is constrained as specified below in item k).

b) A list of integer values specifying layer representation identifiers is derived by invoking the process specified in subclause G.8.1.1 with the output being the list dqIdList. The SVC sequence parameter sets that are referred to by coded slice NAL units with DQId greater than 0 and DQId in the list dqIdList shall have profile_idc equal to 86 or (profile_idc equal to 83 and constraint_set1_flag equal to 1).

c) Only I, P, B, EI, EP, and EB slices shall be present.

d) SVC sequence parameter sets shall have chroma_format_idc equal to 1.

e) SVC sequence parameter sets shall have bit_depth_luma_minus8 equal to 0.

f) SVC sequence parameter sets shall have bit_depth_chroma_minus8 equal to 0.

g) SVC sequence parameter sets shall have separate_colour_plane_flag equal to 0.

h) SVC sequence parameter sets shall have qpprime_y_zero_transform_bypass_flag equal to 0.

i) Picture parameter sets shall have redundant_pic_cnt_present_flag equal to 0.

j) Picture parameter sets shall have num_slice_groups_minus1 equal to 0.

k) A list of integer values specifying layer representation identifiers is derived by invoking the process specified in subclause G.8.1.1 with the output being the list dqIdList. The variable numDQEntries is set equal to the number of elements in the list dqIdList. When numDQEntries is greater than 1, for any element dqIdList[i] with i = 1..(numDQEntries − 1), when MaxTCoeffLevelPredFlag is equal to 0 for any layer representation with DQId in the set specified by dqIdList[k] with k = 0..i, the picture parameter set that is referenced by the coded slice NAL units of the layer representation with DQId equal to dqIdList[i] shall have constrained_intra_pred_flag equal to 1.

l) Arbitrary slice order is not allowed.

m) The level constraints specified in subclause G.10.2 shall be fulfilled.

Conformance of a bitstream to Scalable High profile is indicated by profile_idc equal to 86.

Decoders conforming to the Scalable High profile at a specific level shall be capable of decoding all bitstreams in which both of the following conditions are true:

a) All active SVC sequence parameter sets have one or more of the following conditions fulfilled:

   – profile_idc is equal to 86,

   – profile_idc is equal to 83 and constraint_set1_flag is equal to 1,

   – profile_idc is equal to 77 or 100,

   – profile_idc is equal to 66 or 88 and constraint_set1_flag is equal to 1.

b) level_idc or (level_idc and constraint_set3_flag) for all active SVC sequence parameter sets represent a level less than or equal to the specified level.

G.10.1.3 Scalable High Intra profile

Bitstreams conforming to the Scalable High Intra profile shall obey the following constraints:

a) The base layer bitstream as specified in subclause G.8.8.2 shall obey the following constraints:

   i) All constraints of the High profile specified in subclause A.2.4 shall be obeyed.

ii) Sequence parameter sets should have profile_idc equal to 100 and constraint_set3_flag equal to 1. Sequence parameter sets may have profile_idc equal to 66, 77, or 88 and constraint_set1_flag equal to 1. Sequence parameter sets shall not have profile_idc equal to a value other than 66, 77, 88, or 100.

b) A list of integer values specifying layer representation identifiers is derived by invoking the process specified in subclause G.8.1.1 with the output being the list dqIdList. The SVC sequence parameter sets that are referred to by coded slice NAL units with DQId greater than 0 and DQId in the list dqIdList shall have profile_idc equal to 86 and constraint_set3_flag equal to 1.

c) All constraints of the Scalable High profile specified in subclause G.10.1.2 shall be obeyed.

d) All pictures shall be IDR pictures.

e) SVC sequence parameter sets shall have max_num_ref_frames equal to 0.

f) When vui_parameters_present_flag is equal to 1 and bitstream_restriction_flag is equal to 1, SVC sequence parameter sets shall have max_num_reorder_frames equal to 0.

g) When vui_parameters_present_flag is equal to 1 and bitstream_restriction_flag is equal to 1, SVC sequence parameter sets shall have max_dec_frame_buffering equal to 0.

h) Picture timing SEI messages, whether present in the bitstream (by non-VCL NAL units) or conveyed equivalently by other means not specified in this Recommendation | International Standard, shall have dpb_output_delay equal to 0.

i) The level constraints specified in subclause G.10.2 shall be fulfilled.

Conformance of a bitstream to Scalable High Intra profile is indicated by constraint_set3_flag being equal to 1 with profile_idc equal to 86.

Decoders conforming to the Scalable High Intra profile at a specific level shall be capable of decoding all bitstreams in which both of the following conditions are true:

a) All active SVC sequence parameter sets have profile_idc equal to 86 or 100 and constraint_set3_flag equal to 1.

b) level_idc or (level_idc and constraint_set3_flag) for all active SVC sequence parameter sets represents a level less than or equal to the specified level.

The operation of the deblocking filter process for target representation as specified in subclause G.8.7.2 is not required for decoder conformance to the Scalable High Intra profile.

G.10.2 Levels

The following is specified for expressing the constraints in this subclause:

– Let access unit n be the n-th access unit in decoding order with the first access unit being access unit 0.

– Let picture n be the primary coded picture or the corresponding decoded picture of access unit n.

The variable fR is derived as follows:

– If picture n is a frame, fR is set equal to (1 ÷ 172).

– Otherwise (picture n is a field), fR is set equal to (1 ÷ (172 * 2)).

G.10.2.1 Level limits common to Scalable Baseline, Scalable High, and Scalable High Intra profiles

The variable dqIdMax is set equal to the maximum value of DQId for the layer representation of the access unit.

The variable refLayerDQId is set equal to the value of MaxRefLayerDQId for the layer representation with DQId equal to dqIdMax.

A list of integer values specifying layer representation identifiers for the access unit is derived by invoking the process specified in subclause G.8.1.1 with the output being the list dqIdList. The variable numDQEntries is set equal to the number of elements in the list dqIdList.

A variable dependentDId is derived by the following pseudo-code:

```
dependentDId = 0
for( i = 0; i < numDQEntries; i++ )
    if( ( dqIdList[ i ] % 16 )  = =  0 )                                        (G-365)
        dependentDId++
```

The variable svcPicSizeInMbs is derived as follows:

– If numDQEntries is less than 3, svcPicSizeInMbs is set equal to PicSizeInMbs for the layer representation with DQId equal to dqIdMax.

– Otherwise (numDQEntries is greater than 2), svcPicSizeInMbs is derived by applying the following ordered steps:

  1. svcPicSizeInMbs is set equal to PicSizeInMbs for the layer representation with DQId equal to dqIdMax.

  2. The variable refLayerMbs is set equal to 0.

  3. For each element dqIdList[i] with i = 2..(numDQEntries − 1), with refLayerPicSizeInMbs being the variable PicSizeInMbs for the layer representation with DQId equal to dqIdList[i], the variable refLayerMbs is modified by

     refLayerMbs += refLayerPicSizeInMbs (G-366)

  4. svcPicSizeInMbs is modified by

     svcPicSizeInMbs += (refLayerMbs + 1) >> 1 (G-367)

Bitstreams conforming to the Scalable Baseline, Scalable High, or Scalable High Intra profiles at a specific level shall obey the following constraints:

a) The nominal removal time of access unit n with n > 0 from the CPB as specified in subclause C.1.2, satisfies the constraint that $t_{r,n}(n) - t_r(n - 1)$ is greater than or equal to Max(svcPicSizeInMbs ÷ MaxMBPS, fR), where MaxMBPS is the value specified in Table A-1 that applies to picture n − 1 and svcPicSizeInMbs is derived for picture n − 1.

b) The difference between consecutive output times of pictures from the DPB as specified in subclause C.2.2, satisfies the constraint that $\Delta t_{o,dpb}(n)$ >= Max(svcPicSizeInMbs ÷ MaxMBPS, fR), where MaxMBPS is the value specified in Table A-1 for picture n, and svcPicSizeInMbs is derived for picture n, provided that picture n is a picture that is output and is not the last picture of the bitstream that is output.

c) PicWidthInMbs * FrameHeightInMbs <= MaxFS, where MaxFS is specified in Table A-1. PicWidthInMbs and FrameHeightInMbs are the derived variables for the layer representation with DQId equal to dqIdMax.

d) PicWidthInMbs <= Sqrt(MaxFS * 8), where MaxFS is specified in Table A-1 and PicWidthInMbs is the derived variable for the layer representation with DQId equal to dqIdMax.

e) FrameHeightInMbs <= Sqrt(MaxFS * 8), where MaxFS is specified in Table A-1 and FrameHeightInMbs is the derived variable for the layer representation with DQId equal to dqIdMax.

f) max_dec_frame_buffering <= MaxDpbFrames, where MaxDpbFrames is equal to Min(MaxDpbMbs / (PicWidthInMbs * FrameHeightInMbs), 16) and MaxDpbMbs is specified in Table A-1. PicWidthInMbs and FrameHeightInMbs are the derived variables for the layer representation with DQId equal to dqIdMax.

g) The vertical motion vector component range does not exceed MaxVmvR in units of luma frame samples, where MaxVmvR is specified in Table A-1.

h) The horizontal motion vector range does not exceed the range of −2048 to 2047.75, inclusive, in units of luma samples.

i) For each layer representation, the total number of motion vectors per two macroblocks mbAddrA and mbAddrB with (mbAddrA + 1) equal to mbAddrB does not exceed MaxMvsPer2Mb, where MaxMvsPer2Mb is specified in Table A-1 given the level that is indicated in the SVC sequence parameter set that is referenced by the layer representation. The number of motion vectors for each macroblock is the value of the variable MvCnt after the completion of the base decoding process for slices without resolution change specified in subclause G.8.1.4.1 (when SpatialResolutionChangeFlag is equal to 0) or after completion the base decoding process for slices with resolution change specified in subclause G.8.1.4.2 (when SpatialResolutionChangeFlag is equal to 1).

   NOTE – Due to the constraint specified in subclause G.8.8.1, the number of motion vectors for the layer representation with DQId equal to 0 is additionally constrained as specified in Annex A.

j) The number of bits of macroblock_layer( ) and macroblock_layer_in_scalable_extension( ) data for any macroblock in any layer representation is not greater than $128 + \text{RawMbBits}$. Depending on entropy_coding_mode_flag, the bits of macroblock_layer( ) data are counted as follows:

   – If entropy_coding_mode_flag is equal to 0, the number of bits of macroblock_layer( ) data is given by the number of bits in the macroblock_layer( ) syntax structure for a macroblock.

   – Otherwise (entropy_coding_mode_flag is equal to 1), the number of bits of macroblock_layer( ) data for a macroblock is given by the number of times read_bits(1) is called in subclauses 9.3.3.2.2 and 9.3.3.2.3 when parsing the macroblock_layer( ) associated with the macroblock.

k) The variable dependentDId specified at the beginning of this subclause shall not exceed 3.

l) For each layer representation present in an access unit that has MinNoInterLayerPredFlag equal to 0, the following applies:

   1. The variables numILIntraPredSamples and numRefLayerILIntraPredMbs are derived as specified in the derivation process for variables related to inter-layer intra prediction in subclause G.8.6.2.5 with DQId being the input.

   2. The following constraint shall be obeyed.

$$\text{numRefLayerILIntraPredMbs} * 256 \le 1.5 * \text{numILIntraPredSamples} \quad \text{(G-368)}$$

m) When MaxRefLayerDQId is greater than or equal to 0 for a particular layer representation, the value of level_idc in the SVC sequence parameter set that is referenced by the particular layer representation shall represent a level that is greater than or equal to the level that is represented by the value of level_idc or (level_idc and constraint_set3_flag) in the SVC sequence parameter set that is referenced by the layer representation with DQId equal to MaxRefLayerDQId.

Table A-1 specifies the limits for each level. A definition of all levels identified in the "Level number" column of Table A-1 is specified for the Scalable Baseline, Scalable High, and Scalable High Intra profiles. Each entry in Table A-1 indicates, for the level corresponding to the row of the table, the absence or value of a limit that is imposed by the variable corresponding to the column of the table, as follows:

– If the table entry is marked as "-", no limit is imposed by the value of the variable as a requirement of bitstream conformance to the profile at the specified level.

– Otherwise, the table entry specifies the value of the variable for the associated limit that is imposed as a requirement of bitstream conformance to the profile at the specified level.

In bitstreams conforming to the Scalable Baseline, Scalable High, or Scalable High Intra profiles, the conformance of the bitstream to a specified level is indicated by the syntax element level_idc as follows:

– If level_idc is equal to 9, the indicated level is level 1b.

– Otherwise (level_idc is not equal to 9), level_idc is equal to a value of ten times the level number (of the indicated level) specified in Table A-1.

G.10.2.2 Profile specific level limits

The variable dqIdMax is set equal to the maximum value of DQId for the layer representation of the access unit.

A list of integer values specifying layer representation identifiers for the access unit is derived by invoking the process specified in subclause G.8.1.1 with the output being the list dqIdList. The variable numDQEntries is set equal to the number of elements in the list dqIdList.

The variable numSVCSlices is derived as specified by the following pseudo-code:

```
numSVCSlices = 0
for( i = 0; i < numDQEntries; i++ )                                                    (G-369)
    numSVCSlices += number of slices in layer representation with DQId equal to dqIdList[ i ]
```

The variable svcPicSizeInMbs is derived as specified in subclause G.10.2.1.

The following constraints are specified:

a) In bitstreams conforming to the Scalable Baseline, Scalable High, or Scalable High Intra profiles, the removal time of access unit 0 shall satisfy the constraint that the numSVCSlices variable for picture 0 is less than or equal to $(\text{Max}(\text{svcPicSizeInMbs}, \text{fR} * \text{MaxMBPS}) + \text{MaxMBPS} * (t_r(0) - t_{r,n}(0))) \div \text{SliceRate}$, where MaxMBPS and SliceRate are the values that apply to picture 0. MaxMBPS is specified in Table A-1. For

Scalable Baseline profile, SliceRate is specified in Table G-16. For Scalable High and Scalable High Intra profiles, SliceRate is specified in Table A-4.

b) In bitstreams conforming to the Scalable Baseline, Scalable High, or Scalable High Intra profiles, the difference between consecutive removal times of access units n and $n - 1$ with $n > 0$ shall satisfy the constraint that the numSVCSlices variable for picture n is less than or equal to MaxMBPS * ($t_r(n) - t_r(n - 1)$) ÷ SliceRate, where MaxMBPS and SliceRate are the values that apply to picture n. MaxMBPS is specified in Table A-1. For the Scalable Baseline profile, SliceRate is specified in Table G-16. For the Scalable High and Scalable High Intra profiles, SliceRate is specified in Table A-4.

c) In bitstreams conforming to the Scalable High profile, SVC sequence parameter sets shall have direct_8x8_inference_flag equal to 1 for the levels specified in Table A-4. In bitstreams conforming to Scalable Baseline profile, SVC sequence parameter sets shall have direct_8x8_inference_flag equal to 1.

d) In bitstreams conforming to the Scalable High or Scalable High Intra profiles, SVC sequence parameter sets shall have frame_mbs_only_flag equal to 1 for the levels specified in Table A-4.

e) In bitstreams conforming to the Scalable High profile, for all macroblocks mbAddr and macroblock partitions mbPartIdx, the value of subMbType[mbAddr][mbPartIdx] that is derived as specified in subclause G.8.1.5.1.1 shall not be equal to B_Bi_8x4, B_Bi_4x8, or B_Bi_4x4 for the levels in which MinLumaBiPredSize is shown as 8x8 in Table A-4. In bitstreams conforming to the Scalable Baseline profile, for all macroblocks mbAddr and macroblock partitions mbPartIdx, the value of subMbType[mbAddr][mbPartIdx] that is derived as specified in subclause G.8.1.5.1.1 shall not be equal to B_Bi_8x4, B_Bi_4x8, or B_Bi_4x4.

f) In bitstreams conforming to the Scalable Baseline profile, ($xInt_{max} - xInt_{min} + 6$) * ($yInt_{max} - yInt_{min} + 6$) <= MaxSubMbRectSize in macroblocks coded with macroblock type equal to P_8x8, P_8x8ref0 or B_8x8 for all invocations of the process specified in subclause 8.4.2.2.1 used to generate the predicted luma sample array for a single reference picture list (reference picture list 0 or reference picture list 1) for each 8x8 sub-macroblock with the macroblock partition index mbPartIdx, where NumSubMbPart(sub_mb_type[mbPartIdx]) > 1, where MaxSubMbRectSize is specified in Table G-16 and

– $xInt_{min}$ is the minimum value of $xInt_L$ among all luma sample predictions for the sub-macroblock,

– $xInt_{max}$ is the maximum value of $xInt_L$ among all luma sample predictions for the sub-macroblock,

– $yInt_{min}$ is the minimum value of $yInt_L$ among all luma sample predictions for the sub-macroblock,

– $yInt_{max}$ is the maximum value of $yInt_L$ among all luma sample predictions for the sub-macroblock.

g) In bitstreams conforming to the Scalable Baseline, Scalable High, or Scalable High Intra profiles, for the VCL HRD parameters, BitRate[SchedSelIdx] <= cpbBrVclFactor * MaxBR and CpbSize[SchedSelIdx] <= cpbBrVclFactor * MaxCPB for at least one value of SchedSelIdx, where cpbBrVclFactor is specified in Table G-17. With vui_ext_vcl_hrd_parameters_present_flag[i] being the syntax element, in the SVC VUI parameters extension of the active SVC sequence parameter set, that is associated with the VCL HRD parameters that are used for conformance checking (as specified in Annex C), BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are given as follows:

– If vui_ext_vcl_hrd_parameters_present_flag[i] is equal to 1, BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are given by Equations E-37 and E-38, respectively, using the syntax elements of the hrd_parameters( ) syntax structure that immediately follows vui_ext_vcl_hrd_parameters_present_flag[i].

– Otherwise (vui_ext_vcl_hrd_parameters_present_flag[i] is equal to 0), BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are inferred as specified in subclause E.2.2 for VCL HRD parameters.

MaxBR and MaxCPB are specified in Table A-1 in units of cpbBrVclFactor bits/s and cpbBrVclFactor bits, respectively. The bitstream shall satisfy these conditions for at least one value of SchedSelIdx in the range 0 to cpb_cnt_minus1, inclusive.

h) In bitstreams conforming to the Scalable Baseline, Scalable High, or Scalable High Intra profiles, for the NAL HRD parameters, BitRate[SchedSelIdx] <= cpbBrNalFactor * MaxBR and CpbSize[SchedSelIdx] <= cpbBrNalFactor * MaxCPB for at least one value of SchedSelIdx, where cpbBrNalFactor is specified in Table G-17. With vui_ext_nal_hrd_parameters_present_flag[i] being the syntax element, in the SVC VUI parameters extension of the active SVC sequence parameter set, that is associated with the NAL HRD parameters that are used for conformance checking (as specified in Annex C), BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are given as follows:

– If vui_ext_nal_hrd_parameters_present_flag[i] is equal to 1, BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are given by Equations E-37 and E-38, respectively, using the syntax elements of the hrd_parameters( ) syntax structure that immediately follows vui_ext_nal_hrd_parameters_present_flag[i].

– Otherwise (vui_ext_nal_hrd_parameters_present_flag[i] is equal to 0), BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are inferred as specified in subclause E.2.2 for NAL HRD parameters.

MaxBR and MaxCPB are specified in Table A-1 in units of cpbBrNalFactor bits/s and cpbBrNalFactor bits, respectively. The bitstream shall satisfy these conditions for at least one value of SchedSelIdx in the range 0 to cpb_cnt_minus1, inclusive.

i) In bitstreams conforming to the Scalable Baseline, Scalable High, or Scalable High Intra profiles, the sum of the NumBytesInNALunit variables for access unit 0 is less than or equal to 384 * (Max(svcPicSizeInMbs, fR * MaxMBPS) + MaxMBPS * ($t_r(0) - t_{r,n}(0)$)) ÷ MinCR, where MaxMBPS and MinCR are the values specified in Table A-1 that apply to picture 0.

j) In bitstreams conforming to the Scalable Baseline, Scalable High, or Scalable High Intra profiles, the sum of the NumBytesInNALunit variables for access unit n with n > 0 is less than or equal to 384 * MaxMBPS * ($t_r(n) - t_r(n - 1)$) ÷ MinCR, where MaxMBPS and MinCR are the values specified in Table A-1 that apply to picture n.

k) In bitstreams conforming to Scalable Baseline profile, picture parameter sets shall have entropy_coding_mode_flag equal to 0 and transform_8x8_mode_flag equal to 0 for level 2.1 and below.

l) In bitstreams conforming to Scalable Baseline, Scalable High, and Scalable High Intra profiles, when PicSizeInMbs is greater than 1620 for DQId equal to dqIdMax, the number of macroblocks in any coded slice shall not exceed MaxFS / 4, where MaxFS is specified in Table A-1.

Table A-4 specifies limits for each level that are specific to bitstreams conforming to the Scalable High and Scalable High Intra profiles. Table G-16 specifies limits for each level that are specific to bitstreams conforming to the Scalable Baseline profile. Each entry in Tables A-4 and G-16 indicates, for the level corresponding to the row of the table, the absence or value of a limit that is imposed by the variable corresponding to the column of the table, as follows:

– If the table entry is marked as "-", no limit is imposed by the value of the variable as a requirement of bitstream conformance to the profile at the specified level.

– Otherwise, the table entry specifies the value of the variable for the associated limit that is imposed as a requirement of bitstream conformance to the profile at the specified level.

Table G-16 – Scalable Baseline profile level limits

Level number	SliceRate	MaxSubMbRectSize
1	-	576
1b	-	576
1.1	-	576
1.2	-	576
1.3	-	576
2	-	576
2.1	22	576
2.2	22	576
3	22	576
3.1	60	1152
3.2	60	1152
4	60	1440
4.1	24	1440
4.2	24	1440
5	24	-
5.1	24	-

Table G-17 – Specification of cpbBrVclFactor and cpbBrNalFactor

Profile	cpbBrVclFactor	cpbBrNalFactor
Scalable Baseline Scalable High, or Scalable High Intra	1250	1500

G.11 Byte stream format

The specifications in Annex B apply.

G.12 Hypothetical reference decoder

The specifications in Annex C apply with substituting SVC sequence parameter set for sequence parameter set.

G.13 Supplemental enhancement information

The specifications in Annex D together with the extensions and modifications specified in this subclause apply.

G.13.1 SEI payload syntax

G.13.1.1 Scalability information SEI message syntax

scalability_info(payloadSize) {	**C**	**Descriptor**
temporal_id_nesting_flag	5	u(1)
priority_layer_info_present_flag	5	u(1)
priority_id_setting_flag	5	u(1)
num_layers_minus1	5	ue(v)
for(i = 0; i <= num_layers_minus1; i++) {		
layer_id[i]	5	ue(v)
priority_id[i]	5	u(6)
discardable_flag[i]	5	u(1)
dependency_id[i]	5	u(3)
quality_id[i]	5	u(4)
temporal_id[i]	5	u(3)
sub_pic_layer_flag[i]	5	u(1)
sub_region_layer_flag[i]	5	u(1)
iroi_division_info_present_flag[i]	5	u(1)
profile_level_info_present_flag[i]	5	u(1)
bitrate_info_present_flag[i]	5	u(1)
frm_rate_info_present_flag[i]	5	u(1)
frm_size_info_present_flag[i]	5	u(1)
layer_dependency_info_present_flag[i]	5	u(1)
parameter_sets_info_present_flag[i]	5	u(1)
bitstream_restriction_info_present_flag[i]	5	u(1)
exact_inter_layer_pred_flag[i]	5	u(1)
if(sub_pic_layer_flag[i] || iroi_division_info_present_flag[i])		
exact_sample_value_match_flag[i]	5	u(1)
layer_conversion_flag[i]	5	u(1)
layer_output_flag[i]	5	u(1)
if(profile_level_info_present_flag[i])		
layer_profile_level_idc[i]	5	u(24)

if(bitrate_info_present_flag[i]) {		
avg_bitrate[i]	5	u(16)
max_bitrate_layer[i]	5	u(16)
max_bitrate_layer_representation[i]	5	u(16)
max_bitrate_calc_window[i]	5	u(16)
}		
if(frm_rate_info_present_flag[i]) {		
constant_frm_rate_idc[i]	5	u(2)
avg_frm_rate[i]	5	u(16)
}		
if(frm_size_info_present_flag[i] || iroi_division_info_present_flag[i]) {		
frm_width_in_mbs_minus1[i]	5	ue(v)
frm_height_in_mbs_minus1[i]	5	ue(v)
}		
if(sub_region_layer_flag[i]) {		
base_region_layer_id[i]	5	ue(v)
dynamic_rect_flag[i]	5	u(1)
if(!dynamic_rect_flag[i]) {		
horizontal_offset[i]	5	u(16)
vertical_offset[i]	5	u(16)
region_width[i]	5	u(16)
region_height[i]	5	u(16)
}		
}		
if(sub_pic_layer_flag[i])		
roi_id[i]	5	ue(v)
if (iroi_division_info_present_flag[i]) {		
iroi_grid_flag[i]	5	u(1)
if (iroi_grid_flag[i]) {		
grid_width_in_mbs_minus1[i]	5	ue(v)
grid_height_in_mbs_minus1[i]	5	ue(v)
} else {		
num_rois_minus1[i]	5	ue(v)
for (j = 0; j <= num_rois_minus1[i]; j++) {		
first_mb_in_roi[i][j]	5	ue(v)
roi_width_in_mbs_minus1[i][j]	5	ue(v)
roi_height_in_mbs_minus1[i][j]	5	ue(v)
}		
}		
}		
if(layer_dependency_info_present_flag[i]) {		
num_directly_dependent_layers[i]	5	ue(v)
for(j = 0; j < num_directly_dependent_layers[i]; j++)		
directly_dependent_layer_id_delta_minus1[i][j]	5	ue(v)
} else		
layer_dependency_info_src_layer_id_delta[i]	5	ue(v)
if(parameter_sets_info_present_flag[i]) {		

num_seq_parameter_sets[i]	5	ue(v)
for(j = 0; j < num_seq_parameter_sets[i]; j++)		
seq_parameter_set_id_delta[i][j]	5	ue(v)
num_subset_seq_parameter_sets[i]	5	ue(v)
for(j = 0; j < num_subset_seq_parameter_sets[i]; j++)		
subset_seq_parameter_set_id_delta[i][j]	5	ue(v)
num_pic_parameter_sets_minus1[i]	5	ue(v)
for(j = 0; j <= num_pic_parameter_sets_minus1[i]; j++)		
pic_parameter_set_id_delta[i][j]	5	ue(v)
} else		
parameter_sets_info_src_layer_id_delta[i]	5	ue(v)
if(bitstream_restriction_info_present_flag[i]) {		
motion_vectors_over_pic_boundaries_flag[i]	5	u(1)
max_bytes_per_pic_denom[i]	5	ue(v)
max_bits_per_mb_denom[i]	5	ue(v)
log2_max_mv_length_horizontal[i]	5	ue(v)
log2_max_mv_length_vertical[i]	5	ue(v)
max_num_reorder_frames[i]	5	ue(v)
max_dec_frame_buffering[i]	5	ue(v)
}		
if(layer_conversion_flag[i]) {		
conversion_type_idc[i]	5	ue(v)
for(j=0; j < 2; j++) {		
rewriting_info_flag[i][j]	5	u(1)
if(rewriting_info_flag[i][j]) {		
rewriting_profile_level_idc[i][j]	5	u(24)
rewriting_avg_bitrate[i][j]	5	u(16)
rewriting_max_bitrate[i][j]	5	u(16)
}		
}		
}		
}		
if(priority_layer_info_present_flag) {		
pr_num_dIds_minus1	5	ue(v)
for(i = 0; i <= pr_num_dIds_minus1; i++) {		
pr_dependency_id[i]	5	u(3)
pr_num_minus1[i]	5	ue(v)
for(j = 0; j <= pr_num_minus1[i]; j++) {		
pr_id[i][j]	5	ue(v)
pr_profile_level_idc[i][j]	5	u(24)
pr_avg_bitrate[i][j]	5	u(16)
pr_max_bitrate[i][j]	5	u(16)
}		
}		
}		
if(priority_id_setting_flag) {		
PriorityIdSettingUriIdx = 0		
do		

priority_id_setting_uri[PriorityIdSettingUriIdx]	5	b(8)
while(priority_id_setting_uri[PriorityIdSettingUriIdx++] != 0)		
}		
}		

G.13.1.2 Sub-picture scalable layer SEI message syntax

sub_pic_scalable_layer(payloadSize) {	C	Descriptor
layer_id	5	ue(v)
}		

G.13.1.3 Non-required layer representation SEI message syntax

non_required_layer_rep(payloadSize) {	C	Descriptor
num_info_entries_minus1	5	ue(v)
for(i = 0; i <= num_info_entries_minus1; i++) {		
entry_dependency_id[i **]**	5	u(3)
num_non_required_layer_reps_minus1[i **]**	5	ue(v)
for(j = 0; j <= num_non_required_layer_reps_minus1**[** i **]**; j++) {		
non_required_layer_rep_dependency_id[i **][** j **]**	5	u(3)
non_required_layer_rep_quality_id[i **][** j **]**	5	u(4)
}		
}		
}		

G.13.1.4 Priority layer information SEI message syntax

priority_layer_info(payloadSize) {	C	Descriptor
pr_dependency_id	5	u(3)
num_priority_ids	5	u(4)
for(i = 0; i < num_priority_ids; i++) {		
alt_priority_id[i **]**	5	u(6)
}		
}		

G.13.1.5 Layers not present SEI message syntax

layers_not_present(payloadSize) {	C	Descriptor
num_layers	5	ue(v)
for(i = 0; i < num_layers; i++) {		
layer_id[i]	5	ue(v)
}		
}		

G.13.1.6 Layer dependency change SEI message syntax

layer_dependency_change(payloadSize) {	C	Descriptor
num_layers_minus1	5	ue(v)
for(i = 0; i <= num_layers_minus1; i++) {		
layer_id[i **]**	5	ue(v)
layer_dependency_info_present_flag[i **]**	5	u(1)
if(layer_dependency_info_present_flag[i]) {		
num_directly_dependent_layers[i **]**	5	ue(v)
for (j = 0; j < num_directly_dependent_layers[i]; j++)		
directly_dependent_layer_id_delta_minus1[i **][** j **]**	5	ue(v)
} else {		
layer_dependency_info_src_layer_id_delta_minus1[i **]**	5	ue(v)
}		
}		
}		

G.13.1.7 Scalable nesting SEI message syntax

scalable_nesting(payloadSize) {	C	Descriptor
all_layer_representations_in_au_flag	5	u(1)
if(all_layer_representations_in_au_flag = = 0) {		
num_layer_representations_minus1	5	ue(v)
for(i = 0; i <= num_layer_representations_minus1; i++) {		
sei_dependency_id[i **]**	5	u(3)
sei_quality_id[i **]**	5	u(4)
}		
sei_temporal_id	5	u(3)
}		
while(!byte_aligned( ))		
sei_nesting_zero_bit /* equal to 0 */	5	f(1)
do		
sei_message( )	5	
while(more_rbsp_data( ))		
}		

G.13.1.8 Base layer temporal HRD SEI message syntax

base_layer_temporal_hrd(payloadSize) {	C	Descriptor
num_of_temporal_layers_in_base_layer_minus1	5	ue(v)
for(i = 0; i <= num_of_temporal_layers_in_base_layer_minus1; i++){		
sei_temporal_id[i **]**	5	u(3)
sei_timing_info_present_flag[i **]**	5	u(1)
if(sei_timing_info_present_flag[i]) {		
sei_num_units_in_tick[i **]**	5	u(32)
sei_time_scale[i **]**	5	u(32)

sei_fixed_frame_rate_flag[i]	5	u(1)
}		
sei_nal_hrd_parameters_present_flag[i]	5	u(1)
if(sei_nal_hrd_parameters_present_flag[i])		
hrd_parameters( )	5	
sei_vcl_hrd_parameters_present_flag[i]	5	u(1)
if(sei_vcl_hrd_parameters_present_flag[i])		
hrd_parameters( )	5	
if(sei_nal_hrd_parameters_present_flag[i] || sei_vcl_hrd_parameters_present_flag[i])		
sei_low_delay_hrd_flag[i]	5	u(1)
sei_pic_struct_present_flag[i]	5	u(1)
}		
}		

G.13.1.9 Quality layer integrity check SEI message syntax

quality_layer_integrity_check(payloadSize) {	C	Descriptor
num_info_entries_minus1	5	ue(v)
for(i = 0; i <= num_info_entries_minus1; i++) {		
entry_dependency_id[i]	5	u(3)
quality_layer_crc[i]	5	u(16)
}		
}		

G.13.1.10 Redundant picture property SEI message syntax

redundant_pic_property(payloadSize) {	C	Descriptor
num_dIds_minus1	5	ue(v)
for(i = 0; i <= num_dIds_minus1; i++) {		
dependency_id[i]	5	u(3)
num_qIds_minus1[i]	5	ue(v)
for(j = 0; j <= num_qIds_minus1[i]; j++) {		
quality_id[i][j]	5	u(4)
num_redundant_pics_minus1[i][j]	5	ue(v)
for(k = 0; k <= num_redundant_pics_minus1[i][j]; k++) {		
redundant_pic_cnt_minus1[i][j][k]	5	ue(v)
pic_match_flag[i][j][k]	5	u(1)
if(!pic_match_flag[i][j][k]) {		
mb_type_match_flag[i][j][k]	5	u(1)
motion_match_flag[i][j][k]	5	u(1)
residual_match_flag[i][j][k]	5	u(1)
intra_samples_match_flag[i][j][k]	5	u(1)
}		
}		
}		
}		
}		

G.13.1.11 Temporal level zero dependency representation index SEI message syntax

tl0_dep_rep_index(payloadSize) {	C	Descriptor
tl0_dep_rep_idx	5	u(8)
effective_idr_pic_id	5	u(16)
}		

G.13.1.12 Temporal level switching point SEI message syntax

tl_switching_point(payloadSize) {	C	Descriptor
delta_frame_num	5	se(v)
}		

G.13.2 SEI payload semantics

The semantics of the SEI messages with payloadType in the range of 0 to 23, inclusive, which are specified in subclause D.2, are extended as follows:

– If payloadType is equal to 3, 8, 19, 20, or 22, the following applies:

  – If the SEI message is not included in a scalable nesting SEI message, it applies to the layer representations of the current access unit that have dependency_id equal to 0 and quality_id equal to 0.

    The semantics as specified in subclause D.2 apply to the bitstream that would be obtained by invoking the bitstream extraction process as specified in subclause G.8.8.1 with dIdTarget equal to 0 and qIdTarget equal to 0. All syntax elements and derived variables that are referred to in the semantics in subclause D.2 are syntax elements and variables for layer representations with dependency_id equal to 0 and quality_id equal to 0. All SEI messages that are referred to in subclause D.2 are SEI messages that apply to layer representations with dependency_id equal to 0 and quality_id equal to 0.

  – Otherwise (the SEI message is included in a scalable nesting SEI message), the SEI message applies to all layer representations of the current access unit for which DQId is equal to any value of ((sei_dependency_id[i] << 4) + sei_quality_id[i]) with i in the range of 0 to num_layer_representations_minus1, inclusive.

    For each value of i in the range of 0 to num_layer_representations_minus1, inclusive, the semantics as specified in subclause D.2 apply to the bitstream that would be obtained by invoking the bitstream extraction process as specified in subclause G.8.8.1 with dIdTarget equal to sei_dependency_id[i] and qIdTarget equal to sei_quality_id[i]. All syntax elements and derived variables that are referred to in the semantics in subclause D.2 are syntax elements and variables for layer representations with dependency_id equal to sei_dependency_id[i] and quality_id equal to sei_quality_id[i]. All SEI messages that are referred to in subclause D.2 are SEI messages that apply to layer representations with dependency_id equal to sei_dependency_id[i] and quality_id equal to sei_quality_id[i].

– Otherwise, if payloadType is equal to 2, 6, 7, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 21, or 23, the following applies:

  – If the SEI message is not included in a scalable nesting SEI message, it applies to the dependency representations of the current access unit that have dependency_id equal to 0.

    The semantics as specified in subclause D.2 apply to the bitstream that would be obtained by invoking the bitstream extraction process as specified in subclause G.8.8.1 with dIdTarget equal to 0. All syntax elements and derived variables that are referred to in the semantics in subclause D.2 are syntax elements and variables for dependency representations with dependency_id equal to 0. All SEI messages that are referred to in subclause D.2 are SEI messages that apply to dependency representations with dependency_id equal to 0.

  – Otherwise (the SEI message is included in a scalable nesting SEI message), the scalable nesting SEI message containing the SEI message shall have all_layer_representations_in_au_flag equal to 1 or, when all_layer_representations_in_au_flag is equal to 0, all values of sei_quality_id[i] present in the scalable nesting SEI message shall be equal to 0. The SEI message that is included in the scalable nesting SEI message applies to all dependency representations of the current access unit for which dependency_id is equal to any value of sei_dependency_id[i] with i in the range of 0 to num_layer_representations_minus1, inclusive.

For each value of i in the range of 0 to num_layer_representations_minus1, inclusive, the semantics as specified in subclause D.2 apply to the bitstream that would be obtained by invoking the bitstream extraction process as specified in subclause G.8.8.1 with dIdTarget equal to sei_dependency_id[i]. All syntax elements and derived variables that are referred to in the semantics in subclause D.2 are syntax elements and variables for dependency representations with dependency_id equal to sei_dependency_id[i]. All SEI messages that are referred to in subclause D.2 are SEI messages that apply to dependency representations with dependency_id equal to sei_dependency_id[i].

When payloadType is equal to 10 for the SEI message that is included in a scalable nesting SEI message, the semantics for sub_seq_layer_num of the sub-sequence information SEI message is modified as follows:

sub_seq_layer_num specifies the sub-sequence layer number of the current picture. When the current picture resides in a sub-sequence for which the first picture in decoding order is an IDR picture, the value of sub_seq_layer_num shall be equal to 0. For a non-paired reference field, the value of sub_seq_layer_num shall be equal to 0. sub_seq_layer_num shall be in the range of 0 to 255, inclusive.

– Otherwise, if payloadType is equal to 0 or 1, the following applies:

  – If the SEI message is not included in a scalable nesting SEI message, the following applies. When the SEI message and all other SEI messages with payloadType equal to 0 or 1 not included in a scalable nesting SEI message are used as the buffering period and picture timing SEI messages for checking the bitstream conformance according to Annex C and the decoding process specified in clauses 2-9 is used, the bitstream shall be conforming to this Recommendation | International Standard.

    The value of seq_parameter_set_id in a buffering period SEI message not included in a scalable nesting SEI message shall be equal to the value of seq_parameter_set_id in the picture parameter set that is referenced by the layer representation with DQId equal to 0 of the primary coded picture in the same access unit.

  – Otherwise (the SEI message is included in a scalable nesting SEI message), the following applies. When the SEI message and all other SEI messages with payloadType equal to 0 or 1 included in a scalable nesting SEI message with identical values of sei_temporal_id, sei_dependency_id[i], and sei_quality_id[i] are used as the buffering period and picture timing SEI messages for checking the bitstream conformance according to Annex C, the bitstream that would be obtained by invoking the bitstream extraction process as specified in subclause G.8.8.1 with tIdTarget equal to sei_temporal_id, dIdTarget equal to sei_dependency_id[i], and qIdTarget equal to sei_quality_id[i] shall be conforming to this Recommendation | International Standard.

    In the semantics of subclauses D.2.1 and D.2.2, the syntax elements num_units_in_tick, time_scale, fixed_frame_rate_flag, nal_hrd_parameters_present_flag, vcl_hrd_parameters_present_flag, low_delay_hrd_flag, and pic_struct_present_flag and the derived variables NalHrdBpPresentFlag, VclHrdBpPresentFlag, and CpbDpbDelaysPresentFlag are substituted with the syntax elements vui_ext_num_units_in_tick[i], vui_ext_time_scale[i], vui_ext_fixed_frame_rate_flag[i], vui_ext_nal_hrd_parameters_present_flag[i], vui_ext_vcl_hrd_parameters_present_flag[i], vui_ext_low_delay_hrd_flag[i], and vui_ext_pic_struct_present_flag[i] and the derived variables VuiExtNalHrdBpPresentFlag[i], VuiExtVclHrdBpPresentFlag[i], and VuiExtCpbDpbDelaysPresentFlag[i].

    The value of seq_parameter_set_id in a buffering period SEI message included in a scalable nesting SEI message with the values of sei_dependency_id[i] and sei_quality_id[i] shall be equal to the value of seq_parameter_set_id in the picture parameter set that is referenced by the layer representation with DQId equal to ((sei_dependency_id[i] << 4) + sei_quality_id[i]) of the primary coded picture in the same access unit.

– Otherwise (payloadType is equal to 4 or 5), the corresponding SEI message semantics are not extended.

For the semantics of SEI messages with payloadType in the range of 0 to 23, inclusive, which are specified in subclause D.2, SVC sequence parameter set is substituted for sequence parameter set; the parameters of the picture parameter set RBSP and SVC sequence parameter set RBSP that are in effect are specified in subclause G.7.4.1.2.1.

When an SEI NAL unit contains an SEI message with payloadType in the range of 24 to 35, inclusive, it shall not contain any SEI message with payloadType less than 24 that is not included in a scalable nesting SEI message, and the first SEI message in the SEI NAL unit shall have payloadType in the range of 24 to 35, inclusive.

When an SEI NAL unit contains an SEI message with payloadType equal to 24, 28, or 29, it shall not contain any SEI message with payloadType not equal to 24, 28, or 29.

When a scalable nesting SEI message (payloadType is equal to 30) is present in an SEI NAL unit, it shall be the only SEI message in the SEI NAL unit.

The semantics for SEI messages with payloadType in the range of 24 to 35, inclusive, are specified in the following.

G.13.2.1 Scalability information SEI message semantics

The scalability information SEI message provides scalability information for subsets of the bitstream.

In the following specification of this subclause, a VCL NAL unit of a primary coded picture is also referred to as primary coded VCL NAL unit and a VCL NAL unit of a redundant coded picture is also referred to as redundant coded VCL NAL unit.

A scalability information SEI message shall not be included in a scalable nesting SEI message.

A scalability information SEI message shall not be present in access units that contain primary coded VCL NAL units with IdrPicFlag equal to 0. The set of access units consisting of the access unit associated with the scalability information SEI message and all succeeding access units in decoding order until, but excluding, the next access unit that does not contain any primary coded VCL NAL unit with IdrPicFlag equal to 0 (if present) or the end of the bitstream (otherwise) is referred to as the target access unit set. The scalability information SEI message applies to the target access unit set.

The scalability information SEI message provides information for subsets of the target access unit set. These subsets are referred to as scalable layers. A scalable layer represents a set of NAL units, inside the target access unit set, that consists of VCL NAL units with the same values of dependency_id, quality_id, and temporal_id, as specified later in this subclause, and associated non-VCL NAL units. When present in the target access unit set, the following NAL units are associated non-VCL NAL units for a scalable layer:

– sequence parameter set, subset sequence parameter set, and picture parameter set NAL units that are referenced in the VCL NAL units of the scalable layer (via the syntax element pic_parameter_set_id),

– sequence parameter set extension NAL units that are associated with a sequence parameter set NAL unit referenced in the VCL NAL units of the scalable layer,

– filler data NAL units that are associated with the same values of dependency_id, quality_id, and temporal_id as the VCL NAL units of the scalable layer,

– SEI NAL units containing SEI messages, with payloadType not equal to 24, 28, or 29, that apply to subsets of the bitstream that contain one or more VCL NAL units of the scalable layer,

– access unit delimiter, end of sequence, and end of stream NAL units that are present in access units that contain VCL NAL units of the scalable layer,

– when dependency_id and quality_id are both equal to 0 in the VCL NAL units of a scalable layer, coded slice of an auxiliary coded picture without partitioning NAL units that are present in access units that contain VCL NAL units of the scalable layer.

A scalable layer A is directly dependent on a scalable layer B when any primary coded VCL NAL unit of the scalable layer A references data of any VCL NAL unit of the scalable layer B through inter prediction or inter-layer prediction as specified in the decoding process in subclause G.8, with the following exception: A scalable layer A (identified by layer_id[a]) is not directly dependent on a scalable layer B (identified by layer_id[b]) when dependency_id[a] is equal to dependency_id[b], sub_pic_layer_flag[a] is equal to 1, and one of the following conditions is true:

– sub_pic_layer_flag[b] is equal to 0,

– sub_pic_layer_flag[b] is equal to 1 and (horizontal_offset[a] is not equal to horizontal_offset[b], vertical_offset[a] is not equal to vertical_offset[b], region_width[a] is not equal to region_width[b], or region_height[a] is not equal to region_height[b]).

NOTE 1 – Sub-picture scalable layers with a particular value of dependency_id and a particular sub-picture area are only considered to depend on scalable layers with the same value of dependency_id when these scalable layers are associated with the same sub-picture area.

A scalable layer A is indirectly dependent on a scalable layer B when the scalable layer A is not directly dependent on the scalable layer B but there exists a set of n (with n being greater than 0) scalable layers $\{C_0, .., C_{n-1}\}$ with the following properties: The scalable layer A is directly dependent on the scalable layer C_0, each scalable layer C_i with i in the range of 0 to $n - 2$, inclusive, is directly dependent on the scalable layer C_{i+1}, and the scalable layer C_{n-1} is directly dependent on the scalable layer B.

The representation of a particular scalable layer is the set of NAL units that represents the set union of the particular scalable layer and all scalable layers on which the particular scalable layer directly or indirectly depends. The representation of a scalable layer is also referred to as scalable layer representation. In the following specification of this subclause, the terms representation of a scalable layer and scalable layer representation are also used for referring to the access unit set that can be constructed from the NAL units of the scalable layer representation. A scalable layer representation can be decoded independently of all NAL units that do not belong to the scalable layer representation.

The decoding result of a scalable layer representation is the set of decoded pictures that are obtained by decoding the access unit set of the scalable layer representation.

NOTE 2 – The set of access units that is formed by the representation of a scalable layer with sub_pic_layer_flag[i] equal to 1 does not conform to this Recommendation | International Standard, since the primary coded VCL NAL units with quality_id equal to 0 that belong to such a scalable layer representation do not cover all macroblocks of the layer pictures with dependency_id equal to dependency_id[i] and quality_id equal to 0. For the following specification in this subclause, the decoding result for the representation of a scalable layer with sub_pic_layer_flag[i] equal to 1 is the decoding result that would be obtained for the sub-picture area (as specified later in this subclause) by following the decoding process in subclause G.8 but ignoring the constraint that the layer representations with quality_id equal to 0 of primary coded pictures must cover all macroblocks of the corresponding layer pictures.

Each scalable layer is associated with a unique layer identifier as specified later in this subclause. The representation of a particular scalable layer with a particular layer identifier layerId does not include any scalable layer with a layer identifier greater than layerId, but it may include scalable layers with layer identifiers less than layerId. The scalable layers on which a particular scalable layer depends may be indicated in the scalability information SEI message as specified later in this subclause.

NOTE 3 – When all scalable layers for which scalability information is provided in the scalability information SEI message have sub_pic_layer_flag[i] equal to 0, the unique layer identifier values may be set equal to (128 * dependency_id + 8 * quality_id + temporal_id), with dependency_id, quality_id, and temporal_id being the corresponding syntax elements that are associated with the VCL NAL units of a scalable layer.

temporal_id_nesting_flag indicates whether inter prediction is additionally restricted for the target access unit set. Depending on the value of temporal_id_nesting_flag, the following applies:

– If temporal_id_nesting_flag is equal to 1, the scalability information SEI message indicates that the following constraint is obeyed for all access units sets that can be derived from the target access unit set by invoking the sub-bitstream extraction process as specified in subclause G.8.8.1 with tIdTarget equal to any value in the range of 0 to 7, inclusive, dIdTarget equal to any value in the range of 0 to 7, inclusive, and qIdTarget equal to any value in the range of 0 to 15, inclusive, as the inputs: The values of the samples in the decoded pictures for each access unit auA with temporal_id equal to tIdA and all following access units in decoding order are independent of an access unit auB with temporal_id equal to tIdB and tIdB less than or equal to tIdA, when there exists an access unit auC with temporal_id equal to tIdC and tIdC less than tIdB, that follows the access unit auB and precedes the access unit auA in decoding order.

– Otherwise (temporal_id_nesting_flag is equal to 0), the scalability information SEI message indicates that the constraint specified for temporal_id_nesting_flag equal to 1 may or may not be obeyed.

NOTE 4 – The syntax element temporal_id_nesting_flag is used to indicate that temporal up-switching, i.e., switching from decoding of up to a particular temporal_id value tIdN to decoding of up to a temporal_id value tIdM greater than tIdN, is always possible inside the target access unit set.

priority_layer_info_present_flag equal to 1 specifies that characteristic information for priority layers, as specified later in this subclause, is present in the scalability information SEI message and that priority layer information SEI messages associating an alternative value for priority_id with each layer representation of the primary coded pictures in the target access unit set are present. priority_layer_info_present_flag equal to 0 specifies that characteristic information for priority layers is not present in the scalability information SEI message.

priority_id_setting_flag equal to 1 specifies that syntax elements priority_id_setting_uri[i] are present in the scalability information SEI message and that the description of the method used to calculate the priority_id values is provided by the specified universal resource identifier (URI). priority_id_setting_flag equal to 0 specifies that syntax elements priority_id_setting_uri[i] are not present in the scalability information SEI message.

num_layers_minus1 plus 1 specifies the number of scalable layers for which information is provided in the scalability information SEI message. The value of num_layers_minus1 shall be in the range of 0 to 2047, inclusive.

layer_id[i **]** specifies the layer identifier of the i-th scalable layer specified in the scalability information SEI message. layer_id[i] shall be in the range of 0 to 2047, inclusive.

For the following specification inside this subclause, the scalable layer with layer identifier equal to the current value of layer_id[i] is referred to as the current scalable layer, and the representation of the current scalable layer is referred to as the current scalable layer representation.

priority_id[i] indicates an upper bound for the priority_id values of the current scalable layer representation. All primary coded VCL NAL units of the current scalable layer representation shall have a value of priority_id that is less than or equal to priority_id[i].

discardable_flag[i] equal to 1 indicates that all primary coded VCL NAL units of the current scalable layer have discardable_flag equal to 1. discardable_flag[i] equal to 0 indicates that the current scalable layer may contain one or more primary coded VCL NAL units with discardable_flag equal to 0.

dependency_id[i **]**, **quality_id[** i **]**, and **temporal_id[** i **]** are equal to the values of dependency_id, quality_id, and temporal_id, respectively, of the VCL NAL units of the current scalable layer. All VCL NAL units of a scalable layer have the same values of dependency_id, quality_id, and temporal_id.

When the target access unit set does not contain any primary coded VCL NAL unit with particular values of dependency_id, quality_id, and temporal_id, the scalability information SEI message shall not contain information for a scalable layer with dependency_id[i], quality_id[i], and temporal_id[i] equal to the particular values of dependency_id, quality_id, and temporal_id, respectively.

NOTE 5 – When an application removes NAL units from a scalable bitstream, e.g. in order to adapt the bitstream to a transmission channel or the capabilities of a receiving device, and keeps the present scalability information SEI messages, it might need to modify the content of the scalability information SEI messages in order to obtain a bitstream conforming to this Recommendation | International Standard.

sub_pic_layer_flag[i **]** specifies whether the current scalable layer represents a sub-picture scalable layer as specified subsequently. Depending on sub_pic_layer_flag[i], the following applies:

– If sub_pic_layer_flag[i] is equal to 0, the current scalable layer does not represent a sub-picture scalable layer. The VCL NAL units of the current scalable layer are all VCL NAL units of the target access unit set that have dependency_id, quality_id, and temporal_id equal to dependency_id[i], quality_id[i], and temporal_id[i], respectively.

– Otherwise (sub_pic_layer_flag[i] is equal to 1), the current scalable layer represents a sub-picture scalable layer and is associated with a sub-picture area as specified in the following:

(a) The sub-picture area is a rectangular area of slice group map units inside the layer frames with dependency_id equal to dependency_id[i] and represents a proper subset of the area of the layer frames with dependency_id equal to dependency_id[i]. The sub-picture area associated with a sub-picture scalable layer does not change inside the target access unit set. The sub-picture area is specified by the syntax elements horizontal_offset[i], vertical_offset[i], region_width[i], and region_height[i] as specified later in this subclause.

NOTE 6 – The sub-picture area for a sub-picture scalable layer may additionally be indicated by the presence of sub-picture scalable layer SEI messages with layer_id equal to value of layer_id[i] for the current scalable layer.

(b) When a VCL NAL unit of the target access unit set has dependency_id equal to dependency_id[i] and contains any macroblock that resides inside the sub-picture area, it shall not contain any macroblock that resides outside of the sub-picture area.

(c) The VCL NAL units of the current scalable layer are the coded slice NAL units of the target access unit set that have dependency_id, quality_id, and temporal_id equal to dependency_id[i], quality_id[i], and temporal_id[i], respectively, and for which the macroblock specified by first_mb_in_slice resides inside the specified sub-picture area and the associated prefix NAL units (when present).

(d) For all access units sets that can be derived from the target access unit set by invoking the sub-bitstream extraction process as specified in subclause G.8.8.1 with dIdTarget equal to dependency_id[i] and qIdTarget equal to any value in the range of 0 to 15, inclusive, as the inputs, the following constraint shall be obeyed: No sample value outside the sub-picture area and no sample value at a fractional sample position that is derived using one or more sample values outside the sub-picture area is used, in the decoding process as specified in subclause G.8, for inter prediction of any sample within the sub-picture area.

When the target access unit set contains any primary coded VCL NAL unit with particular values of dependency_id, quality_id, and temporal_id, the scalability information SEI message shall contain information for a exactly one scalable layer with dependency_id[i], quality_id[i], and temporal_id[i] equal to the particular values of dependency_id, quality_id, and temporal_id, respectively, and sub_pic_layer_flag[i] equal to 0.

NOTE 7 – The scalability information SEI message may additionally contain information for one or more scalable layers with dependency_id[i], quality_id[i], and temporal_id[i] equal to the particular values of dependency_id, quality_id, and temporal_id, respectively, and sub_pic_layer_flag[i] equal to 1.

When sub_pic_layer_flag[i] is equal to 1 for the current scalable layer and the target access unit set contains any primary coded VCL NAL unit that has dependency_id equal to dependency_id[i], resides inside the sub-picture area, and has particular values of quality_id and temporal_id, with either quality_id not equal to quality_id[i] or temporal_id not equal to temporal_id[i], the scalability information SEI message shall also contain information for a scalable layer j with dependency_id[j] equal to dependency_id[i], quality_id[j] and temporal_id[j] equal to the particular values of quality_id and temporal_id, respectively, sub_pic_layer_flag[j] equal to 1, and horizontal_offset[j], vertical_offset[j], region_width[j], and region_height[j] equal to horizontal_offset[i], vertical_offset[i], region_width[i], and region_height[i], respectively.

The scalability information SEI message shall not contain information for two or more scalable layers with sub_pic_layer_flag[i] equal to 1 and the same values of dependency_id[i], quality_id[i], temporal_id[i], sub_pic_layer_flag[i], horizontal_offset[i], vertical_offset[i], region_width[i], and region_height[i].

When the scalability information SEI message contains information for two scalable layers A and B (identified by layer_id[a] and layer_id[b], respectively) with dependency_id[a] equal to dependency_id[b], quality_id[a] equal to quality_id[b], temporal_id[a] equal to temporal_id[b], sub_pic_layer_flag[a] equal to 1, and sub_pic_layer_flag[b] equal to 1, and the sub-picture areas associated with the scalable layers A and B overlap, the scalability information SEI message shall also contain information for a scalable layer C (identified by layer_id[c]) with dependency_id[c] equal to dependency_id[b], quality_id[c] equal to quality_id[b], temporal_id[c] equal to temporal_id[b], and sub_pic_layer_flag[c] is equal to 1, and with the scalable layer C being associated with a sub-picture area that represents the intersection of the sub-picture areas associated with the scalable layers A and B.

sub_region_layer_flag[i **]** equal to 1 specifies that the syntax elements base_region_layer_id[i] and dynamic_rect_flag[i] for the current scalable layer are present in the scalability information SEI message. sub_region_layer_flag[i] equal to 0 specifies that the syntax elements base_region_layer_id[i] and dynamic_rect_flag[i] for the current scalable layer are not present in the scalability information SEI message.

When sub_pic_layer_flag[i] is equal to 1, sub_region_layer_flag[i] shall be equal to 1.

iroi_division_info_present_flag[i **]** equal to 1 specifies that the layer pictures with dependency_id equal to dependency_id[i] are divided along slice group map unit boundaries into multiple rectangular regions of interest, referred to as interactive regions of interest (IROIs), and that the IROI division information is explicitly signalled in the scalability information SEI message as specified later in this subclause, and that the syntax elements frame_width_in_mbs_minus1[i] and frame_height_in_mbs_minus1[i] for the current scalable layer are present in the scalability information SEI message. iroi_division_info_present_flag[i] equal to 0 specifies that the IROI division information for the current scalable layer is not present in the scalability information SEI message.

When sub_pic_layer_flag[i] is equal to 1, iroi_division_info_present_flag[i] shall be equal to 0.

When iroi_division_info_present_flag[i] is equal to 1, the following is specified:

(a) When a primary coded VCL NAL unit of the target access unit set has dependency_id equal to dependency_id[i] and contains any macroblock that resides inside a particular IROI, it shall not contain any macroblock that resides outside of the particular IROI.

(b) For all access units sets that can be derived from the target access unit set by invoking the sub-bitstream extraction process as specified in subclause G.8.8.1 with dIdTarget equal to dependency_id[i] and qIdTarget equal to any value in the range of 0 to 15, inclusive, as the inputs, the following constraint shall be obeyed: No sample value outside a particular IROI and no sample value at a fractional sample position that is derived using one or more sample values outside the particular IROI is used, in the decoding process as specified in subclause G.8, for inter prediction of any sample within the particular IROI.

All scalable layers with the same value of dependency_id[i] for which scalability information is present in the scalability information SEI message shall have the same value of iroi_division_info_present_flag[i].

profile_level_info_present_flag[i **]** equal to 1 specifies that profile_idc, constraint_set0_flag, constraint_set1_flag, constraint_set2_flag, constraint_set3_flag, constraint_set4_flag, constraint_set5_flag, reserved_zero_2bits, and level_idc applicable for the current scalable layer representation are indicated by the value of layer_profile_level_idc[i] as specified later in this subclause.

When profile_level_info_present_flag[i] is equal to 0, profile_idc, constraint_set0_flag, constraint_set1_flag, constraint_set2_flag, constraint_set3_flag, constraint_set4_flag, constraint_set5_flag, and level_idc applicable for the current scalable layer representation are not indicated in the scalability information SEI message.

bitrate_info_present_flag[i **]** equal to 1 specifies that the bit rate information for the current scalable layer representation is present in the scalability information SEI message. bitrate_info_present_flag[i] equal to 0 specifies that the bit rate information for the current scalable layer representation is not present in the scalability information SEI message.

frm_rate_info_present_flag[i **]** equal to 1 specifies that the frame rate information for the current scalable layer representation is present in the scalability information SEI message. frm_rate_info_present_flag[i] equal to 0 specifies that the frame rate information for the current scalable layer representation is not present in the scalability information SEI message.

frm_size_info_present_flag[i **]** equal to 1 specifies that the frame size information for the current scalable layer representation is present in the scalability information SEI message. frm_size_info_present_flag[i] equal to 0 specifies that the presence of the frame size information for the current scalable layer representation in the scalability information SEI message is specified by iroi_division_info_present_flag[i].

layer_dependency_info_present_flag[i **]** equal to 1 specifies that one or more syntax elements dependent_layer_id_delta_minus1[i][j] indicating the layer dependency information for the current scalable layer are present in the scalability information SEI message. layer_dependency_info_present_flag[i] equal to 0 specifies that, for the current scalable layer, the syntax element layer_dependency_info_src_layer_id_delta[i] is present in the scalability information SEI message.

parameter_sets_info_present_flag[i **]** equal to 1 specifies that the values of seq_parameter_set_id of the sequence parameter sets and subset sequence parameter sets and the values of pic_parameter_set_id of the picture parameter sets that are referred to in the primary coded VCL NAL units of the current scalable layer representation are present in the scalability information SEI message. parameter_sets_info_present_flag[i] equal to 0 specifies that, for the current scalable layer, the syntax element parameter_sets_info_src_layer_id_delta[i] is present in the scalability information SEI message.

bitstream_restriction_info_present_flag[i **]** equal to 1 specifies that the bitstream restriction information for the current scalable layer representation is present in the scalability information SEI message. bitstream_restriction_info_present_flag[i] equal to 0 specifies that the bitstream restriction information for the current scalable layer representation is not present in the scalability information SEI message.

exact_inter_layer_pred_flag[i **]** equal to 1 indicates that, for all primary coded VCL NAL units with no_inter_layer_pred_flag equal to 0 of the current scalable layer representation, the reference layer representation (specified by the syntax elements ref_layer_dq_id) that is used for inter-layer prediction in the decoding process, as specified in subclause G.8, is the same as the reference layer representation that was used during encoding. exact_inter_layer_pred_flag[i] equal to 0 indicates that, for the primary coded VCL NAL units with no_inter_layer_pred_flag equal to 0 of the current scalable layer representation, the reference layer representations that are used for inter-layer prediction in the decoding process may or may not be the same as the reference layer representations that were used during encoding.

NOTE 8 – A mismatch between the reference layer representation that is used for inter-layer prediction in the decoding process and the reference layer representation that was used during encoding may be a result of a bitstream adaption, in which one or more layer representations that are referred to in inter-layer prediction are removed from the bitstream, any of the primary coded VCL NAL units that refer to any of the removed layer representations by inter-layer prediction is not removed from the bitstream, and the value of the syntax elements ref_layer_dq_id in the primary coded VCL NAL units that refer to any of the removed layer representations is modified in order to obtain a bitstream conforming to this Recommendation | International Standard.

exact_inter_layer_pred_flag[i] should be equal to 1. When the current scalable layer representation does not contain any primary coded VCL NAL unit with no_inter_layer_pred_flag equal to 0, exact_inter_layer_pred_flag[i] shall be equal to 1.

exact_sample_value_match_flag[i **]** indicates whether the values of decoded samples for decoding the representation of the current sub-picture scalable layer (when sub_pic_layer_flag[i] is equal to 1) or any particular IROI within the current scalable layer representation (when iroi_division_info_present_flag[i] is equal to 1) are identical to the values of the same decoded samples that would be obtained by decoding all layer representations, of the primary coded pictures inside the target access unit set, that have DQId less than or equal to 16 * dependency_id[i] + quality_id[i] and temporal_id less than or equal to temporal_id[i].

With picSubset being the set of the primary coded pictures of the current scalable layer representation that contain any VCL NAL unit with dependency_id equal to dependency_id[i], the following applies:

– If sub_pic_layer_flag[i] is equal to 1 (iroi_division_info_present_flag[i] is equal to 0), the following is specified:

1. Let picLRepSubset be the set of primary coded pictures that is formed by all the layer representations, of the target access unit set, that contain any primary coded VCL NAL unit present in the set of pictures picSubset.

    NOTE 9 – picSubset is a proper subset of picLRepSubset. picSubset only contains the primary coded slices of the current (sub-picture) scalable layer representation, picLRepSubset contains all primary coded slices of the corresponding layer representations (i.e. the complete layer representations that contain any slice of picSubset).

2. exact_sample_value_match_flag[i] equal to 1 indicates that the value of each decoded sample inside the sub-picture area for decoding the picture set picSubset is identical to the value of the same decoded sample that would be obtained by decoding the picture set picLRepSubset.

3. exact_sample_value_match_flag[i] equal to 0 indicates that the value of any decoded sample inside the sub-picture area for decoding the picture set picSubset may or may not be identical to the value of the same decoded sample that would be obtained by decoding the picture set picLRepSubset.

– Otherwise (sub_pic_layer_flag[i] is equal to 0 and iroi_division_info_present_flag[i] is equal to 1), for each particular IROI, the following is specified:

1. Let picIROISubset be the set of primary coded VCL NAL units that is obtained by removing all the VCL NAL units from the set of pictures picSubset that do not cover any macroblock inside the IROI.
2. exact_sample_value_match_flag[i] equal to 1 indicates that the value of each decoded sample inside the IROI for decoding the picture set picSubset is identical to the value of the same decoded sample that would be obtained by decoding the picture set picIROISubset.
3. exact_sample_value_match_flag[i] equal to 0 indicates that the value of any decoded sample inside the IROI for decoding the picture set picSubset may or may not be identical to the value of the same decoded sample that would be obtained by decoding the picture set picIROISubset.

NOTE 10 – In the above specification, the decoding result for picIROISubset is the decoding result that would be obtained for the IROI by following the decoding process in subclause G.8 but ignoring the constraint that the layer representations with quality_id equal to 0 of primary coded pictures must cover all macroblocks of the corresponding layer pictures.

NOTE 11 – When disable_deblocking_filter_idc is equal to 1, 2, or 5 in all primary coded slices of the current scalable layer representation that have dependency_id equal to dependency_id[i], exact_sample_value_match_flag should be equal to 1.

layer_conversion_flag[i **]** equal to 1 indicates that the representation of the current scalable layer can be converted into an alternative set of access units that conforms to one or more of the profiles specified in Annex A and gives exactly the same decoding result as the current scalable layer representation and that this conversion can be done without full reconstruction and re-encoding. layer_conversion_flag[i] equal to 0 indicates that such a conversion of the current scalable layer representation may or may not be possible.

layer_output_flag[i **]** equal to 1 indicates that the decoding result for the current scalable layer representation is intended for output. layer_output_flag[i] equal to 0 indicates that the decoding result for the current scalable layer representation is not intended for output.

NOTE 12 – The decoding result for a scalable layer representation with layer_output_flag[i] equal to 0 may be inappropriate for output due to its low visual quality.

layer_profile_level_idc[i] indicates the conformance point of the representation of the current scalable layer. layer_profile_level_idc[i] is the exact copy of the three bytes comprised of profile_idc, constraint_set0_flag, constraint_set1_flag, constraint_set2_flag, constraint_set3_flag, constraint_set4_flag, constraint_set5_flag, reserved_zero_2bits and level_idc, as if these syntax elements were used to specify the profile and level conformance of the representation of the current scalable layer.

NOTE 13 – The representation of a sub-picture scalable layer (sub_pic_layer_flag[i] is equal to 1) does not conform to this Recommendation | International Standard, since the primary coded VCL NAL units with quality_id equal to 0 that belong to a sub-picture scalable layer representation do not cover all macroblocks of the layer pictures with dependency_id equal to dependency_id[i] and quality_id equal to 0. For sub-picture scalable layers, the violation of the constraint that the layer representations with quality_id equal to 0 of primary coded pictures must cover all macroblocks of the corresponding layer pictures is ignored in the conformance point indication by layer_profile_level_idc[i].

avg_bitrate[i **]** indicates the average bit rate of the representation of the current scalable layer. The average bit rate for the representation of the current scalable layer in bits per second is given by BitRateBPS(avg_bitrate[i]) with the function BitRateBPS( ) being specified by:

$$\text{BitRateBPS}(\ x\ ) = (\ x\ \&\ (\ 2^{14} - 1\ )\ ) * 10^{(\ 2 + (\ x >> 14\ )\ )} \quad \text{(G-370)}$$

The average bit rate is derived according to the access unit removal time specified in Annex C of this Recommendation | International Standard. In the following, bTotal is the number of bits in all NAL units of the current scalable layer representation, t_1 is the removal time (in seconds) of the access unit associated with the scalability information SEI message, and t_2 is the removal time (in seconds) of the last access unit (in decoding order) of the target access unit set.

With x specifying the value of avg_bitrate[i], the following applies:

– If t_1 is not equal to t_2, the following condition shall be true:

$$(\ x\ \&\ (\ 2^{14} - 1\ )\ )\ = =\ \text{Round}(\ \text{bTotal} \div (\ (\ t_2 - t_1\ ) * 10^{(\ 2 + (\ x >> 14\ )\ )}\ )\ ) \quad \text{(G-371)}$$

– Otherwise (t_1 is equal to t_2), the following condition shall be true:

$$(\ x\ \&\ (\ 2^{14} - 1\ )\ )\ = =\ 0 \quad \text{(G-372)}$$

max_bitrate_layer[i **]** indicates an upper bound for the bit rate of the current scalable layer in any fixed-size time window, specified by max_bitrate_calc_window[i], of access unit removal time as specified in Annex C. The upper bound for the bit rate of the current scalable layer in bits per second is given by BitRateBPS(max_bitrate_layer[i]) with the function BitRateBPS( ) being specified in Equation G-370. The bit rate values are derived according to the access unit removal time specified in Annex C of this Recommendation | International Standard. In the following, t_1 is any point in time (in seconds), t_2 is set equal to t_1 + max_bitrate_calc_window[i] ÷ 100, and bTotal is the number of bits in all NAL units of the current scalable layer that belong to access units with a removal time greater than or equal to t_1 and less than t_2. With x specifying the value of max_bitrate_layer[i], the following condition shall be obeyed for all values of t_1:

$$(x \,\&\, (2^{14} - 1)) \;>=\; \text{bTotal} \div ((t_2 - t_1) * 10^{(2 + (x >> 14))}) \qquad \text{(G-373)}$$

max_bitrate_layer_representation[i **]** indicates an upper bound for the bit rate of the current scalable layer representation in any fixed-size time window, specified by max_bitrate_calc_window[i], of access unit removal time as specified in Annex C. The upper bound for the bit rate of the current scalable layer representation in bits per second is given by BitRateBPS(max_bitrate_layer_representation[i]) with the function BitRateBPS( ) being specified in Equation G-370. The bit rate values are derived according to the access unit removal time specified in Annex C of this Recommendation | International Standard. In the following, t_1 is any point in time (in seconds), t_2 is set equal to t_1 + max_bitrate_calc_window[i] ÷ 100, and bTotal is the number of bits in all NAL units of the current scalable layer representation that belong to access units with a removal time greater than or equal to t_1 and less than t_2. With x specifying the value of max_bitrate_layer_representation[i], the condition specified in Equation G-373 shall be obeyed.

max_bitrate_calc_window[i **]** specifies the size of the time window that is used for calculating the upper bounds for the bit rate of the current scalable layer (indicated by max_bitrate_layer[i]) and the bit rate of the current scalable layer representation (indicated by max_bitrate_layer_representation[i]) in units of 1/100 second.

constant_frm_rate_idc[i **]** indicates whether the frame rate of the current scalable layer representation is constant. In the following, a temporal segment tSeg is any set of two or more consecutive access units, in decoding order, of the current scalable layer representation, fTotal(tSeg) is the number of frames, complementary field pairs, and non-paired fields in the temporal segment tSeg, t_1(tSeg) is the removal time (in seconds) of the first access unit (in decoding order) of the temporal segment tSeg, t_2(tSeg) is the removal time (in seconds) of the last access unit (in decoding order) of the temporal segment tSeg, and avgFR(tSeg) is the average frame rate in the temporal segment tSeg, which is given by:

$$\text{avgFR(tSeg)} \;==\; \text{Round(fTotal(tSeg) * 256} \div (t_2\text{(tSeg)} - t_1\text{(tSeg)})) \qquad \text{(G-374)}$$

If the current scalable layer representation does only contain one access unit or the value of avgFR(tSeg) is constant over all temporal segments of the scalable layer representation, the frame rate is constant; otherwise, the frame rate is not constant. constant_frm_rate_idc[i] equal to 0 indicates that the frame rate of the current scalable layer representation is not constant. constant_frm_rate_idc[i] equal to 1 indicates that the frame rate of the current scalable layer representation is constant. constant_frm_rate_idc[i] equal to 2 indicates that the frame rate of the current scalable layer representation may or may not be constant. The value of constant_frm_rate_idc[i] shall be in the range of 0 to 2, inclusive.

avg_frm_rate[i **]** indicates the average frame rate, in units of frames per 256 seconds, of the representation of the current scalable layer. With fTotal being the number of frames, complementary field pairs, and non-paired fields in the current scalable layer representation, t_1 being the removal time (in seconds) of the access unit associated with the scalability information SEI message, and t_2 being the removal time (in seconds) of the last access unit (in decoding order) of the target access unit set, the following applies:

– If t_1 is not equal to t_2, the following condition shall be true:

$$\text{avg\_frm\_rate[i]} \;==\; \text{Round(fTotal * 256} \div (t_2 - t_1)) \qquad \text{(G-375)}$$

– Otherwise (t_1 is equal to t_2), the following condition shall be true:

$$\text{avg\_frm\_rate[i]} \;==\; 0 \qquad \text{(G-376)}$$

frm_width_in_mbs_minus1[i **]** and **frm_height_in_mbs_minus1[** i **]** indicate the width and height, respectively, of the decoded pictures for the current scalable layer representation (when sub_pic_layer_flag[i] is equal to 0) or the sub-picture area inside the decoded pictures for the current sub-picture scalable layer (when sub_pic_layer_flag[i] is equal to 1). When frame_mbs_only_flag is equal to 0 for any primary coded VCL NAL unit of the current scalable layer, (frm_height_in_mbs_minus1[i] + 1) % 2 shall be equal to 0.

Let picSubset be the set of the primary coded pictures inside the current scalable layer representation that contain any VCL NAL unit with dependency_id equal to dependency_id[i]. For decoding the picture set picSubset, the following applies:

– If sub_pic_layer_flag[i] is equal to 0, the width and height of a decoded picture are equal to frm_width_in_mbs_minus1[i] + 1 and ((frm_height_in_mbs_minus1[i] + 1) / (1 + field_pic_flag)) macroblocks, respectively, with field_pic_flag being the slice header syntax element in the slices with dependency_id equal to dependency_id[i] of the corresponding primary coded picture. The width and height of the decoded pictures that are indicated by frm_width_in_mbs_minus1[i] and frm_height_in_mbs_minus1[i], respectively, shall be identical to the width and height of the decoded pictures that are specified by the syntax elements pic_width_in_mbs_minus1 and pic_height_in_map_units_minus1, respectively, of the SVC sequence parameter sets referenced in the corresponding coded slice NAL units with dependency_id equal to dependency_id[i].

– Otherwise (sub_pic_layer_flag[i] is equal to 1), the width and height of the sub-picture area inside a decoded picture are equal to frm_width_in_mbs_minus1[i] + 1 and ((frm_height_in_mbs_minus1[i] + 1) / (1 + field_pic_flag)) macroblocks, respectively, with field_pic_flag being the slice header syntax element in the slices with dependency_id equal to dependency_id[i] of the corresponding primary coded picture. The sub-picture area that is indicated by frm_width_in_mbs_minus1[i] and frm_height_in_mbs_minus1[i] shall be less than the area of the decoded pictures, which is specified by the syntax elements pic_width_in_mbs_minus1 and pic_height_in_map_units_minus1 of the SVC sequence parameter sets referenced in the corresponding coded slice NAL units with dependency_id equal to dependency_id[i].

The variable FrmWidthInMbs[i] is set equal to (frm_width_in_mbs_minus1[i] + 1). The variable FrmHeightInMbs[i] is set equal to frm_height_in_mbs_minus1[i] + 1 . The variable FrmSizeInMbs[i] is set equal to (FrmWidthInMbs[i] * FrmHeightInMbs[i]).

base_region_layer_id[i **]** indicates the layer identifier layer_id[b] of the scalable layer b that represents the base region for the current scalable layer as specified in the following. The value of base_region_layer_id[i] shall be in the range of 0 to 2047, inclusive.

Let picSubset be the set of the primary coded pictures, inside the current scalable layer representation, that contain any VCL NAL unit with dependency_id equal to dependency_id[i]. Let basePicSubset be the set of the primary coded pictures, inside the representation of the scalable layer b with layer_id[b] equal to base_region_layer_id[i], that contain any VCL NAL unit with dependency_id equal to dependency_id[b]. Depending on sub_pic_layer_flag[i], the following applies:

– If sub_pic_layer_flag[i] is equal to 0, it is indicated that the decoded pictures for the picture set picSubset represent a subset of the areas that are represented by the decoded pictures for the picture set basePicSubset. The value of dependency_id[b] for the scalable layer b shall be less than the value of dependency_id[i] for the current scalable layer. The area that is represented by the decoded pictures for the picture set picSubset is also referred to as the region represented by the current scalable layer and the area represented by the corresponding decoded pictures for the picture set basePicSubset is also referred to as the base region for the current scalable layer.

– Otherwise (sub_pic_layer_flag[i] is equal to 1), it is indicated that the sub-picture area inside the decoded pictures for the picture set picSubset represents a proper subset of the areas that are represented by the decoded pictures for the picture set basePicSubset. The value of dependency_id[b] shall be equal to the value of dependency_id[i] for the current sub-picture scalable layer. The area that is represented by the sub-picture area inside the decoded pictures for the picture set picSubset is also referred to as the region represented by the current scalable layer and the area represented by the corresponding decoded pictures for the picture set basePicSubset is also referred to as the base region for the current scalable layer.

NOTE 14 – When sub_pic_layer_flag[i] is equal to 1, the base region represents the area of the layer pictures with dependency_id equal to dependency_id[i].

The scalability information SEI message shall contain information for the scalable layer b with layer_id[b] equal to base_region_layer_id[i], the value of sub_pic_layer_flag[b] for the scalable layer b shall be equal to 0, and the value of temporal_id[i] for the scalable layer b shall be equal to the value of temporal_id[i] for the current scalable layer.

dynamic_rect_flag[i **]** equal to 1 indicates that the region represented by the current scalable layer representation is a dynamically changing rectangular subset of the base region. dynamic_rect_flag[i] equal to 0 indicates that the region represented by the current scalable layer representation is a fixed rectangular subset of the base region and is specified by the syntax elements horizontal_offset[i], vertical_offset[i], region_width[i], and region_height[i]. When sub_pic_layer_flag[i] is equal to 1, dynamic_rect_flag[i] shall be equal to 0.

horizontal_offset[i **]**, **vertical_offset[** i **]**, **region_width[** i **]**, and **region_height[** i **]** indicate the position and size of the region represented by the current scalable layer in relation to its base region.

Let picSubset be the set of the primary coded pictures, inside the current scalable layer representation, that contain any VCL NAL unit with dependency_id equal to dependency_id[i]. Let basePicSubset be the set of the primary coded pictures, inside the representation of the scalable layer b with layer_id[b] equal to base_region_layer_id[i], that contain any VCL NAL unit with dependency_id equal to dependency_id[b]. Depending on sub_pic_layer_flag[i], the following applies:

– If sub_pic_layer_flag[i] is equal to 0, the top-left luma frame sample in the decoded pictures for picture set picSubset corresponds to the luma frame sample at the luma frame sample location (horizontal_offset[i], vertical_offset[i]) in the decoded pictures for the picture set basePicSubset. The region represented by the decoded pictures for picture set picSubset represents an area of (region_width[i])x(region_height[i]) luma frame samples in the decoded pictures for the picture set basePicSubset. When frame_mbs_only_flag is equal to 0 for any primary coded VCL NAL unit of the current scalable layer, (vertical_offset[i] % 2) and (region_height[i] % 2) shall both be equal to 0.

– Otherwise (sub_pic_layer_flag[i] is equal to 1), the top-left luma frame sample of the sub-picture area in the decoded pictures for picture set picSubset corresponds to the luma frame sample at the luma frame sample location (horizontal_offset[i], vertical_offset[i]) in the decoded pictures for the picture set basePicSubset. The region represented by the sub-picture area in the decoded pictures for picture set picSubset represents an area of (region_width[i])x(region_height[i]) luma frame samples in the decoded pictures for the picture set basePicSubset. (horizontal_offset[i] % 16) and (region_width[i] % 16) shall both be equal to 0, and depending on the values of frame_mbs_only_flag for the primary coded VCL NAL units of the current scalable layer, the following applies:

  – If frame_mbs_only_flag is equal to 1 for all primary coded VCL NAL units of the current scalable layer, (vertical_offset[i] % 16) and (region_height[i] % 16) shall both be equal to 0.

  – Otherwise (frame_mbs_only_flag is equal to 0 for any primary coded VCL NAL units of the current scalable layer), (vertical_offset[i] % 32) and (region_height[i] % 32) shall both be equal to 0.

When sub_pic_layer_flag[i] is equal to 1 and frm_size_info_present_flag[i] is equal to 1, the values of region_width[i] and region_height[i] shall be equal to (FrmWidthInMbs[i] << 4) and (FrmHeightInMbs[i] << 4), respectively.

roi_id[i] specifies a region-of-interest identifier for the region represented by the current sub-picture scalable layer, which may be used for identifying the purpose of the current sub-picture scalable layer by an application. The value of roi_id[i] shall be in the range of 0 to 63, inclusive.

Let layerIdA and layerIdB be the layer identifiers of two scalable layers A and B, respectively, both of which having sub_pic_layer_flag[i] equal to 1, and roiIdA and roidIdB be the region-of-interest identifiers of the scalable layers A and B, respectively. When layerIdA is less than layerIdB, roiIdA shall not be greater than roidIdB.

iroi_grid_flag[i] specifies how the IROI division information is indicated for the current scalable layer. iroi_grid_flag[i] equal to 1 indicates that all IROIs for the current scalable layer are aligned on a fixed-size grid as specified in the following and that the syntax elements grid_width_in_mbs_minus1[i] and grid_width_in_mbs_minus1[i] for the current scalable layer are present in the scalability information SEI message. iroi_grid_flag[i] equal to 0 indicates that the IROIs for the current scalable layer may or may not be aligned on a fixed-size grid.

All scalable layers with the same value of dependency_id[i] for which scalability information is present in the scalability information SEI message and for which iroi_division_info_present_flag[i] is equal to 1 shall have the same value of iroi_grid_flag[i].

grid_width_in_mbs_minus1[i] and **grid_height_in_mbs_minus1**[i] indicate the size of the IROI grid for the current scalable layer. When frame_mbs_only_flag is equal to 0 for any primary coded VCL NAL unit of the current scalable layer, (grid_height_in_mbs_minus1[i] + 1) % 2 shall be equal to 0.

The value of grid_width_in_mbs_minus1[i] shall be in the range of 0 to FrmWidthInMbs[i] − 1, inclusive. The value of grid_height_in_mbs_minus1[i] shall be in the range of 0 to FrmHeightInMbs[i] − 1, inclusive.

Let numX and numY be equal to (FrmWidthInMbs[i] + grid_width_in_mbs_minus1[i]) / (grid_width_in_mbs_minus1[i] + 1) and (FrmHeightInMbs[i] + grid_height_in_mbs_minus1[i]) / (grid_height_in_mbs_minus1[i] + 1), respectively.

The layer pictures with dependency_id equal to dependency_id[i] are partitioned into (numX * numY) IROIs. Let (xI[k], yI[k]) be the location of the top-left luma sample of the k-th IROI relative to the top-left luma sample of the layer picture and let w[k] and h[k] be the width and height, in luma samples, of the k-th IROI in the layer picture. With field_pic_flag being the slice header syntax element for a particular layer picture with dependency_id equal to dependency_id[i], the location of the top-left luma sample and the width and height of the k-th IROI, with k = 0..(numX * numY − 1), are given by

$$xI[k] = 16 * (k \% numX) * (grid\_width\_in\_mbs\_minus1[i] + 1) \quad (G-377)$$

$$yI[k] = 16 * (k / numX) * (grid\_height\_in\_mbs\_minus1[i] + 1) / (1 + field\_pic\_flag) \quad (G-378)$$

$$w[k] = Min(16 * (grid\_width\_in\_mbs\_minus1[i] + 1), 16 * FrmWidthInMbs[i] − xI[k]) \quad (G-379)$$

$$h[k] = Min(16 * (grid\_height\_in\_mbs\_minus1[i] + 1) / (1 + field\_pic\_flag), 16 * FrmHeightInMbs[i] / (1 + field\_pic\_flag) − yI[k]) \quad (G-380)$$

All scalable layers with the same value of dependency_id[i] for which scalability information is present in the scalability information SEI message and for which iroi_division_info_present_flag[i] is equal to 1 and iroi_grid_flag[i] is equal to 1 shall have the same values of grid_width_in_mbs_minus1[i] and grid_height_in_mbs_minus1[i].

num_rois_minus1[i] plus 1 indicates the number of IROIs for the current scalable layer.

Depending on the primary coded VCL NAL units of the current scalable layer, the following applies:

– If frame_mbs_only_flag is equal to 1 for all primary coded VCL NAL units of the current scalable layer, the value of num_rois_minus1[i] shall be in the range of 0 to FrmSizeInMbs[i] − 1, inclusive.

– Otherwise (frame_mbs_only_flag is equal to 0 for any primary coded VCL NAL unit of the current scalable layer), the value of num_rois_minus1[i] shall be in the range of 0 to FrmSizeInMbs[i] / 2 − 1, inclusive.

All scalable layers with the same value of dependency_id[i] for which scalability information is present in the scalability information SEI message and for which iroi_division_info_present_flag[i] is equal to 1 and iroi_grid_flag[i] is equal to 0 shall have the same value of num_rois_minus1[i].

first_mb_in_roi[i][j] indicates the macroblock address of the first macroblock in the j-th IROI for the current scalable layer. The value of first_mb_in_roi[i][j] shall be in the range of 0 to FrmSizeInMbs[i] − 1, inclusive. When j is greater than 0, the value of first_mb_in_roi[i][j] shall not be equal to any of the values of first_mb_in_roi[i][k] with k = 0..(j − 1).

The variables firstMbY and firstMbInROIFld are derived as

$$firstMbY = first\_mb\_in\_roi[i][j] − (first\_mb\_in\_roi[i][j] \% FrmWidthInMbs[i]) \quad (G-381)$$

$$firstMbInROIFld = (firstMbY >> 1) + (first\_mb\_in\_roi[i][j] \% FrmWidthInMbs[i]) \quad (G-382)$$

When frame_mbs_only_flag is equal to 0 for any primary coded VCL NAL unit of the current scalable layer, (firstMbY % 2) shall be equal to 0.

For each dependency representation that contains any primary coded VCL NAL unit of the current scalable layer, the following applies:

– If field_pic_flag is equal to 0 and MbaffFrameFlag is equal to 0 for the dependency representation, the value of first_mb_in_roi[i][j] shall be equal to the syntax element first_mb_in_slice in the slice that belongs to the dependency representation and covers the top-left macroblock of the j-th IROI.

– Otherwise (field_pic_flag is equal to 1 or MbaffFrameFlag is equal to 1 for the dependency representation), the value of firstMbInROIFld shall be equal to the syntax element first_mb_in_slice in the slice that belongs to the dependency representation and covers the top-left macroblock of the j-th IROI.

roi_width_in_mbs_minus1[i][j] and **roi_height_in_mbs_minus1**[i][j] specify the size of the j-th IROI for the current scalable layer. When frame_mbs_only_flag is equal to 0 for any primary coded VCL NAL unit of the current scalable layer, (roi_height_in_mbs_minus1[i][j] + 1) % 2 shall be equal to 0.

The value of roi_width_in_mbs_minus1[i][j] shall be in the range of 0 to (FrmWidthInMbs[i] − 1 − (first_mb_in_roi[i][j] % FrmWidthInMbs[i])), inclusive. The value of roi_height_in_mbs_minus1[i][j] shall be in the range of 0 to (FrmHeightInMbs[i] − 1 − (firstMbY / FrmWidthInMbs[i])), inclusive.

With field_pic_flag being the slice header syntax element for a particular layer picture with dependency_id equal to dependency_id[i], the width and height of the j-th IROI in the layer pictures with dependency_id equal to dependency_id[i] are equal to 16 * (roi_width_in_mbs_minus1[i][j] + 1) and 16 * (roi_height_in_mbs_minus1[i][j] + 1) / (1 + field_pic_flag), respectively, in units of luma samples.

All scalable layers with the same value of dependency_id[i] for which scalability information is present in the scalability information SEI message and for which iroi_division_info_present_flag[i] is equal to 1 and iroi_grid_flag[i] is equal to 0 shall have the same values of first_mb_in_roi[i][j], roi_width_in_mbs_minus1[i][j], and roi_height_in_mbs_minus1[i][j] with j in the range of 0 to num_rois_minus1[i], inclusive.

num_directly_dependent_layers[i **]** specifies the number of the syntax elements directly_dependent_layer_id_delta_minus1[i][j] that are present for the current scalable layer. The value of num_directly_dependent_layers shall be in the range of 0 to 255, inclusive.

directly_dependent_layer_id_delta_minus1[i **][** j **]** plus 1 indicates the difference between the value of layer_id[i] for the current scalable layer and the layer identifier of a particular scalable layer, on which the current scalable layer directly depends. The value of directly_dependent_layer_id_delta_minus1[i][j] shall be in the range of 0 to layer_id[i] − 1, inclusive. The layer identifier of the particular scalable layer, on which the current scalable layer directly depends, is equal to layer_id[i] − directly_dependent_layer_id_delta_minus1[i][j] − 1. The scalability information SEI message shall contain information for a scalable layer b with layer_id[b] equal to layer_id[i] − directly_dependent_layer_id_delta_minus1[i][j] − 1 and this information shall not contain a value of layer_dependency_info_src_layer_id_delta[i] equal to 0.

Let setOfDepLayers be the set union of the representations of the scalable layers b that have layer_id[b] equal to layer_id[i] − directly_dependent_layer_id_delta_minus1[i][j] − 1, with j = 0..num_directly_dependent_layers[i] − 1. When layer_dependency_info_present_flag[i] is equal to 1, the set setOfDepLayers shall not contain any scalable layer, on which the current scalable layer does not directly or indirectly depends and the current scalable layer shall not depend on any scalable layer that is not included in the set setOfDepLayers.

layer_dependency_info_src_layer_id_delta[i] greater than 0 indicates that the current scalable layer has the same layer dependency information as the scalable layer with layer identifier equal to layer_id[i] − layer_dependency_info_src_layer_id_delta[i]. layer_dependency_info_src_layer_id_delta[i] equal to 0 specifies that the layer dependency information of the current scalable layer is not present in the scalability information SEI message. The value of layer_dependency_info_src_layer_id_delta[i] shall be in the range of 0 to layer_id[i], inclusive. When layer_dependency_info_src_layer_id_delta[i] is greater than 0, the scalability information SEI message shall contain information for a scalable layer b with layer_id[b] equal to layer_id[i] − layer_dependency_info_src_layer_id_delta[i] and this information shall not contain a value of layer_dependency_info_src_layer_id_delta[b] equal to 0.

When layer_dependency_info_present_flag[i] is equal to 0 and layer_dependency_info_src_layer_id_delta[i] is greater than 0, the set of scalable layers on which the current scalable layer depends shall be identical to the set of layers on which the scalable layer b with layer_id[b] equal to layer_id[i] − layer_dependency_info_src_layer_id_delta[i] depends.

NOTE 15 – When layer_dependency_info_src_layer_id_delta[i] equal to 0 is not present for the current scalable layer, the representation of the current scalable layer is specified by the syntax element layer_dependency_info_src_layer_id_delta[i] or by the syntax elements directly_dependent_layer_id_delta_minus1[i][j], with j = 0..num_directly_dependent_layers[i] − 1.

NOTE 16 – A change for the layer dependency information may be signalled by the presence of one or more layer dependency change SEI messages. When a scalability information SEI message specifies that a scalable layer A does not directly or indirectly depend on a scalable layer B, this relationship applies to the complete target access unit set. When a scalability information SEI message specifies that a scalable layer A does directly or indirectly depend on a scalable layer B, a following layer dependency change SEI message may indicate that this dependency does not apply for a subset of the target access unit set.

num_seq_parameter_sets[i **]** indicates the number of different sequence parameter sets that are referred to by the primary coded VCL NAL units of the current scalable layer representation. The value of num_seq_parameter_sets[i] shall be in the range of 0 to 32, inclusive.

seq_parameter_set_id_delta[i **][** j **]** indicates the smallest value of the seq_parameter_set_id of any sequence parameter set required for decoding the representation of the current scalable layer, if j is equal to 0. Otherwise (j is greater than 0), seq_parameter_set_id_delta[i][j] indicates the difference between the value of the seq_parameter_set_id of the j-th required sequence parameter set and the value of the seq_parameter_set_id of the (j − 1)-th required sequence parameter set for decoding the representation of the current scalable layer. The value of seq_parameter_set_id_delta[i][j] shall not be greater than 31. When j is greater than 0, the value of seq_parameter_set_id_delta[i][j] shall not be equal to 0. When parameter_sets_info_present_flag[i] is equal to 1, the primary coded VCL NAL units of the current scalable layer representation shall not refer to any sequence parameter set for which the value of seq_parameter_set_id is not indicated by the syntax elements seq_parameter_set_id_delta[i][j] for the current scalable layer and the syntax elements seq_parameter_set_id_delta[i][j] for the current scalable layer shall not indicate any sequence parameter set that is not referenced in any primary coded VCL NAL unit of the current scalable layer representation.

num_subset_seq_parameter_sets[i] indicates the number of different subset sequence parameter sets that are referred to by the primary coded VCL NAL units of the current scalable layer representation. The value of num_subset_seq_parameter_sets[i] shall be in the range of 0 to 32, inclusive.

subset_seq_parameter_set_id_delta[i][j] indicates the smallest value of the seq_parameter_set_id of any subset sequence parameter set required for decoding the representation of the current scalable layer, if j is equal to 0. Otherwise (j is greater than 0), subset_seq_parameter_set_id_delta[i][j] indicates the difference between the value of the seq_parameter_set_id of the j-th required subset sequence parameter set and the value of the seq_parameter_set_id of the (j − 1)-th required subset sequence parameter set for decoding the representation of the current scalable layer. The value of subset_seq_parameter_set_id_delta[i][j] shall not be greater than 31. When j is greater than 0, the value of subset_seq_parameter_set_id_delta[i][j] shall not be equal to 0. When parameter_sets_info_present_flag[i] is equal to 1, the primary coded VCL NAL units of the current scalable layer representation shall not refer to any subset sequence parameter set for which the value of seq_parameter_set_id is not indicated by the syntax elements subset_seq_parameter_set_id_delta[i][j] for the current scalable layer and the syntax elements subset_seq_parameter_set_id_delta[i][j] for the current scalable layer shall not indicate any subset sequence parameter set that is not referenced in any primary coded VCL NAL unit of the current scalable layer representation.

num_pic_parameter_sets_minus1[i] plus 1 indicates the number of different picture parameter sets that are referred to by the primary coded VCL NAL units of the current scalable layer representation. The value of num_pic_parameter_sets_minus1[i] shall be in the range of 0 to 255, inclusive.

pic_parameter_set_id_delta[i][j] indicates the smallest value of the pic_parameter_set_id of any picture parameter set required for decoding the representation of the current scalable layer, if j is equal to 0. Otherwise (j is greater than 0), pic_parameter_set_id_delta[i][j] indicates the difference between the value of the pic_parameter_set_id of the j-th required picture parameter set and the value of the pic_parameter_set_id of the (j − 1)-th required picture parameter set for decoding the representation of the current scalable layer. The value of pic_parameter_set_id_delta[i][j] shall not be greater than 255. When j is greater than 0, the value of pic_parameter_set_id_delta[i][j] shall not be equal to 0. When parameter_sets_info_present_flag[i] is equal to 1, the primary coded VCL NAL units of the current scalable layer representation shall not refer to any picture parameter set for which the value of pic_parameter_set_id is not indicated by the syntax elements pic_parameter_set_id_delta[i][j] for the current scalable layer and the syntax elements pic_parameter_set_id_delta[i][j] for the current scalable layer shall not indicate any picture parameter set that is not referenced in any primary coded VCL NAL unit of the current scalable layer representation.

parameter_sets_info_src_layer_id_delta[i] greater than 0 indicates that the values of seq_parameter_set_id of the sequence parameter sets and subset sequence parameter sets and the values of pic_parameter_set_id of the picture parameter sets that are referred to by the primary coded VCL NAL units of the current scalable layer representation are the same as those that are referred to by the primary coded VCL NAL units of the representation of the scalable layer b with the layer identifier layer_id[b] equal to layer_id[i] − parameter_sets_info_src_layer_id_delta[i]. When parameter_sets_info_src_layer_id_delta[i] is greater than 0, the scalability information SEI message shall contain information for a scalable layer b with layer_id[b] equal to layer_id[i] − parameter_sets_info_src_layer_id_delta[i] and this information shall not contain a value of parameter_sets_info_src_layer_id_delta[b] equal to 0.

parameter_sets_info_src_layer_id_delta[i] equal to 0 indicates that the values of seq_parameter_set_id of the sequence parameter sets and subset sequence parameter sets and the values of pic_parameter_set_id of the picture parameter sets that are referred to by the primary coded VCL NAL units of the current scalable layer representation are not indicated in the scalability information SEI message.

The value of parameter_sets_info_src_layer_id_delta[i] shall be in the range of 0 to layer_id[i], inclusive.

motion_vectors_over_pic_boundaries_flag[i] indicates the value of motion_vectors_over_pic_boundaries_flag, as specified in subclause E.2.1, that applies to the current scalable layer representation.

max_bytes_per_pic_denom[i] indicates the value of max_bytes_per_pic_denom, as specified in subclause E.2.1, that applies to the current scalable layer representation.

max_bits_per_mb_denom[i] indicates the value of max_bits_per_mb_denom, as specified in subclause E.2.1, that applies to the current scalable layer representation.

log2_max_mv_length_horizontal[i] and **log2_max_mv_length_vertical[i]** indicate the values of log2_max_mv_length_horizontal and log2_max_mv_length_vertical, as specified in subclause E.2.1, that apply to the current scalable layer representation.

NOTE 17 – The maximum absolute value of a decoded vertical or horizontal motion vector component is also constrained by profile and level limits as specified in Annex A and subclause G.10.

max_num_reorder_frames[i] indicates the value of max_num_reorder_frames, as specified in subclause E.2.1, that applies to the current scalable layer representation.

max_dec_frame_buffering[i] indicates the value of max_dec_frame_buffering, as specified in subclause E.2.1, that applies to the current scalable layer representation.

conversion_type_idc[i] equal to 0 indicates that tcoeff_level_prediction_flag is equal to 1 for all primary coded slices of the current scalable layer representation excluding those having no_inter_layer_pred_flag equal to 1 and that the information specified by the syntax elements rewriting_profile_level_idc[i][j], rewriting_avg_bitrate[i][j], and rewriting_max_bitrate[i][j], when present, is correct, though the method for converting the current scalable layer representation into an alternative set of access units that conforms to one or more of the profiles specified in Annex A and gives exactly the same decoding result as the current scalable layer representation is unspecified.

conversion_type_idc[i] equal to 1 indicates that the slice_header_restriction_flag in the subset sequence parameter sets referred to by the primary coded VCL NAL units of the current scalable layer is equal to 1, that slice_skip_flag is equal to 1 for all primary coded VCL NAL units with no_inter_layer_pred_flag equal to 0 in the current scalable layer representation, and that the alternative set of access units obtained by applying the following operations in sequential order to the current scalable layer representation conforms to one or more of the profiles specified in Annex A:

1. For all picture parameter set NAL units referred to by NAL units with nal_unit_type equal to 1 or 5, change the value of seq_parameter_set_id to be equal to the value of seq_parameter_set_id in a subset sequence parameter set NAL unit with profile_idc equal to 83 or 86 that is referred to by slices with nal_unit_type equal to 20 of the current scalable layer.
2. Remove all NAL units with nal_unit_type equal to 20 and slice_skip_flag equal to 1.
3. Remove all NAL units with nal_unit_type equal to 14.
4. Remove all redundant coded VCL NAL units.
5. In each access unit, remove all VCL NAL units with DQId less than DQIdMax, with DQIdMax being the maximum value of DQId in the primary coded slices of the access unit after removing the NAL units with nal_unit_type equal to 20 and slice_skip_flag equal to 1.
6. Remove the NAL unit header SVC extension from NAL units with nal_unit_type equal to 20.
7. For NAL units with nal_unit_type equal to 20 and idr_flag equal to 1, set nal_unit_type equal to 5.
8. For NAL units with nal_unit_type equal to 20 and idr_flag equal to 0, set nal_unit_type equal to 1.
9. Remove all SEI NAL units.
10. Remove all NAL units with nal_unit_type equal to 7.
11. For all NAL units with nal_unit_type equal to 15, set nal_unit_type equal to 7, remove all the syntax elements after the syntax structure seq_parameter_set_data( ) and before the rbsp_trailing_bits( ) syntax structure, replace the three bytes starting from profile_idc as specified by rewriting_profile_level_idc[i][entropy_coding_mode_flag], when present, and change RBSP trailing bits appropriately.

conversion_type_idc[i] equal to 2 indicates that slice_header_restriction_flag in the subset sequence parameter sets referred to by the primary coded VCL NAL units of the current scalable layer is equal to 1, that no_inter_layer_pred_flag is equal to 1 in all primary coded VCL NAL units of the current scalable layer, and that the alternative set of access units obtained by applying the following operations in sequential order to the current scalable layer representation conforms to one or more of the profiles specified in Annex A:

1. Remove all NAL units with nal_unit_type equal to 14.
2. Remove all redundant coded VCL NAL units.
3. In each access unit, remove all VCL NAL units with DQId less than DQIdMax.
4. Remove the NAL unit header SVC extension from NAL units with nal_unit_type equal to 20.
5. For NAL units with nal_unit_type equal to 20 and idr_flag equal to 1, set nal_unit_type equal to 5.
6. For NAL units with nal_unit_type equal to 20 and idr_flag equal to 0, set nal_unit_type equal to 1.
7. Remove all SEI NAL units.
8. Remove all NAL units with nal_unit_type equal to 7.
9. For all NAL units with nal_unit_type equal to 15, set nal_unit_type equal to 7, remove all the syntax elements after the syntax structure seq_parameter_set_data( ) and before the rbsp_trailing_bits( ) syntax structure, replace the three bytes starting from profile_idc as specified by

rewriting_profile_level_idc[i][entropy_coding_mode_flag], when present, and change RBSP trailing bits appropriately.

The value of conversion_type_idc[i] shall be in the range of 0 to 2, inclusive.

For the following syntax elements rewriting_info_flag[i][j], rewriting_profile_level_idc[i [j], rewriting_avg_bitrate[i][j], and rewriting_max_bitrate[i][j], the variable j specifies the value of entropy_coding_mode_flag for all picture parameter set NAL units that are referenced in the VCL NAL units of the alternative set of access units obtained by converting the current scalable layer representation, with values for j equal to 0 or 1 indicating use of the CAVLC or CABAC entropy coding methods, respectively.

NOTE 18 – It might be possible to convert the current scalable layer representation into two alternative sets of access units that conform to one or more of the profiles specified in Annex A, with one of these sets having entropy_coding_mode_flag equal to 0 and the other set having entropy_coding_mode_flag equal to 1 in all picture parameter set NAL units that are referenced in the VCL NAL units of the alternative set of access units.

rewriting_info_flag[i][j] equal to 1 specifies that information about the alternative set of access units obtained by converting the current scalable layer representation is present in the scalability information SEI message. rewriting_info_flag[i][j] equal to 0 specifies that information about the alternative set of access units is not present in the scalability information SEI message. When rewriting_info_flag[i][j] is equal to 1, it is asserted that the information signalled by the syntax elements rewriting_profile_level_idc[i [j], rewriting_avg_bitrate[i][j], and rewriting_max_bitrate[i][j] is correct, though, when conversion_type_idc[i] is equal to 0 or the value of entropy_coding_mode_flag is modified, the method for constructing the alternative set of access units is unspecified.

rewriting_profile_level_idc[i][j] indicates the conformance point of the alternative set of access units for the current scalable layer representation after conversion. rewriting_profile_level_idc[i] is the exact copy of the three bytes consist of profile_idc, constraint_set0_flag, constraint_set1_flag, constraint_set2_flag, constraint_set3_flag, constraint_set4_flag, constraint_set5_flag, reserved_zero_2bits, and level_idc, as if these syntax elements were used to specify the profile and level conformance of the alternative set of access units obtained by converting the scalable layer representation.

rewriting_avg_bitrate[i **]**[j] indicates the average bit rate of the alternative set of access units obtained by converting the representation of the current scalable layer. The average bit rate of the alternative set of access units in bits per second is given by BitRateBPS(rewriting_avg_bitrate[i][j]) with the function BitRateBPS( ) being specified in Equation G-370. The average bit rate is derived according to the access unit removal time specified in Annex C of the Recommendation | International Standard.

rewriting_max_bitrate[i **]**[j] indicates an upper bound for the bit rate of the alternative set of access units obtained by converting the representation of the current scalable layer, in any one-second time window of access unit removal time as specified in Annex C. The upper bound for the bit rate of the alternative set of access units in bits per second is given by BitRateBPS(rewriting_max_bitrate[i][j]) with the function BitRateBPS( ) being specified in Equation G-370.

For the following specification, the terms priority layer, dependency layer, and priority layer representation are defined as follows. A priority layer consists of the set of primary coded VCL NAL units, inside the target access unit set, that are associated with a particular value of dependency_id and a value of alt_priority_id[i], as specified in subclause G.13.2.4, that is less than or equal to a particular priority identifier pId and the set of associated non-VCL NAL units. A priority layer is associated with a particular value of dependency_id and a particular priority layer identifier pId. When present in the target access unit, the following NAL units are associated non-VCL NAL units for a priority layer:

– sequence parameter set, subset sequence parameter set, and picture parameter set NAL units that are referenced in the VCL NAL units of the priority layer (via the syntax element pic_parameter_set_id),

– sequence parameter set extension NAL units that are associated with a sequence parameter set NAL unit referenced in the VCL NAL units of the priority layer,

– filler data NAL units that belong to an access unit containing VCL NAL units of the priority layer and are associated with the same values of dependency_id and quality_id as the VCL NAL units of the priority layer in the same access unit,

– SEI NAL units containing SEI messages, with payloadType not equal to 24, 28, or 29, that apply to subsets of the bitstream that contain one or more VCL NAL units of the priority layer,

– access unit delimiter, end of sequence, and end of stream NAL units that are present in access units that contain VCL NAL units of the priority layer.

The set of NAL units that represents the set union of all priority layers that are associated with the same value of dependency_id is referred to as dependency layer. A dependency layer is associated with a particular value of dependency_id.

A priority layer A is directly dependent on a priority layer B when any VCL NAL unit of the priority layer A references data of any VCL NAL unit of the priority layer B through inter prediction or inter-layer prediction as specified in the decoding process in subclause G.8. A priority layer A is indirectly dependent on a priority layer B when the priority layer A is not directly dependent on the priority layer B but there exists a set of n (with n being greater than 0) priority layers $\{C_0, .., C_{n-1}\}$ with the following properties: The priority layer A is directly dependent on the priority layer C_0, each priority layer C_i with i in the range of 0 to n − 2, inclusive, is directly dependent on the priority layer C_{i+1}, and the priority layer C_{n-1} is directly dependent on the priority layer B.

The representation of a particular priority layer is the set of NAL units that represents the set union of the particular priority layer and all priority layers on which the particular priority layer directly or indirectly depends. The representation of a priority layer is also referred to as priority layer representation. In the following specification of this subclause, the terms representation of a priority layer and priority layer representation are also used for referring to the access unit set that can be constructed from the NAL units of the priority layer representation. A priority layer representation can be decoded independently of all NAL units that do not belong to the priority layer representation.

pr_num_dIds_minus1 plus 1 specifies the number of dependency layers for which the priority layer characteristic information as specified by the following syntax elements is present in the scalability information SEI message. The value of pr_num_dIds_minus1 shall be in the range of 0 to 7, inclusive.

pr_dependency_id[i **]** specifies the value of dependency_id of the dependency layer for which the priority layer characteristic information is signalled by the following syntax elements. When i is greater than 0, the value of pr_dependency_id[i] shall not be equal to any of the values of pr_dependency_id[j] with j = 0..(i − 1).

pr_num_minus1[i **]** plus 1 specifies the number of priority layers with dependency_id equal to pr_dependency_id[i] for which priority layer characteristic information as specified by the following syntax elements is present in the scalability information SEI message. The value of pr_num_minus1[i] shall be in the range of 0 to 63, inclusive.

pr_id[i **][** j **]** specifies the priority identifier pId for a priority layer with dependency_id equal to pr_dependency_id[i]. The value of pr_id[i][j] shall be in the range of 0 to 63, inclusive. The target access unit set shall contain one or more primary coded VCL NAL units that are associated with dependency_id equal to pr_dependency_id[i] and alt_priority_id[i] equal to pr_id[i][j], where the value of alt_priority_id[i] that is associated with a primary coded VCL NAL unit is specified in subclause G.13.2.4. When j is greater than 0, the value of pr_id[i][j] shall not be equal to any of the values of pr_id[i][k] with k = 0..(j − 1).

For the following specification inside the subclause, the priority layer with dependency_id equal to the current value of pr_dependency_id[i] and the priority layer identifier pId equal to the current value of pr_id[i][j] is referred to as the current priority layer and the representation of the current priority layer is referred to as the current priority layer representation.

pr_profile_level_idc[i **][** j **]** indicates the conformance point of the current priority layer representation. pr_profile_level_idc[i] is the exact copy of the three bytes consisting of profile_idc, constraint_set0_flag, constraint_set1_flag, constraint_set2_flag, constraint_set3_flag, constraint_set4_flag, constraint_set5_flag, reserved_zero_2bits, and level_idc, as if these syntax elements were used to specify the profile and level conformance of the current priority layer representation.

pr_avg_bitrate[i **][** j **]** indicates the average bit rate of the current priority layer representation. The average bit rate of the current priority layer representation in bits per second is given by BitRateBPS(pr_avg_bitrate[i][j]) with the function BitRateBPS( ) being specified in Equation G-370. The average bit rate is derived according to the access unit removal time specified in Annex C of this Recommendation | International Standard. In the following, bTotal is the number of bits in all NAL units of the current priority layer representation, t_1 is the removal time (in seconds) of the access unit associated with the scalability information SEI message, and t_2 is the removal time (in seconds) of the last access unit (in decoding order) of the target access unit set.

With x specifying the value of pr_avg_bitrate[i], the following applies:

– If t_1 is not equal to t_2, the condition specified in Equation G-371 shall be fulfilled.

– Otherwise (t_1 is equal to t_2), the condition specified in Equation G-372 shall be fulfilled.

pr_max_bitrate[i **][** j **]** indicates an upper bound for the bit rate of the current priority layer representation in any one-second time window of access unit removal time as specified in Annex C. The upper bound for the bit rate of the current priority layer representation in bits per second is given by BitRateBPS(pr_max_bitrate[i][j]) with the function BitRateBPS( ) being specified in Equation G-370. The bit rate values are derived according to the access unit removal time specified in Annex C of this Recommendation | International Standard. In the following, t_1 is any point in time (in seconds), t_2 is set equal to $t_1 + 1$, and bTotal is the number of bits in all NAL units of the current priority layer representation that belong to access units with a removal time greater than or equal to t_1 and less than t_2. With x specifying the value of pr_max_bitrate[i][j], the condition specified in Equation G-373 shall be obeyed.

priority_id_setting_uri[PriorityIdSettingUriIdx **]** is the PriorityIdSettingUriIdx-th byte of a null-terminated string encoded in UTF-8 characters, specifying the universal resource identifier (URI) of the description of the method used to calculate the priority_id values in the NAL unit headers for the target access unit set.

G.13.2.2 Sub-picture scalable layer SEI message semantics

The sub-picture scalable SEI message provides a mechanism for associating a slice group set indicated in a motion-constrained slice group set SEI message with a sub-picture scalable layer.

In the following specification of this subclause, the terms scalable layer, sub-picture scalable layer, and primary coded VCL NAL unit are used as specified in subclause G.13.2.1.

A sub-picture scalable layer SEI message shall not be succeeded, in decoding order, by a scalability information SEI message inside the same access unit.

When a sub-picture scalable SEI message is present, the following applies:

– If the sub-picture scalable layer SEI message is included in a scalable nesting SEI message, a motion-constrained slice group set SEI message, which is also referred to as the associated motion-constrained slice group set SEI message, shall be present in the same scalable nesting SEI message and it shall immediately precede the sub-picture scalable layer SEI message in decoding order. The scalable nesting SEI message that contains the sub-picture scalable layer SEI message shall contain num_layer_representations_minus1 equal to 0 and sei_quality_id[0] equal to 0. The variable depId is set equal to the value of sei_dependency_id[0] that is present in the scalable nesting SEI message containing the sub-picture scalable layer SEI message.

– Otherwise (the sub-picture scalable layer SEI message is not included in a scalable nesting SEI message), the sub-picture scalable layer SEI message shall be the first SEI payload in an SEI NAL unit and the NAL unit immediately preceding the SEI NAL unit containing the sub-picture scalable layer SEI message shall be an SEI NAL unit that contains a motion-constrained slice group set SEI message, which is also referred to as associated motion-constrained slice group set SEI message, as last SEI payload. The variable depId is set equal to 0.

The slice group set identified by the associated motion-constrained slice group set SEI message is referred to as the associated slice group set of the sub-picture scalable layer SEI message.

The access unit associated with the sub-picture scalable layer SEI message shall not contain any primary coded VCL NAL unit that has dependency_id equal to depId and IdrPicFlag equal to 0. The set of access units consisting of the access unit associated with the sub-picture scalable layer SEI message and all succeeding access units in decoding order until, but excluding, the next access unit that contains any primary coded VCL NAL unit with dependency_id equal to depId and IdrPicFlag equal to 1 or that does not contain any primary coded VCL NAL units with IdrPicFlag equal to 0 (if present) or the end of the bitstream (otherwise) is referred to as the target access unit set. The sub-picture scalable layer SEI message applies to the target access unit set.

NOTE – The set of primary coded pictures in the target access unit set for a sub-picture scalable layer SEI message is identical to the target picture set for the associated motion-constrained slice group set SEI message.

layer_id indicates, when the access unit containing the sub-picture scalable layer SEI message belongs to the target access unit set of a scalability information SEI message, the layer identifier of the sub-picture scalable layer to which the coded slice NAL units in the associated slice group set belong. The value of layer_id shall be in the range of 0 to 2047, inclusive.

The access unit containing the sub-picture scalable layer SEI message may or may not belong to the target access unit set of a scalability information SEI message. When the access unit containing the sub-picture scalable layer SEI message belongs to the target access unit set of a scalability information SEI message, the corresponding scalability information SEI message may or may not contain information for a scalable layer i with layer identifier layer_id[i] equal to layer_id. When the access unit containing the sub-picture scalable layer SEI message belongs to the target access unit set of a scalability information SEI message and the corresponding scalability information SEI message contains information for a scalable layer i with layer identifier layer_id[i] equal to layer_id, which is referred to as the current scalable layer in the following, the following applies. The information for the current scalable layer in the scalability SEI shall contain sub_pic_layer_flag[i] equal to 1. The sub-picture area for the current scalable layer i, which is specified by the syntax elements horizontal_offset[i], vertical_offset[i], region_width[i], and region_height[i] in the scalability information SEI message, shall be identical to the area specified by the associated slice group set.

G.13.2.3 Non-required layer representation SEI message semantics

The non-required layer representation SEI message provides a mechanism for indicating which layer representations of the current primary coded picture are not required for decoding dependency representations with a particular value of dependency_id of the current primary coded picture and succeeding primary coded pictures, in decoding order.

The non-required layer representation SEI message shall not be included in a scalable nesting SEI message.

num_info_entries_minus1 plus 1 specifies the number of dependency_id values for which non-required layer representations are indicated in the SEI message. The value of num_info_entries_minus1 shall be in the range of 0 to 7, inclusive.

entry_dependency_id[i **]** specifies the dependency_id value for which non-required layer representations are indicated by the following syntax elements. The instances of entry_dependency_id[i] shall appear in increasing order of their values.

The dependency representation of the primary coded picture with dependency_id equal to entry_dependency_id[i] is referred to as the target dependency representation.

The target dependency representation may or may not be present in the access unit.

num_non_required_layer_reps_minus1[i **]** plus 1 specifies the number of non-required layer representations for the target dependency representation that are indicated in the SEI message. The value of num_non_required_layer_reps_minus1[i] shall be in the range of 0 to 127, inclusive.

non_required_layer_rep_dependency_id[i **][** j **]** indicates the value of dependency_id of the j-th non-required layer representation for the target dependency representation.

non_required_layer_rep_quality_id[i **][** j **]** indicates the value of quality_id of the j-th non-required layer representation for the target dependency representation.

The i-th non-required layer representation for the target dependency representation is the layer representation of the primary coded picture that has dependency_id equal to non_required_layer_rep_dependency_id[i][j] and quality_id equal to non_required_layer_rep_quality_id[i][j]. A non-required layer representation for the target dependency representation is not required for decoding the target dependency representation and any dependency representation with dependency_id equal to entry_dependency_id[i] of primary coded pictures that follow the current primary picture in decoding order.

When DependencyIdMax is equal to entry_dependency_id[i], the VCL NAL units of the non-required layer representations shall not be referenced through inter or inter-layer prediction in the decoding process as specified in subclause G.8.

NOTE – In addition to the i-th non-required layer representation for the target dependency representation, those layer representations that have dependency_id equal to non_required_layer_rep_dependency_id[i][j] and quality_id greater than non_required_layer_rep_quality_id[i][j] are also non-required layer representations for the target dependency representation.

The i-th non-required layer representation may or may not be present in the access unit.

G.13.2.4 Priority layer information SEI message semantics

The priority layer information SEI message provides a mechanism for signalling alternative priority_id values for VCL NAL units of the primary coded picture. The alternative values for priority_id indicate priority layers.

The priority layer information SEI message shall not be included in a scalable nesting SEI message.

pr_dependency_id specifies the value of dependency_id for the VCL NAL units for which alternative values for priority_id are indicated.

num_priority_ids specifies the number of layer representations with dependency_id equal to pr_dependency_id for which alternative values of priority_id are indicated.

alt_priority_id[i **]** specifies the alternative value for priority_id for the VCL NAL units of the primary coded picture that have dependency_id equal to pr_dependency_id and quality_id equal to i.

The layer representation of the primary coded picture with dependency_id equal to pr_dependency_id and quality_id equal to i may or may not be present in the access unit.

G.13.2.5 Layers not present SEI message semantics

The layers not present SEI message provides a mechanism for signalling that NAL units of particular scalable layers indicated by the preceding scalability information SEI message are not present in a particular set of access units.

In the following specification of this subclause, the terms scalable layer and primary coded VCL NAL unit are used as specified in subclause G.13.2.1.

A layers not present SEI message shall not be included in a scalable nesting SEI message.

A layers not present SEI message shall not be present in an access unit that does not belong to the target access unit set of any scalability information SEI message. A layers not present SEI message shall not be succeeded, in decoding order, by a scalability information SEI message inside the same access unit. The set of access units consisting of the access unit associated with the layers not present SEI message and all succeeding access units in decoding order until, but excluding, the next access unit that contains a layers not present SEI message or that does not contain any primary coded VCL NAL units with IdrPicFlag equal to 0 (if present), or the end of the bitstream (otherwise) is referred to as the target access unit set. The layers not present SEI message applies to the target access unit set.

A layers not present SEI message refers to the most recent scalability information SEI message in decoding order. Each scalable layer that is referred to in this subclause is a scalable layer indicated in the most recent scalability information SEI message in decoding order. Each layer identifier for a scalable layer that is referred to in this subclause is a layer identifier for a scalable layer indicated in the most recent scalability information SEI message in decoding order.

NOTE 1 – Layers not present SEI messages do not have a cumulative effect.

num_layers specifies the number of syntax elements layer_id[i] that are present in the layers not present SEI message. The value of num_layers shall be in the range of 0 to 2047, inclusive.

layer_id[i **]** indicates the layer identifier of a scalable layer for which no VCL NAL units are present in the target access unit set. The value of layer_id[i] shall be in the range of 0 to 2047, inclusive. The value of layer_id[i] shall be equal to one of the values of layer_id[i] in the most recent scalability information SEI message. The target access unit set shall not contain any VCL NAL unit of the scalable layer having a layer identifier equal to layer_id[i]. When i is greater than 0, the value of layer_id[i] shall not be equal to any of the values of layer_id[j] with j = 0..(i − 1).

NOTE 2 – When an application removes NAL units from a scalable bitstream, e.g. in order to adapt the bitstream to a transmission channel or the capabilities of a receiving device, and keeps the present layers not present SEI messages, it might need to modify the content of some of the layers not present SEI messages and remove some other layers not present SEI messages in order to obtain a bitstream conforming to this Recommendation | International Standard.

G.13.2.6 Layer dependency change SEI message semantics

The layer dependency change SEI message provides a mechanism for signalling that the interdependencies between particular scalable layers indicated by the preceding scalability information SEI message are changed for a particular set of access units.

In the following specification of this subclause, the terms scalable layer, representation of a scalable layer, scalable layer representation, and primary coded VCL NAL unit are used as specified in subclause G.13.2.1.

A layer dependency change SEI message shall not be included in a scalable nesting SEI message.

A layer dependency change SEI message shall not be present in an access unit that does not belong to the target access unit set of any scalability information SEI message. A layer dependency change SEI message shall not be succeeded, in decoding order, by a scalability information SEI message or a layers not present SEI message inside the same access unit. The set of access units consisting of the access unit associated with the layer dependency change SEI message and all succeeding access units in decoding order until, but excluding, the next access unit that contains a layer dependency change SEI message or a layers not present SEI message or that does not contain any primary coded VCL NAL units with IdrPicFlag equal to 0 (if present), or the end of the bitstream (otherwise) is referred to as the target access unit set. The layer dependency change SEI message applies to the target access unit set.

A layer dependency change SEI message refers to the most recent scalability information SEI message in decoding order. Each scalable layer that is referred to in this subclause is a scalable layer indicated in the most recent scalability information SEI message in decoding order. Each layer identifier for a scalable layer that is referred to in this subclause is a layer identifier for a scalable layer indicated in the most recent scalability information SEI message in decoding order.

NOTE 1 – Layer dependency change SEI messages do not have a cumulative effect.

The presence of the layer dependency change SEI message specifies the following. For a scalable layer with a layer identifier equal to any value of layer_id[i] present in the layer dependency change SEI message, the layer dependency relationship is changed for the target access unit set relative to the layer dependency relationship specified by the most recent scalability information SEI message in decoding order. For a scalable layer with a layer identifier not equal to any value of layer_id[i] present in the layer dependency change SEI message, the layer dependency relationship remains the same as the one specified in the most recent scalability information SEI message in decoding order.

When, according to the layer dependency information indicated in the most recent scalability information SEI message in decoding order, a scalable layer A does not directly or indirectly depend on another scalable layer B, the layer dependency change SEI message shall not specify that the scalable layer A directly or indirectly depends on the scalable layer B.

When a scalable layer is considered to directly or indirectly depend on another scalable layer is specified in subclause G.13.2.1, with the target access unit set being the target access unit set for the layer dependency change SEI message.

num_layers_minus1 plus 1 specifies the number of scalable layers for which a layer dependency information change relative to the most recent scalability information SEI message, in decoding order, is indicated in the layer dependency change SEI message. The value of num_layers_minus1 is in the range of 0 to 2047, inclusive.

layer_id[i] indicates the layer identifier of the scalable layer for which a layer dependency information change is indicated by the following syntax elements. The value of layer_id[i] shall be in the range of 0 to 2047, inclusive. The value of layer_id[i] shall be equal to one of the values of layer_id[i] in the most recent scalability information SEI message. When i is greater than 0, the value of layer_id[i] shall not be equal to any of the values of layer_id[j] with j = 0..(i − 1).

NOTE 2 – When an application removes NAL units from a scalable bitstream, e.g. in order to adapt the bitstream to a transmission channel or the capabilities of a receiving device, and keeps the present layer dependency change SEI messages, it might need to modify the content of some of the layer dependency change SEI messages and remove some other layer dependency change SEI messages in order to obtain a bitstream conforming to this Recommendation | International Standard.

For the following specification of this subclause, the scalable layer with layer identifier equal to the current value of layer_id[i] is referred to as the current scalable layer and the representation of the current scalable layer is referred to as current scalable layer representation.

layer_dependency_info_present_flag[i] equal to 1 specifies that one or more syntax elements dependent_layer_id_delta_minus1[i][j] indicating the layer dependency information for the current scalable layer are present in the layer dependency change SEI message. layer_dependency_info_present_flag[i] equal to 0 specifies that the syntax element layer_dependency_info_src_layer_id_delta_minus1[i] for the current scalable layer is present in the layer dependency change SEI message.

num_directly_dependent_layers[i] specifies the number of the syntax elements directly_dependent_layer_id_delta_minus1[i][j] that are present for the current scalable layer. The value of num_directly_dependent_layers shall be in the range of 0 to 255, inclusive.

directly_dependent_layer_id_delta_minus1[i][j] plus 1 indicates the difference between the value of layer_id[i] for the current scalable layer and the layer identifier of a particular scalable layer, on which the current scalable layer directly depends. The value of directly_dependent_layer_id_delta_minus1[i][j] shall be in the range of 0 to layer_id[i] − 1, inclusive. The layer identifier of the particular scalable layer, on which the current scalable layer directly depends, is equal to layer_id[i] − directly_dependent_layer_id_delta_minus1 − 1. The most recent scalability information SEI message in decoding order shall contain information for a scalable layer b with layer_id[b] equal to layer_id[i] − directly_dependent_layer_id_delta_minus1[i][j] − 1 and this information shall not contain a value of layer_dependency_info_src_layer_id_delta[i] equal to 0.

Let setOfDepLayers be the set union of the representations of the scalable layers b that have layer_id[b] equal to layer_id[i] − directly_dependent_layer_id_delta_minus1[i][j] − 1, with j = 0..num_directly_dependent_layers[i] − 1. When layer_dependency_info_present_flag[i] is equal to 1, the set setOfDepLayers shall not contain any scalable layer, on which the current scalable layer does not directly or indirectly depends inside the target access unit set and the current scalable layer shall not depend on any scalable layer, inside the target access unit set, that is not included in the set setOfDepLayers.

layer_dependency_info_src_layer_id_delta_minus1[i] indicates that the current scalable layer has the same layer dependency information as the scalable layer with layer identifier equal to layer_id[i] − layer_dependency_info_src_layer_id_delta_minus1[i] − 1. The value of layer_dependency_info_src_layer_id_delta_minus1[i] shall be in the range of 0 to layer_id[i] − 1, inclusive. The most recent scalability information SEI message in decoding order shall contain information for a scalable layer b with layer_id[b] equal to layer_id[i] − layer_dependency_info_src_layer_id_delta_minus1[i] − 1 and this information shall not contain a value of layer_dependency_info_src_layer_id_delta[b] equal to 0.

When layer_dependency_info_present_flag[i] is equal to 0, the set of scalable layers on which the current scalable layer depends inside the target access unit set shall be identical to the set of layers on which the scalable layer b with layer_id[b] equal to layer_id[i] − layer_dependency_info_src_layer_id_delta_minus1[i] − 1 depends inside the target access unit set.

G.13.2.7 Scalable nesting SEI message semantics

The scalable nesting SEI message provides a mechanism for associating SEI messages with subsets of a bitstream.

A scalable nesting SEI message shall contain one or more SEI messages with payloadType not equal to 30 and it shall not contain any SEI message with payloadType equal to 30. An SEI message contained in a scalable nesting SEI message is referred to as a nested SEI message. An SEI message not contained in a scalable nesting SEI message is referred to as a non-nested SEI message. The scope to which the nested SEI message applies is indicated by the syntax elements all_layer_representations_in_au_flag, num_layer_representations_minus1, sei_dependency_id[i], sei_quality_id[i], and sei_temporal_id, when present.

A buffering period SEI message and an SEI message of any other type shall not be nested in the same scalable nesting SEI message. A picture timing SEI message and an SEI message of any other type shall not be nested in the same scalable nesting SEI message.

all_layer_representations_in_au_flag equal to 1 specifies that the nested SEI message applies to all layer representations of the access unit. all_layer_representations_in_au_flag equal to 0 specifies that the scope of the nested SEI message is specified by the syntax elements num_layer_representations_minus1, sei_dependency_id[i], sei_quality_id[i], and sei_temporal_id.

num_layer_representations_minus1 plus 1 specifies, when num_layer_representations_minus1 is present, the number of syntax element pairs sei_dependency_id[i] and sei_quality_id[i] that are present in the scalable nesting SEI message. When num_layer_representations_minus1 is not present, it shall be inferred to be equal to (numSVCLayers − 1) with numSVCLayers being the number of layer representations that are present in the primary coded picture of the access unit. The value of num_layer_representations_minus1 shall be in the range of 0 to 127, inclusive.

sei_dependency_id[i] and **sei_quality_id**[i] indicate the dependency_id and the quality_id values, respectively, of the layer representations to which the nested SEI message applies. The access unit may or may not contain layer representations with dependency_id equal to sei_dependency_id[i] and quality_id equal to sei_quality_id[i]. When i is greater than 0, the value of (16 * sei_dependency_id[i] + sei_quality_id[i]) shall not be equal to any of the values of (16 * sei_dependency_id[j] + sei_quality_id[j]) with j = 0..(i − 1).

When num_layer_representations_minus1 is not present, the values of sei_dependency_id[i] and sei_quality_id[i] for i in the range of 0 to num_layer_representations_minus1 (with num_layer_representations_minus1 being the inferred value), inclusive, shall be inferred as specified in the following:

1. Let setDQId be the set of the values DQId for all layer representations that are present in the primary coded picture of the access unit.
2. For i proceeding from 0 to num_layer_representations_minus1, inclusive, the following applies:
   a. sei_dependency_id[i] and sei_quality_id[i] are inferred to be equal to (minDQId >> 4) and (minDQId & 15), respectively, with minDQId being the smallest value (smallest value of DQId) in the set setDQId.
   b. The smallest value (smallest value of DQId) of the set setDQId is removed from setDQId and thus the number of elements in the set setDQId is decreased by 1.

sei_temporal_id indicates the temporal_id value of the bitstream subset to which the nested SEI message applies. When sei_temporal_id is not present, it shall be inferred to be equal to temporal_id of the access unit.

When the nested SEI message is a buffering period SEI message or a picture timing SEI message (i.e., payloadType is equal to 0 or 1 for the nested SEI message), sei_temporal_id indicates the bitstream subset for which the nested buffering period SEI message or picture timing SEI message applies. For a buffering period SEI message or picture timing SEI message that is nested in a scalable nesting SEI message, sei_dependency_id[i], sei_quality_id[i], and sei_temporal_id specify the greatest values of dependency_id, quality_id, and temporal_id, respectively, of the bitstream subsets to which the nested buffering period SEI message or picture timing SEI message applies. The bitstream may or may not contain access units with temporal_id equal to sei_temporal_id.

When the scalable nesting SEI message contains one or more SEI messages with payloadType not equal to 0 or 1, sei_temporal_id shall be equal to the value of temporal_id for the access unit associated with the scalable nesting SEI message. For an nested SEI message with payloadType not equal to 0 or 1, the values of sei_dependency_id[i], sei_quality_id[i], and sei_temporal_id, present in or inferred for the associated scalable nesting SEI message, indicate the values of dependency_id, quality_id, and temporal_id, respectively, of the VCL NAL units to which the nested SEI message applies.

sei_nesting_zero_bit shall be equal to 0.

G.13.2.8 Base layer temporal HRD SEI message semantics

The base layer temporal HRD SEI message provides HRD parameters for subsets of the base layer bitstream.

The base layer temporal HRD SEI message shall not be included in a scalable nesting SEI message. The base layer temporal HRD SEI message shall not be present in access units that do not contain VCL NAL units of the primary coded picture with nal_unit_type equal to 5.

When present, this SEI message applies to the target access unit set that consists of the current access unit and all subsequent access units in decoding order until, but excluding, the next access unit containing a NAL unit of the primary coded picture with nal_unit_type equal to 5 (if present) or the end of the bitstream (otherwise).

num_of_temporal_layers_in_base_layer_minus1 plus 1 specifies the number of bitstream subsets inside the target access unit set for which the following syntax elements are specified in the base layer temporal HRD SEI message. The value of num_of_temporal_layers_in_base_layer_minus1 shall be in the range of 0 to 7, inclusive.

sei_temporal_id[i **]** specifies the temporal_id value of the i-th bitstream subset. When i is greater than 0, the value of sei_temporal_id[i] shall not be equal to any of the values of sei_temporal_id[j] with j = 0..(i − 1).

Access units with temporal_id equal to sei_temporal_id[i] may or may not be present in the target access unit set. When access units with temporal_id equal to sei_temporal_id[i] are not present in the target access unit set, the i-th bitstream subset is considered as not existing.

When access units with temporal_id equal to sei_temporal_id[i] are present in the target access unit set, the i-th bitstream subset is the bitstream subset that is obtained by invoking the bitstream extraction process as specified in subclause G.8.8.1 for the target access unit set with tIdTarget equal to sei_temporal_id[i], dIdTarget equal to 0, and qIdTarget equal to 0 as the inputs.

sei_timing_info_present_flag[i **]** equal to 1 specifies that sei_num_units_in_tick[i], sei_time_scale[i], and sei_fixed_frame_rate_flag[i] are present in the base layer temporal HRD SEI message. sei_timing_info_present_flag[i] equal to 0 specifies that sei_num_units_in_tick[i], sei_time_scale[i], and sei_fixed_frame_rate_flag[i] are not present in the base layer temporal HRD SEI message.

The following syntax elements for the i-th bitstream subset are specified using references to Annex E. For these syntax elements the same semantics and constraints as the ones specified in Annex E apply, as if these syntax elements sei_num_units_in_tick[i], sei_time_scale[i], sei_fixed_frame_rate_flag[i], sei_nal_hrd_parameters_present_flag[i], sei_vcl_hrd_parameters_present_flag[i], sei_low_delay_hrd_flag[i], and sei_pic_struct_present_flag[i] were present as num_units_in_tick, time_scale, fixed_frame_rate_flag, nal_hrd_parameters_present_flag, vcl_hrd_parameters_present_flag, low_delay_hrd_flag, and pic_struct_present_flag, respectively, in the VUI parameters of the active SVC sequence parameter sets for the i−th bitstream subset.

The parameters for the i-th bitstream subset that are coded in the base layer temporal HRD SEI message shall be correct, as if these parameters are used for conformance checking (as specified in Annex C) of the i-th bitstream subset.

sei_num_units_in_tick[i **]** indicates the value of num_units_in_tick, as specified in subclause E.2.1, that applies to the i-th bitstream subset.

sei_time_scale[i **]** indicates the value of time_scale, as specified in subclause E.2.1, that applies to the i-th bitstream subset.

sei_fixed_frame_rate_flag[i **]** indicates the value of fixed_frame_rate_flag, as specified in subclause E.2.1, that applies to the i-th bitstream subset.

sei_nal_hrd_parameters_present_flag[i **]** indicates the value of nal_hrd_parameters_present_flag, as specified in subclause E.2.1, that applies to the i-th bitstream subset. When sei_nal_hrd_parameters_present_flag[i] is equal to 1, the NAL HRD parameters that apply to the i-th bitstream subset immediately follow the sei_nal_hrd_parameters_present_flag[i].

sei_vcl_hrd_parameters_present_flag[i **]** indicates the value of vcl_hrd_parameters_present_flag, as specified in subclause E.2.1, that applies to the i-th bitstream subset. When sei_vcl_hrd_parameters_present_flag[i] is equal to 1, the VCL HRD parameters that apply to the i-th bitstream subset immediately follow the sei_vcl_hrd_parameters_present_flag[i].

sei_low_delay_hrd_flag[i **]** indicates the value of low_delay_hrd_flag, as specified in subclause E.2.1, that applies to the i-th bitstream subset.

sei_pic_struct_present_flag[i **]** indicates the value of pic_struct_present_flag, as specified in subclause E.2.1, that applies to the i-th bitstream subset.

G.13.2.9 Quality layer integrity check SEI message semantics

The quality layer integrity check SEI message provides a mechanism for detecting whether VCL NAL units with quality_id greater than 0 of the primary coded picture have been removed from the bitstream.

The quality layer integrity check SEI message shall not be included in a scalable nesting SEI message.

num_info_entries_minus1 plus 1 specifies the number of syntax element pairs entry_dependency_id[i] and quality_layer_crc[i] that are present in the quality layer integrity check SEI message. The value of num_info_entries_minus1 shall be in the range of 0 to 7, inclusive.

entry_dependency_id[i **]** specifies the dependency_id value of the dependency representation for which quality_layer_crc[i] is indicated. The instances of entry_dependency_id[i] shall appear in increasing order of their values. The dependency representation of the primary coded picture that has dependency_id equal to entry_dependency_id[i] is referred to as target dependency representation.

The target dependency representation may or may not be present in the access unit.

quality_layer_crc[i **]** specifies the cyclic redundancy check for all the VCL NAL units with quality_id greater than 0 in the target dependency representation.

Let crcVal be a variable that is derived as specified by the following ordered steps:

1. Let the variable qNalUnits[ ] be the one-dimensional array of bytes that contains a concatenation, in decoding order, of the bytes of the nal_unit( ) syntax structures of all VCL NAL units with quality_id greater than 0 in the target dependency representation, in decoding order.

2. Let the variable pLen be the sum of the NumBytesInNALunit variables of all VCL NAL units with quality_id greater than 0 in the target dependency representation.

3. The value of crcVal is derived as specified by the following pseudo-code process:

```
qNalUnits[ pLen      ] = 0
qNalUnits[ pLen + 1 ] = 0
crcVal = 65535
for( bitIdx = 0; bitIdx < ( pLen + 2 ) * 8; bitIdx++ ) {                    (G-383)
   crcMsb = ( crcVal >> 15 ) & 1
   bitVal = ( qNalUnits[ bitIdx >> 3 ] >> ( 7 − ( bitIdx & 7 ) ) ) & 1
   crcVal = ( ( ( crcVal << 1 ) + bitVal ) & 65535 ) ^ ( crcMsb * 4129 )
}
```

When the target dependency representation is present in the access unit, a value of quality_layer_crc[i] not equal to crcVal indicates that one or more VCL NAL units with quality_id greater than 0 of the target dependency representation have been removed from the bitstream and that the output pictures may show undesirable visual artefacts.

G.13.2.10 Redundant picture property SEI message semantics

The redundant picture property SEI message indicates properties for layer representations of redundant coded pictures. In the following, a layer representation of a redundant coded picture is also referred to as redundant coded layer representation and a layer representation of the primary coded picture is also referred to as primary coded layer representation.

The redundant picture property SEI message shall not be included in a scalable nesting SEI message.

num_dIds_minus1 plus 1 specifies the number of dependency_id values for which properties of redundant coded layer representations are indicated in the redundant picture property SEI message. The value of num_dIds_minus1 shall be in the range of 0 to 7, inclusive.

dependency_id[i **]** specifies the dependency_id value of the redundant coded layer representations for which properties are indicated by the following syntax elements. When i is greater than 0, the value of dependency_id[i] shall not be equal to any of the values of dependency_id[j] with j = 0..(i − 1).

num_qIds_minus1[i **]** plus 1 specifies the number of quality_id values for which properties of redundant coded layer representations with dependency_id equal to dependency_id[i] are indicated by the following syntax elements. The value of num_qIds_minus1[i] shall be in the range of 0 to 15, inclusive.

quality_id[i **][** j **]** specifies the quality_id value of the redundant coded layer representations with dependency_id equal to dependency_id[i] for which properties are indicated by the following syntax elements. When j is greater than 0, the value of quality_id[i][j] shall not be equal to any of the values of quality_id[i][k] with k = 0..(j − 1).

num_redundant_pics_minus1[i][j] plus 1 specifies the number of redundant coded layer representations with dependency_id equal to dependency_id[i] and quality_id equal to quality_id[i][j] for which properties are indicated by the following syntax elements. The value of num_redundant_pics_minus1[i][j] shall be in the range of 0 to 127, inclusive.

redundant_pic_cnt_minus1[i][j][k] plus 1 specifies the redundant_pic_cnt value of the redundant coded layer representation with dependency_id equal to dependency_id[i] and quality_id equal to quality_id[i][j] for which properties are indicated by the following syntax elements. The value of redundant_pic_cnt_minus1[i][j][k] shall be in the range of 0 to 126, inclusive. When k is greater than 0, the value of redundant_pic_cnt_minus1[i][j][k] shall not be equal to any of the values of redundant_pic_cnt_minus1[i][j][m] with m = 0..(k − 1).

The redundant coded layer representation having dependency_id equal to dependency_id[i], quality_id equal to quality_id[i][j], and redundant_pic_cnt equal to (redundant_pic_cnt_minus1[i][j][k] + 1) is referred to as the target redundant coded layer representation. The primary coded layer representation (redundant_pic_cnt is equal to 0) having dependency_id equal to dependency_id[i] and quality_id equal to quality_id[i][j] is referred to as the target primary coded layer representation.

The target redundant coded layer representation may or may not be present in the access unit. The target primary coded layer representation may or may not be present in the access unit.

For the following specification, the picture that only consists of the target redundant coded layer representation and the primary coded layer representations with DQId less than (dependency_id[i] << 4) + quality_id[i] is referred to as target redundant coded picture and the picture that only consists of the target primary coded layer representation and the primary coded layer representations with DQId less than (dependency_id[i] << 4) + quality_id[i] is referred to as target primary coded picture.

For the following specification, the arrays mbType, subMbType, predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, rS_L, rS_{Cb}, rS_{Cr}, cS_L, cS_{Cb}, and cS_{Cr} represent the corresponding arrays of the collective term currentVars after completion of the target macroblock decoding process as specified in subclause G.8.1.5.6.

pic_match_flag[i][j][k] equal to 1 indicates that the target redundant coded layer representation is an exact copy of the target primary coded layer representation, with the only difference in the value of redundant_pic_cnt.

mb_type_match_flag[i][j][k] equal to 1 indicates that the array mbType for the target redundant coded picture is identical to the array mbType for the target primary coded picture.

motion_match_flag[i][j][k] equal to 1 indicates that, for each macroblock mbAddr in the target layer representation of the target primary coded picture for which the derived macroblock type mbType[mbAddr] represents a P or B macroblock type, the variables and arrays mbType[mbAddr], subMbType[mbAddr], predFlagL0[mbAddr], predFlagL1[mbAddr], refIdxL0[mbAddr], refIdxL1[mbAddr], mvL0[mbAddr], and mvL1[mbAddr] for the target redundant coded picture are identical to the corresponding variables and arrays for the target primary coded picture.

residual_match_flag[i][j][k] equal to 1 indicates that, for each macroblock mbAddr in the target layer representation of the target primary coded picture for which the derived macroblock type mbType[mbAddr] represents a P or B macroblock type, the associated reconstructed residual sample values in the arrays rS_L, rS_{Cb}, and rS_{Cr} for the target redundant coded picture are identical or close to the corresponding reconstructed residual sample values for the target primary coded picture.

intra_samples_match_flag[i][j][k] equal to 1 indicates that, for each macroblock mbAddr in the target layer representation of the target primary coded picture for which the derived macroblock type mbType[mbAddr] represents an I macroblock type, the associated reconstructed sample values in the arrays cS_L, cS_{Cb}, and cS_{Cr} for the target redundant coded picture are identical or close to the corresponding reconstructed sample values for the target primary coded picture.

G.13.2.11 Temporal level zero dependency representation index SEI message semantics

The temporal level zero dependency representation index SEI message provides a mechanism for detecting whether a dependency representation with temporal_id equal to 0 required for decoding the current access unit is available when NAL unit losses are expected during transport.

Let setOfDId be a set of dependency_id values that is derived as follows:

– If the temporal level zero dependency representation index SEI message is not included in a scalable nesting SEI message, setOfDId consists of exactly one value, which is equal to 0.

– Otherwise (the temporal level zero dependency representation index SEI message is included in a scalable nesting SEI message), setOfDId consists of the values sei_dependency_id[i] for all i in the range of 0 to num_layer_representations_minus1, inclusive, that are present in the scalable nesting SEI message associated with the temporal level zero dependency representation index SEI message. For the scalable nesting SEI message that contains the temporal level zero dependency representation index SEI message, all_layer_representations_in_au_flag shall be equal to 1 or the value of sei_quality_id[i] shall be equal to 0 for all values of i in the range of 0 to num_layer_representations_minus1, inclusive.

All dependency representations that are referred to in the following specification inside this subclause are dependency representations of a primary coded picture. Unless specified otherwise, all dependency representation that are referred to in the following are dependency representations of the primary coded picture of the access unit that is associated with the temporal level zero dependency representation index SEI message.

The dependency representations of the access unit that have dependency_id equal to any value of the set setOfDId are also referred to as associated dependency representations.

For each value of dId in the set setOfDId, the access unit may or may not contain a dependency representation with dependency_id equal to dId.

tl0_dep_rep_idx indicates the temporal level zero index for the associated dependency representations, if temporal_id is equal to 0. Otherwise (temporal_id is greater than 0), tl0_dep_rep_idx indicates the temporal level zero index of the dependency representations of the most recent access unit with temporal_id equal to 0 in decoding order that have the same value of dependency_id as any of the associated dependency representations.

For each value of dId in the set setOfDId, the following applies:

– If the dependency representation with dependency_id equal to dId contains a NAL unit with nal_unit_type equal to 5 or a NAL unit with nal_unit_type equal to 20 and idr_flag equal to 1, tl0_dep_rep_idx shall be equal to 0.

– Otherwise (the dependency representation with dependency_id equal to dId does not contain a NAL unit with nal_unit_type equal to 5 or a NAL unit with nal_unit_type equal to 20 and idr_flag equal to 1), the following is specified:

  1. Let prevTL0AU be the most recent access unit in decoding order that has temporal_id equal to 0 and for which the primary coded picture contains a dependency representation with dependency_id equal to dId.

  2. Let prevTL0DepRep be the dependency representation with dependency_id equal to dId of the primary coded picture in access unit prevTL0AU.

  3. Let prevTL0DepRepIdx be equal to the value of tl0_dep_rep_idx that is associated with the dependency representation prevTL0DepRep, as indicated by a corresponding temporal level zero dependency representation index SEI message.

  4. Depending on temporal_id of the current access unit, the following applies:

     – If temporal_id of the current access unit is equal to 0, tl0_dep_rep_idx shall be equal to (prevTL0DepRepIdx + 1) % 256.

     – Otherwise (temporal_id of the current access unit is greater than 0), tl0_dep_rep_idx shall be equal to prevTL0DepRepIdx.

When the temporal level zero dependency representation index SEI message is associated with a particular dependency representation depRepA that has dependency_id equal dIdA and IdrPicFlag equal to 0, a temporal level zero dependency representation index SEI message shall also be associated with the previous dependency representation dIdB in decoding order that has dependency_id equal to dIdA and IdrPicFlag equal to 1 and all dependency representations with dependency_id equal to dIdA and temporal_id equal to 0 that follow the dependency representation dIdB and precede the dependency representation dIdA in decoding order.

NOTE – For the tl0_dep_rep_idx mechanism to be effectively used, transport mechanisms should ensure that the information is present in every packet that carries data for the particular values of dependency_id.

effective_idr_pic_id indicates the latest value of idr_pic_id in decoding order present in this access unit or any preceding access unit for dependency representations indicated by sei_dependency_id[i].

For each value of dId in the set setOfDId, the following applies:

– If the dependency representation with dependency_id equal to dId contains a NAL unit with nal_unit_type equal to 5 or a NAL unit with nal_unit_type equal to 20 and idr_flag equal to 1, effective_idr_pic_id shall be equal to idr_pic_id of the dependency representation with dependency_id equal to dId.

– Otherwise (the dependency representation with dependency_id equal to dId does not contain a NAL unit with nal_unit_type equal to 5 or a NAL unit with nal_unit_type equal to 20 and idr_flag equal to 1), effective_idr_pic_id shall be equal to idr_pic_id of the previous dependency representation in decoding order with dependency_id equal to dId that contains a NAL unit with nal_unit_type equal to 5 or a NAL unit with nal_unit_type equal to 20 and idr_flag equal to 1.

G.13.2.12 Temporal level switching point SEI message semantics

The temporal level switching point SEI message provides a mechanism for identifying temporal level switching points. If a dependency representation is associated with a temporal level switching point SEI message, then it is a temporal level switching point as specified subsequently and constrained by delta_frame_num. Otherwise, the dependency representation may or may not be a temporal level switching point.

All dependency representations that are referred to in the following specification of this subclause are dependency representations of primary coded pictures.

In the following, let tId be the value of temporal_id of the access unit that is associated with the temporal level switching point SEI message.

NOTE 1 – Let dId be the value of dependency_id that a bitstream adaptation process has used to generate a bitstream subset subBitstreamA that contains dependency representations with dependency_id less than or equal to dId and temporal_id less than tId of an input bitstream (that is conforming to this Recommendation | International Standard) until the current access unit, exclusive. The bitstream adaptation process can infer from a temporal level switching point SEI message whether or not the bitstream subset containing subBitstreamA and the dependency representations with dependency_id less than or equal to dId and temporal_id less than or equal to tId of the input bitstream starting from the current access unit, inclusive, is conforming to this Recommendation | International Standard.

The temporal level switching point SEI message shall not be present in access units with temporal_id equal to 0.

The temporal level switching point SEI message shall be included in a scalable nesting SEI message. For the scalable nesting SEI message that contains the temporal level switching point SEI message, all_layer_representations_in_au_flag shall be equal to 1 or the value of sei_quality_id[i] shall be equal to 0 for all values of i in the range of 0 to num_layer_representations_minus1, inclusive.

The following semantics apply independently to each value of sei_dependency_id[i] indicated by the scalable nesting SEI message containing the temporal level switching point SEI message. The current access unit, i.e., the access unit associated with the temporal level switching point SEI message, may or may not contain a dependency representation with dependency_id equal to sei_dependency_id[i]. When the current access unit contains a dependency representation with dependency_id equal to sei_dependency_id[i], the following semantics apply.

The following semantics are specified in a way that they apply to a bitstream conforming to this Recommendation | International Standard for which DependencyIdMax for the current access unit is equal to sei_dependency_id[i].

Let the switch-to dependency representation be the dependency representation in the current access unit that has dependency_id equal to sei_dependency_id[i] and let maxFrameNum be the value of MaxFrameNum for the SVC sequence parameter set that is the active SVC sequence parameter set for the current access unit (with DependencyIdMax equal to sei_dependency_id[i]).

delta_frame_num indicates the difference between the frame_num values of the switch-to dependency representation and the dependency representation with dependency_id equal to sei_dependency_id[i] in the required access unit, as specified subsequently. The value of delta_frame_num shall be in the range of 1 − maxFrameNum to maxFrameNum − 1, inclusive.

Let currFrameNum be the frame_num value of the switch-to dependency representation. The variable requiredFrameNum is set equal to currFrameNum − delta_frame_num. Let lastIdrAU be the most recent access unit in decoding order that contains a dependency representation with dependency_id equal to sei_dependency_id[i] and IdrPicFlag equal to 1. The bitstream shall contain an access unit that succeeds the access unit lastIdrAU and precedes the current access unit in decoding order and contains a dependency representation with frame_num equal to requiredFrameNum and dependency_id equal to sei_dependency_id[i]. The most recent access unit in decoding order that contains a dependency representation with frame_num equal to requiredFrameNum and dependency_id equal to sei_dependency_id[i] is referred to as the required access unit. The required access unit shall have a value of temporal_id that is equal to tId − 1.

The current access unit and all subsequent access units in decoding order for which temporal_id is less than or equal to tId shall not reference any of the following access units through inter prediction in the decoding process specified in subclause G.8:

– access units that precede the required access unit in decoding order and have temporal_id equal to tId − 1,

– access units that precede the current access unit in decoding order and have temporal_id equal to tId.

NOTE 2 – The set of access units consisting of the current access unit and all access units with temporal_id less than or equal to tId that follow the current access unit in decoding order can be decoded when all of the following access units, which precede the current access unit in decoding order, have been decoded: all access units required for decoding the required access unit (i.e., all access units that are directly or indirectly referenced through inter prediction in the decoding process for the required access unit), the required access unit, and all access units with temporal_id less than tId that succeed the required access unit and precede the current access unit in decoding order.

G.14 Video usability information

The specifications in Annex E apply with substituting SVC sequence parameter set for sequence parameter set. The VUI parameters and the constraints specified in Annex E apply to coded video sequences for which the SVC sequence parameter set becomes the active SVC sequence parameter set.

Additionally, the following applies.

G.14.1 SVC VUI parameters extension syntax

svc_vui_parameters_extension( ) {	**C**	**Descriptor**
vui_ext_num_entries_minus1	0	ue(v)
for(i = 0; i <= vui_ext_num_entries_minus1; i++) {		
vui_ext_dependency_id[i **]**	0	u(3)
vui_ext_quality_id[i **]**	0	u(4)
vui_ext_temporal_id[i **]**	0	u(3)
vui_ext_timing_info_present_flag[i **]**	0	u(1)
if(vui_ext_timing_info_present_flag[i]) {		
vui_ext_num_units_in_tick[i **]**	0	u(32)
vui_ext_time_scale[i **]**	0	u(32)
vui_ext_fixed_frame_rate_flag[i **]**	0	u(1)
}		
vui_ext_nal_hrd_parameters_present_flag[i **]**	0	u(1)
if(vui_ext_nal_hrd_parameters_present_flag[i])		
hrd_parameters( )	0	
vui_ext_vcl_hrd_parameters_present_flag[i **]**	0	u(1)
if(vui_ext_vcl_hrd_parameters_present_flag[i])		
hrd_parameters( )	0	
if(vui_ext_nal_hrd_parameters_present_flag[i] || vui_ext_vcl_hrd_parameters_present_flag[i])		
vui_ext_low_delay_hrd_flag[i **]**	0	u(1)
vui_ext_pic_struct_present_flag[i **]**	0	u(1)
}		
}		

G.14.2 SVC VUI parameters extension semantics

The SVC VUI parameters extension specifies timing information, HRD parameter sets, and the presence of picture structure information for subsets of coded video sequences (including the complete coded video sequences) conforming one or more of the profiles specified in Annex G. In Annex C it is specified which of the HRD parameter sets specified in the SVC VUI parameters extension are used for conformance checking.

vui_ext_num_entries_minus1 plus 1 specifies the number of information entries that are present in the SVC VUI parameters extension syntax structure. The value of vui_ext_num_entries_minus1 shall be in the range of 0 to 1023, inclusive. Each information entry is associated with particular values of temporal_id, dependency_id, and quality_id and may indicate timing information, NAL HRD parameters, VCL HRD parameters, and the presence of picture structure information for a particular subset of coded video sequences as specified in the following.

vui_ext_dependency_id[i] and **vui_ext_quality_id**[i] indicate the maximum value of DQId for the i-th subset of coded video sequences. The maximum value of DQId for the i-th subset of coded video sequences is derived by vui_ext_dependency_id[i] + (vui_ext_quality_id[i] << 4).

vui_ext_temporal_id[i] indicates the maximum value of temporal_id for the i-th subset of coded video sequences.

The SVC VUI parameters extension syntax structure shall not contain two or more information entries with identical values of vui_ext_dependency_id[i], vui_ext_quality_id[i], and vui_ext_temporal_id[i].

The following syntax elements apply to the coded video sequences that are obtained by the invoking the sub-bitstream extraction process as specified in subclause G.8.8.1 with tIdTarget equal to vui_ext_temporal_id[i], dIdTarget equal to vui_ext_dependency_id[i], and qIdTarget equal to vui_ext_quality_id[i] as the inputs and the i-th subset of coded video sequences as the output.

vui_ext_timing_info_present_flag[i] equal to 1 specifies that vui_ext_num_units_in_tick[i], vui_ext_time_scale[i], and vui_ext_fixed_frame_rate_flag[i] for the i-th subset of coded video sequences are present in the SVC VUI parameters extension. vui_ext_timing_info_present_flag[i] equal to 0 specifies that vui_ext_num_units_in_tick[i], vui_ext_time_scale[i], and vui_ext_fixed_frame_rate_flag[i] for the i-th subset of coded video sequences are not present in the SVC VUI parameters extension.

The following syntax elements for the i-th subset of coded video sequences are specified using references to Annex E. For these syntax elements the same semantics and constraints as the ones specified in Annex E apply, as if these syntax elements vui_ext_num_units_in_tick[i], vui_ext_time_scale[i], vui_ext_fixed_frame_rate_flag[i], vui_ext_nal_hrd_parameters_present_flag[i], vui_ext_vcl_hrd_parameters_present_flag[i], vui_ext_low_delay_hrd_flag[i], and vui_ext_pic_struct_present_flag[i] were present as the syntax elements num_units_in_tick, time_scale, fixed_frame_rate_flag, nal_hrd_parameters_present_flag, vcl_hrd_parameters_present_flag, low_delay_hrd_flag, and pic_struct_present_flag, respectively, in the VUI parameters of the active SVC sequence parameter sets for the i-th subset of coded video sequences.

vui_ext_num_units_in_tick[i] specifies the value of num_units_in_tick, as specified in subclause E.2.1, for the i-th subset of coded video sequences.

vui_ext_time_scale[i] specifies the value of time_scale, as specified in subclause E.2.1, for the i-th subset of coded video sequences.

vui_ext_fixed_frame_rate_flag[i] specifies the value of fixed_frame_rate_flag, as specified in subclause E.2.1, for the i-th subset of coded video sequences.

vui_ext_nal_hrd_parameters_present_flag[i] specifies the value of nal_hrd_parameters_present_flag, as specified in subclause E.2.1, for the i-th subset of coded video sequences.

When vui_ext_nal_hrd_parameters_present_flag[i] is equal to 1, NAL HRD parameters (subclauses E.1.2 and E.2.2) for the i-th subset of coded video sequences immediately follow the flag.

The variable VuiExtNalHrdBpPresentFlag[i] is derived as follows:

– If any of the following is true, the value of VuiExtNalHrdBpPresentFlag[i] shall be set equal to 1:
  – vui_ext_nal_hrd_parameters_present_flag[i] is present in the bitstream and is equal to 1,
  – for the i-th subset of coded video sequences, the need for presence of buffering periods for NAL HRD operation to be present in the bitstream in buffering period SEI messages is determined by the application, by some means not specified in this Recommendation | International Standard.
– Otherwise, the value of VuiExtNalHrdBpPresentFlag[i] shall be set equal to 0.

vui_ext_vcl_hrd_parameters_present_flag[i] specifies the value of vcl_hrd_parameters_present_flag, as specified in subclause E.2.1, for the i-th subset of coded video sequences.

When vui_ext_vcl_hrd_parameters_present_flag[i] is equal to 1, VCL HRD parameters (subclauses E.1.2 and E.2.2) for the i-th subset of coded video sequences immediately follow the flag.

The variable VuiExtVclHrdBpPresentFlag[i] is derived as follows:

– If any of the following is true, the value of VuiExtVclHrdBpPresentFlag[i] shall be set equal to 1:
  – vui_ext_vcl_hrd_parameters_present_flag[i] is present in the bitstream and is equal to 1,
  – for the i-th subset of coded video sequences, the need for presence of buffering periods for VCL HRD operation to be present in the bitstream in buffering period SEI messages is determined by the application, by some means not specified in this Recommendation | International Standard.
– Otherwise, the value of VuiExtVclHrdBpPresentFlag[i] shall be set equal to 0.

The variable VuiExtCpbDpbDelaysPresentFlag[i] is derived as follows:

– If any of the following is true, the value of VuiExtCpbDpbDelaysPresentFlag[i] shall be set equal to 1:
  – vui_ext_nal_hrd_parameters_present_flag[i] is present in the bitstream and is equal to 1,

  - vui_ext_vcl_hrd_parameters_present_flag[i] is present in the bitstream and is equal to 1,
  - for the i-th subset of coded video sequences, the need for presence of CPB and DPB output delays to be present in the bitstream in picture timing SEI messages is determined by the application, by some means not specified in this Recommendation | International Standard.
- Otherwise, the value of VuiExtCpbDpbDelaysPresentFlag[i] shall be set equal to 0.

vui_ext_low_delay_hrd_flag[i] specifies the value of low_delay_hrd_flag, as specified in subclause E.2.1, for the i-th subset of coded video sequences.

vui_ext_pic_struct_present_flag[i] specifies the value of pic_struct_present_flag, as specified in subclause E.2.1, for the i-th subset of coded video sequences.

Annex H

Multiview video coding

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies multiview video coding, referred to as MVC.

H.1 Scope

Bitstreams and decoders conforming to the profile specified in this annex are completely specified in this annex with reference made to clauses 2-9 and Annexes A-E.

H.2 Normative references

The specifications in clause 2 apply.

H.3 Definitions

For the purpose of this annex, the following definitions apply in addition to the definitions in clause 3. These definitions are either not present in clause 3 or replace definitions in clause 3.

H.3.1 **access unit**: A set of *NAL units* that are consecutive in *decoding order* and contain exactly one *primary coded picture* consisting of one or more *view components*. In addition to the *primary coded picture*, an access unit may also contain one or more *redundant coded pictures*, one *auxiliary coded picture*, or other *NAL units* not containing *slices* or *slice data partitions* of a *coded picture*. The decoding of an access unit always results in one *decoded picture* consisting of one or more *decoded view components*.

H.3.2 **anchor access unit**: An *access unit* in which the *primary coded picture* is an *anchor picture*.

H.3.3 **anchor picture**: A *coded picture* in which all *slices* may reference only *slices* within the same *access unit*, i.e., *inter-view prediction* may be used, but no *inter prediction* is used, and all following *coded pictures* in output order do not use *inter prediction* from any *picture* prior to the *coded picture* in *decoding order*. The value of anchor_pic_flag is equal to 1 for all the *prefix NAL units* (when present) and all the slice extension NAL units that are contained in an anchor picture.

H.3.4 **anchor view component**: A *view component* in an *anchor picture*. All *view components* in an *anchor picture* are anchor view components.

H.3.5 **associated NAL unit**: A *NAL unit* that immediately follows a *prefix NAL unit* in *decoding order*.

H.3.6 **base view**: A *view* that has the minimum value of *view order index* in a *coded video sequence*. The base view can be decoded independently of other *views*, does not use *inter-view prediction,* and contains *VCL NAL units* only with nal_unit_type equal to 1, 5, or 14. The *bitstream subset* corresponding to the base view conforms to one or more of the *profiles* specified in Annex A. There is only one base view in a *coded video sequence*.

H.3.7 **bitstream subset**: A *bitstream* that is derived as a *subset* from a *bitstream* by discarding zero or more *NAL units*. A *bitstream subset* is also referred to as a *sub-bitstream*.

H.3.8 **coded slice MVC extension NAL unit**: A *coded slice NAL unit* that has nal_unit_type equal to 20.

H.3.9 **decoded view component**: A decoded view component is derived by decoding a *view component*. A decoded view component is either a decoded *frame view component*, or a decoded *field view component*.

H.3.10 **direct prediction**: An *inter prediction* or *inter-view 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 mode and temporal direct prediction mode.

H.3.11 **field view component**: A *view component* of a *field*.

H.3.12 **frame view component**: A *view component* of a *frame*.

H.3.13 **instantaneous decoding refresh (IDR) view component**: A *view component* in an *IDR picture*. All *view components* in an *IDR picture* are IDR view components. IDR view components are also *anchor view* *components*, and *inter-view prediction* may be used for IDR view components that are part of a *non-base view*.

H.3.14 inter-view coding: Coding of a *block*, *macroblock*, *slice*, or *picture* that uses *inter-view prediction*.

H.3.15 inter-view only reference component: A *view component* coded with nal_ref_idc equal to 0 and inter_view_flag equal to 1. An inter-view only reference component contains samples that may be used for *inter-view prediction* in the *decoding process* of subsequent *view components* in *decoding order*, but are not used for *inter prediction* by any *view components*. Inter-view only reference components are *non-reference pictures*.

H.3.16 inter-view prediction: A *prediction* derived from decoded samples of *inter-view reference components* or *inter-view only reference components* for decoding another *view component* in the same *access unit*.

H.3.17 inter-view prediction reference: A collective term for *inter-view reference components* or *inter-view only reference components*.

H.3.18 inter-view reference component: A *view component* coded with nal_ref_idc greater than 0 and inter_view_flag equal to 1. An inter-view reference component contains samples that may be used for *inter prediction* of subsequent *pictures* in *decoding order* and *inter-view prediction* of subsequent *view components* in *decoding order*. Inter-view reference components are *reference pictures*.

H.3.19 list 0 (list 1) prediction: *Inter prediction or inter-view prediction* of the content of a *slice* using a *reference index* pointing into *reference picture list 0 (list 1)*.

H.3.20 macroblock partition: A *block* of *luma* samples and two corresponding *blocks* of *chroma* samples resulting from a *partitioning* of a *macroblock* for *inter prediction or inter-view prediction*.

H.3.21 motion vector: A two-dimensional vector used for *inter prediction or inter-view* prediction that provides an offset from the coordinates in the *decoded view component* to the coordinates in a *reference picture* or *inter-view only reference component*.

H.3.22 MVC sequence parameter set: A collective term for *sequence parameter set* or *subset sequence parameter set*.

H.3.23 MVC sequence parameter set RBSP: A collective term for sequence parameter set RBSP or subset sequence parameter set RBSP.

H.3.24 non-anchor access unit: An *access unit* that is not an *anchor access unit*.

H.3.25 non-anchor picture: A *coded picture* that is not an *anchor picture*.

H.3.26 non-anchor view component: A *view component* that is not an *anchor view component*.

H.3.27 non-base view: A *view* that is not the *base view*. *VCL NAL units* of a non-base view have nal_unit_type equal to 20. Decoding of *view components* in a non-base view may require the use of *inter-view prediction*.

H.3.28 non-reference picture: A *view component* coded with nal_ref_idc equal to 0. A non-reference picture is not used for *inter prediction* in the *decoding process* of any other *view components*.

H.3.29 operation point: An operation point is identified by a temporal_id value representing the *target temporal level* and a set of view_id values representing the *target output views*. One *operation point* is associated with a *bitstream subset*, which consists of the *target output views* and all other *views* the *target output views* depend on, that is derived using the *sub-bitstream* extraction process as specified in subclause H.8.5.3 with tIdTarget equal to the temporal_id value and viewIdTargetList consisting of the set of view_id values as inputs. More than one *operation point* may be associated with the same *bitstream subset*. When the specification states "an *operation point* is decoded" it refers to the decoding of a *bitstream subset* corresponding to the *operation point* and subsequent output of the *target output views*.

H.3.30 picture order count: A variable that is associated with each *field view component* and each *field* of a *frame view component* and has a value that is non-decreasing with increasing *field* position in *output order* in the same *view* relative to the first output *field* of the previous *IDR view component* in *decoding order* in the same *view* or relative to the first output *field of the* previous *view component*, in *decoding order* in the same *view*, that contains a *memory management control operation* that marks all *reference pictures* in the *view* as "unused for reference".

H.3.31 prefix NAL unit: A *NAL unit* with nal_unit_type equal to 14 that immediately precedes in *decoding order* a *NAL unit* with nal_unit_type equal to 1 or 5. The *NAL unit* that immediately follows in *decoding order* the prefix NAL unit is referred to as the *associated NAL unit*. The prefix NAL unit contains data associated with the *associated NAL unit*, which are considered to be part of the *associated NAL unit*.

H.3.32 reference picture: A *view component* coded with nal_ref_idc greater than 0. A reference picture contains samples that may be used for *inter prediction* in the *decoding process* of subsequent *view components* in *decoding order*. A reference picture may be an *inter-view reference component*, in which case the samples contained in the reference picture may also be used for *inter-view prediction* in the *decoding process* of subsequent *view components* in *decoding order*.

H.3.33 reference picture list: A list of *reference pictures* and *inter-view only reference components* that are used for *inter prediction* or *inter-view prediction* of a *P* or *B slice.* For the *decoding process* of a *P slice,* there is one reference picture list. For the *decoding process* of a *B slice*, there are two reference picture lists.

H.3.34 reference picture list 0: A *reference picture list* used for *inter prediction* or *inter-view prediction* of a *P* or *B slice*. All *inter prediction* or *inter-view prediction* used for *P slices* uses reference picture list 0. Reference picture list 0 is one of two *reference picture lists* used for *inter prediction* or *inter-view prediction* for a *B slice*, with the other being *reference picture list 1*.

H.3.35 reference picture list 1: A *reference picture list* used for *inter prediction* or *inter-view prediction* of a *B slice*. Reference picture list 1 is one of two *reference picture lists* used for *inter prediction* or *inter-view prediction* for a *B slice*, with the other being *reference picture list 0*.

H.3.36 reference picture marking: Specifies, in the bitstream, how the *decoded view components* are marked for *inter prediction* or *inter-view prediction*.

H.3.37 reference view index: An index into a list of *anchor view components* or a list of *non-anchor view components* that are specified in the sequence parameter set MVC extension syntax structure and can be used for *inter-view prediction* as *list 0 prediction* or *list 1 prediction*.

H.3.38 sub-bitstream: A *subset* of a *bitstream*. A *sub-bitstream* is also referred to as a *bitstream subset*.

H.3.39 subset: A subset contains only elements that are also contained in the set from which the subset is derived. The subset may be identical to the set from which it is derived.

H.3.40 subset sequence parameter set: A *syntax structure* containing *syntax elements* that apply to zero or more *non-base views* 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* of *I, P, and B slices* of a *non-base view component*.

H.3.41 target output view: A *view* that is to be output. The target output views are either indicated by external means or, when not indicated by external means, the target output view is the *base view*.

NOTE – The output views may be requested by a receiver and may be negotiated between the receiver and the sender.

H.3.42 target temporal level: The *target temporal level* of an *operation point* is the greatest value of temporal_id of all *VCL NAL units* in the *bitstream subset* associated with the *operation point*.

H.3.43 view: A sequence of *view components* associated with an identical value of view_id.

H.3.44 view component: A *coded representation* of a *view* in a single *access unit*.

H.3.45 view order index: An index that indicates the *decoding order* of *view components* in an *access unit*.

H.4 Abbreviations

The specifications in clause 4 apply.

H.5 Conventions

The specifications in clause 5 apply.

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

The specifications in clause 6 apply with substitution of MVC sequence parameter set for sequence parameter set.

H.7 Syntax and semantics

This clause specifies syntax and semantics for coded video sequences that conform to one or more of the profiles specified in this annex.

H.7.1 Method of specifying syntax in tabular form

The specifications in subclause 7.1 apply.

H.7.2 Specification of syntax functions, categories, and descriptors

The specifications in subclause 7.2 apply.

H.7.3 Syntax in tabular form

H.7.3.1 NAL unit syntax

The syntax table is specified in subclause 7.3.1.

H.7.3.1.1 NAL unit header MVC extension syntax

nal_unit_header_mvc_extension( ) {	**C**	**Descriptor**
non_idr_flag	All	u(1)
priority_id	All	u(6)
view_id	All	u(10)
temporal_id	All	u(3)
anchor_pic_flag	All	u(1)
inter_view_flag	All	u(1)
reserved_one_bit	All	u(1)
}		

H.7.3.2 Raw byte sequence payloads and RBSP trailing bits syntax

H.7.3.2.1 Sequence parameter set RBSP syntax

The syntax table is specified in subclause 7.3.2.1.

H.7.3.2.1.1 Sequence parameter set data syntax

The syntax table is specified in subclause 7.3.2.1.1.

H.7.3.2.1.1.1 Scaling list syntax

The syntax table is specified in subclause 7.3.2.1.1.1.

H.7.3.2.1.2 Sequence parameter set extension RBSP syntax

The syntax table is specified in subclause 7.3.2.1.2.

H.7.3.2.1.3 Subset sequence parameter set RBSP syntax

The syntax table is specified in subclause 7.3.2.1.3.

H.7.3.2.1.4 Sequence parameter set MVC extension syntax

seq_parameter_set_mvc_extension( ) {	C	Descriptor
num_views_minus1	0	ue(v)
for(i = 0; i <= num_views_minus1; i++)		
view_id[i **]**	0	ue(v)
for(i = 1; i <= num_views_minus1; i++) {		
num_anchor_refs_l0[i **]**	0	ue(v)
for(j = 0; j < num_anchor_refs_l0[i]; j++)		
anchor_ref_l0[i **][** j **]**	0	ue(v)
num_anchor_refs_l1[i **]**	0	ue(v)
for(j = 0; j < num_anchor_refs_l1[i]; j++)		
anchor_ref_l1[i **][** j **]**	0	ue(v)
}		
for(i = 1; i <= num_views_minus1; i++) {		
num_non_anchor_refs_l0[i **]**	0	ue(v)
for(j = 0; j < num_non_anchor_refs_l0[i]; j++)		
non_anchor_ref_l0[i **][** j **]**	0	ue(v)
num_non_anchor_refs_l1[i **]**	0	ue(v)
for(j = 0; j < num_non_anchor_refs_l1[i]; j++)		
non_anchor_ref_l1[i **][** j **]**	0	ue(v)
}		
num_level_values_signalled_minus1	0	ue(v)
for(i = 0; i <= num_level_values_signalled_minus1; i++) {		
level_idc[i **]**	0	u(8)
num_applicable_ops_minus1[i **]**	0	ue(v)
for(j = 0; j <= num_applicable_ops_minus1[i]; j++) {		
applicable_op_temporal_id[i **][** j **]**	0	u(3)
applicable_op_num_target_views_minus1[i **][** j **]**	0	ue(v)
for(k = 0; k <= applicable_op_num_target_views_minus1[i][j]; k++)		
applicable_op_target_view_id[i **][** j **][** k **]**	0	ue(v)
applicable_op_num_views_minus1[i **][** j **]**	0	ue(v)
}		
}		
}		

H.7.3.2.2 Picture parameter set RBSP syntax

The syntax table is specified in subclause 7.3.2.2.

H.7.3.2.3 Supplemental enhancement information RBSP syntax

The syntax table is specified in subclause 7.3.2.3.

H.7.3.2.3.1 Supplemental enhancement information message syntax

The syntax table is specified in subclause 7.3.2.3.1.

H.7.3.2.4 Access unit delimiter RBSP syntax

The syntax table is specified in subclause 7.3.2.4.

H.7.3.2.5 End of sequence RBSP syntax

The syntax table is specified in subclause 7.3.2.5.

H.7.3.2.6 End of stream RBSP syntax

The syntax table is specified in subclause 7.3.2.6.

H.7.3.2.7 Filler data RBSP syntax

The syntax table is specified in subclause 7.3.2.7.

H.7.3.2.8 Slice layer without partitioning RBSP syntax

The syntax table is specified in subclause 7.3.2.8.

H.7.3.2.9 Slice data partition RBSP syntax

Slice data partition syntax is not present in coded video sequences conforming to one or more of the profiles specified in this annex.

H.7.3.2.10 RBSP slice trailing bits syntax

The syntax table is specified in subclause 7.3.2.10.

H.7.3.2.11 RBSP trailing bits syntax

The syntax table is specified in subclause 7.3.2.11.

H.7.3.2.12 Prefix NAL unit RBSP syntax

The syntax table is specified in subclause 7.3.2.12.

H.7.3.2.13 Slice layer extension RBSP syntax

The syntax table is specified in subclause 7.3.2.13.

H.7.3.3 Slice header syntax

The syntax table is specified in subclause 7.3.3.

H.7.3.3.1 Reference picture list modification syntax

The syntax table is specified in subclause 7.3.3.1.

H.7.3.3.1.1 Reference picture list MVC modification syntax

ref_pic_list_mvc_modification( ) {	C	Descriptor
if(slice_type % 5 != 2 && slice_type % 5 != 4) {		
ref_pic_list_modification_flag_l0	2	u(1)
if(ref_pic_list_modification_flag_l0)		
do {		
modification_of_pic_nums_idc	2	ue(v)
if(modification_of_pic_nums_idc = = 0 || modification_of_pic_nums_idc = = 1)		
abs_diff_pic_num_minus1	2	ue(v)
else if(modification_of_pic_nums_idc = = 2)		
long_term_pic_num	2	ue(v)
else if (modification_of_pic_nums_idc = = 4 || modification_of_pic_nums_idc = = 5)		
abs_diff_view_idx_minus1	2	ue(v)
} while(modification_of_pic_nums_idc != 3)		
}		
if(slice_type % 5 = = 1) {		
ref_pic_list_modification_flag_l1	2	u(1)
if(ref_pic_list_modification_flag_l1)		
do {		
modification_of_pic_nums_idc	2	ue(v)
if(modification_of_pic_nums_idc = = 0 || modification_of_pic_nums_idc = = 1)		
abs_diff_pic_num_minus1	2	ue(v)
else if(modification_of_pic_nums_idc = = 2)		
long_term_pic_num	2	ue(v)
else if (modification_of_pic_nums_idc = = 4 || modification_of_pic_nums_idc = = 5)		
abs_diff_view_idx_minus1	2	ue(v)
} while(modification_of_pic_nums_idc != 3)		
}		
}		

H.7.3.3.2 Prediction weight table syntax

The syntax table is specified in subclause 7.3.3.2.

H.7.3.3.3 Decoded reference picture marking syntax

The syntax table is specified in subclause 7.3.3.3.

H.7.3.4 Slice data syntax

The syntax table is specified in subclause 7.3.4.

H.7.3.5 Macroblock layer syntax

The syntax table is specified in subclause 7.3.5.

H.7.3.5.1 Macroblock prediction syntax

The syntax table is specified in subclause 7.3.5.1.

H.7.3.5.2 Sub-macroblock prediction syntax

The syntax table is specified in subclause 7.3.5.2.

H.7.3.5.3 Residual data syntax

The syntax table is specified in subclause 7.3.5.3.

H.7.3.5.3.1 Residual luma syntax

The syntax table is specified in subclause 7.3.5.3.1.

H.7.3.5.3.2 Residual block CAVLC syntax

The syntax table is specified in subclause 7.3.5.3.2.

H.7.3.5.3.3 Residual block CABAC syntax

The syntax table is specified in subclause 7.3.5.3.3.

H.7.4 Semantics

Semantics associated with the syntax structures and syntax elements within these structures (in subclause H.7.3 and in subclause 7.3 by reference in subclause H.7.3) are specified in this subclause and by reference to subclause 7.4. When the semantics of a syntax element are specified using a table or a set of tables, any values that are not specified in the table(s) shall not be present in the bitstream unless otherwise specified in this Recommendation | International Standard.

H.7.4.1 NAL unit semantics

The semantics for the syntax elements in subclause H.7.3.1 are specified in subclause 7.4.1. The following specifications additionally apply.

For NAL units with nal_unit_type equal to 14, nal_ref_idc shall be identical to the value of nal_ref_idc for the associated NAL unit, which follows the NAL unit with nal_unit_type equal to 14 in decoding order.

The value of nal_ref_idc shall be identical for all VCL NAL units of a view component.

H.7.4.1.1 NAL unit header MVC extension semantics

The syntax elements non_idr_flag, priority_id, view_id, temporal_id, anchor_pic_flag, and inter_view_flag, when present in a prefix NAL unit, are considered to apply to the associated NAL unit.

non_idr_flag equal to 0 specifies that the current access unit is an IDR access unit.

The value of non_idr_flag shall be the same for all VCL NAL units of an access unit. When non_idr_flag is equal to 0 for a prefix NAL unit, the associated NAL unit shall have nal_unit_type equal to 5. When non_idr_flag is equal to 1 for a prefix NAL unit, the associated NAL unit shall have nal_unit_type equal to 1.

When nal_unit_type is equal to 1 and the NAL unit is not immediately preceded by a NAL unit with nal_unit_type equal to 14, non_idr_flag shall be inferred to be equal to 1. When nal_unit_type is equal to 5 and the NAL unit is not immediately preceded by a NAL unit with nal_unit_type equal to 14, non_idr_flag shall be inferred to be equal to 0.

When nal_ref_idc is equal to 0, the value of non_idr_flag shall be equal to 1.

For NAL units in which non_idr_flag is present, the variable IdrPicFlag derived in subclause 7.4.1 is modified by setting it equal to 1 when non_idr_flag is equal to 0, and setting it equal to 0 when non_idr_flag is equal to 1.

priority_id specifies a priority identifier for the NAL unit. A lower value of priority_id specifies a higher priority. The assignment of values to priority_id is constrained by the sub-bitstream extraction process as specified in subclause H.8.5.3.

When nal_unit_type is equal to 1 or 5 and the NAL unit is not immediately preceded by a NAL unit with nal_unit_type equal to 14, priority_id shall be inferred to be equal to 0.

NOTE 1 – The syntax element priority_id is not used by the decoding process specified in this Recommendation | International Standard. The syntax element priority_id may be used as determined by the application within the specified constraints.

view_id specifies a view identifier for the NAL unit. NAL units with the same value of view_id belong to the same view. The assignment of values to view_id is constrained by the sub-bitstream extraction process as specified in subclause H.8.5.3.

When nal_unit_type is equal to 1 or 5 and the NAL unit is not immediately preceded by a NAL unit with nal_unit_type equal to 14, the value of view_id shall be inferred to be equal to 0. When the bitstream does contain NAL units with nal_unit_type equal to 1 or 5 that are not immediately preceded by a NAL unit with nal_unit_type equal to 14, it shall not contain data that result in a value of view_id for a view component of any non-base view that is equal to 0.

The variable VOIdx, representing the view order index of the view identified by view_id, is set equal to the value of i for which the syntax element view_id[i] included in the referred subset sequence parameter set is equal to view_id.

temporal_id specifies a temporal identifier for the NAL unit.

When nal_unit_type is equal to 1 or 5 and the NAL unit is not immediately preceded by a NAL unit with nal_unit_type equal to 14, temporal_id shall be inferred to be equal to the value of temporal_id for the non-base views in the same access unit.

The value of temporal_id shall be the same for all prefix and coded slice MVC extension NAL units of an access unit. When an access unit contains any NAL unit with nal_unit_type equal to 5 or non_idr_flag equal to 0, temporal_id shall be equal to 0.

The assignment of values to temporal_id is further constrained by the sub-bitstream extraction process as specified in subclause H.8.5.3.

anchor_pic_flag equal to 1 specifies that the current access unit is an anchor access unit.

When nal_unit_type is equal to 1 or 5 and the NAL unit is not immediately preceded by a NAL unit with nal_unit_type equal to 14, anchor_pic_flag shall be inferred to be equal to the value of anchor_pic_flag for the non-base views in the same access unit.

When non_idr_flag is equal to 0, anchor_pic_flag shall be equal to 1.

When nal_ref_idc is equal to 0, the value of anchor_pic_flag shall be equal to 0.

The value of anchor_pic_flag shall be the same for all VCL NAL units of an access unit.

inter_view_flag equal to 0 specifies that the current view component is not used for inter-view prediction by any other view component in the current access unit. inter_view_flag equal to 1 specifies that the current view component may be used for inter-view prediction by other view components in the current access unit.

When nal_unit_type is equal to 1 or 5 and the NAL unit is not immediately preceded by a NAL unit with nal_unit_type equal to 14, inter_view_flag shall be inferred to be equal to 1.

The value of inter_view_flag shall be the same for all VCL NAL units of a view component.

reserved_one_bit shall be equal to 1. The value 0 for reserved_one_bit may be specified by future extension of this Recommendation | International Standard. Decoders shall ignore the value of reserved_one_bit.

H.7.4.1.2 Order of NAL units and association to coded pictures, access units, and video sequences

This subclause specifies constraints on the order of NAL units in the bitstream. Any order of NAL units in the bitstream obeying these constraints is referred to in the text as the decoding order of NAL units. Within a NAL unit, the syntax in subclauses 7.3, D.1, E.1, H.7.3, H.13.1, and H.14.1 specifies the decoding order of syntax elements. Decoders shall be capable of receiving NAL units and their syntax elements in decoding order.

H.7.4.1.2.1 Order of MVC sequence parameter set RBSPs and picture parameter set RBSPs and their activation

NOTE 1 – The sequence and picture parameter set mechanism decouples the transmission of infrequently changing information from the transmission of coded macroblock data. Sequence and picture parameter sets may, in some applications, be conveyed "out-of-band" using a reliable transport mechanism.

A picture parameter set RBSP includes parameters that can be referred to by the coded slice NAL units of one or more view components of one or more coded pictures.

Each picture parameter set RBSP is initially considered not active at the start of the operation of the decoding process. At most one picture parameter set RBSP is considered as the active picture parameter set RBSP at any given moment during the operation of the decoding process, and when any particular picture parameter set RBSP becomes the active picture parameter set RBSP, the previously-active picture parameter set RBSP (if any) is deactivated.

In addition to the active picture parameter set RBSP, zero or more picture parameter set RBSPs may be specifically active for view components (with a particular value of VOIdx less than VOIdxMax) that may be referred to through inter-view prediction in decoding the view component with VOIdx equal to VOIdxMax. Such a picture parameter set RBSP is referred to as active view picture parameter set RBSP for the particular value of VOIdx. The restrictions on active picture parameter set RBSPs also apply to active view picture parameter set RBSPs for a particular value of VOIdx less than VOIdxMax.

When a picture parameter set RBSP (with a particular value of pic_parameter_set_id) is not the active picture parameter set RBSP and it is referred to by a coded slice NAL unit with VOIdx equal to VOIdxMax (using that value of pic_parameter_set_id), it is activated. This picture parameter set RBSP is called the active picture parameter set RBSP until it is deactivated when another picture parameter set RBSP becomes the active picture parameter set RBSP. A picture parameter set RBSP, with that particular value of pic_parameter_set_id, shall be available to the decoding process prior to its activation.

When a picture parameter set RBSP (with a particular value of pic_parameter_set_id) is not the active view picture parameter set for a particular value of VOIdx less than VOIdxMax and it is referred to by a coded slice NAL unit with the particular value of VOIdx (using that value of pic_parameter_set_id), it is activated for view components with the particular value of VOIdx. This picture parameter set RBSP is called the active view picture parameter set RBSP for the particular value of VOIdx until it is deactivated when another picture parameter set RBSP becomes the active view picture parameter set RBSP for the particular value of VOIdx. A picture parameter set RBSP, with that particular value of pic_parameter_set_id, shall be available to the decoding process prior to its activation.

Any picture parameter set NAL unit containing the value of pic_parameter_set_id for the active picture parameter set RBSP for a coded picture shall have the same content as that of the active picture parameter set RBSP for this coded picture unless it follows the last VCL NAL unit of this coded picture and precedes the first VCL NAL unit of another coded picture. Any picture parameter set NAL unit containing the value of pic_parameter_set_id for the active view picture parameter set RBSP for a particular value of VOIdx less than VOIdxMax for a coded picture shall have the same content as that of the active view picture parameter set RBSP for the particular value of VOIdx for this coded picture unless it follows the last VCL NAL unit of this coded picture and precedes the first VCL NAL unit of another coded picture.

When a picture parameter set NAL unit with a particular value of pic_parameter_set_id is received, its content replaces the content of the previous picture parameter set NAL unit, in decoding order, with the same value of pic_parameter_set_id (when a previous picture parameter set NAL unit with the same value of pic_parameter_set_id was present in the bitstream).

NOTE 2 – A decoder must be capable of simultaneously storing the contents of the picture parameter sets for all values of pic_parameter_set_id. The content of the picture parameter set with a particular value of pic_parameter_set_id is overwritten when a new picture parameter set NAL unit with the same value of pic_parameter_set_id is received.

An MVC sequence parameter set RBSP includes parameters that can be referred to by one or more picture parameter set RBSPs or one or more buffering period SEI messages.

Each MVC sequence parameter set RBSP is initially considered not active at the start of the operation of the decoding process. At most one MVC sequence parameter set RBSP is considered as the active MVC sequence parameter set RBSP at any given moment during the operation of the decoding process, and when any particular MVC sequence parameter set RBSP becomes the active MVC sequence parameter set RBSP, the previously-active MVC sequence parameter set RBSP (if any) is deactivated.

In addition to the active MVC sequence parameter set RBSP, zero or more MVC sequence parameter set RBSPs may be specifically active for view components (with a particular value of VOIdx less than VOIdxMax) that may be referred to through inter-view prediction in decoding the view component with VOIdx equal to VOIdxMax. Such an MVC sequence parameter set RBSP is referred to as the active view MVC sequence parameter set RBSP for the particular value of VOIdx. The restrictions on active MVC sequence parameter set RBSPs also apply to active view MVC sequence parameter set RBSPs for a particular value of VOIdx less than VOIdxMax.

For the following specification, the activating buffering period SEI message is specified as follows:

– If VOIdxMax is equal to VOIdxMin and the access unit contains a buffering period SEI message not included in an MVC scalable nesting SEI message, this buffering period SEI message is the activating buffering period SEI message.

– Otherwise if VOIdxMax is not equal to VOIdxMin and the access unit contains a buffering period SEI message included in an MVC scalable nesting SEI message and associated with the operation point being decoded, this buffering period SEI message is the activating buffering period SEI message.

– Otherwise, the access unit does not contain an activating buffering period SEI message.

When a sequence parameter set RBSP (nal_unit_type is equal to 7) with a particular value of seq_parameter_set_id is not already the active MVC sequence parameter set RBSP and it is referred to by activation of a picture parameter set RBSP (using that value of seq_parameter_set_id) and the picture parameter set RBSP is activated by a coded slice NAL unit with nal_unit_type equal to 1 or 5 (the picture parameter set RBSP becomes the active picture parameter set RBSP and VOIdxMax is equal to VOIdxMin) and the access unit does not contain an activating buffering period SEI message, it is activated. This sequence parameter set RBSP is called the active MVC sequence parameter set RBSP until it is deactivated when another MVC sequence parameter set RBSP becomes the active MVC sequence parameter set RBSP.

A sequence parameter set RBSP, with that particular value of seq_parameter_set_id, shall be available to the decoding process prior to its activation.

When a sequence parameter set RBSP (nal_unit_type is equal to 7) with a particular value of seq_parameter_set_id is not already the active MVC sequence parameter set RBSP and it is referred to by an activating buffering period SEI message (using that value of seq_parameter_set_id) that is not included in an MVC scalable nesting SEI message and VOIdxMax is equal to VOIdxMin, it is activated. This sequence parameter set RBSP is called the active MVC sequence parameter set RBSP until it is deactivated when another MVC sequence parameter set RBSP becomes the active MVC sequence parameter set RBSP. A sequence parameter set RBSP, with that particular value of seq_parameter_set_id, shall be available to the decoding process prior to its activation.

When a subset sequence parameter set RBSP (nal_unit_type is equal to 15) with a particular value of seq_parameter_set_id is not already the active MVC sequence parameter set RBSP and it is referred to by activation of a picture parameter set RBSP (using that value of seq_parameter_set_id) and the picture parameter set RBSP is activated by a coded slice MVC extension NAL unit (nal_unit_type is equal to 20) with VOIdx equal to VOIdxMax (the picture parameter set RBSP becomes the active picture parameter set RBSP) and the access unit does not contain an activating buffering period SEI message, it is activated. This subset sequence parameter set RBSP is called the active MVC sequence parameter set RBSP until it is deactivated when another MVC sequence parameter set RBSP becomes the active MVC sequence parameter set RBSP. A subset sequence parameter set RBSP, with that particular value of seq_parameter_set_id, shall be available to the decoding process prior to its activation.

When a subset sequence parameter set RBSP (nal_unit_type is equal to 15) with a particular value of seq_parameter_set_id is not already the active MVC sequence parameter set RBSP and it is referred to by an activating buffering period SEI message (using that value of seq_parameter_set_id) that is included in an MVC scalable nesting SEI message, it is activated. This subset sequence parameter set RBSP is called the active MVC sequence parameter set RBSP until it is deactivated when another MVC sequence parameter set RBSP becomes the active MVC sequence parameter set RBSP. A subset sequence parameter set RBSP, with that particular value of seq_parameter_set_id, shall be available to the decoding process prior to its activation.

NOTE 3 – The active MVC sequence parameter set RBSP is either a sequence parameter set RBSP or a subset sequence parameter set RBSP. Sequence parameter set RBSPs are activated by coded slice NAL units with nal_unit_type equal to 1 or 5 or buffering period SEI messages that are not included in an MVC scalable nesting SEI message. Subset sequence parameter sets are activated by coded slice MVC extension NAL units (nal_unit_type equal to 20) or buffering period SEI messages that are included in an MVC scalable nesting SEI message. A sequence parameter set RBSP and a subset sequence parameter set RBSP may have the same value of seq_parameter_set_id.

For the following specification, the activating view buffering period SEI message for a particular value of VOIdx is specified as follows:

– If the access unit contains one or more than one buffering period SEI message included in an MVC scalable nesting SEI message and associated with an operation point for which the greatest VOIdx in the associated bitstream subset is equal to the particular value of VOIdx, the first of these buffering period SEI messages, in decoding order, is the activating view buffering period SEI message for the particular value of VOIdx.

– Otherwise, if the access unit contains a buffering period SEI message not included in an MVC scalable nesting SEI message, this buffering period SEI message is the activating view buffering period SEI message for the particular value of VOIdx equal to VOIdxMin.

– Otherwise, the access unit does not contain an activating buffering period SEI message for the particular value of VOIdx.

When a sequence parameter set RBSP (nal_unit_type is equal to 7) with a particular value of seq_parameter_set_id is not already the active view MVC sequence parameter set RBSP for VOIdx equal to VOIdxMin and VOIdxMax is greater than VOIdxMin and it is referred to by activation of a picture parameter set RBSP (using that value of seq_parameter_set_id) and the picture parameter set RBSP is activated by a coded slice NAL unit with nal_unit_type equal to 1 or 5 (the picture parameter set RBSP becomes the active view picture parameter set RBSP for VOIdx equal to VOIdxMin), it is activated for view components with VOIdx equal to VOIdxMin. This sequence parameter set RBSP is called the active view MVC sequence parameter set RBSP for VOIdx equal to VOIdxMin until it is deactivated when another MVC sequence parameter set RBSP becomes the active view MVC sequence parameter set RBSP for VOIdx equal to VOIdxMin or when decoding an access unit with VOIdxMax equal to VOIdxMin, whichever is earlier. A sequence parameter set RBSP, with that particular value of seq_parameter_set_id, shall be available to the decoding process prior to its activation.

When a sequence parameter set RBSP (nal_unit_type is equal to 7) with a particular value of seq_parameter_set_id is not already the active view MVC sequence parameter set RBSP for VOIdx equal to VOIdxMin and VOIdxMax is greater than VOIdxMin and it is referred to by an activating view buffering period SEI message (using that value of seq_parameter_set_id) that is not included in an MVC scalable nesting SEI message, the sequence parameter set RBSP is activated for view components with VOIdx equal to VOIdxMin. This sequence parameter set RBSP is called the active view MVC sequence parameter set RBSP for VOIdx equal to VOIdxMin until it is deactivated when another MVC sequence parameter set RBSP becomes the active view MVC sequence parameter set RBSP for VOIdx equal to VOIdxMin or when decoding an access unit with VOIdxMax equal to VOIdxMin. A sequence parameter set RBSP, with that particular value of seq_parameter_set_id, shall be available to the decoding process prior to its activation.

When a subset sequence parameter set RBSP (nal_unit_type is equal to 15) with a particular value of seq_parameter_set_id is not already the active view MVC sequence parameter set RBSP for a particular value of VOIdx less than VOIdxMax and it is referred to by activation of a picture parameter set RBSP (using that value of seq_parameter_set_id) and the picture parameter set RBSP is activated by a coded slice MVC extension NAL unit (nal_unit_type equal to 20) with the particular value of VOIdx (the picture parameter set RBSP becomes the active view picture parameter set RBSP for the particular value of VOIdx), it is activated for view components with the particular value of VOIdx. This subset sequence parameter set RBSP is called the active view MVC sequence parameter set RBSP for the particular value of VOIdx until it is deactivated when another MVC sequence parameter set RBSP becomes the active view MVC sequence parameter set RBSP for the particular value of VOIdx or when decoding an access unit with VOIdxMax less than or equal to the particular value of VOIdx. A subset sequence parameter set RBSP, with that particular value of seq_parameter_set_id, shall be available to the decoding process prior to its activation.

When a subset sequence parameter set RBSP (nal_unit_type is equal to 15) with a particular value of seq_parameter_set_id is not already the active view MVC sequence parameter set RBSP for a particular value of VOIdx less than VOIdxMax and it is referred to by an activating view buffering period SEI message (using that value of seq_parameter_set_id) that is included in an MVC scalable nesting SEI message and associated with the particular value of VOIdx, this subset sequence parameter set RBSP is activated for view components with the particular value of VOIdx. This subset sequence parameter set RBSP is called the active view MVC sequence parameter set RBSP for the particular value of VOIdx until it is deactivated when another MVC sequence parameter set RBSP becomes the active view MVC sequence parameter set RBSP for the particular value of VOIdx or when decoding an access unit with VOIdxMax less than or equal to the particular value of VOIdx. A subset sequence parameter set RBSP, with that particular value of seq_parameter_set_id, shall be available to the decoding process prior to its activation.

An MVC sequence parameter set RBSP that includes a value of profile_idc not specified in Annex A or Annex H shall not be referred to by activation of a picture parameter set RBSP as the active picture parameter set RBSP or as active view picture parameter set RBSP (using that value of seq_parameter_set_id) or referred to by a buffering period SEI message (using that value of seq_parameter_set_id). An MVC sequence parameter set RBSP including a value of profile_idc not specified in Annex A or Annex H is ignored in the decoding for profiles specified in Annex A or Annex H.

It is a requirement of bitstream conformance that the following constraints are obeyed:

– For each particular value of VOIdx, all coded slice NAL units of a coded video sequence shall refer to the same value of seq_parameter_set_id (via the picture parameter set RBSP that is referred to by the value of pic_parameter_set_id).

– The value of seq_parameter_set_id in a buffering period SEI message that is not included in an MVC scalable nesting SEI message shall be identical to the value of seq_parameter_set_id in the picture parameter set RBSP that is referred to by coded slice NAL units (with nal_unit_type equal to 1 or 5) (via the value of pic_parameter_set_id) in the same access unit.

– The value of seq_parameter_set_id in a buffering period SEI message that is included in an MVC scalable nesting SEI message and is associated with a particular value of VOIdx shall be identical to the value of seq_parameter_set_id in the picture parameter set RBSP that is referred to by coded slice NAL units with the particular value of VOIdx (via the value of pic_parameter_set_id) in the same access unit.

The active view MVC sequence parameter set RBSPs for different values of VOIdx may be the same MVC sequence parameter set RBSP. The active MVC sequence parameter set RBSP and an active view MVC sequence parameter set RBSP for a particular value of VOIdx may be the same MVC sequence parameter set RBSP.

When the active MVC sequence parameter set RBSP for a coded picture is a sequence parameter set RBSP, any sequence parameter set RBSP in the coded video sequence containing this coded picture and with the value of seq_parameter_set_id for the active MVC sequence parameter set RBSP shall have the same content as that of the active MVC sequence parameter set RBSP.

When the active MVC sequence parameter set RBSP for a coded picture is a subset sequence parameter set RBSP, any subset sequence parameter set RBSP in the coded video sequence containing this coded picture and with the value of seq_parameter_set_id for the active MVC sequence parameter set RBSP shall have the same content as that of the active MVC sequence parameter set RBSP.

For each particular value of VOIdx, the following applies:

– When the active view MVC sequence parameter set RBSP for a coded picture is a sequence parameter set RBSP, any sequence parameter set RBSP in the coded video sequence containing this coded picture and with the value of seq_parameter_set_id for the active view MVC sequence parameter set RBSP shall have the same content as that of the active view MVC sequence parameter set RBSP.

– When the active view MVC sequence parameter set RBSP for a coded picture is a subset sequence parameter set RBSP, any subset sequence parameter set RBSP in the coded video sequence containing this coded picture and with the value of seq_parameter_set_id for the active view MVC sequence parameter set RBSP shall have the same content as that of the active view MVC sequence parameter set RBSP.

NOTE 4 – If picture parameter set RBSPs or MVC sequence parameter set RBSPs are conveyed within the bitstream, these constraints impose an order constraint on the NAL units that contain the picture parameter set RBSPs or MVC sequence parameter set RBSPs, respectively. Otherwise (picture parameter set RBSPs or MVC sequence parameter set RBSPs are conveyed by other means not specified in this Recommendation | International Standard), they must be available to the decoding process in a timely fashion such that these constraints are obeyed.

When a sequence parameter set NAL unit with a particular value of seq_parameter_set_id is received, its content replaces the content of the previous sequence parameter set NAL unit, in decoding order, with the same value of seq_parameter_set_id (when a previous sequence parameter set NAL unit with the same value of seq_parameter_set_id was present in the bitstream). When a subset sequence parameter set NAL unit with a particular value of seq_parameter_set_id is received, its content replaces the content of the previous subset sequence parameter set NAL unit, in decoding order, with the same value of seq_parameter_set_id (when a previous subset sequence parameter set NAL unit with the same value of seq_parameter_set_id was present in the bitstream).

NOTE 5 – A decoder must be capable of simultaneously storing the contents of the sequence parameter sets and subset sequence parameter sets for all values of seq_parameter_set_id. The content of the sequence parameter set with a particular value of seq_parameter_set_id is overwritten when a new sequence parameter set NAL unit with the same value of seq_parameter_set_id is received, and the content of the subset sequence parameter set with a particular value of seq_parameter_set_id is overwritten when a new subset sequence parameter set NAL unit with the same value of seq_parameter_set_id is received.

When present, a sequence parameter set extension RBSP includes parameters having a similar function to those of a sequence parameter set RBSP. For purposes of establishing constraints on the syntax elements of the sequence parameter set extension RBSP and for purposes of determining activation of a sequence parameter set extension RBSP, the sequence parameter set extension RBSP shall be considered part of the preceding sequence parameter set RBSP with the same value of seq_parameter_set_id. When a sequence parameter set RBSP is present that is not followed by a sequence parameter set extension RBSP with the same value of seq_parameter_set_id prior to the activation of the sequence parameter set RBSP, the sequence parameter set extension RBSP and its syntax elements shall be considered not present for the active MVC sequence parameter set RBSP. The contents of sequence parameter set extension RBSPs only apply when the base view, which conforms to one or more of the profiles specified in Annex A, of a coded video sequence conforming to one or more profiles specified in Annex H is decoded. Subset sequence parameter set RBSPs shall not be followed by a sequence parameter set extension RBSP.

NOTE 6 – Sequence parameter sets extension RBSPs are not considered to be part of a subset sequence parameter set RBSP and subset sequence parameter set RBSPs must not be followed by a sequence parameter set extension RBSP.

For view components with VOIdx equal to VOIdxMax, all constraints that are expressed on the relationship between the values of the syntax elements (and the values of variables derived from those syntax elements) in MVC sequence parameter sets and picture parameter sets and other syntax elements are expressions of constraints that apply only to the active MVC sequence parameter set and the active picture parameter set. For view components with a particular value of VOIdx less than VOIdxMax, all constraints that are expressed on the relationship between the values of the syntax elements (and the values of variables derived from those syntax elements) in MVC sequence parameter sets and picture parameter sets and other syntax elements are expressions of constraints that apply only to the active view MVC sequence parameter set and the active view picture parameter set for the particular value of VOIdx. If any MVC sequence parameter set RBSP having profile_idc equal to the value of one of the profile_idc values specified in Annex A or Annex H is present that is never activated in the bitstream (i.e., it never becomes the active MVC sequence parameter set or an active view MVC sequence parameter set), its syntax elements shall have values that would conform to the specified constraints if it were activated by reference in an otherwise-conforming bitstream. If any picture parameter set RBSP is present that is never activated in the bitstream (i.e., it never becomes the active picture parameter set or an active view picture parameter set), its syntax elements shall have values that would conform to the specified constraints if it were activated by reference in an otherwise-conforming bitstream.

During operation of the decoding process (see subclause H.8), for view components with VOIdx equal to VOIdxMax, the values of parameters of the active picture parameter set and the active MVC sequence parameter set shall be considered in effect. For view components with a particular value of VOIdx less than VOIdxMax, the values of the parameters of the active view picture parameter set and the active view MVC sequence parameter set for the particular value of VOIdx shall be considered in effect. For interpretation of SEI messages that apply to the entire access unit or the view component with VOIdx equal to VOIdxMax, the values of the parameters of the active picture parameter set and the active MVC sequence parameter set for the same access unit shall be considered in effect unless otherwise specified in the SEI message semantics. For interpretation of SEI messages that apply to view components with a particular value of VOIdx less than VOIdxMax, the values of the parameters of the active view picture parameter set and the active view MVC sequence parameter set for the particular value of VOIdx for the same access unit shall be considered in effect unless otherwise specified in the SEI message semantics.

H.7.4.1.2.2 Order of access units and association to coded video sequences

The specification of subclause 7.4.1.2.2 applies with the following modifications.

The first access unit of the bitstream shall only contain coded slice NAL units with nal_unit_type equal to 5 or non_idr_flag equal to 0.

The order of NAL units and coded pictures and their association to access units is described in subclause H.7.4.1.2.3.

H.7.4.1.2.3 Order of NAL units and coded pictures and association to access units

The specification of subclause 7.4.1.2.3 applies with the following modifications.

NOTE – Some bitstreams that conform to one or more profiles specified in this annex do not conform to any profile specified in Annex A (prior to operation of the base view extraction process specified in subclause H.8.5.4). As specified in subclauses 7.4.1 and 7.4.1.2.3, for the profiles specified in Annex A, NAL units with nal_unit_type equal to 20 are classified as non-VCL NAL units that must be preceded within each access unit by at least one NAL unit with nal_unit_type in the range of 1 to 5, inclusive. For this reason, any bitstream that conforms to one or more profiles specified in this annex does not conform to any profile specified in Annex A when it contains any of the following:

- any access unit that does not contain any NAL units with nal_unit_type equal to 1 or 5, but contains one or more NAL units with nal_unit_type equal to 6, 7, 8, 9, or 15;
- any access unit in which one or more NAL units with nal_unit_type equal to 7, 8, or 15 is present after the last NAL unit in the access unit with nal_unit_type equal to 1 or 5.

The association of VCL NAL units to primary or redundant coded pictures is specified in subclause H.7.4.1.2.5.

The constraints for the detection of the first VCL NAL unit of a primary coded picture are specified in subclause H.7.4.1.2.4.

The constraint expressed in subclause 7.4.1.2.3 on the order of a buffering period SEI message is replaced by the following constraints.

- When an SEI NAL unit containing a buffering period SEI message is present, the following applies:
  - If the buffering period SEI message is the only buffering period SEI message in the access unit and it is not included in an MVC scalable nesting SEI message, the buffering period SEI message shall be the first SEI message payload of the first SEI NAL unit in the access unit.
  - Otherwise (the buffering period SEI message is not the only buffering period SEI message in the access unit or it is included in an MVC scalable nesting SEI message), the following constraints are specified:
    - When a buffering period SEI message that is not included in an MVC scalable nesting SEI message is present, this buffering period SEI message shall be the only SEI message payload of the first SEI NAL unit in the access unit.
    - An MVC scalable nesting SEI message that includes a buffering period SEI message shall not include any other SEI messages and shall be the only SEI message inside the SEI NAL unit.
    - All SEI NAL units that precede an SEI NAL unit that contains an MVC scalable nesting SEI message with a buffering period SEI message as payload in an access unit shall only contain buffering period SEI messages or MVC scalable nesting SEI messages with a buffering period SEI message as payload.

Each prefix NAL unit shall be immediately followed by a NAL unit with nal_unit_type equal to 1 or 5.

H.7.4.1.2.4 Detection of the first VCL NAL unit of a primary coded picture

This subclause specifies constraints on VCL NAL unit syntax that are sufficient to enable the detection of the first VCL NAL unit of each primary coded picture.

The first VCL NAL unit of the primary coded picture of the current access unit, in decoding order, shall be different from the last VCL NAL unit of the primary coded picture of the previous access unit, in decoding order, in one or more of the following ways:

- view_id of the first VCL NAL unit of the primary coded picture of the current access unit is different from view_id of the last VCL NAL unit of the primary coded picture of the previous access unit, and VOIdx of the first VCL

NAL unit of the primary coded picture of the current access unit is less than VOIdx of the last VCL NAL unit of the primary coded picture of the previous access unit.

– view_id of the first VCL NAL unit of the primary coded picture of the current access unit is equal to view_id of the last VCL NAL unit of the primary coded picture of the previous access unit, and any of the conditions specified in subclause 7.4.1.2.4 is fulfilled.

H.7.4.1.2.5 Order of VCL NAL units and association to coded pictures

Each VCL NAL unit is part of a coded picture.

Let voIdx be the value of VOIdx of any particular VCL NAL unit. The order of the VCL NAL units within a coded picture is constrained as follows:

– For all VCL NAL units following this particular VCL NAL unit, the value of VOIdx shall be greater than or equal to voIdx.

For each set of VCL NAL units within a view component, the following applies:

– If arbitrary slice order, as specified in Annex A or subclause H.10, is allowed, coded slice NAL units of a view component may have any order relative to each other.

– Otherwise (arbitrary slice order is not allowed), coded slice NAL units of a slice group shall not be interleaved with coded slice NAL units of another slice group and the order of coded slice NAL units within a slice group shall be in the order of increasing macroblock address for the first macroblock of each coded slice NAL unit of the same slice group.

NAL units having nal_unit_type equal to 12 may be present in the access unit but shall not precede the first VCL NAL unit of the primary coded picture within the access unit.

NAL units having nal_unit_type equal to 0 or in the range of 24 to 31, inclusive, which are unspecified, may be present in the access unit but shall not precede the first VCL NAL unit of the primary coded picture within the access unit.

NAL units having nal_unit_type in the range of 21 to 23, inclusive, which are reserved, shall not precede the first VCL NAL unit of the primary coded picture within the access unit (when specified in the future by ITU-T | ISO/IEC).

H.7.4.2 Raw byte sequence payloads and RBSP trailing bits semantics

H.7.4.2.1 Sequence parameter set RBSP semantics

The semantics specified in subclause 7.4.2.1 apply.

H.7.4.2.1.1 Sequence parameter set data semantics

For all syntax elements other than max_num_ref_frames, the semantics specified in subclause 7.4.2.1.1 apply with the substitution of MVC sequence parameter set for sequence parameter set. All constraints specified in subclause 7.4.2.1.1 apply only to the view components for which the MVC sequence parameter set is the active MVC sequence parameter set or the active view MVC sequence parameter set as specified in subclause H.7.4.1.2.1.

For each coded video sequence, the active MVC sequence parameter set and all active view MVC sequence parameter sets (if any) shall have the same values of pic_width_in_mbs_minus1, pic_height_in_map_units_minus1, and frame_mbs_only_flag.

When the seq_parameter_set_data( ) syntax structure is present in a subset sequence parameter set RBSP and vui_parameters_present_flag is equal to 1, timing_info_present_flag shall be equal to 0, nal_hrd_parameters_present_flag shall be equal to 0, vcl_hrd_parameters_present_flag shall be equal to 0, and pic_struct_present_flag shall be equal to 0. The value of 1 for timing_info_present_flag, nal_hrd_parameters_present_flag, vcl_hrd_parameters_present_flag, and pic_struct_present_flag for subset sequence parameter set RBSPs is reserved for future use by ITU-T | ISO/IEC. When timing_info_present_flag is equal to 1, decoders shall ignore the values of the directly following num_units_in_tick, time_scale, fixed_frame_rate_flag syntax elements. When nal_hrd_parameters_present_flag is equal to 1, decoders shall ignore the value of the syntax elements in the directly following hrd_parameters( ) syntax structure. When vcl_hrd_parameters_present_flag is equal to 1, decoders shall ignore the value of the syntax elements in the directly following hrd_parameters( ) syntax structure.

If max_num_ref_frames is included in a sequence parameter set, the semantics specified in subclause 7.4.2.1.1 apply. Otherwise (max_num_ref_frames is included in a subset sequence parameter set), the following is specified:

max_num_ref_frames specifies the maximum number of short-term and long-term reference frames, complementary reference field pairs, and non-paired reference fields that may be used by the decoding process for inter prediction of any view component in the coded video sequence. max_num_ref_frames also determines the sliding window size of the sliding window operation as specified in subclause H.8.3. The value of max_num_ref_frames shall be in the range of 0 to 16, inclusive.

H.7.4.2.1.1.1 Scaling list semantics

The semantics specified in subclause 7.4.2.1.1.1 apply.

H.7.4.2.1.2 Sequence parameter set extension RBSP semantics

The semantics specified in subclause 7.4.2.1.2 apply. Additionally, the following applies.

Sequence parameter set extension RBSPs can only follow sequence parameter set RBSPs in decoding order. Subset sequence parameter set RBSPs shall not be followed by a sequence parameter set extension RBSP. The contents of sequence parameter set extension RBSPs only apply when the base view, which conforms to one or more of the profiles specified in Annex A, of a coded video sequence conforming to one or more profiles specified in Annex H is decoded.

H.7.4.2.1.3 Subset sequence parameter set RBSP semantics

The semantics specified in subclause 7.4.2.1.3 apply.

H.7.4.2.1.4 Sequence parameter set MVC extension semantics

The sequence parameter set MVC extension specifies inter-view dependency relationships for the coded video sequence. The sequence parameter set MVC extension also specifies level values for a subset of the operation points for the coded video sequence. All sequence parameter set MVC extensions that are referred to by a coded video sequence shall be identical.

Some views identified by view_id[i] may not be present in the coded video sequence.

> NOTE 1 – Some views or temporal subsets described by the sequence parameter set MVC extension may have been removed from the original coded video sequence, hence may not be present in the coded video sequence. However, the information in the sequence parameter set MVC extension always applies to the remaining views and temporal subsets.

num_views_minus1 plus 1 specifies the maximum number of coded views in the coded video sequence. The value of num_view_minus1 shall be in the range of 0 to 1023, inclusive.

> NOTE 2 – The actual number of views in the coded video sequence may be less than num_views_minus1 plus 1.

view_id[i **]** specifies the view_id of the view with VOIdx equal to i. The value of view_id[i] shall be in the range of 0 to 1023, inclusive.

num_anchor_refs_l0[i **]** specifies the number of view components for inter-view prediction in the initial reference picture list RefPicList0 (which is derived as specified in subclause H.8.2.1) in decoding anchor view components with VOIdx equal to i. The value of num_anchor_refs_l0[i] shall not be greater than Min(15, num_views_minus1). The value of num_anchor_refs_l0[0] shall be equal to 0.

anchor_ref_l0[i **][** j **]** specifies the view_id of the j-th view component for inter-view prediction in the initial reference picture list RefPicList0 (which is derived as specified in subclause H.8.2.1) in decoding anchor view components with VOIdx equal to i. The value of anchor_ref_l0[i][j] shall be in the range of 0 to 1023, inclusive.

num_anchor_refs_l1[i **]** specifies the number of view components for inter-view prediction in the initial reference picture list RefPicList1 (which is derived as specified in subclause H.8.2.1) in decoding anchor view components with VOIdx equal to i. The value of num_anchor_refs_l1[i] shall not be greater than Min(15, num_views_minus1). The value of num_anchor_refs_l1[0] shall be equal to 0.

anchor_ref_l1[i **][** j **]** specifies the view_id of the j-th view component for inter-view prediction in the initial reference picture list RefPicList1 (which is derived as specified in subclause H.8.2.1) in decoding an anchor view component with VOIdx equal to i. The value of anchor_ref_l1[i][j] shall be in the range of 0 to 1023, inclusive.

num_non_anchor_refs_l0[i **]** specifies the number of view components for inter-view prediction in the initial reference picture list RefPicList0 (which is derived as specified in subclause H.8.2.1) in decoding non-anchor view components with VOIdx equal to i. The value of num_non_anchor_refs_l0[i] shall not be greater than Min(15, num_views_minus1). The value of num_non_anchor_refs_l0[0] shall be equal to 0.

non_anchor_ref_l0[i **][** j **]** specifies the view_id of the j-th view component for inter-view prediction in the initial reference picture list RefPicList0 (which is derived as specified in subclause H.8.2.1) in decoding non-anchor view components with VOIdx equal to i. The value of non_anchor_ref_l0[i][j] shall be in the range of 0 to 1023, inclusive.

num_non_anchor_refs_l1[i **]** specifies the number of view components for inter-view prediction in the initial reference picture list RefPicList1 (which is derived as specified in subclause H.8.2.1) in decoding non-anchor view components with VOIdx equal to i. The value of num_non_anchor_refs_l1[i] shall not be greater than Min(15, num_views_minus1). The value of num_non_anchor_refs_l1[0] shall be equal to 0.

non_anchor_ref_l1[i **][** j **]** specifies the view_id of the j-th view component for inter-view prediction in the initial reference picture list RefPicList1 (which is derived as specified in subclause H.8.2.1) in decoding non-anchor view components with VOIdx equal to i. The value of non_anchor_ref_l1[i][j] shall be in the range of 0 to 1023, inclusive.

For any particular view with view_id equal to vId1 and VOIdx equal to vOIdx1 and another view with view_id equal to vId2 and VOIdx equal to vOIdx2, when vId2 is equal to the value of one of non_anchor_ref_l0[vOIdx1][j] for all j in the range of 0 to num_non_anchor_refs_l0[vOIdx1], exclusive, or one of non_anchor_ref_l1[vOIdx1][j] for all j in the range of 0 to num_non_anchor_refs_l1[vOIdx1], exclusive, vId2 shall also be equal to the value of one of anchor_ref_l0[vOIdx1][j] for all j in the range of 0 to num_anchor_refs_l0[vOIdx1], exclusive, or one of anchor_ref_l1[vOIdx1][j] for all j in the range of 0 to num_anchor_refs_l1[vOIdx1], exclusive.

NOTE 3 – The inter-view dependency for non-anchor view components is a subset of that for anchor view components.

num_level_values_signalled_minus1 plus 1 specifies the number of level values signalled for the coded video sequence. The value of num_level_values_signalled_minus1 shall be in the range of 0 to 63, inclusive.

level_idc[i **]** specifies the i-th level value signalled for the coded video sequence.

num_applicable_ops_minus1[i **]** plus 1 specifies the number of operation points to which the level indicated by level_idc[i] applies. The value of num_applicable_ops_minus1[i] shall be in the range of 0 to 1023, inclusive.

applicable_op_temporal_id[i **][** j **]** specifies the temporal_id of the j-th operation point to which the level indicated by level_idc[i] applies.

applicable_op_num_target_views_minus1[i **][** j **]** plus 1 specifies the number of target output views for the j-th operation point to which the level indicated by level_idc[i] applies. The value of applicable_op_num_target_views_minus1[i][j] shall be in the range of 0 to 1023, inclusive.

applicable_op_target_view_id[i **][** j **][** k **]** specifies the k-th target output view for the j-th operation point to which the level indicated by level_idc[i] applies. The value of applicable_op_target_view_id[i][j][k] shall be in the range of 0 to 1023, inclusive.

Let maxTId be the greatest temporal_id of all NAL units in the coded video sequence, and vId be view_id of any view in the coded video sequence. There shall be one set of applicable_op_temporal_id[i][j], applicable_op_num_target_views_minus1[i][j], and applicable_op_target_view_id[i][j][k], for any i and j and all k for the i and j, in which applicable_op_temporal_id[i][j] is equal to maxTId, applicable_op_num_target_views_minus1[i][j] is equal to 0, and applicable_op_target_view_id[i][j][k] is equal to vId.

NOTE 4 – The above constraint ensures that the level that applies to each operation point consisting of only one target output view with the greatest highest temporal_id in the coded video sequence is signalled by one of the level_idc[i] for all i.

NOTE 5 – Some operation points identified by applicable_op_temporal_id[i][j], applicable_op_num_target_views_minus1[i][j], and applicable_op_target_view_id[i][j][k], for all i, j, and k, may not be present in the coded video sequence.

applicable_op_num_views_minus1[i **][** j **]** plus 1 specifies the number of views required for decoding the target output views corresponding to the j-th operation point to which the level indicated by level_idc[i] applies. The number of views specified by applicable_op_num_views_minus1 includes the target output views and the views that the target output views depend on as specified by the sub-bitstream extraction process in subclause H.8.5 with tIdTarget equal to applicable_op_temporal_id[i][j] and viewIdTargetList equal to the list of applicable_op_target_view_id[i][j][k] for all k in the range of 0 to applicable_op_num_target_views_minus1[i][j], inclusive, as inputs. The value of applicable_op_num_views_minus1[i][j] shall be in the range of 0 to 1023, inclusive.

H.7.4.2.2 Picture parameter set RBSP semantics

The semantics specified in subclause 7.4.2.2 apply with substituting MVC sequence parameter set for sequence parameter set. All constraints specified in subclause 7.4.2.2 apply only to the view components for which the picture parameter set is the active picture parameter set or the active view picture parameter set as specified in subclause H.7.4.1.2.1.

weighted_bipred_idc has the same semantics as specified in subclause 7.4.2.2 with the following modification.

When there is at least one inter-view prediction reference, which belongs to the same access unit as the current view component, in RefPicList0 or RefPicList1, weighted_bipred_idc shall not be equal to 2.

H.7.4.2.3 Supplemental enhancement information RBSP semantics

The semantics specified in subclause 7.4.2.3 apply.

H.7.4.2.3.1 Supplemental enhancement information message semantics

The semantics specified in subclause 7.4.2.3.1 apply.

H.7.4.2.4 Access unit delimiter RBSP semantics

The semantics specified in subclause 7.4.2.4 apply.

NOTE – The value of primary_pic_type applies to the slice_type values in all slice headers of the primary coded picture, including the slice_type syntax elements in all NAL units with nal_unit_type equal to 1, 5, or 20. NAL units with nal_unit_type equal to 2 are not present in bitstreams conforming to any of the profiles specified in this annex.

H.7.4.2.5 End of sequence RBSP semantics

The semantics specified in subclause 7.4.2.5 apply.

H.7.4.2.6 End of stream RBSP semantics

The semantics specified in subclause 7.4.2.6 apply.

H.7.4.2.7 Filler data RBSP semantics

The semantics specified in subclause 7.4.2.7 apply with the following addition.

Filler data NAL units shall be considered to contain the syntax elements priority_id, view_id, and temporal_id with values that are inferred as follows:

1. Let prevMvcNalUnit be the most recent NAL unit in decoding order that has nal_unit_type equal to 14 or 20.

   NOTE – The most recent NAL unit in decoding order with nal_unit_type equal to 14 or 20 always belongs to the same access unit as the filler data NAL unit.

2. The values of priority_id, view_id, and temporal_id for the filler data NAL unit are inferred to be equal to the values of priority_id, view_id, and temporal_id, respectively, of the NAL unit prevMvcNalUnit.

H.7.4.2.8 Slice layer without partitioning RBSP semantics

The semantics specified in subclause 7.4.2.8 apply.

H.7.4.2.9 Slice data partition RBSP semantics

Slice data partition syntax is not present in bitstreams conforming to one or more of the profiles specified in Annex H.

H.7.4.2.10 RBSP slice trailing bits semantics

The semantics specified in subclause 7.4.2.10 apply with the following modifications.

Let NumBytesInVclNALunits be the sum of the values of NumBytesInNALunit for all VCL NAL units of a view component and let BinCountsInNALunits be the number of times that the parsing process function DecodeBin( ), specified in subclause 9.3.3.2, is invoked to decode the contents of all VCL NAL units of the view component. When entropy_coding_mode_flag is equal to 1, it is a requirement of bitstream conformance that BinCountsInNALunits shall not exceed $(32 \div 3) * \text{NumBytesInVclNALunits} + (\text{RawMbBits} * \text{PicSizeInMbs}) \div 32$.

NOTE – The constraint on the maximum number of bins resulting from decoding the contents of the slice layer NAL units of a view component can be met by inserting a number of cabac_zero_word syntax elements to increase the value of NumBytesInVclNALunits. Each cabac_zero_word is represented in a NAL unit by the three-byte sequence 0x000003 (as a result of the constraints on NAL unit contents that result in requiring inclusion of an emulation_prevention_three_byte for each cabac_zero_word).

H.7.4.2.11 RBSP trailing bits semantics

The semantics specified in subclause 7.4.2.11 apply.

H.7.4.2.12 Prefix NAL unit RBSP semantics

The semantics specified in subclause 7.4.2.12 apply.

H.7.4.2.13 Slice layer extension RBSP semantics

The semantics specified in subclause 7.4.2.13 apply.

H.7.4.3 Slice header semantics

The semantics specified in subclause 7.4.3 apply with the following modifications.

All constraints specified in subclause 7.4.3 apply only to the view components with the same value of VOIdx.

The value of the following MVC sequence parameter set syntax elements shall be the same across all coded slice NAL units of an access unit: chroma_format_idc.

The value of the following slice header syntax elements shall be the same across all coded slice NAL units of an access unit: field_pic_flag and bottom_field_flag.

frame_num is used as an identifier for view components and is represented by log2_max_frame_num_minus4 + 4 bits in the bitstream.

frame_num is constrained as specified in subclause 7.4.3 where this constraint applies to view components with view_id equal to the current value of view_id.

direct_spatial_mv_pred_flag has the same semantics as specified in subclause 7.4.3 with the following modification.

When RefPicList1[0] is an inter-view reference component or an inter-view only reference component, which belongs to the same access unit as the current view component, direct_spatial_mv_pred_flag shall be equal to 1.

num_ref_idx_l0_active_minus1 has the same semantics as specified in subclause 7.4.3 with the following modification.

The range of num_ref_idx_l0_active_minus1 is specified as follows:

– If num_views_minus1 is equal to 1, the following applies:
  – If field_pic_flag is equal to 0, num_ref_idx_l0_active_minus1 shall be in the range of 0 to 7, inclusive. When MbaffFrameFlag is equal to 1, num_ref_idx_l0_active_minus1 is the maximum index value for the decoding of frame macroblocks and 2 * num_ref_idx_l0_active_minus1 + 1 is the maximum index value for the decoding of field macroblocks.
  – Otherwise (field_pic_flag is equal to 1), num_ref_idx_l0_active_minus1 shall be in the range of 0 to 15, inclusive.
– Otherwise (num_views_minus1 is greater than 1), the following applies:
  – If field_pic_flag is equal to 0, num_ref_idx_l0_active_minus1 shall be in the range of 0 to 15, inclusive. When MbaffFrameFlag is equal to 1, num_ref_idx_l0_active_minus1 is the maximum index value for the decoding of frame macroblocks and 2 * num_ref_idx_l0_active_minus1 + 1 is the maximum index value for the decoding of field macroblocks.
  – Otherwise (field_pic_flag is equal to 1), num_ref_idx_l0_active_minus1 shall be in the range of 0 to 31, inclusive.

num_ref_idx_l1_active_minus1 has the same semantics as specified in subclause 7.4.3 with the following modification.

The range of num_ref_idx_l1_active_minus1 is constrained as specified in the semantics for num_ref_idx_l0_active_minus1 in this subclause with l0 and list 0 replaced by l1 and list 1, respectively.

H.7.4.3.1 Reference picture list modification semantics

The semantics specified in subclause 7.4.3.1 apply with the following modifications.

ref_pic_list_modification_flag_l0 equal to 1 specifies that the syntax element modification_of_pic_nums_idc is present for specifying reference picture list 0. ref_pic_list_modification_flag_l0 equal to 0 specifies that this syntax element is not present.

When ref_pic_list_modification_flag_l0 is equal to 1, the number of times that modification_of_pic_nums_idc is not equal to 3 following ref_pic_list_modification_flag_l0 shall not exceed num_ref_idx_l0_active_minus1 + 1.

When RefPicList0[num_ref_idx_l0_active_minus1] in the initial reference picture list produced as specified in subclause H.8.2.1 is equal to "no reference picture", ref_pic_list_modification_flag_l0 shall be equal to 1 and modification_of_pic_nums_idc shall not be equal to 3 until RefPicList0[num_ref_idx_l0_active_minus1] in the modified list produced as specified in subclause H.8.2.2 is not equal to "no reference picture".

ref_pic_list_modification_flag_l1 equal to 1 specifies that the syntax element modification_of_pic_nums_idc is present for specifying reference picture list 1. ref_pic_list_modification_flag_l1 equal to 0 specifies that this syntax element is not present.

When ref_pic_list_modification_flag_l1 is equal to 1, the number of times that modification_of_pic_nums_idc is not equal to 3 following ref_pic_list_modification_flag_l1 shall not exceed num_ref_idx_l1_active_minus1 + 1.

When decoding a slice with slice_type equal to 1 or 6 and RefPicList1[num_ref_idx_l1_active_minus1] in the initial reference picture list produced as specified in subclause H.8.2.1 is equal to "no reference picture", ref_pic_list_modification_flag_l1 shall be equal to 1 and modification_of_pic_nums_idc shall not be equal to 3 until RefPicList1[num_ref_idx_l1_active_minus1] in the modified list produced as specified in subclause H.8.2.2 is not equal to "no reference picture".

H.7.4.3.1.1 Reference picture list MVC modification semantics

The semantics specified in subclause 7.4.3.1 apply with the following modified semantics of modification_of_pic_nums_idc. In addition, the semantics of abs_diff_view_idx_minus1 specified below apply.

modification_of_pic_nums_idc together with abs_diff_pic_num_minus1, long_term_pic_num, or abs_diff_view_idx_minus1 specifies which of the reference pictures or inter-view only reference components are re-mapped. The values of modification_of_pic_nums_idc are specified in Table H-1. The value of the first modification_of_pic_nums_idc that follows immediately after ref_pic_list_modification_flag_l0 or ref_pic_list_modification_flag_l1 shall not be equal to 3.

Table H-1 – modification_of_pic_nums_idc operations for modification of reference picture lists

modification_of_pic_nums_idc	Modification specified
0	abs_diff_pic_num_minus1 is present and corresponds to a difference to subtract from a picture number prediction value
1	abs_diff_pic_num_minus1 is present and corresponds to a difference to add to a picture number prediction value
2	long_term_pic_num is present and specifies the long-term picture number for a reference picture
3	End loop for modification of the initial reference picture list
4	abs_diff_view_idx_minus1 is present and corresponds to a difference to subtract from a prediction value of the reference view index
5	abs_diff_view_idx_minus1 is present and corresponds to a difference to add to a prediction value of the reference view index

abs_diff_view_idx_minus1 plus 1 specifies the absolute difference between the reference view index to put to the current index in the reference picture list and the prediction value of the reference view index.

Let currVOIdx be the VOIdx of the current view component, and let intViewIdx be the reference view index of the target inter-view prediction reference to put to the current index in RefPicListX (X is 0 or 1). Depending on whether the current view component is an anchor view component, the following applies:

– If the current view component is an anchor view component, the view_id of the target inter-view prediction reference is equal to anchor_ref_lX[currVOIdx][intViewIdx]. For anchor view components with VOIdx equal to currVOIdx, abs_diff_view_idx_minus1 shall be in the range of 0 to Max(0, num_anchor_refs_lX[currVOIdx] − 1), inclusive.

– Otherwise (the current view component is not an anchor view component), the view_id of the target inter-view prediction reference is equal to non_anchor_ref_lX[currVOIdx] [intViewIdx]. For non-anchor view components with VOIdx equal to currVOIdx, abs_diff_view_idx_minus1 shall be in the range of 0 to Max(0, num_non_anchor_refs_lX[currVOIdx] − 1), inclusive.

The allowed values of abs_diff_view_idx_minus1 are further restricted as specified in subclause H.8.2.2.3.

H.7.4.3.2 Prediction weight table semantics

The semantics specified in subclause 7.4.3.2 apply.

H.7.4.3.3 Decoded reference picture marking semantics

The semantics specified in subclause 7.4.3.3 apply to each view independently, with "sequence parameter set" being replaced by "MVC sequence parameter set", and "primary coded picture" being replaced by "view component of the primary coded picture".

H.7.4.4 Slice data semantics

The semantics specified in subclause 7.4.4 apply.

H.7.4.5 Macroblock layer semantics

The semantics specified in subclause 7.4.5 apply.

H.7.4.5.1 Macroblock prediction semantics

The semantics specified in subclause 7.4.5.1 apply.

H.7.4.5.2 Sub-macroblock prediction semantics

The semantics specified in subclause 7.4.5.2 apply.

H.7.4.5.3 Residual data semantics

The semantics specified in subclause 7.4.5.3 apply.

H.7.4.5.3.1 Residual luma semantics

The semantics specified in subclause 7.4.5.3.1 apply.

H.7.4.5.3.2 Residual block CAVLC semantics

The semantics specified in subclause 7.4.5.3.2 apply.

H.7.4.5.3.3 Residual block CABAC semantics

The semantics specified in subclause 7.4.5.3.3 apply.

H.8 MVC decoding process

This subclause specifies the decoding process for an access unit of a coded video sequence conforming to one or more of the profiles specified in Annex H. Specifically, this subclause specifies how the decoded picture with multiple view components is derived from syntax elements and global variables that are derived from NAL units in an access unit when the decoder is decoding the operation point identified by the target temporal level and the target output views.

The decoding process is specified such that all decoders shall produce numerically identical results for the target output views. Any decoding process that produces identical results for the target output views to the process described here conforms to the decoding process requirements of this Recommendation | International Standard.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the decoding process specified in this subclause and all child processes invoked from the process specified in this subclause are the syntax elements and derived upper-case variables for the current access unit.

The target output views are either specified by external means not specified in this Specification, or, when not specified by external means, there shall be one target output view which is the base view. Let OutputVOIdxList be the list of VOIdx values, in increasing order of VOIdx, of all the target output views in one access unit. The list OutputVOIdxList shall not change within a coded video sequence.

All sub-bitstreams that can be derived using the sub-bitstream extraction process with pIdTarget equal to any value in the range of 0 to 63, inclusive, tIdTarget equal to any value in the range of 0 to 7, inclusive, viewIdTargetList consisting of any one or more viewIdTarget's identifying the views in the bitstream as inputs as specified in subclause H.8.5 shall result in a set of coded video sequences, with each coded video sequence conforming to one or more of the profiles specified in Annex A and Annex H.

Let vOIdxList be a list of integer values specifying the VOIdx values of the view components of the access unit. The variable VOIdxMax is set equal to the maximum value of the entries in the list vOIdxList, and the variable vOIdxMin is set to the minimum value of the entries in the list vOIdxList. VOIdxMax shall be the same for all access units within a coded video sequence. vOIdxMin shall be the same for all anchor access units within a coded video sequence. When the current access unit is an anchor access unit, the variable VOIdxMin is set to vOIdxMin.

The multiview video decoding process specified in this subclause is repeatedly invoked for each view component with VOIdx from vOIdxMin to VOIdxMax, inclusive, which is present in the list vOIdxList, in increasing order of VOIdx.

Outputs of the multiview video decoding process are decoded samples of the current primary coded picture including all decoded view components.

For each view component, the specifications in clause 8 apply, with the decoding processes for picture order count, reference picture lists construction and decoded reference picture marking being modified in subclauses H.8.1, H.8.2 and H.8.3, respectively. The MVC inter prediction and inter-view prediction process is specified in subclause H.8.4. Additionally, the specification of bitstream subsets is specified in subclause H.8.5.

H.8.1 MVC decoding process for picture order count

The process specified in this subclause is invoked for a particular view with view order index VOIdx. The specifications in subclause 8.2.1 apply independently for each view, with "picture" being replaced by "view component", "frame" being replaced by "frame view component", and "field" being replaced by "field view component".

The following constraints shall be obeyed:

– When the view components of an access unit have field_pic_flag equal to 0 or (field_pic_flag equal to 1 and bottom_field_flag equal to 0), it is a requirement of bitstream conformance that the bitstream shall not contain data that result in different values of TopFieldOrderCnt for the view components of the access unit.

– When the view components of an access unit have field_pic_flag equal to 0 or (field_pic_flag equal to 1 and bottom_field_flag equal to 1), it is a requirement of bitstream conformance that the bitstream shall not contain data that result in different values of BottomFieldOrderCnt for the view components of the access unit.

H.8.2 MVC decoding process for reference picture lists construction

This process is invoked at the beginning of the decoding process for each P, SP or B slice.

During the invocation of this process, when subclauses 8.2.4.1 and 8.2.4.2 are invoked, only the reference pictures having the same value of view_id as the current slice are considered. All subclauses of clause 8 are invoked with "picture" being replaced by "view component", "frame" being replaced by "frame view component", and "field" being replaced by "field view component".

Decoded reference pictures are marked as "used for short-term reference" or "used for long-term reference" as specified in subclause H.8.3. Short-term reference pictures are identified by the values of frame_num. Long-term reference pictures are assigned a long-term frame index as specified in subclause H.8.3.

In addition to reference pictures marked as "used for short-term reference" or "used for long-term reference", inter-view reference components and inter-view only reference components of the current access unit may also be included in a reference picture list. Inter-view reference components and inter-view only reference components are identified by the value of view_id.

Subclause 8.2.4.1 is invoked to specify

– the assignment of variables FrameNum, FrameNumWrap, and PicNum to each of the short-term reference pictures, and

– the assignment of variable LongTermPicNum to each of the long-term reference pictures.

Reference pictures and, when present, inter-view only reference components, are addressed through reference indices as specified in subclause 8.2.4.1. A reference index is an index into a reference picture list. When decoding a P or SP slice, there is a single reference picture list RefPicList0. When decoding a B slice, there is a second independent reference picture list RefPicList1 in addition to RefPicList0.

At the beginning of the decoding process for each slice, reference picture list RefPicList0, and for B slices RefPicList1, are derived as specified by the following ordered steps:

1. Depending on non_idr_flag, the following applies:
   – If non_idr_flag is equal to 1, the initial reference picture list RefPicList0 and for B slices RefPicList1 are derived as specified in subclause 8.2.4.2.
   – Otherwise (non_idr_flag is equal to 0), all (num_ref_idx_l0_active_minus1 + 1) entries of the initial reference picture list RefPicList0 are set equal to "no reference picture" and, for B slices, all (num_ref_idx_l1_active_minus1 + 1) entries of the initial reference picture list RefPicList1 are set equal to "no reference picture".
2. Inter-view reference components or inter-view only reference components are appended to the initial reference picture list RefPicList0 and for B slices RefPicList1 as specified in subclause H.8.2.1.
3. When ref_pic_list_modification_flag_l0 is equal to 1 or, when decoding a B slice, ref_pic_list_modification_flag_l1 is equal to 1, the reference picture list RefPicList0 and for B slices RefPicList1 are modified as specified in subclause H.8.2.2.

NOTE – The modification process for reference picture lists specified in subclause H.8.2.2 allows the contents of RefPicList0 and for B slices RefPicList1 to be modified in a flexible fashion. In particular, it is possible for a reference picture that is currently marked "used for reference" to be inserted into RefPicList0 and for B slices RefPicList1 even when the reference picture is not in the initial reference picture list derived as specified in subclauses 8.2.4.2 and H.8.2.1.

The number of entries in the modified reference picture list RefPicList0 is num_ref_idx_l0_active_minus1 + 1, and for B slices the number of entries in the modified reference picture list RefPicList1 is num_ref_idx_l1_active_minus1 + 1. A reference picture or inter-view only reference component may appear at more than one index in the modified reference picture lists RefPicList0 or RefPicList1.

During the invocation of the process specified in subclause H.8.2.1, an inter-view prediction reference appended to RefPicListX (with X being 0 or 1) may not exist. However, an inter-view prediction reference that does not exist shall not be in the modified RefPicListX after the invocation of the process specified in subclause H.8.2.2.

When anchor_pic_flag is equal to 1, the bitstream shall not contain data that result in any entry of the reference picture list RefPicList0 or, for B slices, any entry of the reference picture list RefPicList1 that does not represent a view component of the current access unit.

H.8.2.1 Initialisation process for reference picture list for inter-view prediction references

Inputs to this process are a reference picture list RefPicListX (with X being 0 or 1), inter_view_flag and view dependency information that has been decoded from the seq_parameter_set_mvc_extension( ).

The output of this process is a possibly modified reference picture list RefPicListX, which is still referred to as the initial reference picture list RefPicListX.

With i being the value of VOIdx for the current slice, inter-view reference components and inter-view only reference components (the corresponding NAL units have inter_view_flag equal to 1) are appended to the reference picture list as specified in the following.

– If the current slice has anchor_pic_flag equal to 1, for each value of reference view index j from 0 to num_anchor_refs_lX[i] − 1, inclusive, in ascending order of j, the inter-view prediction reference with view_id equal to anchor_ref_lX[i][j] from the same access unit as the current slice is appended to RefPicListX.

– Otherwise (the current slice has anchor_pic_flag equal to 0), for each value of reference view index j from 0 to num_non_anchor_refs_lX[i] − 1, inclusive, in ascending order of j, the inter-view prediction reference with view_id equal to non_anchor_ref_lX[i][j] from the same access unit as the current slice is appended to RefPicListX.

NOTE 1 – View components with inter_view_flag equal to 0 are not appended to the reference picture list.

NOTE 2 – When a NAL unit with nal_unit_type equal to 1 or 5 is not immediately preceded by a NAL unit with nal_unit_type equal to 14, the value of inter_view_flag is inferred to be equal to 1. Encoders that do not encode a prefix NAL unit before each NAL unit with nal_unit_type equal to 1 or 5 and devices that remove prefix NAL units from a bitstream should take into consideration this inferred value to avoid potential mismatches in the reference picture lists between the encoder and decoder.

Inter-view reference components and inter-view only reference components are appended to the reference picture list starting from the first entry position of "no reference picture" in the initial reference picture list RefPicListX or starting from the entry position num_ref_idx_lX_active_minus1+1 of the initial reference picture list RefPicListX, whichever is the earliest position.

When the number of entries in the initial reference picture list RefPicListX is greater than (num_ref_idx_lX_active_minus1 + 1), the extra entries past position num_ref_idx_lX_active_minus1 are discarded from the initial reference picture list RefPicListX.

H.8.2.2 Modification process for reference picture lists

Input to this process is reference picture list RefPicList0 and, when decoding a B slice, also reference picture list RefPicList1.

Outputs of this process are a possibly modified reference picture list RefPicList0 and, when decoding a B slice, also a possibly modified reference picture list RefPicList1.

When ref_pic_list_modification_flag_l0 is equal to 1, the following ordered steps are specified:

1. Let refIdxL0 be an index into the reference picture list RefPicList0. It is initially set equal to 0.

2. The corresponding syntax elements modification_of_pic_nums_idc are processed in the order they occur in the bitstream. For each of these syntax elements, the following applies:
   – If modification_of_pic_nums_idc is equal to 0 or equal to 1, the process specified in subclause H.8.2.2.1 is invoked with RefPicList0 and refIdxL0 given as input, and the output is assigned to RefPicList0 and refIdxL0.
   – Otherwise, if modification_of_pic_nums_idc is equal to 2, the process specified in subclause H.8.2.2.2 is invoked with RefPicList0 and refIdxL0 given as input, and the output is assigned to RefPicList0 and refIdxL0.
   – Otherwise, if modification_of_pic_nums_idc is equal to 4 or equal to 5, the process specified in subclause H.8.2.2.3 is invoked with RefPicList0 and refIdxL0 given as input, and the output is assigned to RefPicList0 and refIdxL0.
   – Otherwise (modification_of_pic_nums_idc is equal to 3), the modification process for reference picture list RefPicList0 is finished.

When ref_pic_list_modification_flag_l1 is equal to 1, the following ordered steps are specified:

1. Let refIdxL1 be an index into the reference picture list RefPicList1. It is initially set equal to 0.
2. The corresponding syntax elements modification_of_pic_nums_idc are processed in the order they occur in the bitstream. For each of these syntax elements, the following applies:
   – If modification_of_pic_nums_idc is equal to 0 or equal to 1, the process specified in subclause H.8.2.2.1 is invoked with RefPicList1 and refIdxL1 given as input, and the output is assigned to RefPicList1 and refIdxL1.
   – Otherwise, if modification_of_pic_nums_idc is equal to 2, the process specified in subclause H.8.2.2.2 is invoked with RefPicList1 and refIdxL1 given as input, and the output is assigned to RefPicList1 and refIdxL1.
   – Otherwise, if modification_of_pic_nums_idc is equal to 4 or equal to 5, the process specified in subclause H.8.2.2.3 is invoked with RefPicList1 and refIdxL1 given as input, and the output is assigned to RefPicList1 and refIdxL1.
   – Otherwise (modification_of_pic_nums_idc is equal to 3), the modification process for reference picture list RefPicList1 is finished.

H.8.2.2.1 Modification process of reference picture lists for short-term reference pictures for inter prediction

Inputs to this process are an index refIdxLX and a reference picture list RefPicListX (with X being 0 or 1).

Outputs of this process are an incremented index refIdxLX and a modified reference picture list RefPicListX.

The variable picNumLXNoWrap is derived as follows:

– If modification_of_pic_nums_idc is equal to 0,

```
if( picNumLXPred − ( abs_diff_pic_num_minus1 + 1 ) < 0 )
      picNumLXNoWrap = picNumLXPred − ( abs_diff_pic_num_minus1 + 1 ) + MaxPicNum          (H-1)
else
      picNumLXNoWrap = picNumLXPred − ( abs_diff_pic_num_minus1 + 1 )
```

– Otherwise (modification_of_pic_nums_idc is equal to 1),

```
if( picNumLXPred + ( abs_diff_pic_num_minus1 + 1 )  >=  MaxPicNum )
      picNumLXNoWrap = picNumLXPred + ( abs_diff_pic_num_minus1 + 1 ) − MaxPicNum          (H-2)
else
      picNumLXNoWrap = picNumLXPred + ( abs_diff_pic_num_minus1 + 1 )
```

picNumLXPred is the prediction value for the variable picNumLXNoWrap. When the process specified in this subclause is invoked the first time for a slice (that is, for the first occurrence of modification_of_pic_nums_idc equal to 0 or 1 in the ref_pic_list_modification( ) syntax), picNumL0Pred and picNumL1Pred are initially set equal to CurrPicNum. After each assignment of picNumLXNoWrap, the value of picNumLXNoWrap is assigned to picNumLXPred.

The variable picNumLX is derived as specified by the following pseudo-code:

```
if( picNumLXNoWrap > CurrPicNum )
      picNumLX = picNumLXNoWrap − MaxPicNum                                  (H-3)
else
      picNumLX = picNumLXNoWrap
```

picNumLX shall be equal to the PicNum of a reference picture that is marked as "used for short-term reference" and shall not be equal to the PicNum of a short-term reference picture that is marked as "non-existing".

The following procedure is conducted to place the picture with short-term picture number picNumLX into the index position refIdxLX, shift the position of any other remaining pictures to later in the list, and increment the value of refIdxLX:

```
for( cIdx = num_ref_idx_lX_active_minus1 + 1; cIdx > refIdxLX; cIdx−− )
      RefPicListX[ cIdx ] = RefPicListX[ cIdx − 1]
RefPicListX[ refIdxLX++ ] = short-term reference picture with PicNum equal to picNumLX
nIdx = refIdxLX
for( cIdx = refIdxLX; cIdx <= num_ref_idx_lX_active_minus1 + 1; cIdx++ )     (H-4)
      if( PicNumF( RefPicListX[ cIdx ] ) != picNumLX  | |  viewID(RefPicListX[ cIdx ] ) != currViewID )
        RefPicListX[ nIdx++ ] = RefPicListX[ cIdx ]
```

In the above, the function viewID(refpic) returns the view_id of the reference picture refpic, the variable currViewID is equal to the view_id of the current slice, and the function PicNumF(RefPicListX[cIdx]) is derived as follows:

– If the reference picture RefPicListX[cIdx] is marked as "used for short-term reference", PicNumF(RefPicListX[cIdx]) is the PicNum of the picture RefPicListX[cIdx].

– Otherwise (the reference picture RefPicListX[cIdx] is not marked as "used for short-term reference"), PicNumF(RefPicListX[cIdx]) is equal to MaxPicNum.

NOTE 1 – The value of picNumLX can never be equal to MaxPicNum.

NOTE 2 – Within this pseudo-code procedure, the length of the list RefPicListX is temporarily made one element longer than the length needed for the final list. After the execution of this procedure, only elements 0 through num_ref_idx_lX_active_minus1 of the list need to be retained.

H.8.2.2.2 Modification process of reference picture lists for long-term reference pictures for inter prediction

Inputs to this process are an index refIdxLX (with X being 0 or 1) and reference picture list RefPicListX.

Outputs of this process are an incremented index refIdxLX and a modified reference picture list RefPicListX.

The following procedure is conducted to place the picture with long-term picture number long_term_pic_num into the index position refIdxLX, shift the position of any other remaining pictures to later in the list, and increment the value of refIdxLX:

```
for( cIdx = num_ref_idx_lX_active_minus1 + 1; cIdx > refIdxLX; cIdx−− )
      RefPicListX[ cIdx ] = RefPicListX[ cIdx − 1]
RefPicListX[ refIdxLX++ ] = long-term reference picture with LongTermPicNum equal to long_term_pic_num
nIdx = refIdxLX
for( cIdx = refIdxLX; cIdx <= num_ref_idx_lX_active_minus1 + 1; cIdx++ )     (H-5)
      if( LongTermPicNumF( RefPicListX[ cIdx ] ) != long_term_pic_num | |
          viewID(RefPicListX[ cIdx ] ) != currViewID )
          RefPicListX[ nIdx++ ] = RefPicListX[ cIdx ]
```

In the above, the function viewID(refpic) returns the view_id of the reference picture refpic, the variable currViewID is equal to the view_id of the current slice, and the function LongTermPicNumF(RefPicListX[cIdx]) is derived as follows:

– If the reference picture RefPicListX[cIdx] is marked as "used for long-term reference", LongTermPicNumF(RefPicListX[cIdx]) is the LongTermPicNum of the picture RefPicListX[cIdx].

– Otherwise (the reference picture RefPicListX[cIdx] is not marked as "used for long-term reference"), LongTermPicNumF(RefPicListX[cIdx]) is equal to 2 * (MaxLongTermFrameIdx + 1).

NOTE 1 – The value of long_term_pic_num can never be equal to 2 * (MaxLongTermFrameIdx + 1).

NOTE 2 – Within this pseudo-code procedure, the length of the list RefPicListX is temporarily made one element longer than the length needed for the final list. After the execution of this procedure, only elements 0 through num_ref_idx_lX_active_minus1 of the list need to be retained.

H.8.2.2.3 Modification process for reference picture lists for inter-view prediction references

Inputs to this process are reference picture list RefPicListX (with X being 0 or 1) and an index refIdxLX into this list.

Outputs of this process are a modified reference picture list RefPicListX (with X being 0 or 1) and an incremented index refIdxLX.

Let currVOIdx be the variable VOIdx of the current slice. The variable maxViewIdx is derived as follows:

– If the current slice has anchor_pic_flag equal to 1, maxViewIdx is set equal to num_anchor_refs_lX[currVOIdx] − 1.

– Otherwise (the current slice has anchor_pic_flag equal to 0), maxViewIdx is set equal to num_non_anchor_refs_lX[currVOIdx] − 1.

The variable picViewIdxLX is derived as follows:

– If modification_of_pic_nums_idc is equal to 4,

```
if( picViewIdxLXPred − ( abs_diff_view_idx_minus1 + 1 ) < 0 )
    picViewIdxLX = picViewIdxLXPred − ( abs_diff_view_idx_minus1 + 1 ) + maxViewIdx + 1        (H-6)
else
    picViewIdxLX = picViewIdxLXPred − ( abs_diff_view_idx_minus1 + 1 )
```

– Otherwise (modification_of_pic_nums_idc is equal to 5),

```
if( picViewIdxLXPred + ( abs_diff_view_idx_minus1 + 1 ) >= maxViewIdx + 1 )
    picViewIdxLX = picViewIdxLXPred + ( abs_diff_view_idx_minus1 + 1 ) − ( maxViewIdx + 1 )    (H-7)
else
    picViewIdxLX = picViewIdxLXPred + ( abs_diff_view_idx_minus1 + 1 )
```

picViewIdxLXPred is the prediction value for the variable picViewIdxLX. When the process specified in this subclause is invoked the first time for a slice (that is, for the first occurrence of modification_of_pic_nums_idc equal to 4 or 5 in the ref_pic_list_modification( ) syntax), picViewIdxL0Pred and picViewIdxL1Pred are initially set equal to −1. After each assignment of picViewIdxLX, the value of picViewIdxLX is assigned to picViewIdxLXPred.

The bitstream shall not contain data that result in picViewIdxLX less than 0 or picViewIdxLX greater than maxViewIdx.

The variable targetViewID is derived as follows:

– If the current slice has anchor_pic_flag equal to 1,

```
targetViewID = anchor_refs_lX[ currVOIdx ][ picViewIdxLX ]        (H-8)
```

– Otherwise (the current slice has anchor_pic_flag equal to 0),

```
targetViewID = non_anchor_refs_lX[ currVOIdx ][ picViewIdxLX ]        (H-9)
```

The following procedure is conducted to place the inter-view prediction reference with reference view index equal to picViewIdxLX into the index position refIdxLX and shift the position of any other remaining pictures to later in the list:

```
for( cIdx = num_ref_idx_lX_active_minus1 + 1; cIdx > refIdxLX; cIdx−− )
    RefPicListX[ cIdx ] = RefPicListX[ cIdx − 1]
RefPicListX[ refIdxLX++ ] = inter-view prediction reference with view_id equal to targetViewID
nIdx = refIdxLX
for( cIdx = refIdxLX; cIdx <= num_ref_idx_lX_active_minus1 + 1; cIdx++ )        (H-10)
    if( viewID(RefPicListX[ cIdx ]) != targetViewID | | PictureOrderCnt(RefPicListX[ cIdx ]) != currPOC )
        RefPicListX[ nIdx++ ] = RefPicListX[ cIdx ]
```

In the above, the function viewID(refpic) returns the view_id of the reference picture refpic, the variable currViewID is equal to the view_id of the current slice, and the variable currPOC is equal to PicOrderCnt( ) of the current slice.

H.8.3 MVC decoded reference picture marking process

The process specified in this subclause is invoked for a particular view with view order index VOIdx. The specifications in subclause 8.2.5 apply with "picture" being replaced by "view component", "frame" being replaced by "frame view component", and "field" being replaced by "field view component". During the invocation of the process for a particular view, only view components of the particular view are considered. The marking of view components of other views is not changed.

NOTE – A view component of a picture may have a different marking status than other view components of the same picture.

H.8.4 MVC inter prediction and inter-view prediction process

For both inter-prediction and inter-view prediction, the specifications in subclause 8.4 apply. For the invocation of the MVC inter prediction and inter-view prediction process as specified in this subclause, the inter-view reference components and inter-view only reference components that are included in the reference picture lists are considered as not being marked as "used for short-term reference" or "used for long-term reference".

NOTE – This implies that when RefPicList1[0] represents an inter-view reference component or an inter-view only reference component, the variable colZeroFlag in subclause 8.4.1.2.2 is always derived to be equal to 0.

H.8.5 Specification of bitstream subsets

Subclauses H.8.5.1 and H.8.5.2 specify the processes for deriving required anchor and non-anchor view components, respectively, that are used in the sub-bitstream extraction process. Subclause H.8.5.3 specifies the sub-bitstream extraction process. Subclause H.8.5.4 specifies the base view bitstream subset. Subclause H.8.5.5 gives an informative example for creation of a base view in case the orignal base view in the input bitstream to the bitstream extraction process is not included in the output bitstream subset.

H.8.5.1 Derivation process for required anchor view components

This process is recursively invoked to derive the set of required anchor view components for a specified view. The view_id's of all views for which the anchor view components are required for the specified view are marked as "required for anchor" and their corresponding VOIdx values are included in VOIdxList.

Input to this process is a variable viewId, representing a view with view_id equal to viewId, with its corresponding view order index denoted by vOIdx.

Outputs of this process are the view_id equal to viewId being marked as "required for anchor", a possibly updated VOIdxList, and additional invocations of the derivation process based on the inter-view dependency for anchor view components in the view with view_id equal to viewId as specified in the sequence parameter set MVC extension.

The following ordered steps are specified:

1. Mark the view_id equal to viewId as "required for anchor" and add vOIdx to VOIdxList if the same value is not already included in VOIdxList.

2. Depending on num_anchor_refs_l0[vOIdx] and num_anchor_refs_l1[vOIdx], the following applies:

   – If both num_anchor_refs_l0[vOIdx] and num_anchor_refs_l1[vOIdx] are equal to 0, terminate this process.

   – Otherwise (num_anchor_refs_l0[vOIdx] or num_anchor_refs_l1[vOIdx] is not equal to 0), the following ordered steps are specified:

     a. When num_anchor_refs_l0[vOIdx] is not equal to 0, invoke the process specified in subclause H.8.5.1 for each viewId equal to anchor_ref_l0[vOIdx][i] for all i in the range of 0 to num_anchor_refs_l0[vOIdx] − 1, inclusive, in ascending order of i.

     b. When num_anchor_refs_l1[vOIdx] is not equal to 0, invoke the process specified in subclause H.8.5.1 for each viewId equal to anchor_ref_l1[vOIdx][i] for all i in the range of 0 to num_anchor_refs_l1[vOIdx] − 1, inclusive, in ascending order of i.

H.8.5.2 Derivation process for required non-anchor view components

This process is recursively invoked to derive the set of required non-anchor view components for a specified view. The view_id's of all views for which the non-anchor view components are required for the specified view are marked as "required for non-anchor".

Input to this process is a variable viewId, representing a view with view_id equal to viewId, with its corresponding view order index denoted by vOIdx.

Outputs of this process are the view_id equal to viewId being marked as "required for non-anchor" and additional invocations of the derivation process based on the inter-view dependency for non-anchor view components in the view with view_id equal to viewId as specified in the sequence parameter set MVC extension.

The following ordered steps are specified:

1. Mark the view_id equal to viewId as "required for non-anchor".

2. Depending on num_non_anchor_refs_l0[vOIdx] and num_non_anchor_refs_l1[vOIdx], the following applies:

   – If both num_non_anchor_refs_l0[vOIdx] and num_non_anchor_refs_l1[vOIdx] are equal to 0, terminate this process.

   – Otherwise (num_non_anchor_refs_l0[vOIdx] or num_non_anchor_l1[vOIdx] is not equal to 0), the following ordered steps are specified:

     a. When num_non_anchor_refs_l0[vOIdx] is not equal to 0, invoke the process specified in subclause H.8.5.2 for each viewId equal to non_anchor_ref_l0[vOIdx][i] for all i in the range of 0 to num_non_anchor_l0[vOIdx] − 1, inclusive, in ascending order of i.

     b. When num_non_anchor_refs_l1[vOIdx] is not equal to 0, invoke the process specified in subclause H.8.5.2 for each viewId equal to non_anchor_ref_l1[vOIdx][i] for all i in the range of 0 to num_non_anchor_l1[vOIdx] − 1, inclusive, in ascending order of i.

H.8.5.3 Sub-bitstream extraction process

It is requirement of bitstream conformance that any sub-bitstream that is the output of the process specified in this subclause with pIdTarget equal to any value in the range of 0 to 63, inclusive, tIdTarget equal to any value in the range of 0 to 7, inclusive, viewIdTargetList consisting of any one or more values of viewIdTarget identifying the views in the bitstream, shall be conforming to this Recommendation | International Standard.

NOTE 1 – A conforming bitstream contains one or more coded slice NAL units with priority_id equal to 0 and temporal_id equal to 0.

NOTE 2 – It is possible that not all operation points of sub-bitstreams resulting from the sub-bitstream extraction process have an applicable level_idc or level_idc[i]. In this case, each coded video sequence in a sub-bitstream must still conform to one or more of the profiles specified in Annex A and Annex H, but may not satisfy the level constraints specified in subclauses A.3 and H.10.2, respectively.

Inputs to this process are:

– a variable pIdTarget (when present),

– a variable tIdTarget (when present),

– a list viewIdTargetList consisting of one or more values of viewIdTarget (when present).

Outputs of this process are a sub-bitstream and a list of VOIdx values VOIdxList.

When pIdTarget is not present as input to this subclause, pIdTarget is inferred to be equal to 63.

When tIdTarget is not present as input to this subclause, tIdTarget is inferred to be equal to 7.

When viewIdTargetList is not present as input to this subclause, there shall be one value of viewIdTarget inferred in viewIdTargetList and the value of viewIdTarget is inferred to be equal to view_id of the base view.

The sub-bitstream is derived by applying the following operations in sequential order:

1. Let VOIdxList be empty and minVOIdx be the VOIdx value of the base view.

2. For each value of viewIdTarget included in viewIdTargetList, invoke the process specified in subclause H.8.5.1 with the value of viewIdTarget as input.

3. For each value of viewIdTarget included in viewIdTargetList, invoke the process specified in subclause H.8.5.2 with the value of viewIdTarget as input.

4. Mark all VCL NAL units and filler data NAL units for which any of the following conditions are true as "to be removed from the bitstream":

   – priority_id is greater than pIdTarget,

   – temporal_id is greater than tIdTarget,

   – anchor_pic_flag is equal to 1 and view_id is not marked as "required for anchor",

   – anchor_pic_flag is equal to 0 and view_id is not marked as "required for non-anchor",

   – nal_ref_idc is equal to 0 and inter_view_flag is equal to 0 and view_id is not equal to any value in the list viewIdTargetList.

5. Remove all access units for which all VCL NAL units are marked as "to be removed from the bitstream".

6. Remove all VCL NAL units and filler data NAL units that are marked as "to be removed from the bitstream".

7. When VOIdxList contains only one value of VOIdx that is equal to minVOIdx, remove the following NAL units:

   – all NAL units with nal_unit_type equal to 14 or 15,

   – all NAL units with nal_unit_type equal to 6 in which the first SEI message has payloadType in the range of 36 to 44, inclusive.

   NOTE 3 – When VOIdxList contains only one value of VOIdx equal to minVOIdx, the sub-bitstream contains only the base view or only a temporal subset of the base view.

8. Let maxTId be the maximum temporal_id of all the remaining VCL NAL units. Remove all NAL units with nal_unit_type equal to 6 that only contain SEI messages that are part of an MVC scalable nesting SEI message with any of the following properties:

   – operation_point_flag is equal to 0 and all_view_components_in_au_flag is equal to 0 and none of sei_view_id[i] for all i in the range of 0 to num_view_components_minus1, inclusive, corresponds to a VOIdx value included in VOIdxList,

   – operation_point_flag is equal to 1 and either sei_op_temporal_id is greater than maxTId or the list of sei_op_view_id[i] for all i in the range of 0 to num_view_components_op_minus1, inclusive, is not a subset of viewIdTargetList (i.e., it is not true that sei_op_view_id[i] for any i in the range of 0 to num_view_components_op_minus1, inclusive, is equal to a value in viewIdTargetList).

9. Remove each view scalability information SEI message and each operation point not present SEI message, when present.

10. When VOIdxList does not contain a value of VOIdx equal to minVOIdx, the view with VOIdx equal to the minimum VOIdx value included in VOIdxList is converted to the base view of the extracted sub-bitstream. An informative procedure that outlines key processing steps to create a base view is described in subclause H.8.5.5.

    NOTE 4 – When VOIdxList does not contain a value of VOIdx equal to minVOIdx, the resulting sub-bitstream according to the operation steps 1-9 above does not contain a base view that conforms to one or more profiles specified in Annex A. In this case, by this operation step, the remaining view with the new minimum VOIdx value is converted to be the new base view that conforms to one or more profiles specified in Annex A.

H.8.5.4 Specification of the base view bitstream

A bitstream that conforms to one or more profiles as specified in Annex H shall contain a base view bitstream that conforms to one or more of the profiles specified in Annex A. This base view bitstream is derived by invoking the sub-bitstream extraction process as specified in subclause H.8.5.3 with no input and the base view bitstream being the output.

NOTE – Although all multiview bitstreams that conform to one or more of the profiles specified in this annex contain a base view bitstream that conforms to one or more of the profiles specified in Annex A, the complete multiview bitstream (prior to operation of the base view extraction process specified in this subclause) may not conform to any profile specified in Annex A.

H.8.5.5 Creation of a base view during sub-bitstream extraction (informative)

According to the sub-bitstream extraction process specified in subclause H.8.5.3, the resulting sub-bitstream shall contain a base view. When the resulting bitstream does not contain a base view, the following procedure may be used to create a base view during sub-bitstream extraction.

When VOIdxList does not contain a value of VOIdx equal to minVOIdx, let newBaseViewId be equal to the view_id for which the VOIdx value is equal to the minimum VOIdx value included in VOIdxList, and apply the following operations in sequential order:

1. Remove all NAL units with nal_unit_type equal to 7.

2. For all subset sequence parameter set NAL units (with nal_unit_type equal to 15) that are referred to by at least one remaining VCL NAL unit with view_id equal to newBaseViewId, apply the following operations in sequential order:

   a. Set nal_unit_type to 7.

   b. Set profile_idc to 100.

   c. Set level_idc to level_idc[i], with i equal to the value that for one value of j in the range of 0 to num_applicable_ops_minus1[i], inclusive, applicable_op_temporal_id[i][j] is equal to maxTId, applicable_op_num_target_views_minus1[i][j] is equal to 0, and applicable_op_target_view_id[i][j][k] for k equal to 0 is equal to newBaseViewId.

d. Remove all the syntax elements after the syntax structure seq_parameter_set_data( ) and before the syntax structure rbsp_trailing_bits( ), and change RBSP trailing bits appropriately.

3. Remove all SEI NAL units (with nal_unit_type equal to 6) for which the first contained SEI message has payloadType in the range of 0 to 23, inclusive.

4. For each SEI NAL unit (with nal_unit_type equal to 6) containing an MVC scalable nesting SEI message, the following operations are applied in sequential order:

   a. When none of the following properties is true for the MVC scalable nesting SEI message, the SEI NAL unit is removed:

      – operation_point_flag is equal to 0 and all_view_components_in_au_flag is equal to 1,

      – operation_point_flag is equal to 0, all_view_components_in_au_flag is equal to 0, and at least one of the values of sei_view_id[i] for all i in the range of 0 to num_view_components_minus1, inclusive, is equal to the value of one of the viewIdTarget's in viewIdTargetList,

      – operation_point_flag is equal to 1, sei_op_temporal_id is equal to or less than maxtIdT, and the list of sei_op_view_id[i] for all i in the range of 0 to num_view_components_op_minus1, inclusive, is a subset of viewIdTargetList (i.e., it is true that sei_op_view_id[i] for any i in the range of 0 to num_view_components_op_minus1, inclusive, is equal to a value in viewIdTargetList).

   b. When the SEI NAL unit is not removed, the following applies:

      – If VOIdxList contains only one VOIdx value, the SEI NAL unit is replaced by an SEI NAL unit containing only the original nested SEI message not as part of an MVC scalable nesting SEI message.

      – Otherwise (VOIdxList contains more than one VOIdx value), when any of the following properties is true for the MVC scalable nesting SEI message, a new SEI NAL unit containing only the nested SEI message not as part of an MVC scalable nesting SEI message is generated and inserted immediately before the original SEI NAL unit in decoding order:

         – operation_point_flag is equal to 0 and all_view_components_in_au_flag is equal to 1,

         – operation_point_flag is equal to 0, all_view_components_in_au_flag is equal to 0, and for the values of sei_view_id[i] for all i in the range of 0 to num_view_components_minus1, inclusive, one is equal to newBaseViewId, and at least another one is equal to the value of one of the viewIdTarget's in viewIdTargetList.

5. When VOIdxList contains only one value of VOIdx, remove the following NAL units:

   – all NAL units with nal_unit_type equal to 15,

   – all NAL units with nal_unit_type equal to 6 in which the first SEI message has payloadType in the range of 36 to 44, inclusive.

6. For each NAL unit nalUnit with nal_unit_type equal to 20 and view_id equal to newBaseViewId, the following operations are applied in sequential order:

   a. Depending on non_idr_flag, the following applies:

      – If non_idr_flag is equal to 0, set nal_unit_type equal to 5.

      – Otherwise (non_idr_flag is equal to 1), set nal_unit_type equal to 1.

   b. When VOIdxList contains more than one VOIdx value, generate a prefix NAL unit with the same NAL unit header (including NAL unit header MVC extension) as the NAL unit nalUnit, except that nal_unit_type is set to 14 and priority_id may be changed, and insert the prefix NAL unit immediately before the NAL unit nalUnit in decoding order. After the last application of this operation, at least one of all the inserted prefix NAL units by the applications of this operation shall have priority_id equal to 0.

   c. Remove the NAL unit header MVC extension of nalUnit.

H.9 Parsing process

The specifications in clause 9 apply.

H.10 Profiles and levels

The specifications in Annex A apply. Additional profiles and specific values of profile_idc are specified in the following.

The profiles that are specified in subclause H.10.1 are also referred to as the profiles specified in Annex H.

H.10.1 Profiles

All constraints for picture parameter sets that are specified in the following are constraints for picture parameter sets that become the active picture parameter set or an active view picture parameter set inside the bitstream. All constraints for MVC sequence parameter sets that are specified in the following are constraints for MVC sequence parameter sets that become the active MVC sequence parameter set or an active view MVC sequence parameter set inside the bitstream.

H.10.1.1 Multiview High profile

Bitstreams conforming to the Multiview High profile shall obey the following constraints:

- The base view bitstream as specified in subclause H.8.5.4 shall obey all constraints of the Progressive High profile specified in subclause A.2.4.1 and all active sequence parameter sets shall fulfill one or more of the following conditions:
  - profile_idc is equal to 100 or 77 and constraint_set4_flag is equal to 1,
  - (profile_idc is equal to 66 or constraint_set0_flag is equal to 1) and constraint_set1_flag is equal to 1,
  - profile_idc is equal to 77 and constraint_set0_flag is equal to 1,
  - profile_idc is equal to 88, constraint_set1_flag is equal to 1, and constraint_set4_flag is equal to 1.
- Only I, P, and B slice types may be present.
- NAL unit streams shall not contain nal_unit_type values in the range of 2 to 4, inclusive.
- MVC sequence parameter sets shall have frame_mbs_only_flag equal to 1.
- Arbitrary slice order is not allowed.
- Picture parameter sets shall have num_slice_groups_minus1 equal to 0 only.
- Picture parameter sets shall have redundant_pic_cnt_present_flag equal to 0 only.
- MVC sequence parameter sets shall have chroma_format_idc in the range of 0 to 1 inclusive.
- MVC sequence parameter sets shall have bit_depth_luma_minus8 equal to 0 only.
- MVC sequence parameter sets shall have bit_depth_chroma_minus8 equal to 0 only.
- MVC sequence parameter sets shall have qpprime_y_zero_transform_bypass_flag equal to 0 only.
- The level constraints specified for the Multiview High profile in subclause H.10.2 shall be fulfilled.

Conformance of a bitstream to the Multiview High profile is indicated by profile_idc being equal to 118.

Decoders conforming to the Multiview High profile at a specific level shall be capable of decoding all bitstreams in which both of the following conditions are true:

a) All active MVC sequence parameter sets have one or more of the following conditions fulfilled:
   - profile_idc is equal to 118,
   - profile_idc is equal to 100 or 77 and constraint_set4_flag is equal to 1,
   - profile_idc is equal to 88, constraint_set1_flag is equal to 1, and constraint_set4_flag is equal to 1,
   - profile_idc is equal to 77 and constraint_set0_flag is equal to 1,
   - (profile_idc is equal to 66 or constraint_set0_flag is equal to 1) and constraint_set1_flag is equal to 1.

b) All active MVC sequence parameter sets have one or more of the following conditions fulfilled:
   - level_idc or (level_idc and constraint_set3_flag) represent a level less than or equal to the specific level,
   - level_idc[i] or (level_idc[i] and constraint_set3_flag) represent a level less than or equal to the specific level.

H.10.1.2 Stereo High profile

Bitstreams conforming to the Stereo High profile shall obey the following constraints:

– The base view bitstream as specified in subclause H.8.5.4 shall obey all constraints of the High profile specified in subclause A.2.4 and all active sequence parameter sets shall fulfill one of the following conditions:
  – profile_idc is equal to 77 or constraint_set1_flag is equal to 1,
  – profile_idc is equal to 100.
– Only I, P, and B slice types may be present.
– NAL unit streams shall not contain nal_unit_type values in the range of 2 to 4, inclusive.
– Arbitrary slice order is not allowed.
– Picture parameter sets shall have num_slice_groups_minus1 equal to 0 only.
– Picture parameter sets shall have redundant_pic_cnt_present_flag equal to 0 only.
– MVC sequence parameter sets shall have chroma_format_idc in the range of 0 to 1 inclusive.
– MVC sequence parameter sets shall have bit_depth_luma_minus8 equal to 0 only.
– MVC sequence parameter sets shall have bit_depth_chroma_minus8 equal to 0 only.
– MVC sequence parameter sets shall have qpprime_y_zero_transform_bypass_flag equal to 0 only.
– When num_views_minus1 is present in an MVC sequence parameter set, its value shall be less than 2.
– For each access unit, the value of level_idc for all active view MVC sequence parameter set RBSPs shall be the same as the value of level_idc for the active MVC sequence parameter set RBSP.
– The level constraints specified for the Stereo High profile in subclause H.10.2 shall be fulfilled.

Conformance of a bitstream to the Stereo High profile is indicated by profile_idc being equal to 128.

Decoders conforming to the Stereo High profile at a specific level shall be capable of decoding all bitstreams in which both of the following conditions are true:

a) All active MVC sequence parameter sets have one or more of the following conditions fulfilled:
   – profile_idc is equal to 128,
   – profile_idc is equal to 118 and constraint_set5_flag is equal to 1,
   – profile_idc is equal to 100,
   – profile_idc is equal to 77 or constraint_set1_flag is equal to 1.
b) All active MVC sequence parameter sets have one or more of the following conditions fulfilled:
   – level_idc or (level_idc and constraint_set3_flag) represent a level less than or equal to the specific level,
   – level_idc[i] or (level_idc[i] and constraint_set3_flag) represent a level less than or equal to the specific level.

H.10.2 Levels

The following is specified for expressing the constraints in this subclause:

– Let access unit n be the n-th access unit in decoding order with the first access unit being access unit 0.
– Let picture n be the primary coded picture or the corresponding decoded picture of access unit n.

Let the variable fR be derived as follows:

– If picture n is a frame, fR is set equal to 1 ÷ 172.
– Otherwise (picture n is a field), fR is set equal to 1 ÷ (172 * 2).

The value of mvcScaleFactor is set equal to 2.

The value of NumViews is set equal to applicable_op_num_views_minus1[i][j] plus 1, which indicates the number of views required for decoding the target output views corresponding to the j-th operation point for level_idc[i] as signalled in the subset sequence parameter set.

H.10.2.1 Level limits common to Multiview High and Stereo High profiles

Bitstreams conforming to the Multiview High profile at a specified level shall obey the following constraints:

a) The nominal removal time of access unit n (with n > 0) from the CPB as specified in subclause C.1.2, satisfies the constraint that $t_{r,n}(n) - t_r(n - 1)$ is greater than or equal to Max(NumViews * PicSizeInMbs ÷ (mvcScaleFactor * MaxMBPS), fR), where MaxMBPS is the value specified in Table A-1 that applies to picture n − 1, and PicSizeInMbs is the number of macroblocks in a single view component of picture n − 1.

b) The difference between consecutive output times of pictures from the DPB as specified in subclause C.2.2, satisfies the constraint that $\Delta t_{o,dpb}(n)$ >= Max(NumViews * PicSizeInMbs ÷ (mvcScaleFactor * MaxMBPS), fR), where MaxMBPS is the value specified in Table A-1 for picture n, and PicSizeInMbs is the number of macroblocks of a single view component of picture n, provided that picture n is a picture that is output and is not the last picture of the bitstream that is output.

c) PicWidthInMbs * FrameHeightInMbs <= MaxFS, where MaxFS is specified in Table A-1.

d) PicWidthInMbs <= Sqrt(MaxFS * 8), where MaxFS is specified in Table A-1.

e) FrameHeightInMbs <= Sqrt(MaxFS * 8), where MaxFS is specified in Table A-1.

f) max_dec_frame_buffering <= MaxDpbFrames, where MaxDpbFrames is equal to Min(mvcScaleFactor * MaxDpbMbs / (PicWidthInMbs * FrameHeightInMbs), Max(1, Ceil(log2(NumViews))) * 16) and MaxDpbMbs is specified in Table A-1.

g) The vertical motion vector component range does not exceed MaxVmvR in units of luma frame samples, where MaxVmvR is specified in Table A-1.

h) The horizontal motion vector range does not exceed the range of −2048 to 2047.75, inclusive, in units of luma samples.

i) Let setOf2Mb be the set of unsorted pairs of macroblocks that contains the unsorted pairs of macroblocks (mbA, mbB) of a coded video sequence for which any of the following conditions are true:

   – mbA and mbB are macroblocks that belong to the same slice and are consecutive in decoding order,

   – separate_colour_plane_flag is equal to 0, mbA is the last macroblock (in decoding order) of a slice, and mbB is the first macroblock (in decoding order) of the next slice in decoding order,

   – separate_colour_plane_flag is equal to 1, mbA is the last macroblock (in decoding order) of a slice with a particular value of colour_plane_id, and mbB is the first macroblock (in decoding order) of the next slice with the same value of colour_plane_id in decoding order.

   NOTE 1 – In the two above conditions, the macroblocks mbA and mbB can belong to different pictures.

   For each unsorted pair of macroblocks (mbA, mbB) of the set setOf2Mb, the total number of motion vectors (given by the sum of the number of motion vectors for macroblock mbA and the number of motion vectors for macroblock mbB) does not exceed MaxMvsPer2Mb, where MaxMvsPer2Mb is specified in Table A-1. The number of motion vectors for each macroblock is the value of the variable MvCnt after the completion of the intra or inter prediction process for the macroblock.

   NOTE 2 – When separate_colour_plane_flag is equal to 0, the constraint specifies that the total number of motion vectors for two consecutive macroblocks in decoding order must not exceed MaxMvsPer2Mb. When separate_colour_plane_flag is equal to 1, the constraint specifies that the total number of motion vectors for two consecutive macroblocks with the same value of colour_plane_id in decoding order must not exceed MaxMvsPer2Mb. For macroblocks that are consecutive in decoding order but are associated with a different value of colour_plane_id, no constraint for the total number of motion vectors is specified.

j) The number of bits of macroblock_layer( ) data for any macroblock is not greater than 128 + RawMbBits. Depending on entropy_coding_mode_flag, the bits of macroblock_layer( ) data are counted as follows:

   – If entropy_coding_mode_flag is equal to 0, the number of bits of macroblock_layer( ) data is given by the number of bits in the macroblock_layer( ) syntax structure for a macroblock.

   – Otherwise (entropy_coding_mode_flag is equal to 1), the number of bits of macroblock_layer( ) data for a macroblock is given by the number of times read_bits(1) is called in subclauses 9.3.3.2.2 and 9.3.3.2.3 when parsing the macroblock_layer( ) associated with the macroblock.

k) The removal time of access unit 0 shall satisfy the constraint that the number of slices in picture 0 is less than or equal to mvcScaleFactor * (Max(PicSizeInMbs, fR * MaxMBPS) + MaxMBPS * ($t_r(0) - t_{r,n}(0)$)) ÷

SliceRate, where MaxMBPS and SliceRate are the values specified in Tables A-1 and A-4, respectively, that apply to picture 0 and PicSizeInMbs is the number of macroblocks in a single view component of picture 0.

l) The removal time of access unit 0 shall satisfy the constraint that the number of slices in each view component of picture 0 is less than or equal to $(\mathrm{Max}(\mathrm{PicSizeInMbs}, \mathrm{fR} * \mathrm{MaxMBPS}) + \mathrm{MaxMBPS} * (t_r(0) - t_{r,n}(0))) \div \mathrm{SliceRate}$, where MaxMBPS and SliceRate are the values specified in Tables A-1 and A-4, respectively, that apply to picture 0 and PicSizeInMbs is the number of macroblocks in a single view component of picture 0.

m) The difference between consecutive removal times of access units n and n − 1 with n > 0 shall satisfy the constraint that the number of slices in picture n is less than or equal to $\mathrm{mvcScaleFactor} * \mathrm{MaxMBPS} * (t_r(n) - t_r(n - 1)) \div \mathrm{SliceRate}$, where SliceRate is the value specified in Table A-4 that applies to picture n.

n) The difference between consecutive removal times of access units n and n − 1 with n > 0 shall satisfy the constraint that the number of slices in each view component of picture n is less than or equal to $\mathrm{MaxMBPS} * (t_r(n) - t_r(n - 1)) \div \mathrm{SliceRate}$, where SliceRate is the value specified in Table A-4 that applies to picture n.

o) MVC sequence parameter sets shall have direct_8x8_inference_flag equal to 1 for the levels specified in Table A-4.

p) The value of sub_mb_type[mbPartIdx] with mbPartIdx = 0..3 in B macroblocks with mb_type equal to B_8x8 shall not be equal to B_Bi_8x4, B_Bi_4x8, or B_Bi_4x4 for the levels in which MinLumaBiPredSize is shown as 8x8 in Table A-4.

q) For the VCL HRD parameters, BitRate[SchedSelIdx] <= cpbBrVclFactor * MaxBR and CpbSize[SchedSelIdx] <= cpbBrVclFactor *MaxCPB for at least one value of SchedSelIdx, where cpbBrVclFactor is equal to 1250. With vui_mvc_vcl_hrd_parameters_present_flag[i] being the syntax element, in the MVC VUI parameters extension of the active MVC sequence parameter set, that is associated with the VCL HRD parameters that are used for conformance checking (as specified in Annex C), BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are given as follows:

   – If vui_mvc_vcl_hrd_parameters_present_flag equal to 1, BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are given by Equations E-37 and E-38, respectively, using the syntax elements of the hrd_parameters( ) syntax structure that immediately follows vui_mvc_vcl_hrd_parameters_present_flag.

   – Otherwise (vui_mvc_vcl_hrd_parameters_present_flag equal to 0), BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are inferred as specified in subclause E.2.2 for VCL HRD parameters.

   MaxBR and MaxCPB are specified in Table A-1 in units of cpbBrVclFactor bits/s and cpbBrVclFactor bits, respectively. The bitstream shall satisfy these conditions for at least one value of SchedSelIdx in the range 0 to cpb_cnt_minus1, inclusive.

r) For the NAL HRD parameters, BitRate[SchedSelIdx] <= cpbBrNalFactor * MaxBR and CpbSize[SchedSelIdx] <= cpbBrNalFactor *MaxCPB for at least one value of SchedSelIdx, where cpbBrNalFactor is equal to 1500. With vui_mvc_nal_hrd_parameters_present_flag[i] being the syntax element, in the MVC VUI parameters extension of the active MVC sequence parameter set, that is associated with the NAL HRD parameters that are used for conformance checking (as specified in Annex C), BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are given as follows:

   – If vui_mvc_nal_hrd_parameters_present_flag equal to 1, BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are given by Equations E-37 and E-38, respectively, using the syntax elements of the hrd_parameters( ) syntax structure that immediately follows vui_mvc_nal_hrd_parameters_present_flag.

   – Otherwise (vui_mvc_nal_hrd_parameters_present_flag equal to 0), BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are inferred as specified in subclause E.2.2 for NAL HRD parameters.

   MaxBR and MaxCPB are specified in Table A-1 in units of cpbBrNalFactor bits/s and cpbBrNalFactor bits, respectively. The bitstream shall satisfy these conditions for at least one value of SchedSelIdx in the range 0 to cpb_cnt_minus1, inclusive.

s) The sum of the NumBytesInNALunit variables for access unit 0 is less than or equal to $384 * \mathrm{mvcScaleFactor} * (\mathrm{Max}(\mathrm{PicSizeInMbs}, \mathrm{fR} * \mathrm{MaxMBPS}) + \mathrm{MaxMBPS} * (t_r(0) - t_{r,n}(0))) \div \mathrm{MinCR}$, where MaxMBPS and MinCR are the values specified in Table A-1 that apply to picture 0 and PicSizeInMbs is the number of macroblocks in a single view component of picture 0.

t) The sum of the NumBytesInNALunit variables for the VCL NAL units of each view component of access unit 0 is less than or equal to 384 * (Max(PicSizeInMbs, fR * MaxMBPS) + MaxMBPS * ($t_r(0) - t_{r,n}(0)$)) ÷ MinCR, where MaxMBPS and MinCR are the values specified in Table A-1 that apply to picture 0 and PicSizeInMbs is the number of macroblocks in a single view component of picture 0.

u) The sum of the NumBytesInNALunit variables for access unit n with n > 0 is less than or equal to 384 * mvcScaleFactor * MaxMBPS * ($t_r(n) - t_r(n - 1)$) ÷ MinCR, where MaxMBPS and MinCR are the values specified in Table A-1 that apply to picture n.

v) The sum of the NumBytesInNALunit variables for the VCL NAL units of each view component of access unit n with n > 0 is less than or equal to 384 * MaxMBPS * ($t_r(n) - t_r(n - 1)$) ÷ MinCR, where MaxMBPS and MinCR are the values specified in Table A-1 that apply to picture n.

w) When PicSizeInMbs is greater than 1620, the number of macroblocks in any coded slice shall not exceed MaxFS / 4, where MaxFS is specified in Table A-1.

x) max_num_ref_frames shall be less than or equal to MaxDpbFrames / mvcScaleFactor for each view component, where MaxDpbFrames is specified in item f).

Table A-1 specifies the limits for each level. A definition of all levels identified in the "Level number" column of Table A-1 is specified for the Multiview High and Stereo High profiles. Table A-4 specifies limits for each level that are specific to bitstreams conforming to the Multiview High and Stereo High profiles. Each entry in Tables A-1 and A-4 indicates, for the level corresponding to the row of the table, the absence or value of a limit that is imposed by the variable corresponding to the column of the table, as follows:

– If the table entry is marked as "-", no limit is imposed by the value of the variable as a requirement of bitstream conformance to the profile at the specified level.

– Otherwise, the table entry specifies the value of the variable for the associated limit that is imposed as a requirement of bitstream conformance to the profile at the specified level.

For coded video sequences conforming to the Multiview High or Stereo High profile, the level_idc value is specified as follows:

– If level_idc is not equal to 0, level_idc indicates the level that applies to the coded video sequence operating with all the views being target output views.

  NOTE 3 – A level_idc value that is not equal to zero may indicate a higher level than necessary to decode the coded video sequence operating with all the views being target output views. This may occur when a subset of views or temporal subsets are removed from a coded video sequence according to the sub-bitstream extraction process specified in subclause H.8.5.3, and the level_idc value is not updated accordingly.

– Otherwise (level_idc is equal to 0), the level that applies to the coded video sequence operating with all the views being target output views is unspecified.

  NOTE 4 – When profile_idc is equal to 118 or 128 and level_idc is equal to 0, there may exist a level indicated by level_idc[i] that is applicable to the coded video sequence operating with all the views being target output views. This may occur when a subset of views or temporal subsets are removed from a coded video sequence according to the sub-bitstream extraction process specified in subclause H.8.5.3, and a particular value of level_idc[i] corresponds to the resulting coded video sequence.

In bitstreams conforming to the Multiview High or Stereo High profiles, the conformance of the bitstream to a specified level is indicated by the syntax element level_idc or level_idc[i] as follows:

– If level_idc or level_idc[i] is equal to 9, the indicated level is level 1b.

– Otherwise (level_idc or level_idc[i] is not equal to 9), level_idc or level_idc[i] is equal to a value of ten times the level number (of the indicated level) specified in Table A-1.

H.10.2.2 Profile specific level limits

a) In bitstreams conforming to the Stereo High profile, MVC sequence parameter sets shall have frame_mbs_only_flag equal to 1 for the levels specified in Table A-4.

H.11 Byte stream format

The specifications in Annex B apply.

H.12 MVC hypothetical reference decoder

The specifications in Annex C apply with substituting MVC sequence parameter set for sequence parameter set.

H.13 MVC SEI messages

The specifications in Annex D together with the extensions and modifications specified in this subclause apply.

H.13.1 SEI message syntax

H.13.1.1 Parallel decoding information SEI message syntax

parallel_decoding_info(payloadSize) {	C	Descriptor
seq_parameter_set_id	5	ue(v)
for(i = 1; i <= num_views_minus1; i++) {		
if(anchor_pic_flag) {		
for(j = 0; j <= num_anchor_refs_l0[i]; j++)		
pdi_init_delay_anchor_minus2_l0[i **][** j **]**	5	ue(v)
for(j = 0; j <= num_anchor_refs_l1[i]; j++)		
pdi_init_delay_anchor_minus2_l1[i **][** j **]**	5	ue(v)
}		
else {		
for(j = 0; j <= num_non_anchor_refs_l0[i]; j++)		
pdi_init_delay_non_anchor_minus2_l0[i **][** j **]**	5	ue(v)
for(j = 0; j <= num_non_anchor_refs_l1[i]; j++)		
pdi_init_delay_non_anchor_minus2_l1[i **][** j **]**	5	ue(v)
}		
}		
}		

H.13.1.2 MVC scalable nesting SEI message syntax

mvc_scalable_nesting(payloadSize) {	C	Descriptor
operation_point_flag	5	u(1)
if (!operation_point_flag) {		
all_view_components_in_au_flag	5	u(1)
if(!all_view_components_in_au_flag) {		
num_view_components_minus1	5	ue(v)
for(i = 0; i <= num_view_components_minus1; i++)		
sei_view_id[i **]**	5	u(10)
}		
} else {		
num_view_components_op_minus1	5	ue(v)
for(i = 0; i <= num_view_components_op_minus1; i++)		
sei_op_view_id[i **]**	5	u(10)
sei_op_temporal_id	5	u(3)
}		
while(!byte_aligned( ))		
sei_nesting_zero_bit /* equal to 0 */	5	f(1)
sei_message( )	5	
}		

H.13.1.3 View scalability information SEI message syntax

view_scalability_info(payloadSize) {	C	Descriptor
num_operation_points_minus1	5	ue(v)
for(i = 0; i <= num_operation_points_minus1; i++) {		
operation_point_id[i **]**	5	ue(v)
priority_id[i **]**	5	u(5)
temporal_id[i **]**	5	u(3)
num_target_output_views_minus1[i **]**	5	ue(v)
for(j = 0; j <= num_target_output_views_minus1[i]; j++)		
view_id[i **][** j **]**	5	ue(v)
profile_level_info_present_flag[i **]**	5	u(1)
bitrate_info_present_flag[i **]**	5	u(1)
frm_rate_info_present_flag[i **]**	5	u(1)
if(!num_target_output_views_minus1[i])		
view_dependency_info_present_flag[i **]**	5	u(1)
parameter_sets_info_present_flag[i **]**	5	u(1)
bitstream_restriction_info_present_flag[i]	5	u(1)
if (profile_level_info_present_flag[i])		
op_profile_level_idc[i **]**	5	u(24)
if(bitrate_info_present_flag[i]) {		
avg_bitrate[i **]**	5	u(16)
max_bitrate[i **]**	5	u(16)
max_bitrate_calc_window[i **]**	5	u(16)
}		
if(frm_rate_info_present_flag[i]) {		
constant_frm_rate_idc[i **]**	5	u(2)
avg_frm_rate[i **]**	5	u(16)
}		
if(view_dependency_info_present_flag[i]) {		
num_directly_dependent_views[i **]**	5	ue(v)
for(j = 0; j < num_directly_dependent_views[i]; j++)		
directly_dependent_view_id[i **][** j **]**	5	ue(v)
} else		
view_dependency_info_src_op_id[i **]**	5	ue(v)
if(parameter_sets_info_present_flag[i]) {		
num_seq_parameter_sets[i **]**	5	ue(v)
for(j = 0; j < num_seq_parameter_sets[i]; j++)		
seq_parameter_set_id_delta[i **][** j **]**	5	ue(v)
num_subset_seq_parameter_sets[i **]**	5	ue(v)
for(j = 0; j < num_subset_seq_parameter_sets[i]; j++)		
subset_seq_parameter_set_id_delta[i **][** j **]**	5	ue(v)
num_pic_parameter_sets_minus1[i **]**	5	ue(v)
for(j = 0; j <= num_pic_parameter_sets_minus1[i]; j++)		
pic_parameter_set_id_delta[i **][** j **]**	5	ue(v)
} else		
parameter_sets_info_src_op_id[i **]**	5	ue(v)

if(bitstream_restriction_info_present_flag[i]) {		
motion_vectors_over_pic_boundaries_flag[i **]**	5	u(1)
max_bytes_per_pic_denom[i **]**	5	ue(v)
max_bits_per_mb_denom[i **]**	5	ue(v)
log2_max_mv_length_horizontal[i **]**	5	ue(v)
log2_max_mv_length_vertical[i **]**	5	ue(v)
max_num_reorder_frames[i **]**	5	ue(v)
max_dec_frame_buffering[i **]**	5	ue(v)
}		
}		
}		

H.13.1.4 Multiview scene information SEI message syntax

multiview_scene_info(payloadSize) {	C	Descriptor
max_disparity	5	ue(v)
}		

H.13.1.5 Multiview acquisition information SEI message syntax

multiview_acquisition_info(payloadSize) {	C	Descriptor
num_views_minus1		ue(v)
intrinsic_param_flag	5	u(1)
extrinsic_param_flag	5	u(1)
if (instrinsic_param_flag) {		
intrinsic_params_equal	5	u(1)
prec_focal_length	5	ue(v)
prec_principal_point	5	ue(v)
prec_skew_factor	5	ue(v)
if(intrinsic_params_equal)		
num_of_param_sets = 1		
else		
num_of_param_sets = num_views_minus1 + 1		
for(i = 0; i < num_of_param_sets; i++) {		
sign_focal_length_x[i **]**	5	u(1)
exponent_focal_length_x[i **]**	5	u(6)
mantissa_focal_length_x[i **]**	5	u(v)
sign_focal_length_y[i **]**	5	u(1)
exponent_focal_length_y[i **]**	5	u(6)
mantissa_focal_length_y[i **]**	5	u(v)
sign_principal_point_x[i **]**	5	u(1)
exponent_principal_point_x[i **]**	5	u(6)
mantissa_principal_point_x[i **]**	5	u(v)
sign_principal_point_y[i **]**	5	u(1)
exponent_principal_point_y[i **]**	5	u(6)
mantissa_principal_point_y[i **]**	5	u(v)
sign_skew_factor[i **]**	5	u(1)

exponent_skew_factor[i]	5	u(6)
mantissa_skew_factor[i]	5	u(v)
}		
}		
if(extrinsic_param_flag) {		
prec_rotation_param	5	ue(v)
prec_translation_param	5	ue(v)
for(i = 0; i <= num_views_minus1; i++) {		
for (j = 1; j <= 3; j++) { /* row */		
for (k = 1; k <= 3; k++) { /* column */		
sign_r[i][j][k]	5	u(1)
exponent_r[i][j][k]	5	u(6)
mantissa_r[i][j][k]	5	u(v)
}		
sign_t[i][j]	5	u(1)
exponent_t[i][j]	5	u(6)
mantissa_t[i][j]	5	u(v)
}		
}		
}		
}		

H.13.1.6 Non-required view component SEI message syntax

non_required_view_component(payloadSize) {	**C**	**Descriptor**
num_info_entries_minus1	5	ue(v)
for(i = 0; i <= num_info_entries_minus1; i++) {		
view_order_index[i]	5	ue(v)
num_non_required_view_components_minus1[i]	5	ue(v)
for(j = 0; j <= num_non_required_view_components_minus1[i]; j++)		
index_delta_minus1[i][j]	5	ue(v)
}		
}		

H.13.1.7 View dependency change SEI message syntax

view_dependency_change(payloadSize) {	C	Descriptor
seq_parameter_set_id	5	ue(v)
anchor_update_flag	5	u(1)
non_anchor_update_flag	5	u(1)
if(anchor_update_flag)		
for(i = 1; i <= num_views_minus1; i++) {		
for(j = 0; j < num_anchor_refs_l0[i]; j++)		
anchor_ref_l0_flag[i **][** j **]**	5	u(1)
for(j = 0; j < num_anchor_refs_l1[i]; j++)		
anchor_ref_l1_flag[i **][** j **]**	5	u(1)
}		
if(non_anchor_update_flag)		
for(i = 1; i <= num_views_minus1; i++) {		
for(j = 0; j < num_non_anchor_refs_l0[i]; j++)		
non_anchor_ref_l0_flag[i **][** j **]**	5	u(1)
for(j = 0; j < num_non_anchor_refs_l1[i]; j++)		
non_anchor_ref_l1_flag[i **][** j **]**	5	u(1)
}		
}		

H.13.1.8 Operation point not present SEI message syntax

operation_point_not_present(payloadSize) {	C	Descriptor
num_operation_points	5	ue(v)
for(k = 0; k < num_operation_points; k++)		
operation_point_not_present_id[k **]**	5	ue(v)
}		

H.13.1.9 Base view temporal HRD SEI message syntax

base_view_temporal_hrd (payloadSize) {	C	Descriptor
num_of_temporal_layers_in_base_view_minus1	5	ue(v)
for(i = 0; i <= num_of_temporal_layers_in_base_view_minus1; i++) {		
sei_mvc_temporal_id[i **]**	5	u(3)
sei_mvc_timing_info_present_flag[i **]**	5	u(1)
if(sei_mvc_timing_info_present_flag[i]) {		
sei_mvc_num_units_in_tick[i **]**	5	u(32)
sei_mvc_time_scale[i **]**	5	u(32)
sei_mvc_fixed_frame_rate_flag[i **]**	5	u(1)
}		
sei_mvc_nal_hrd_parameters_present_flag[i **]**	5	u(1)
if(sei_mvc_nal_hrd_parameters_present_flag[i])		
hrd_parameters( )	5	
sei_mvc_vcl_hrd_parameters_present_flag[i **]**	5	u(1)

if(sei_mvc_vcl_hrd_parameters_present_flag[i])		
hrd_parameters( )	5	
if(sei_mvc_nal_hrd_parameters_present_flag[i] || sei_mvc_vcl_hrd_parameters_present_flag[i])		
sei_mvc_low_delay_hrd_flag[i **]**	5	u(1)
sei_mvc_pic_struct_present_flag[i **]**	5	u(1)
}		
}		

H.13.2 SEI message semantics

Depending on payloadType, the corresponding SEI message semantics are extended as follows:

– If payloadType is equal to 2, 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22 or 23, the following applies:

  – If the SEI message is not included in an MVC scalable nesting SEI message, it applies to the view component of the current access unit with VOIdx equal to VOIdxMin.

  – Otherwise (the SEI message is included in an MVC scalable nesting SEI message), it applies to all view components of the current access unit when all_view_components_in_au_flag is equal to 1, or it applies to all view components of the current access unit with view_id equal to sei_view_id[i] for any i in the range of 0 to num_view_components_minus1, inclusive, when all_view_components_in_au_flag is equal to 0. When payloadType is equal to 10 for the SEI message that is included in an MVC scalable nesting SEI message, the semantics for sub_seq_layer_num of the sub-sequence information SEI message is modified as follows:

    sub_seq_layer_num specifies the sub-sequence layer number of the current picture. When the current picture resides in a sub-sequence for which the first picture in decoding order is an IDR picture, the value of sub_seq_layer_num shall be equal to 0. For a non-paired reference field, the value of sub_seq_layer_num shall be equal to 0. sub_seq_layer_num shall be in the range of 0 to 255, inclusive.

– Otherwise, if payloadType is equal to 0 or 1, the following applies:

  – If the SEI message is not included in an MVC scalable nesting SEI message, the following applies. When the SEI message and all other SEI messages with payloadType equal to 0 or 1 not included in an MVC scalable nesting SEI message are used as the buffering period and picture timing SEI messages for checking the bitstream conformance according to Annex C and the decoding process specified in clauses 2-9 is used, the bitstream shall be conforming to this Recommendation | International Standard.

  – Otherwise (the SEI message is included in an MVC scalable nesting SEI message), the following applies. When the SEI message and all other SEI messages with payloadType equal to 0 or 1 included in an MVC scalable nesting SEI message with identical values of sei_op_temporal_id and sei_op_view_id[i] for all i in the range of 0 to num_view_components_op_minus1, inclusive, are used as the buffering period and picture timing SEI messages for checking the bitstream conformance according to Annex C, the bitstream that would be obtained by invoking the bitstream extraction process as specified in subclause H.8.3 with tIdTarget equal to sei_op_temporal_id and viewIdTargetList equal to sei_op_view_id[i] for all i in the range of 0 to num_view_components_op_minus1, inclusive, shall be conforming to this Recommendation | International Standard.

    In the semantics of subclauses D.2.1 and D.2.2, the syntax elements num_units_in_tick, time_scale, fixed_frame_rate_flag, nal_hrd_parameters_present_flag, vcl_hrd_parameters_present_flag, low_delay_hrd_flag, and pic_struct_present_flag and the derived variables NalHrdBpPresentFlag, VclHrdBpPresentFlag, and CpbDpbDelaysPresentFlag are substituted with the syntax elements vui_mvc_num_units_in_tick[i], vui_mvc_time_scale[i], vui_mvc_fixed_frame_rate_flag[i], vui_mvc_nal_hrd_parameters_present_flag[i], vui_mvc_vcl_hrd_parameters_present_flag[i], vui_mvc_low_delay_hrd_flag[i], and vui_mvc_pic_struct_present_flag[i] and the derived variables VuiMvcNalHrdBpPresentFlag[i], VuiMvcVclHrdBpPresentFlag[i], and VuiMvcCpbDpbDelaysPresentFlag[i].

    The values of seq_parameter_set_id's in all buffering period SEI messages included in MVC scalable nesting SEI messages and associated with operation points for which the greatest VOIdx values in the associated bitstream subsets are identical shall be identical.

– Otherwise (all remaining payloadType values), the corresponding SEI message semantics are not extended.

For the semantics of SEI messages with payloadType in the range of 0 to 23, inclusive, which are specified in subclause D.2, MVC sequence parameter set is substituted for sequence parameter set; the parameters of MVC sequence parameter set RBSP and picture parameter set RBSP that are in effect are specified in subclauses H.7.4.2.1 and H.7.4.2.2, respectively.

Coded video sequences conforming to one or more of the profiles specified in Annex H shall not include SEI NAL units that contain SEI messages with payloadType in the range of 24 to 35, inclusive.

When an SEI NAL unit contains an SEI message with payloadType in the range of 36 to 44, inclusive, it shall not contain any SEI messages with payloadType less than 36 and the first SEI message in the SEI NAL unit shall have payloadType in the range of 36 to 44, inclusive.

When an MVC scalable nesting SEI message (payloadType equal to 37) or a view scalability information SEI message (payloadType equal to 38) or an operation point not present SEI message (payloadType equal to 43) is present in an SEI NAL unit, it shall be the only SEI message in the SEI NAL unit.

H.13.2.1 Parallel decoding information SEI message semantics

The parallel decoding information SEI message may be associated with any access unit. The information signalled in the SEI message applies to all the access units from the access unit the SEI message is associated with to the next access unit, in decoding order, containing an SEI message of the same type, exclusively, or to the end of the coded video sequence, whichever is earlier in decoding order.

Some view components for which the parallel decoding information is signalled in a parallel decoding information SEI message may not be present in the coded video sequence.

seq_parameter_set_id specifies a subset sequence parameter set that contains the inter-view dependency relationship information. The value of seq_parameter_set_id shall be equal to the value of seq_parameter_set_id in the picture parameter set referenced by a view component of the primary coded picture of the access unit containing the parallel decoding information SEI message. The value of seq_parameter_set_id shall be in the range of 0 to 31, inclusive.

NOTE 1 – The inter-view dependency relationship is signalled in the sequence parameter set MVC extension, which is identical for all subset sequence parameter sets that may be activated during the decoding process for the coded video sequence.

pdi_init_delay_anchor_minus2_l0[i][j] specifies the unavailable reference area in the view component with view_id equal to anchor_ref_l0[i][j] that shall not be used for inter-view reference by the coded anchor view component with view_id equal to view_id[i], where anchor_ref_l0[i][j] and view_id[i] are both from the MVC sequence parameter set that has a sequence parameter set identifier equal to the syntax element seq_parameter_set_id contained in the current SEI message. The unavailable reference area is a rectangular area with coordinates (0, (CurrMbAddr / PicWidthInMbs + pdi_init_delay_anchor_minus2_l0[i][j] + 2) * 16) as the top left corner and (PicWidthInSamples,PicHeightInSamples) as the bottom right corner. When decoding the coded view component with view_id equal to view_id[i], samples from the unavailable reference area from the view component with view_id equal to anchor_ref_l0[i][j] shall not be referred to by the inter-view prediction process. The value of pdi_init_delay_anchor_minus2_l0[i][j] shall be in the range of 0 to PicHeightInMbs − 2, inclusive.

pdi_init_delay_anchor_minus2_l1[i][j] specifies the unavailable reference area in the view component with view_id equal to anchor_ref_l1[i][j] that shall not be used for inter-view reference by the coded anchor view component with view_id equal to view_id[i], where anchor_ref_lX[i][j] and view_id[i] are both from the MVC sequence parameter set that has a sequence parameter set identifier equal to the syntax element seq_parameter_set_id contained in the current SEI message. The unavailable reference area is a rectangular area with coordinates (0, (CurrMbAddr / PicWidthInMbs + pdi_init_delay_anchor_minus2_l1[i][j] + 2) * 16) as the top left corner and (PicWidthInSamples,PicHeightInSamples) as the bottom right corner. When decoding the coded view component with view_id equal to view_id[i], samples from the unavailable reference area from the view component with view_id equal to anchor_ref_l1[i][j] shall not be referred to by the inter-view prediction process. The value of pdi_init_delay_anchor_minus2_l1[i][j] shall be in the range of 0 to PicHeightInMbs − 2, inclusive.

pdi_init_delay_non_anchor_minus2_l0[i][j] specifies the unavailable reference area in the view component with view_id equal to non_anchor_ref_l0[i][j] that shall not be used for inter-view reference by the coded non-anchor view component with view_id equal to view_id[i], where non_anchor_ref_l0[i][j] and view_id[i] are both from the MVC sequence parameter set that has a sequence parameter set identifier equal to the syntax element seq_parameter_set_id contained in the current SEI message. The unavailable reference area is a rectangular area with coordinates (0, (CurrMbAddr / PicWidthInMbs + pdi_init_delay_non_anchor_minus2_l0[i][j] + 2) * 16) as the top left corner and (PicWidthInSamples,PicHeightInSamples) as the bottom right corner. When decoding the coded view component with view_id equal to view_id[i], samples from the unavailable reference area from the view component with view_id equal to non_anchor_ref_l0[i][j] shall not be referred to by the inter-view prediction process. The value of pdi_init_delay_non_anchor_minus2_l0[i][j] shall be in the range of 0 to PicHeightInMbs − 2, inclusive.

pdi_init_delay_non_anchor_minus2_l1[i][j] specifies the unavailable reference area in the view component with view_id equal to non_anchor_ref_l1[i][j] that shall not be used for inter-view reference by the coded anchor view component with view_id equal to view_id[i], where non_anchor_ref_lX[i][j] and view_id[i] are both from the MVC sequence parameter set that has a sequence parameter set identifier equal to the syntax element seq_parameter_set_id contained in the current SEI message. The unavailable reference area is a rectangular area with coordinates (0, (CurrMbAddr / PicWidthInMbs + pdi_init_delay_non_anchor_minus2_l1[i][j] + 2) * 16) as the top left corner and (PicWidthInSamples,PicHeightInSamples) as the bottom right corner. When decoding the coded view component with view_id equal to view_id[i], samples from the unavailable reference area from the view component with view_id equal to non_anchor_ref_l1[i][j] shall not be referred to by the inter-view prediction process. The value of pdi_init_delay_non_anchor_minus2_l1[i][j] shall be in the range of 0 to PicHeightInMbs − 2, inclusive.

H.13.2.2 MVC scalable nesting SEI message semantics

An MVC nesting SEI message shall contain one and only one SEI message of payloadType less than or equal to 23, which is referred to as the nested SEI message. The scope to which the nested SEI message applies is indicated by the syntax elements operation_point_flag, all_view_components_in_au_flag, num_view_components_minus1, sei_view_id[i] for all i, num_view_components_op_minus1, sei_op_view_id[i] for all i, and sei_op_temporal_id.

Some view components to which the nested SEI message applies may not be present in the access unit containing the MVC scalable nesting SEI message.

operation_point_flag equal to 1 specifies that the nested SEI message applies to the current access unit when the associated operation point identified by sei_op_temporal_id and sei_op_view_id[i] for all i in the range of 0 to num_view_components_op_minus1, inclusive, is decoded. operation_point_flag equal to 0 specifies that the nested SEI message applies to the view components identified by all_view_components_in_au_flag, num_view_components_minus1, and sei_view_id[i] for all i in the range of 0 to num_view_components_minus1, inclusive, regardless of which operation point is decoded.

If the nested SEI message has payloadType equal to 0 or 1, operation_point_flag shall be equal to 1. Otherwise (the nested SEI message has payloadType not equal to 0 or 1), operation_point_flag shall be equal to 0.

all_view_components_in_au_flag equal to 1 specifies that the nested SEI message applies to all view components of the access unit. all_view_components_in_au_flag equal to 0 specifies that the applicable scope of the nested SEI message is signalled by the syntax elements num_view_components_minus1 and sei_view_id[i] for all i in the range of 0 to num_view_components_minus1, inclusive.

num_view_components_minus1 plus 1 specifies the number of view components to which the nested SEI message applies when operation_point_flag is equal to 0 and all_view_components_in_au_flag is equal to 0. The value of num_view_components_minus1 shall be in the range of 0 to 1023, inclusive.

sei_view_id[i] specifies the view_id of the i-th view component to which the nested SEI message applies when operation_point_flag is equal to 0 and all_view_components_in_au_flag is equal to 0.

num_view_components_op_minus1 plus 1 specifies the number of view components of the operation point to which the nested SEI message applies when operation_point_flag is equal to 1. The value of num_view_components_op_minus1 shall be in the range of 0 to 1023, inclusive.

sei_op_view_id[i] specifies the view_id of the i-th view component to which the nested SEI message applies when operation_point_flag is equal to 1.

sei_op_temporal_id specifies the maximum temporal_id of the bitstream subset to which the nested SEI message applies when operation_point_flag is equal to 1.

sei_nesting_zero_bit is equal to 0.

H.13.2.3 View scalability information SEI message semantics

When present, this SEI message shall be associated with an IDR access unit. The semantics of the message are valid for the current coded video sequence. A view scalability information SEI message contains view and scalability information for a subset of the operation points in the coded video sequence. Each operation point is associated with an operation point identifier. The sub-bitstream for an operation point is referred to as the operation point representation or the representation of the operation point. Information such as bit rate and frame rate, among others, are signalled for the representations of the subset of the operation points.

NOTE 1 – Any operation point for which view and scalability information is signalled in a view scalability information SEI message (i.e. identified by a value of operation_point_id[i]) must be present in the coded video sequence. When an application keeps a view scalability information SEI message in a sub-bitstream extracted according to the process specified in subclause H.8.5.3, and after the extraction any operation point for which view and scalability information is signalled in the original SEI message becomes not present in the coded video sequence, the application must change the content of the view scalability information SEI message to fulfil the condition stated by the first sentence in this note.

num_operation_points_minus1 plus 1 specifies the number of operation points that are present in the coded video sequence and for which the view scalability information is signalled by the following syntax elements. The value of num_operation_points_minus1 shall be in the range of 0 to 1023, inclusive.

The bitstream subset corresponding to an operation point is defined as the operation point representation or the representation of the operation point. The representation of the operation point identified by operation_point_id[i] is the output of the sub-bitstream extraction process specified in subclause H.8.5.3 with tIdTarget equal to temporal_id[i] and viewIdTargetList consisting of view_id[i][j] for all j in the range of 0 to num_target_output_views_minus1[i], inclusive, as the inputs.

operation_point_id[i **]** specifies the identifier of the operation point. Each operation point is associated with a unique operation point identifier. The value of operation_point_id[i] shall be in the range of 0 to 65535, inclusive.

In the following semantics in this subclause, the operation point with identifier equal to operation_point_id[i] is referred to as the current operation point.

priority_id[i **]** and **temporal_id[** i **]** specify the maximum value of priority_id and temporal_id, respectively, of the NAL units in the representation of the current operation point.

num_target_output_views_minus1[i **]** plus 1 specifies the number of target output views for the current operation point. The value of num_target_output_views_minus1[i] shall be in the range of 0 to 1023, inclusive.

view_id[i **][** j **]** specifies the identifier of the j-th target output view for the current operation point. The value of view_id[i][j] shall be in the range of 0 to 1023, inclusive.

profile_level_info_present_flag[i **]** equal to 1 specifies that the profile and level information for the representation of the current operation point is present in the SEI message. profile_level_info_present_flag[i] equal to 0 specifies that the profile and level information for the current operation point is not present in the SEI message.

bitrate_info_present_flag[i **]** equal to 1 specifies that the bitrate information for the current operation point is present in the SEI message. bitrate_info_present_flag[i] equal to 0 specifies that the bitrate information for the current operation point is not present in the SEI message.

frm_rate_info_present_flag[i **]** equal to 1 specifies that the frame rate information for the current operation point is present in the SEI message. frm_rate_info_present_flag[i] equal to 0 specifies that the frame rate information for the current operation point is not present in the SEI message.

view_dependency_info_present_flag[i **]** equal to 1 specifies that information on the views the target output view of the current operation point directly depends on is present in the SEI message. View A is directly dependent on view point B if there is at least one view component of view A using a view component of view B for inter-view prediction reference. view_dependency_info_present_flag[i] equal to 0 specifies that view_dependency_info_src_op_id[i] is present in the SEI message. When not present, view_dependency_info_present_flag[i] shall be inferred to be equal to 0.

NOTE 2 – The inter-view dependency relationship signalled in sequence parameter set MVC extension is an upper bound, in the sense that whenever view A may depend on view B at any access unit, it is specified as view A depends on view B. Therefore, the dependency relationship is indicated by sequence parameter set MVC extension when view A depends on view B at only one of all access units in the coded video sequence, or even when view A actually does not depend on view B at any access unit but when generating the sequence parameter set MVC extension the encoder thought view A might depend on view B. The dependency relationship signalled here can be more refined. For example, when view A depends on view B at access units with temporal_id equal to 0 but not at other access units, this can be indicated through the view dependency information signalled in this SEI message for operation points with view A as the target output view and with different maximum values of temporal_id.

parameter_sets_info_present_flag[i **]** equal to 1 specifies that the values of seq_parameter_set_id of the sequence parameter sets and subset sequence parameter sets and the values of pic_parameter_set_id of the picture parameter sets that are referred to by the VCL NAL units of the representation of the current operation point are present in the SEI message. parameter_sets_info_present_flag[i] equal to 0 specifies that parameter_sets_info_src_op_id[i] is present in the SEI message.

bitstream_restriction_info_present_flag[i **]** equal to 1 specifies that the bitstream restriction information for the representation of the current operation point is present in the SEI message. bitstream_restriction_info_present_flag[i] equal to 0 specifies that the bitstream restriction information for the representation of the current operation point is not present in the SEI message.

op_profile_level_idc[i **]** specifies the profile and level compliancy of the representation of the current operation point. op_profile_level_idc[i] is the exact copy of the three bytes comprised of profile_idc, constraint_set0_flag, constraint_set1_flag, constraint_set2_flag, constraint_set3_flag, constraint_set4_flag, constraint_set5_flag, reserved_zero_2bits, and level_idc, if these syntax elements were used to specify the profile and level compliancy of the representation of the current operation point as specified in Annexes A and H.

avg_bitrate[i] specifies the average bit rate of the representation of the current operation point. The average bit rate for the representation of the current operation point in bits per second is given by BitRateBPS(avg_bitrate[i]) with the function BitRateBPS( ) being specified by the following equation.

$$\text{BitRateBPS}(x) = (x \,\&\, (2^{14} - 1)) * 10^{(2 + (x >> 14))} \quad \text{(H-11)}$$

All NAL units of the representation of the current operation point are taken into account in the calculation. The average bit rate is derived according to the access unit removal time specified in Annex C. In the following, bTotal is the number of bits in all NAL units of the representation of the current operation point in the current coded video sequence. t_1 is the removal time (in seconds) of the current access unit, and t_2 is the removal time (in seconds) of the last access unit (in decoding order) of the current coded video sequence.

With x specifying the value of avg_bitrate[i], the following appplies.

– If t_1 is not equal to t_2, the following condition shall be true.

$$(x \,\&\, (2^{14} - 1)) == \text{Round}(\text{bTotal} \div ((t_2 - t_1) * 10^{(2 + (x >> 14))})) \quad \text{(H-12)}$$

– Otherwise (t_1 is equal to t_2), the following condition shall be true.

$$(x \,\&\, (2^{14} - 1)) == 0 \quad \text{(H-13)}$$

max_bitrate[i] specifies the maximum bit rate of the representation of the current operation point, given by BitRateBPS(max_bitrate_layer_representation[i]), in bits per second, with the function BitRateBPS( ) being specified in Equation H-11. The maximum bit rate of the representation of the current operation point is calculated based on a time window specified by max_bitrate_calc_window[i].

max_bitrate_calc_window[i] specifies the length of the time window, in units of 1/100 second, based on which max_bitrate[i] is calculated.

constant_frm_rate_idc[i] specifies whether the frame rate of the representation of the current operation point is constant. If the value of avg_frm_rate as specified below is constant whichever a temporal section of the operation point representation is used for the calculation, the frame rate is constant, otherwise the frame rate is non-constant. constant_frm_rate_idc[i] equal to 0 specifies that the frame rate is not constant, constant_frm_rate_idc[i] equal to 1 specifies that the frame rate is constant, and constant_frm_rate_idc[i] equal to 2 specifies that the frame rate may be or may not be constant. The value of constant_frm_rate_idc[i] shall be in the range of 0 to 2, inclusive.

avg_frm_rate[i] specifies the average frame rate, in units of frames per 256 seconds, of the representation of the current operation point. The semantics of avg_frm_rate[i] is identical to the semantics of average_frame_rate in sub-sequence layer characteristics SEI message when accurate_statistics_flag is equal to 1, except that herein the set of NAL units in the range of sub-sequence layers is replaced by the set of NAL units of the representation of the current operation point.

num_directly_dependent_views[i] specifies the number of views that the target output view of the current operation point is directly dependent on within the representation of the current operation point. The value of num_directly_dependent_views[i] shall be in the range of 0 to 16, inclusive.

directly_dependent_view_id[i][j] specifies the view_id of the j-th view that the target output view of the current operation point is directly dependent on within the representation of the current operation point. The value of directly_dependent_view_id[i][j] shall be in the range of 0 to 1023, inclusive.

view_dependency_info_src_op_id[i] specifies that the views the target output view of the current operation point directly depends on within the representation of the current operation point are the same as the views the target output view of the operation point with identifier equal to view_dependency_info_src_op_id[i] directly depends on within the representation of the operation point with identifier equal to view_dependency_info_src_op_id[i], if view_dependency_info_src_op_id[i] is not equal to operation_point_id[i]. Otherwise (view_dependency_info_src_op_id[i] is equal to operation_point_id[i]), information on the views the target output view of the current operation point directly depends on is not present in the SEI message. The value of view_dependency_info_src_op_id[i] shall be in the range of 0 to 65535, inclusive.

num_seq_parameter_sets[i] specifies the number of different sequence parameter sets that are referred to by the VCL NAL units of the representation of the current operation point. The value of num_seq_parameter_sets[i] shall be in the range of 0 to 32, inclusive.

seq_parameter_set_id_delta[i][j] specifies the smallest value of the seq_parameter_set_id of all sequence parameter sets required for decoding the representation of the current operation point, if j is equal to 0. Otherwise (j is greater than 0), seq_parameter_set_id_delta[i][j] specifies the difference between the value of the seq_parameter_set_id of the j-th required sequence parameter set and the value of the seq_parameter_set_id of the (j−1)-th required sequence parameter set for decoding the representation of the current operation point. The sequence parameter sets are logically ordered in ascending order of the value of seq_parameter_set_id. The value of seq_parameter_set_id_delta[i][j] shall be in the range of 0 to 31, inclusive.

num_subset_seq_parameter_sets[i] specifies the number of different subset sequence parameter sets that are referred to by the VCL NAL units of the representation of the current operation point. The value of num_subset_seq_parameter_sets[i] shall be in the range of 0 to 32, inclusive.

subset_seq_parameter_set_id_delta[i][j] specifies the smallest value of the seq_parameter_set_id of all subset sequence parameter sets required for decoding the representation of the current operation point, if j is equal to 0. Otherwise (j is greater than 0), subset_seq_parameter_set_id_delta[i][j] specifies the difference between the value of the seq_parameter_set_id of the j-th required subset sequence parameter set and the value of the seq_parameter_set_id of the (j−1)-th required subset sequence parameter set for decoding the representation of the current operation point. The subset sequence parameter sets are logically ordered in ascending order of the value of seq_parameter_set_id. The value of subset_seq_parameter_set_id_delta[i][j] shall be in the range of 0 to 31, inclusive.

num_pic_parameter_sets_minus1[i] plus 1 specifies the number of different picture parameter sets that are referred to by the VCL NAL units of the representation of the current operation point. The value of num_pic_parameter_sets_minus1[i] shall be in the range of 0 to 255, inclusive.

pic_parameter_set_id_delta[i][j] specifies the smallest value of the pic_parameter_set_id of all picture parameter sets required for decoding the representation of the current operation point, if j is equal to 0. Otherwise (j is greater than 0), pic_parameter_set_id_delta[i][j] specifies the difference between the value of the pic_parameter_set_id of the j-th required picture parameter set and the value of the pic_parameter_set_id of the (j−1)-th required picture parameter set for decoding the representation of the current operation point. The picture parameter sets are logically ordered in ascending order of the value of pic_parameter_set_id. The value of pic_parameter_set_id_delta[i][j] shall be in the range of 0 to 255, inclusive.

parameter_sets_info_src_op_id[i] specifies that the values of seq_parameter_set_id of the sequence parameter sets and subset sequence parameter sets and the values of pic_parameter_set_id of the picture parameter sets that are referred to by the VCL NAL units of the representation of the current operation point are the same as those for the representation of the operation point with identifier equal to parameter_sets_info_src_op_id[i], if parameter_sets_info_src_op_id[i] is not equal to operation_point_id[i]. Otherwise (parameter_sets_info_src_op_id[i] is equal to operation_point_id[i]), parameter_sets_info_src_op_id[i] specifies that the values of seq_parameter_set_id of the sequence parameter sets and subset sequence parameter sets and the values of pic_parameter_set_id of the picture parameter sets that are referred to by the VCL NAL units of the representation of the current operation point are not present in the SEI message. The value of parameter_sets_info_src_op_id[i] shall be in the range of 0 to 65535, inclusive.

motion_vectors_over_pic_boundaries_flag[i] specifies the value of motion_vectors_over_pic_boundaries_flag, as specified in subclause E.2.1, for the current operation point representation. When the motion_vectors_over_pic_boundaries_flag[i] syntax element is not present, motion_vectors_over_pic_boundaries_flag value for the current operation point representation shall be inferred to be equal to 1.

max_bytes_per_pic_denom[i] specifies the max_bytes_per_pic_denom value, as specified in subclause E.2.1, for the current operation point representation. When the max_bytes_per_pic_denom[i] syntax element is not present, the value of max_bytes_per_pic_denom for the current operation point representation shall be inferred to be equal to 2. The value of max_bytes_per_pic_denom[i] shall be in the range of 0 to 16, inclusive.

max_bits_per_mb_denom[i] specifies the max_bits_per_mb_denom value, as specified in subclause E.2.1, for the current operation point representation. When the max_bits_per_mb_denom[i] is not present, the value of max_bits_per_mb_denom for the current operation point representation shall be inferred to be equal to 1. The value of max_bits_per_mb_denom[i] shall be in the range of 0 to 16, inclusive.

log2_max_mv_length_horizontal[i] and **log2_max_mv_length_vertical[i]** specify the values of log2_max_mv_length_horizontal and log2_max_mv_length_vertical, as specified in subclause E.2.1, for the current operation point representation. When log2_max_mv_length_horizontal[i] is not present, the values of log2_max_mv_length_horizontal and log2_max_mv_length_vertical for the current operation point representation shall be inferred to be equal to 16. The value of log2_max_mv_length_horizontal[i] shall be in the range of 0 to 16, inclusive. The value of log2_max_mv_length_vertical[i] shall be in the range of 0 to 16, inclusive.

NOTE 3 – The maximum absolute value of a decoded vertical or horizontal motion vector component is also constrained by profile and level limits as specified in Annex A or subclause H.10.2.

max_num_reorder_frames[i] specifies the value of max_num_reorder_frames, as specified in subclause E.2.1, for the current operation point representation. The value of max_num_reorder_frames[i] shall be in the range of 0 to 16, inclusive. When the max_num_reorder_frames[i] syntax element is not present, the value of max_num_reorder_frames for the current operation point representation shall be inferred to be equal to 16.

max_dec_frame_buffering[i] specifies the value of max_dec_frame_buffering, as specified in subclause E.2.1, for the current operation point representation. The value of max_dec_frame_buffering[i] shall be in the range of 0 to MaxDpbFrames (as specified in subclauses A.3.1, A.3.2, or H.10.2), inclusive. When the max_dec_frame_buffering[i] syntax element is not present, the value of max_dec_frame_buffering for the current operation point representation shall be inferred to be equal to MaxDpbFrames.

H.13.2.4 Multiview scene information SEI message semantics

The multiview scene information SEI message indicates the maximum disparity among multiple view components in an access unit. The maximum disparity could be used for processing the decoded view components prior to rendering on a 3D display. When present, the multiview scene information SEI message shall be associated with an IDR access unit. The information signalled in the SEI message applies to the coded video sequence.

The actual maximum disparity value may be less than the one signalled in the multiview scene information SEI message, due to that some views in the coded video sequence may have been removed from the original bitstream to produce an extracted sub-bitstream according to the process specified in subclause H.8.5.3.

max_disparity specifies the maximum disparity, in units of luma samples, between spatially adjacent view components among the total set of view components in an access unit. The value of max_disparity shall be in the range of 0 to 1023, inclusive.

NOTE – The maximum disparity depends on the baseline distance between spatially adjacent views and the spatial resolution of each view. Therefore, if either the number of views or spatial resolution is changed, the maximum disparity should also be changed accordingly.

H.13.2.5 Multiview acquisition information SEI message semantics

The multiview acquisition information SEI message specifies various parameters of the acquisition environment. Specifically, intrinsic and extrinsic camera parameters are specified. These parameters could be used for processing the decoded view components prior to rendering on a 3D display. When present, the multiview acquisition information SEI message shall be associated with an IDR access unit. The information signalled in the SEI message applies to the coded video sequence.

Some of the views for which the multiview acquisition information is included in a multiview acquisition information SEI message may not be present in the coded video sequence.

The extrinsic camera parameters are specified according to a right-handed coordinate system, where the upper left corner of the image is the origin, i.e., the (0, 0) coordinate, with the other corners of the image having non-negative coordinates. With these specifications, a 3-dimensional world point, wP=[x y z] is mapped to a 2-dimensional camera point, cP[i] = [u v 1], for the i-th camera according to:

$$s * cP[i] = A[i] * R^{-1}[i] * (wP - T[i]) \quad \text{(H-14)}$$

where A[i] denotes the intrinsic camera parameter matrix, R^{-1}[i] denotes the inverse of the rotation matrix R[i], T[i] denotes the translation vector, and s (a scalar value) is an arbitrary scale factor chosen to make the third coordinate of cP[i] equal to 1. The elements of A[i], R[i], T[i] are determined according to the syntax elements signalled in this SEI message and as specified below.

num_views_minus1 shall be equal to the value of the syntax element num_views_minus1 in the active MVC sequence parameter set for the coded video sequence. The value of num_views_minus1 shall be in the range of 0 to 1023, inclusive.

intrinsic_param_flag equal to 1 indicates the presence of intrinsic camera parameters. intrinsic_param_flag equal to 0 indicates the absence of intrinsic camera parameters.

extrinsic_param_flag equal to 1 indicates the presence of extrinsic camera parameters. extrinsic_param_flag equal to 0 indicates the absence of extrinsic camera parameters.

intrinsic_params_equal equal to 1 indicates that the intrinsic camera parameters are equal for all cameras and only one set of intrinsic camera parameters are present. intrinsic_params_equal equal to 0 indicates that the intrinsic camera parameters are different for each camera and that a set of intrinsic camera parameters are present for each camera.

prec_focal_length specifies the exponent of the maximum allowable truncation error for focal_length_x[i] and focal_length_y[i] as given by $2^{-\text{prec\_focal\_length}}$. The value of prec_focal_length shall be in the range of 0 to 31, inclusive.

prec_principal_point specifies the exponent of the maximum allowable truncation error for principal_point_x[i] and principal_point_y[i] as given by $2^{-\text{prec\_principal\_point}}$. The value of prec_principal_point shall be in the range of 0 to 31, inclusive.

prec_skew_factor specifies the exponent of the maximum allowable truncation error for skew factor as given by $2^{-\text{prec\_skew\_factor}}$. The value of prec_skew_factor shall be in the range of 0 to 31, inclusive.

sign_focal_length_x[i **]** equal to 0 indicates that the sign of the focal length of the i-th camera in the horizontal direction is positive. sign_focal_length_x[i] equal to 1 indicates that the sign is negative.

exponent_focal_length_x[i **]** specifies the exponent part of the focal length of the i-th camera in the horizontal direction. The value of exponent_focal_length_x[i] shall be in the range of 0 to 62, inclusive. The value 63 is reserved for future use by ITU-T | ISO/IEC. Decoders shall treat the value 63 as indicating an unspecified focal length.

mantissa_focal_length_x[i **]** specifies the mantissa part of the focal length of the i-th camera in the horizontal direction. The length of the mantissa_focal_length_x[i] syntax element is variable and determined as follows:

– If exponent_focal_length_x[i] = = 0, the length is Max(0, prec_focal_length − 30).

– Otherwise (0 < exponent_focal_length_x[i] < 63), the length is Max(0, exponent_focal_length_x[i] + prec_focal_length − 31).

sign_focal_length_y[i **]** equal to 0 indicates that the sign of the focal length of the i-th camera in the vertical direction is positive. sign_focal_length_y[i] equal to 1 indicates that the sign is negative.

exponent_focal_length_y[i **]** specifies the exponent part of the focal length of the i-th camera in the vertical direction. The value of exponent_focal_length_y[i] shall be in the range of 0 to 62, inclusive. The value 63 is reserved for future use by ITU-T | ISO/IEC. Decoders shall treat the value 63 as indicating an unspecified focal length.

mantissa_focal_length_y[i **]** specifies the mantissa part of the focal length of the i-th camera in the vertical direction. The length of the mantissa_focal_length_y[i] syntax element is variable and determined as follows:

– If exponent_focal_length_y[i] = = 0, the length is Max(0, prec_focal_length − 30).

– Otherwise (0 < exponent_focal_length_y[i] < 63), the length is Max(0, exponent_focal_length_y[i] + prec_focal_length − 31).

sign_principal_point_x[i **]** equal to 0 indicates that the sign of the principal point of the i-th camera in the horizontal direction is positive. sign_principal_point_x[i] equal to 1 indicates that the sign is negative.

exponent_principal_point_x[i **]** specifies the exponent part of the principal point of the i-th camera in the horizontal direction. The value of exponent_principal_point_x[i] shall be in the range of 0 to 62, inclusive. The value 63 is reserved for future use by ITU-T | ISO/IEC. Decoders shall treat the value 63 as indicating an unspecified principal point.

mantissa_principal_point_x[i **]** specifies the mantissa part of the principal point of the i-th camera in the horizontal direction. The length of the mantissa_principal_point_x[i] syntax element in units of bits is variable and is determined as follows:

– If exponent_principal_point_x[i] = = 0, the length is Max(0, prec_principal_point − 30).

– Otherwise (0 < exponent_principal_point_x[i] < 63), the length is Max(0, exponent_principal_point_x[i] + prec_principal_point − 31).

sign_principal_point_y[i **]** equal to 0 indicates that the sign of the principal point of the i-th camera in the vertical direction is positive. sign_principal_point_y[i] equal to 1 indicates that the sign is negative.

exponent_principal_point_y[i **]** specifies the exponent part of the principal point of the i-th camera in the vertical direction. The value of exponent_principal_point_y[i] shall be in the range of 0 to 62, inclusive. The value 63 is reserved for future use by ITU-T | ISO/IEC. Decoders shall treat the value 63 as indicating an unspecified principal point.

mantissa_principal_point_y[i **]** specifies the mantissa part of the principal point of the i-th camera in the vertical direction. The length of the mantissa_principal_point_y[i] syntax element in units of bits is variable and is determined as follows:

– If exponent_principal_point_y[i] = = 0, the length is Max(0, prec_principal_point − 30).

– Otherwise (0 < exponent_principal_point_y[i] < 63), the length is Max(0, exponent_principal_point_y[i] + prec_principal_point − 31).

sign_skew_factor[i **]** equal to 0 indicates that the sign of the skew factor of the i-th camera is positive. sign_skew_factor[i] equal to 1 indicates that the sign is negative.

exponent_skew_factor[i **]** specifies the exponent part of the skew factor of the i-th camera. The value of exponent_skew_factor[i] shall be in the range of 0 to 62, inclusive. The value 63 is reserved for future use by

ITU-T | ISO/IEC. Decoders shall treat the value 63 as indicating an unspecified skew factor.

mantissa_skew_factor[i **]** specifies the mantissa part of the skew factor of the i-th camera. The length of the mantissa_skew_factor[i] syntax element is variable and determined as follows:

– If exponent_skew_factor[i] = = 0, the length is Max(0, prec_skew_factor − 30).

– Otherwise (0 < exponent_skew_factor[i] < 63), the length is Max(0, exponent_skew_factor[i] + prec_skew_factor − 31).

The intrinsic matrix A[i] for i-th camera is represented by

$$\begin{bmatrix} \text{focalLengthX[i]} & \text{skewFactor[i]} & \text{principalPointX[i]} \\ 0 & \text{focalLengthY[i]} & \text{principalPointY[i]} \\ 0 & 0 & 1 \end{bmatrix} \quad \text{(H-15)}$$

prec_rotation_param specifies the exponent of the maximum allowable truncation error for r[i][j][k] as given by $2^{-\text{prec\_rotation\_param}}$. The value of prec_rotation_param shall be in the range of 0 to 31, inclusive.

prec_translation_param specifies the exponent of the maximum allowable truncation error for t[i][j] as given by $2^{-\text{prec\_translation\_param}}$. The value of prec_ translation_param shall be in the range of 0 to 31, inclusive.

sign_r[i **][** j **][** k **]** equal to 0 indicates that the sign of (j, k) component of the rotation matrix for the i-th camera is positive. sign_r[i][j][k] equal to 1 indicates that the sign is negative.

exponent_r[i **][** j **][** k **]** specifies the exponent part of (j, k) component of the rotation matrix for the i-th camera. The value of exponent_r[i][j][k] shall be in the range of 0 to 62, inclusive. The value 63 is reserved for future use by ITU-T | ISO/IEC. Decoders shall treat the value 63 as indicating an unspecified rotation matrix.

mantissa_r[i **][** j **][** k **]** specifies the mantissa part of (j, k) component of the rotation matrix for the i-th camera. The length of the mantissa_r[i][j][k] syntax element in units of bits is variable and determined as follows:

– If exponent_r[i] = = 0, the length is Max(0, prec_rotation_param − 30).

– Otherwise (0 < exponent_r[i] < 63), the length is Max(0, exponent_r[i] + prec_rotation_param − 31).

The rotation matrix R[i] for i-th camera is represented as follows:

$$\begin{bmatrix} \text{rE[i][0][0]} & \text{rE[i][0][1]} & \text{rE[i][0][2]} \\ \text{rE[i][1][0]} & \text{rE[i][1][1]} & \text{rE[i][1][2]} \\ \text{rE[i][2][0]} & \text{rE[i][2][1]} & \text{rE[i][2][2]} \end{bmatrix} \quad \text{(H-16)}$$

sign_t[i **][** j **]** equal to 0 indicates that the sign of the j-th component of the translation vector for the i-th camera is positive. sign_t[i][j] equal to 1 indicates that the sign is negative.

exponent_t[i **][** j **]** specifies the exponent part of the j-th component of the translation vector for the i-th camera. The value of exponent_t[i][j] shall be in the range of 0 to 62, inclusive. The value 63 is reserved for future use by ITU-T | ISO/IEC. Decoders shall treat the value 63 as indicating an unspecified translation vector.

mantissa_t[i **][** j **]** specifies the mantissa part of the j-th component of the translation vector for the i-th camera. The length v of the mantissa_t[i][j] syntax element in units of bits is variable and is determined as follows:

– If exponent_t[i] = = 0, the length v = Max(0, prec_translation_param − 30).

– Otherwise (0 < exponent_t[i] < 63), the length v = Max(0, exponent_t[i] + prec_translation_param − 31).

The translation vector T[i] for the i-th camera is represented by:

$$\begin{bmatrix} \text{tE[i][0]} \\ \text{tE[i][1]} \\ \text{tE[i][2]} \end{bmatrix} \quad \text{(H-17)}$$

The association between the camera parameter variables and corresponding syntax elements is specified by Table H-2. Each component of the intrinsic and rotation matrices and the translation vector is obtained from the variables specified in Table H-2 as the variable x computed as follows:

– If $0 < e < 63$, $x = (-1)^s * 2^{e-31} * (1 + n \div 2^v)$.

– Otherwise (e is equal to 0), $x = (-1)^s * 2^{-(30+v)} * n$.

NOTE – The above specification is similar to that found in IEC 60559:1989, *Binary floating-point arithmetic for microprocessor systems*.

Table H-2 – Association between camera parameter variables and syntax elements.

x	s	e	n
focalLengthX[i **]**	sign_focal_length_x[i]	exponent_focal_length_x[i]	mantissa_focal_length_x[i]
focalLengthY[i **]**	sign_focal_length_y[i]	exponent_focal_length_y[i]	mantissa_focal_length_y[i]
principalPointX[i **]**	sign_principal_point_x[i]	exponent_principal_point_x[i]	mantissa_principal_point_x[i]
principalPointY[i **]**	sign_principal_point_y[i]	exponent_principal_point_y[i]	mantissa_principal_point_y[i]
skewFactor[i **]**	sign_skew_factor[i]	exponent_skew_factor[i]	mantissa_skew_factor[i]
rE[i **][** j **][** k **]**	sign_r[i][j][k]	exponent_r[i][j][k]	mantissa_r[i][j][k]
tE[i **][** j **]**	sign_t[i][j]	exponent_t[i][j]	mantissa_t[i][j]

H.13.2.6 Non-required view component SEI message semantics

This SEI message indicates non-required view components within the associated access unit. A view component is a non-required view component for a target view component if it is not needed for decoding the target view component and subsequent view components with the same view_id in decoding order within the coded video sequence.

Some of the view components indicated by view_order_index[i] or index_delta_minus1[i][j] may not be present in the associated access unit.

num_info_entries_minus1 plus 1 specifies the number of target view components for which non-required view components are indicated. The value of num_info_entries_minus1 shall be in the range of 0 to num_views_minus1 − 1, inclusive.

view_order_index[i **]** specifies the view order index of the i-th target view component for which non-required view components are indicated. The i-th target view component has view_id equal to view_id[view_order_index [i]]. The value of view_order_index[i] shall be in the range of 1 to num_views_minus1, inclusive.

num_non_required_view_components_minus1[i] plus 1 specifies the number of non-required view components for the i-th target view component. The value of num_non_required_view_components_minus1[i] shall be in the range of 0 to view_order_index[i] − 1, inclusive.

index_delta_minus1[i **][** j **]** plus 1 specifies the difference between the view order index of the i-th target view component and the view order index of the j-th non-required view component for the i-th target view component. The view order index of the j-th non-required view component for the i-th target view component is view_order_index[i] − index_delta_minus1[i][j] − 1. The value of index_delta_minus1[i][j] shall be in the range of 0 to view_order_index[i] − 1, inclusive.

H.13.2.7 View dependency change SEI message semantics

This SEI message indicates that the view dependency information changes starting with the current access unit containing the SEI message and is always interpreted with respect to the active MVC sequence parameter set. When present, the view dependency change SEI message applies to the target access unit set that consists of the current access unit and all the subsequent access units, in decoding order, until the next view dependency change SEI message or the end of the coded video sequence, whichever is earlier in decoding order.

If, according to the view dependency information indicated in the active MVC sequence parameter set, view component A does not directly or indirectly depend on view component B and vice versa, the view dependency change SEI message shall not specify view dependency relationship between view components A and B.

NOTE 1 – The dependent views for any view are always a subset of those indicated by the active MVC sequence parameter set.

NOTE 2 – View dependency change SEI messages do not have a cumulative effect.

Some of the views indicated by the following syntax elements may not be present in the target access unit set.

seq_parameter_set_id specifies a subset sequence parameter set that contains the inter-view dependency relationship information. The value of seq_parameter_set_id shall be equal to the value of seq_parameter_set_id in the picture parameter set referenced by a view component of the primary coded picture of the access unit containing the view dependency change SEI message. The value of seq_parameter_set_id shall be in the range of 0 to 31, inclusive.

anchor_update_flag equal to 1 indicates that there are updates for the dependencies for anchor view components relative to the dependencies defined in the active MVC sequence parameter set. anchor_update_flag equal to 0 indicates that there is no change for the dependencies for anchor view components relative to the dependencies defined in the active MVC sequence parameter set.

non_anchor_update_flag equal to 1 indicates that there are updates for the dependencies for non-anchor view components relative to the dependencies defined in the active MVC sequence parameter set. non_anchor_update_flag equal to 0 indicates that there is no change for the dependencies for non-anchor view components relative to the dependencies defined in the active MVC sequence parameter set.

anchor_ref_l0_flag[i **][** j **]** equal to 0 indicates that the j-th inter-view prediction reference in the initial reference picture list RefPicList0 (which is derived as specified in subclause H.8.2.1) for any anchor view component with view order index equal to i will not be present in the final RefPicList0 after reference picture list modification for the anchor view component. anchor_ref_l0_flag[i][j] equal to 1 indicates that the j-th inter-view prediction reference in the initial reference picture list RefPicList0 for at least one anchor view component with view order index equal to i will be present in the final RefPicList0 after reference picture list modification for the anchor view component.

anchor_ref_l1_flag[i **][** j **]** equal to 0 indicates that the j-th inter-view prediction reference in the initial reference picture list RefPicList1 (which is derived as specified in subclause H.8.2.1) for any anchor view component with view order index equal to i will not be present in the final RefPicList1 after reference picture list modification for the anchor view component. anchor_ref_l1_flag[i][j] equal to 1 indicates that the j-th inter-view prediction reference in the initial reference picture list RefPicList1 for at least one anchor view component with view order index equal to i will be present in the final RefPicList1 after reference picture list modification for the anchor view component.

non_anchor_ref_l0_flag[i **][** j **]** equal to 0 indicates that the j-th inter-view prediction reference in the initial reference picture list RefPicList0 (which is derived as specified in subclause H.8.2.1) for any non-anchor view component with view order index equal to i will not be present in the final RefPicList0 after reference picture list modification for the non-anchor view component. non-anchor_ref_l0_flag[i][j] equal to 1 indicates that the j-th inter-view prediction reference in the initial reference picture list RefPicList0 for at least one non-anchor view component with view order index equal to i will be present in the final RefPicList0 after reference picture list modification for the non-anchor view component.

non_anchor_ref_l1_flag[i **][** j **]** equal to 0 indicates that the j-th inter-view prediction reference in the initial reference picture list RefPicList1 (which is derived as specified in subclause H.8.2.1) for any non-anchor view component with view order index equal to i will not be present in the final RefPicList1 after reference picture list modification for the non-anchor view component. non-anchor_ref_l1_flag[i][j] equal to 1 indicates that the j-th inter-view prediction reference in the initial reference picture list RefPicList1 for at least one non-anchor view component with view order index equal to i will be present in the final RefPicList1 after reference picture list modification for the non-anchor view component.

H.13.2.8 Operation point not present SEI message semantics

This SEI message indicates operation points that are not present in the bitstream starting with the current access unit, and is interpreted with respect to the previous view scalability information SEI message in decoding order. The message remains effective until the next SEI message of the same type or the end of the coded video sequence, whichever is earlier in decoding order.

NOTE 1– Operation point not present SEI messages do not have a cumulative effect.

NOTE 2 – Any operation point identified by a value of operation_point_id[i] in the previous view scalability information SEI message, in decoding order, and not identified by a value of operation_point_not_present_id[k] must be present in the coded video sequence. Therefore, when an application keeps an operation point not present SEI message in a sub-bitstream extracted according to the process specified in subclause H.8.5.3, the application may need to change the content of the operation point not present SEI message according to the semantics.

num_operation_points specifies the number of operation points that are indicated not to be present by the SEI message. num_operation_points equal to 0 indicates that all operation points indicated by the view scalability information SEI message are present. The value of num_operation_points shall be in the range of 0 to the value of num_operation_points_minus1 in the previous view scalability information SEI message in decoding order, inclusive.

operation_point_not_present_id[k **]** identifies an operation point that is not present. operation_point_not_present_id[k] shall be equal to the value of one of the operation_point_id[i] syntax elements of the previous view scalability information SEI message in decoding order. The value of operation_point_not_present_id[k] shall be in the range of 0 to 65535, inclusive.

H.13.2.9 Base view temporal HRD SEI message semantics

When present, this SEI message shall be associated with an IDR access unit. The SEI message applies to the coded video sequence. Some temporal subsets identified by sei_mvc_temporal_id[i] may not be present in the coded video sequence.

num_of_temporal_layers_in_base_view_minus1 plus 1 specifies the number of temporal bitstream subsets in the coded video sequence for which the following syntax elements apply. The value of num_of_temporal_layers_in_base_view_minus1 shall be in the range of 0 to 7, inclusive.

sei_mvc_temporal_id[i **]** specifies the temporal_id value of the i-th temporal bitstream subset.

Let the i-th bitstream subset for the coded video sequence that is obtained by invoking the sub-bitstream extraction process as specified in subclause H.8.5.3 with tIdTarget equal to sei_mvc_temporal_id[i] as input.

sei_mvc_timing_info_present_flag[i **]** equal to 1 specifies that sei_mvc_num_units_in_tick[i], sei_mvc_time_scale[i], and sei_mvc_fixed_frame_rate_flag[i] are present in the base view temporal HRD SEI message. sei_mvc_timing_info_present_flag[i] equal to 0 specifies that sei_mvc_num_units_in_tick[i], sei_mvc_time_scale[i], and sei_mvc_fixed_frame_rate_flag[i] are not present in the base view temporal HRD SEI message.

The following syntax elements for the i-th bitstream subset are specified using references to Annex E. For these syntax elements the same semantics and constraints as the ones specified in Annex E apply, as if these syntax elements sei_mvc_num_units_in_tick[i], sei_mvc_time_scale[i], sei_mvc_fixed_frame_rate_flag[i], sei_mvc_nal_hrd_parameters_present_flag[i], sei_mvc_vcl_hrd_parameters_present_flag[i], sei_mvc_low_delay_hrd_flag[i], and sei_mvc_pic_struct_present_flag[i] were present as num_units_in_tick, time_scale, fixed_frame_rate_flag, nal_hrd_parameters_present_flag, vcl_hrd_parameters_present_flag, low_delay_hrd_flag, and pic_struct_present_flag, respectively, in the VUI parameters of the active MVC sequence parameter sets for the i-th bitstream subset.

The parameters for the i-th bitstream subset that are coded in the base view temporal HRD SEI message shall be correct, as if these parameters are used for conformance checking (as specified in Annex C) of the i-th bitstream subset.

sei_mvc_num_units_in_tick[i **]** indicates the value of num_units_in_tick, as specified in subclause E.2.1, that applies to the i-th bitstream subset.

sei_mvc_time_scale[i **]** indicates the value of time_scale, as specified in subclause E.2.1, that applies to the i-th bitstream subset.

sei_mvc_fixed_frame_rate_flag[i **]** indicates the value of fixed_frame_rate_flag, as specified in subclause E.2.1, that applies to the i-th bitstream subset.

sei_mvc_nal_hrd_parameters_present_flag[i **]** indicates the value of nal_hrd_parameters_present_flag, as specified in subclause E.2.1, that applies to the i-th bitstream subset. When sei_mvc_nal_hrd_parameters_present_flag[i] is equal to 1, the NAL HRD parameters that apply to the i-th bitstream subset immediately follow the sei_mvc_nal_hrd_parameters_present_flag[i].

sei_mvc_vcl_hrd_parameters_present_flag[i **]** indicates the value of vcl_hrd_parameters_present_flag, as specified in subclause E.2.1, that applies to the i-th bitstream subset. When sei_mvc_vcl_hrd_parameters_present_flag[i] is equal to 1, the VCL HRD parameters that apply to the i-th bitstream subset immediately follow the sei_mvc_vcl_hrd_parameters_present_flag[i].

sei_mvc_low_delay_hrd_flag[i **]** indicates the value of low_delay_hrd_flag, as specified in subclause E.2.1, that applies to the i-th bitstream subset.

sei_mvc_pic_struct_present_flag[i **]** indicates the value of pic_struct_present_flag, as specified in subclause E.2.1, that applies to the i-th bitstream subset.

H.14 Video usability information

The specifications in Annex E apply with substituting MVC sequence parameter set for sequence parameter set.

With maxVOIdx being the maximum value of view order index for the views that reference the MVC sequence parameter set containing the vui_parameters( ) syntax structure, the VUI parameters and the constraints specified in Annex E apply to all views with a value of view order index that is less than or equal to maxVOIdx.

Additionally, the following applies.

H.14.1 MVC VUI parameters extension syntax

mvc_vui_parameters_extension( ) {	**C**	**Descriptor**
vui_mvc_num_ops_minus1	0	ue(v)
for(i = 0; i <= vui_mvc_num_ops_minus1; i++) {		
vui_mvc_temporal_id[i **]**	0	u(3)

vui_mvc_num_target_output_views_minus1[i **]**	5	ue(v)
for(j = 0; j <= vui_mvc_num_target_output_views_minus1[i]; j++)		
vui_mvc_view_id[i **][** j **]**	5	ue(v)
vui_mvc_timing_info_present_flag[i **]**	0	u(1)
if(vui_mvc_timing_info_present_flag[i]) {		
vui_mvc_num_units_in_tick[i **]**	0	u(32)
vui_mvc_time_scale[i **]**	0	u(32)
vui_mvc_fixed_frame_rate_flag[i **]**	0	u(1)
}		
vui_mvc_nal_hrd_parameters_present_flag[i **]**	0	u(1)
if(vui_mvc_nal_hrd_parameters_present_flag[i])		
hrd_parameters( )	0	
vui_mvc_vcl_hrd_parameters_present_flag[i **]**	0	u(1)
if(vui_mvc_vcl_hrd_parameters_present_flag[i])		
hrd_parameters( )	0	
if(vui_mvc_nal_hrd_parameters_present_flag[i] || vui_mvc_vcl_hrd_parameters_present_flag[i])		
vui_mvc_low_delay_hrd_flag[i **]**	0	u(1)
vui_mvc_pic_struct_present_flag[i **]**	0	u(1)
}		
}		

H.14.2 MVC VUI parameters extension semantics

The MVC VUI parameters extension specifies VUI parameters that apply to one or more operation points for the coded video sequence. In Annex C it is specified which of the HRD parameter sets specified in the MVC VUI parameters extension are used for conformance checking. All MVC VUI parameters extensions that are referred to by a coded video sequence shall be identical.

Some views identified by vui_mvc_view_id[i][j] may not be present in the coded video sequence. Some temporal subsets identified by vui_mvc_temporal_id[i] may not be present in the coded video sequence.

vui_mvc_num_ops_minus1 plus 1 specifies the number of operation points for which timing information, NAL HRD parameters, VCL HRD parameters, and the pic_struct_present_flag may be present. The value of vui_mvc_num_ops_minus1 shall be in the range of 0 to 1023, inclusive.

vui_mvc_temporal_id[i **]** indicates the maximum value of temporal_id for all VCL NAL units in the representation of the i-th operation point.

vui_mvc_num_target_output_views_minus1[i **]** plus one specifies the number of target output views for the i-th operation point. The value of vui_mvc_num_target_output_views_minus1[i] shall be in the range of 0 to 1023, inclusive.

vui_mvc_view_id[i **][** j **]** indicates the j-th target output view in the i-th operation point. The value of vui_mvc_view_id[i] shall be in the range of 0 to 1023, inclusive.

The following syntax elements apply to the coded video sequence that is obtained by the sub-bitstream extraction process as specified in subclause H.8.5.3 with tIdTarget equal to vui_mvc_temporal_id[i] and viewIdTargetList containing vui_mvc_view_id[i][j] for all j in the range of 0 to vui_mvc_num_target_output_views_minus1[i], inclusive, as the inputs and the i-th sub-bitstream as the output.

vui_mvc_timing_info_present_flag[i **]** equal to 1 specifies that vui_mvc_num_units_in_tick[i], vui_mvc_time_scale[i], and vui_mvc_fixed_frame_rate_flag[i] for the i-th sub-bitstream are present in the MVC VUI parameters extension. vui_mvc_timing_info_present_flag[i] equal to 0 specifies that vui_mvc_num_units_in_tick[i], vui_mvc_time_scale[i], and vui_mvc_fixed_frame_rate_flag[i] for the i-th sub-bitstream are not present in the MVC VUI parameters extension.

The following syntax elements for the i-th sub-bitstream are specified using references to Annex E. For these syntax elements the same semantics and constraints as the ones specified in Annex E apply, as if these syntax elements vui_mvc_num_units_in_tick[i], vui_mvc_time_scale[i], vui_mvc_fixed_frame_rate_flag[i], vui_mvc_nal_hrd_parameters_present_flag[i], vui_mvc_vcl_hrd_parameters_present_flag[i], vui_mvc_low_delay_hrd_flag[i], and vui_mvc_pic_struct_present_flag[i] were present as the syntax elements num_units_in_tick, time_scale, fixed_frame_rate_flag, nal_hrd_parameters_present_flag, vcl_hrd_parameters_present_flag, low_delay_hrd_flag, and pic_struct_present_flag, respectively, in the VUI parameters of the active MVC sequence parameter sets for the i-th sub-bitstream.

vui_mvc_num_units_in_tick[i **]** specifies the value of num_units_in_tick, as specified in subclause E.2.1, for the i-th sub-bitstream.

vui_mvc_time_scale[i **]** specifies the value of time_scale, as specified in subclause E.2.1, for the i-th sub-bitstream.

vui_mvc_fixed_frame_rate_flag[i **]** specifies the value of fixed_frame_rate_flag, as specified in subclause E.2.1, for the i-th sub-bitstream.

vui_mvc_nal_hrd_parameters_present_flag[i **]** specifies the value of nal_hrd_parameters_present_flag, as specified in subclause E.2.1, for the i-th sub-bitstream.

When vui_mvc_nal_hrd_parameters_present_flag[i] is equal to 1, NAL HRD parameters (subclauses E.1.2 and E.2.2) for the i-th sub-bitstream immediately follow the flag.

The variable VuiMvcNalHrdBpPresentFlag[i] is derived as follows:

– If any of the following is true, the value of VuiMvcNalHrdBpPresentFlag[i] shall be set equal to 1:
  – vui_mvc_nal_hrd_parameters_present_flag[i] is present in the bitstream and is equal to 1,
  – for the i-th sub-bitstream, the need for presence of buffering periods for NAL HRD operation to be present in the bitstream in buffering period SEI messages is determined by the application, by some means not specified in this Recommendation | International Standard.
– Otherwise, the value of VuiMvcNalHrdBpPresentFlag[i] shall be set equal to 0.

vui_mvc_vcl_hrd_parameters_present_flag[i **]** specifies the value of vcl_hrd_parameters_present_flag, as specified in subclause E.2.1, for the i-th sub-bitstream.

When vui_mvc_vcl_hrd_parameters_present_flag[i] is equal to 1, VCL HRD parameters (subclauses E.1.2 and E.2.2) for the i-th sub-bitstream immediately follow the flag.

The variable VuiMvcVclHrdBpPresentFlag[i] is derived as follows:

– If any of the following is true, the value of VuiMvcVclHrdBpPresentFlag[i] shall be set equal to 1:
  – vui_mvc_vcl_hrd_parameters_present_flag[i] is present in the bitstream and is equal to 1,
  – for the i-th sub-bitstream, the need for presence of buffering periods for VCL HRD operation to be present in the bitstream in buffering period SEI messages is determined by the application, by some means not specified in this Recommendation | International Standard.
– Otherwise, the value of VuiMvcVclHrdBpPresentFlag[i] shall be set equal to 0.

The variable VuiMvcCpbDpbDelaysPresentFlag[i] is derived as follows:

– If any of the following is true, the value of VuiMvcCpbDpbDelaysPresentFlag[i] shall be set equal to 1:
  – vui_mvc_nal_hrd_parameters_present_flag[i] is present in the bitstream and is equal to 1,
  – vui_mvc_vcl_hrd_parameters_present_flag[i] is present in the bitstream and is equal to 1,
  – for the i-th sub-bitstream, the need for presence of CPB and DPB output delays to be present in the bitstream in picture timing SEI messages is determined by the application, by some means not specified in this Recommendation | International Standard.
– Otherwise, the value of VuiMvcCpbDpbDelaysPresentFlag[i] shall be set equal to 0.

vui_mvc_low_delay_hrd_flag[i **]** specifies the value of low_delay_hrd_flag, as specified in subclause E.2.1, for the i-th sub-bitstream.

vui_mvc_pic_struct_present_flag[i **]** specifies the value of pic_struct_present_flag, as specified in subclause E.2.1, for the i-th sub-bitstream.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	**Audiovisual and multimedia systems**
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Terminals and subjective and objective assessment methods
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2012