ITUPublications
Recommendations

International Telecommunication Union
Standardization Sector

Recommendation

ITU-T H.266 (V3) (09/2023)

SERIES H: Audiovisual and multimedia systems

Infrastructure of audiovisual services – Coding of moving video

Versatile video coding

ITU-T H-SERIES RECOMMENDATIONS

Audiovisual and multimedia systems

CHARACTERISTICS OF VISUAL TELEPHONE SYSTEMS	H.100-H.199
INFRASTRUCTURE OF AUDIOVISUAL SERVICES	H.200-H.499
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
Telepresence, immersive environments, virtual and extended reality	H.420-H.439
Supplementary services for multimedia	H.450-H.499
MOBILITY AND COLLABORATION PROCEDURES	H.500-H.549
VEHICULAR GATEWAYS AND INTELLIGENT TRANSPORTATION SYSTEMS (ITS)	H.550-H.599
BROADBAND, TRIPLE-PLAY AND ADVANCED MULTIMEDIA SERVICES	H.600-H.699
IPTV MULTIMEDIA SERVICES AND APPLICATIONS FOR IPTV	H.700-H.799
E-HEALTH MULTIMEDIA SYSTEMS, SERVICES AND APPLICATIONS	H.800-H.899

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

Recommendation ITU-T H.266 (V3)

Versatile video coding

Summary

Recommendation ITU-T H.266 specifies a video coding technology known as *Versatile Video Coding* and it has been designed with two primary goals. The first of these is to specify a video coding technology with a compression capability that is substantially beyond that of the prior generations of such standards, and the second is for this technology to be highly versatile for effective use in a broadened range of applications than that addressed by prior standards. Some key application areas for the use of this standard particularly include ultra-high-definition video (e.g., with 3840×2160 or 7620×4320 picture resolution and bit depth of 10 bits as specified in Rec. ITU-R BT.2100), video with a high dynamic range and wide colour gamut (e.g., with the perceptual quantization or hybrid log-gamma transfer characteristics specified in Rec. ITU-R BT.2100), and video for immersive media applications such as 360° omnidirectional video projected using a common projection format such as the equirectangular or cubemap projection formats, in addition to the applications that have commonly been addressed by prior video coding standards.

The third edition adds the specification of a new level (level 15.5) for the video profiles to provide a suitable label for bitstreams that can exceed the limits of all other specified levels, additional supplement enhancement information, and corrections to various minor defects in the prior content of the Recommendation.

This Recommendation was developed collaboratively with ISO/IEC JTC 1/SC 29, and corresponds with ISO/IEC 23090-3 as technically aligned twin text.

History *

Edition	Recommendation	Approval	Study Group	Unique ID
1.0	ITU-T H.266	2020-08-29	16	11.1002/1000/14336
2.0	ITU-T H.266 (V2)	2022-04-29	16	11.1002/1000/14948
3.0	ITU-T H.266 (V3)	2023-09-29	16	11.1002/1000/15648

* To access the Recommendation, type the URL https://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID.

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/software copyrights, 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 appropriate ITU-T databases available via the ITU-T website at http://www.itu.int/ITU-T/ipr/.

CONTENTS

Page

List of Tables

Page

Page

Page

Page

List of Figures

Page

Recommendation ITU-T H.266 (V3)

Versatile video coding

Introduction

Purpose

This Recommendation | International Standard specifies Recommendation ITU-T H.266 | International Standard ISO/IEC 23090-3, a video coding technology known as *Versatile Video Coding*. It has been designed with two primary goals. The first of these is to specify a video coding technology with a compression capability that is substantially beyond that of the prior generations of such standards, and the second is for this technology to be highly versatile for effective use in a broader range of applications than that addressed by prior standards. Some key application areas for the use of this standard particularly include ultra-high-definition video (e.g., with 3840×2160 or 7620×4320 picture resolution and bit depth of 10 bits as specified in Rec. ITU-R BT.2100), video with a high dynamic range and wide colour gamut (e.g., with the perceptual quantization or hybrid log-gamma transfer characteristics specified in Rec. ITU-R BT.2100), and video for immersive media applications such as 360° omnidirectional video projected using a common projection format such as the equirectangular or cubemap projection formats, in addition to the applications that have commonly been addressed by prior video coding standards.

Profiles, tiers, and levels

This Recommendation | International Standard is designed to be versatile in the sense that it serves a wide range of applications, bit rates, resolutions, qualities, and services. Applications include, but are not limited to, video coding for digital storage media, television broadcasting, video streaming services, real-time communication. In the course of creating this Recommendation | International Standard, 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 Recommendation | International Standard is designed to facilitate video data interchange among different applications.

Considering the practicality of implementing the full syntax of this Recommendation | International Standard, however, a limited number of subsets of the syntax are also stipulated by means of "profiles", "tiers", 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 in 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 economical 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, "tiers" and "levels" are specified within each profile. A level of a tier is a specified set of constraints imposed on values of the syntax elements in the bitstream. Some of these constraints are expressed as simple limits on values, while others 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). A level specified for a lower tier is more constrained than a level specified for a higher tier.

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.

Encoding process, decoding process, and use of VUI parameters and SEI messages

Any encoding process that produces bitstream data that conforms to the specified bitstream syntax format requirements of this Recommendation | International Standard is considered to be in conformance with the requirements of this Recommendation | International Standard. The decoding process is specified such that all decoders that conform to a specified combination of capabilities known as the profile, tier, and level will produce numerically identical cropped decoded output pictures when invoking the decoding process associated with that profile for a bitstream conforming to that profile, tier and level. Any decoding process that produces identical cropped decoded output pictures to those produced by the process described herein (with the correct output order or output timing, as specified) is considered to be in conformance with the requirements of this Recommendation | International Standard.

Rec. ITU-T H.274 | ISO/IEC 23002-7 specifies the syntax and semantics of the video usability information (VUI) parameters and supplemental enhancement information (SEI) messages that do not affect the conformance specifications in Annex C. These VUI parameters and SEI messages may be used together with this Recommendation | International Standard.

Versions of this Recommendation | International Standard

Rec. ITU-T H.266 | ISO/IEC 23090-3 version 1 refers to the first approved version of this Recommendation | International Standard. The first edition published by ITU-T as Rec. ITU-T H.266 (08/2020) and by ISO/IEC as ISO/IEC 23090-3:2021 corresponded to the first version.

Rec. ITU-T H.266 | ISO/IEC 23090-3 version 2 refers to the integrated text additionally containing operation range extensions, a new level (level 6.3), additional supplement enhancement information, and corrections to various minor defects in the prior content of the Specification. The second edition published by ITU-T as Rec. ITU-T H.266 (04/2022) and by ISO/IEC as ISO/IEC 23090-3:2022 corresponded to the second version.

Rec. ITU-T H.266 | ISO/IEC 23090-3 version 3 (the current version) refers to the integrated text containing the specification of a new level (level 15.5) for the video profiles to provide a suitable label for bitstreams that can exceed the limits of all other specified levels, additional supplement enhancement information, and corrections to various minor defects in the prior content of the Specification. The third edition published by ITU-T as Rec. ITU-T H.266 (09/2023) corresponds to the third version. At the time of publication of this edition by ITU-T, a corresponding third edition of ISO/IEC 23090-3 was in preparation for publication by ISO/IEC.

Overview of the design characteristics

The coded representation specified in the syntax is designed to enable a high compression capability for a desired image or video quality. The algorithm is typically not mathematically lossless, as the exact source sample values are typically not preserved through the encoding and decoding processes, although some modes are included that provide lossless coding capability. A number of techniques are specified to enable highly efficient compression. Encoding algorithms (not specified within the scope of this Recommendation | International Standard) may select between inter, intra, intra block copy (IBC), and palette coding for block-shaped regions of each picture. Inter coding uses motion vectors for block-based inter-picture 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 within the same picture, and intra block copy coding uses block displacement vectors to reference previously decoded regions of the same picture to exploit statistical similarities among different areas of the same picture. Motion vectors, intra prediction modes, and IBC block vectors are specified for a variety of block sizes in the picture. The prediction residual can then be further compressed using a spatial transform to remove spatial correlation inside a block before it is quantized, producing a possibly irreversible process that typically discards less important visual information while forming a close approximation to the source samples. Finally, the motion vectors, intra prediction modes, and block vectors can also be further compressed using a variety of prediction mechanisms, and, after prediction, are combined with the quantized transform coefficient information and encoded using arithmetic coding.

How to read this document

It is suggested that the reader starts with clause 1 and moves on to clause 3. Clause 6 should be read for the geometrical relationship of the source, input, and output of the decoder. Clause 7 specifies the order to parse syntax elements from the bitstream. See clauses 7.1 to 7.3 for syntactical order and clause 7.4 for semantics; e.g., the scope, restrictions, and conditions that are imposed on the syntax elements. The actual parsing for most syntax elements is specified in clause 9. Finally, clause 8 specifies how the syntax elements are mapped into decoded samples. Throughout reading this document, the reader should refer to clauses 2, 4, and 5 as needed. Annexes A through D also form an integral part of this Recommendation | International Standard.

Annex A specifies profiles, each being tailored to certain application domains, and defines the so-called tiers and 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, bitstream conformance, decoder conformance, and the use of the hypothetical reference decoder to check bitstream and decoder conformance. Annex D specifies syntax and semantics for supplemental enhancement information (SEI) message payloads that affect the conformance specifications in Annex C. Rec. ITU-T H.274 | ISO/IEC 23002-7 specifies the syntax and semantics of the video usability information (VUI) parameters as well as SEI messages that do not affect the conformance specifications in Annex C. These VUI parameters and SEI messages may be used together with this Recommendation | International Standard.

1 Scope

This Recommendation | International Standard specifies a video coding technology known as *Versatile Video Coding* (VVC), comprising a video coding technology with a compression capability that is substantially beyond that of the prior generations of such standards and with sufficient versatility for effective use in a broad range of applications.

Only the syntax format, semantics, and associated decoding process requirements are specified, while other matters such as pre-processing, the encoding process, system signalling and multiplexing, data loss recovery, post-processing, and video display are considered to be outside the scope of this Recommendation | International Standard. Additionally, the internal processing steps performed within a decoder are also considered to be outside the scope of this Recommendation | International Standard; only the externally observable output behaviour is required to conform to the specifications 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 include, but are not limited to, video coding for digital storage media, television broadcasting and real-time communication. In the course of creating this Recommendation | International Standard, 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 Recommendation | International Standard is designed to facilitate video data interchange among different applications.

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.

2.1 Identical Recommendations | International Standards

– None.

2.2 Paired Recommendations | International Standards equivalent in technical content

– Rec. ITU-T H.274 | ISO/IEC 23002-7 (in force) *Versatile supplemental enhancement information messages for coded video bitstreams.*

2.3 Additional references

– Rec. ITU-T T.35 (in force), *Procedure for the allocation of ITU-T defined codes for non standard facilities.*

– ISO/IEC 23001-11 (in force), *Information Technology – MPEG Systems technologies — Part 11: Energy-efficient media consumption (green metadata).*

– ISO/IEC 23090-13 (in force), *Information technology – Coded representation of immersive media – Part 13: Video decoding interface for immersive media.*

3 Definitions

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

3.1 **AC transform coefficient**: Any *transform coefficient* for which the *frequency index* in at least one of the two dimensions is non-zero.

3.2 **access unit (AU)**: A set of *PUs* that belong to different *layers* and contain *coded pictures* associated with the same time for output from the *DPB*.

3.3 **adaptation parameter set (APS)**: A *syntax structure* containing *syntax elements* that apply to zero or more *slices* as determined by zero or more *syntax elements* found in *slice headers.*

3.4 **adaptive colour transform (ACT)**: A cross-*component transform* applied to the decoded *residual* of a *coding unit* in the 4:4:4 colour format prior to reconstruction and loop filtering.

3.5 **adaptive loop filter (ALF)**: A filtering process that is applied as part of the *decoding process* and is controlled by parameters conveyed in an *APS*.

3.6 **ALF APS**: An *APS* that controls the *ALF* process.

3.7 **associated GDR picture**: The previous *GDR picture* (when present) in *decoding order*, for a particular picture with nuh_layer_id equal to a particular value layerId, that has nuh_layer_id equal to layerId and between which and the particular *picture* in *decoding order* there is no *IRAP picture* with nuh_layer_id equal to layerId.

3.8 **associated GDR subpicture**: The previous *GDR subpicture* (when present) in *decoding order*, for a particular subpicture with nuh_layer_id equal to a particular value layerId and subpicture index equal to a particular value subpicIdx, that has nuh_layer_id equal to layerId and subpicture index equal to subpicIdx and between which and the particular sub*picture* in *decoding order* there is no *IRAP subpicture* with nuh_layer_id equal to layerId and subpicture index equal to subpicIdx.

3.9 **associated IRAP picture**: The previous *IRAP picture* (when present) in *decoding order*, for a particular picture with nuh_layer_id equal to a particular value layerId, that has nuh_layer_id equal to layerId and between which and the particular *picture* in *decoding order* there is no *GDR picture* with nuh_layer_id equal to layerId.

3.10 **associated IRAP subpicture**: The previous *IRAP subpicture* (when present) in *decoding order*, for a particular subpicture with nuh_layer_id equal to a particular value layerId and subpicture index equal to a particular value subpicIdx, that has nuh_layer_id equal to layerId and subpicture index equal to subpicIdx and between which and the particular sub*picture* in *decoding order* there is no *GDR subpicture* with nuh_layer_id equal to layerId and subpicture index equal to subpicIdx.

3.11 **associated non-VCL NAL unit**: A *non-VCL NAL unit* (when present) for a *VCL NAL unit* where the *VCL NAL unit* is the *associated VCL NAL unit* of the *non-VCL NAL unit*.

3.12 **associated VCL NAL unit**: The preceding *VCL NAL unit* in *decoding order* for a *non-VCL NAL unit* with nal_unit_type equal to EOS_NUT, EOB_NUT, SUFFIX_APS_NUT, SUFFIX_SEI_NUT, FD_NUT, RSV_NVCL_27, UNSPEC_30, or UNSPEC_31; or otherwise the next *VCL NAL unit* in *decoding order*.

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

3.14 **bin string**: An intermediate binary representation of values of *syntax elements* from the *binarization* of the *syntax element*.

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

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

3.17 **binary split**: A split of a rectangular MxN *block* of samples into two *blocks* where a vertical split results in a first (M / 2)xN *block* and a second (M / 2)xN *block*, and a horizontal split results in a first Mx(N / 2) *block* and a second Mx(N / 2) *block*.

3.18 **bi-predictive (B) slice**: A *slice* that is decoded using *intra prediction* or using *inter prediction* with at most two *motion vectors* and *reference indices* to *predict* the sample values of each *block*.

3.19 **bitstream**: A sequence of bits, in the form of a *NAL unit stream* or a *byte stream*, that forms the representation of a sequence of *AUs* forming one or more coded video sequences *(CVSs)*.

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

3.21 **block vector**: A two-dimensional vector that provides an offset from the coordinates of the current *coding block* to the coordinates of the reference block in the same decoded *slice*.

3.22 **byte**: A sequence of 8 bits, within which, when written or read as a sequence of bit values, the left-most and right-most bits represent the most and least significant bits, respectively.

3.23 **byte stream**: An encapsulation of a *NAL unit stream* into a series of *bytes* containing *start code prefixes* and *NAL units*.

3.24 **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*, and 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.

3.25 **chroma**: A sample array or single sample representing one of the two colour difference signals related to the primary colours, represented by the symbols 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.

3.26 **clean random access (CRA) picture**: An *IRAP picture* for which each *VCL NAL unit* has nal_unit_type equal to CRA_NUT.

NOTE – A CRA picture does not use inter prediction in its decoding process, and could be the first picture in the bitstream in decoding order, or could appear later in the bitstream. A CRA picture could have associated RADL or RASL pictures. When a CRA picture has NoOutputBeforeRecoveryFlag equal to 1, the associated RASL pictures are not output by the decoder, because they might not be decodable, as they could contain references to pictures that are not present in the bitstream.

3.27 **clean random access (CRA) PU**: A *PU* in which the *coded picture* is a *CRA picture*.

3.28 **clean random access (CRA) subpicture**: An *IRAP subpicture* for which each *VCL NAL unit* has nal_unit_type equal to CRA_NUT.

3.29 **coded layer video sequence (CLVS)**: A sequence of *PUs* with the same value of nuh_layer_id that consists, in *decoding order*, of a *CLVSS PU*, followed by zero or more *PUs* that are not *CLVSS PUs*, including all subsequent *PUs* up to but not including any subsequent *PU* that is a *CLVSS PU*.

NOTE – A CLVSS PU could be an IDR PU, a CRA PU, or a GDR PU. The value of NoOutputBeforeRecoveryFlag is equal to 1 for each IDR PU, and each CRA PU that has HandleCraAsClvsStartFlag equal to 1, and each CRA or GDR PU that is the first PU in the layer of the bitstream in decoding order or the first PU in the layer of the bitstream that follows an EOS NAL unit in the layer in decoding order.

3.30 **coded layer video sequence start (CLVSS) PU**: A *PU* in which the *coded picture* is a *CLVSS picture*.

3.31 **coded layer video sequence start (CLVSS) picture**: A *coded picture* that is an *IRAP picture* with NoOutputBeforeRecoveryFlag equal to 1 or a *GDR picture* with NoOutputBeforeRecoveryFlag equal to 1.

3.32 **coded picture**: A *coded representation* of a *picture* comprising *VCL NAL units* with a particular value of nuh_layer_id within an *AU* and containing all *CTUs* of the *picture*.

3.33 **coded picture buffer (CPB)**: A first-in first-out buffer containing *DUs* in *decoding order* specified in the *hypothetical reference decoder* in Annex C.

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

3.35 **coded slice NAL unit**: A *NAL unit* that contains a coded *slice*.

3.36 **coded video sequence (CVS)**: A sequence of *AUs* that consists, in *decoding order*, of a *CVSS AU*, followed by zero or more *AUs* that are not *CVSS AUs*, including all subsequent *AUs* up to but not including any subsequent *AU* that is a *CVSS AU*.

3.37 **coded video sequence start (CVSS) AU**: An *IRAP AU* or *GDR AU* for which the *coded picture* in each *PU* is a *CLVSS picture*.

3.38 **coding block**: An MxN *block* of samples for some values of M and N such that the division of a *CTB* into *coding blocks* is a *partitioning*.

3.39 **coding tree block (CTB)**: An N×N *block* of samples for some value of N such that the division of a *component* into *CTBs* is a *partitioning*.

3.40 **coding tree unit (CTU)**: A *CTB* of *luma* samples, two corresponding *CTBs* of *chroma* samples of a *picture* that has three sample arrays, or a *CTB* of samples of a monochrome *picture*, and *syntax structures* used to code the samples.

3.41 **coding unit (CU)**: A *coding block* of *luma* samples, two corresponding *coding blocks* of *chroma* samples of a *picture* that has three sample arrays in the single tree mode, or a *coding block* of *luma* samples of a *picture* that has three sample arrays in the dual tree mode, or two *coding blocks* of *chroma* samples of a *picture* that has three sample arrays in the dual tree mode, or a *coding block* of samples of a monochrome *picture*, and *syntax structures* used to code the samples.

3.42 **component**: An array or single sample from one of the three arrays (*luma* and two *chroma*) that compose a *picture* in 4:2:0, 4:2:2, or 4:4:4 colour format or the array or a single sample of the array that compose a *picture* in monochrome format.

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

3.44 **deblocking filter**: A filtering process that is applied as part of the *decoding process* in order to minimize the appearance of visual artefacts at the boundaries between *blocks*.

3.45 **decoded picture**: A *picture* produced by applying the *decoding process* to a *coded picture*.

3.46 **decoded picture buffer (DPB)**: A buffer holding *decoded pictures* for reference, output reordering, or output delay specified for the *hypothetical reference decoder*.

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

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

3.49 **decoding process**: The process specified in this Specification that reads a *bitstream* and derives *decoded pictures* from it.

3.50 **decoding unit (DU)**: An *AU* if DecodingUnitHrdFlag is equal to 0 or a subset of an *AU* otherwise, consisting of one or more *VCL NAL units* in an *AU* and the *associated non-VCL NAL units*.

3.51 **emulation prevention byte**: A *byte* equal to 0x03 that is present within a *NAL unit* when the *syntax elements* of the *bitstream* form certain patterns of *byte* values in a manner that ensures that no sequence of consecutive *byte-aligned bytes* in the *NAL unit* can contain a *start code prefix*.

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

3.53 **encoding process**: A process not specified in this Specification that produces a *bitstream* conforming to this Specification.

3.54 **filler data NAL units**: *NAL units* with nal_unit_type equal to FD_NUT.

3.55 **flag**: A variable or single-bit *syntax element* that can take one of the two possible values: 0 and 1.

3.56 **frequency index**: A one-dimensional or two-dimensional index associated with a *transform coefficient* prior to the application of a *transform* in the *decoding process.*

3.57 **gradual decoding refresh (GDR) AU**: An *AU* in which there is a *PU* for each *layer* present in the *CVS* and the *coded picture* in each present *PU* is a *GDR picture*.

3.58 **gradual decoding refresh (GDR) PU**: A *PU* in which the *coded picture* is a *GDR picture*.

3.59 **gradual decoding refresh (GDR) picture:** A *picture* for which each VCL NAL unit has nal_unit_type equal to GDR_NUT.

NOTE – The value of pps_mixed_nalu_types_in_pic_flag for a GDR picture is equal to 0. When pps_mixed_nalu_types_in_pic_flag is equal to 0 for a picture, and any slice of the picture has nal_unit_type equal to GDR_NUT, all other slices of the picture have the same value of nal_unit_type, and the picture is known to be a GDR picture after receiving the first slice.

3.60 **gradual decoding refresh (GDR) subpicture:** A *subpicture* for which each VCL NAL unit has nal_unit_type equal to GDR_NUT.

3.61 **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.

3.62 **hypothetical stream scheduler (HSS)**: A hypothetical delivery mechanism used for checking the conformance of a *bitstream* or a *decoder* with regards to the timing and data flow of the input of a *bitstream* into the *hypothetical reference decoder*.

3.63 **instantaneous decoding refresh (IDR) picture**: An *IRAP picture* for which each *VCL NAL unit* has nal_unit_type equal to IDR_W_RADL or IDR_N_LP.

NOTE – An IDR picture does not use inter prediction in its decoding process, and could be the first picture in the bitstream in decoding order, or could appear later in the bitstream. Each IDR picture is the first picture of a CVS in decoding order. When an IDR picture for which each VCL NAL unit has nal_unit_type equal to IDR_W_RADL, it could have associated RADL pictures. When an IDR picture for which each VCL NAL unit has nal_unit_type equal to IDR_N_LP, it does not have any associated leading pictures. An IDR picture does not have associated RASL pictures.

3.64 **instantaneous decoding refresh (IDR) PU**: A *PU* in which the *coded picture* is an *IDR picture*.

3.65 **instantaneous decoding refresh (IDR) subpicture**: An *IRA*P subpi*cture* for which each *VCL NAL unit* has nal_unit_type equal to IDR_W_RADL or IDR_N_LP.

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

3.67 **inter prediction**: A *prediction* derived from blocks of sample values of one or more *reference pictures* as determined by motion vectors.

3.68 **inter-layer reference picture (ILRP)**: A *picture* in the same *AU* with the current *picture*, with nuh_layer_id less than the nuh_layer_id of the current *picture*, and is marked as "used for long-term reference".

3.69 **intra block copy (IBC) prediction**: A *prediction* derived from blocks of sample values of the same decoded *slice* as determined by block vectors.

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

3.71 **intra prediction**: A *prediction* derived from neighbouring sample values of the same decoded *slice*.

3.72 **intra random access point (IRAP) AU**: An *AU* in which there is a *PU* for each layer present in the CVS and the *coded picture* in each *PU* is an *IRAP picture*.

3.73 **intra random access point (IRAP) picture**: A *coded picture* for which all *VCL NAL units* have the same value of nal_unit_type in the range of IDR_W_RADL to CRA_NUT, inclusive.

NOTE 1 – An IRAP picture could be a CRA picture or an IDR picture. An IRAP picture does not use inter prediction from reference pictures in the same layer in its decoding process. The first picture in the bitstream in decoding order is an IRAP or GDR picture. For a single-layer bitstream, provided the necessary parameter sets are available when they need to be referenced, the IRAP picture and all subsequent non-RASL pictures in the CLVS in decoding order are correctly decodable without performing the decoding process of any pictures that precede the IRAP picture in decoding order.

NOTE 2 – The value of pps_mixed_nalu_types_in_pic_flag for an IRAP picture is equal to 0. When pps_mixed_nalu_types_in_pic_flag is equal to 0 for a picture, and any slice of the picture has nal_unit_type in the range of IDR_W_RADL to CRA_NUT, inclusive, all other slices of the picture have the same value of nal_unit_type, and the picture is known to be an IRAP picture after receiving the first slice.

3.74 **intra random access point (IRAP) PU**: A *PU* in which the *coded picture* is an *IRAP picture*.

3.75 **intra random access point (IRAP) subpicture**: A *subpicture* for which all *VCL NAL units* have the same value of nal_unit_type in the range of IDR_W_RADL to CRA_NUT, inclusive.

3.76 **intra (I) slice**: A *slice* that is decoded using *intra prediction* only.

3.77 **layer**: A set of *VCL NAL units* that all have a particular value of nuh_layer_id and the *associated non-VCL NAL units*.

3.78 **leading picture**: A *picture* that precedes the *associated IRAP picture* in *output order*.

NOTE – As constrained in clause 7.4.2.2, a leading picture associated with an IRAP picture is either a RADL picture or a RASL picture. Consequently, a non-leading picture associated with an IRAP picture, e.g., a trailing picture, follows the IRAP picture in output order.

3.79 **leading subpicture**: A *subpicture* that precedes the *associated IRAP subpicture* in *output order*.

3.80 **leaf**: A terminating node of a tree that is a root node of a tree of depth 0.

3.81 **level**: A defined set of constraints on the values that may be taken by the *syntax elements* and variables of this Specification, or the value of a *transform coefficient* prior to *scaling*.

NOTE – 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 could, within the specified constraints, support a different level for each supported profile.

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

3.83 **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*).

3.84 **LMCS APS**: An *APS* that controls the *LMCS* process.

3.85 **long-term reference picture (LTRP)**: A *picture* with nuh_layer_id equal to the nuh_layer_id of the current *picture* and marked as "used for long-term reference".

3.86 **luma**: A sample array or single sample representing the monochrome signal related to the primary colours, represented by the symbol or subscript Y or L.

NOTE – The term luma is used rather than the term luminance in order to avoid implying 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.

3.87 **luma mapping with chroma scaling (LMCS)**: A process that is applied as part of the *decoding process* that maps *luma* samples to particular values and in some cases also applies a scaling operation to the values of *chroma* samples.

3.88 **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*.

3.89 **multi-type tree**: A *tree* in which a parent node can be split either into two child nodes using a *binary split* or into three child nodes using a *ternary split*, each of which could become the parent node for another split into either two or three child nodes.

3.90 **network abstraction layer (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*.

3.91 **network abstraction layer (NAL) unit stream**: A sequence of *NAL units*.

3.92 **operation point (OP)**: A temporal subset of an OLS, identified by an OLS index and a highest value of TemporalId.

3.93 **output layer**: A layer of an output layer set that is output.

3.94 **output layer set (OLS)**: A set of layers for which one or more layers are specified as the output layers.

3.95 **output layer set (OLS) layer index**: An index, of a layer in an OLS, to the list of layers in the OLS.

3.96 **output order**: The order of pictures or subpictures within a CLVS indicated by increasing POC values, and for decoded pictures that are output from DPB, this is the order in which the *decoded pictures* are output from the *DPB*.

3.97 **output time**: A time when a *decoded picture* is to be output from the *DPB* (for the *decoded pictures* that are to be output from the *DPB*) as specified by the *HRD* according to the output timing *DPB* operation.

3.98 **palette**: A set of representative *component* values.

3.99 **palette prediction**: A *prediction* derived from one or more *palettes*.

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

3.101 **picture**: 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.

NOTE – A picture is either a frame or a field. However, in one CVS, either all pictures are frames or all pictures are fields.

3.102 **picture header (PH)**: A *syntax structure* containing *syntax elements* that apply to all *slices* of a coded picture.

3.103 **picture-level slice index**: An index, defined when pps_rect_slice_flag is equal to 1, of a slice to the list of slices in a picture in the order as the slices are signalled in the PPS when pps_single_slice_per_subpic_flag is equal to 0, or in the order of increasing subpicture indices of the subpicture corresponding to the slices when pps_single_slice_per_subpic_flag is equal to 1.

3.104 **picture order count (POC)**: A variable that is associated with each *picture*, uniquely identifies the associated *picture* among all *pictures* in the *CLVS*, and, when the associated *picture* is to be output from the *DPB*, indicates the position of the associated *picture* in *output order* relative to the *output order* positions of the other *pictures* in the same *CLVS* that are to be output from the *DPB*.

3.105 **picture parameter set (PPS)**: A *syntax structure* containing *syntax elements* that apply to zero or more entire *coded pictures* as determined by a *syntax element* found in each *picture header*.

3.106 **picture unit (PU)**: A set of *NAL units* that are associated with each other according to a specified classification rule, are consecutive in *decoding order*, and contain exactly one *coded picture*.

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

3.108 **prediction process**: The use of a *predictor* to provide an estimate of the data element (e.g., sample value or motion vector) currently being decoded.

3.109 **predictive (P) slice**: A *slice* that is decoded using *intra prediction* or using *inter prediction* with at most one *motion vector* and *reference index* to *predict* the sample values of each *block*.

3.110 **predictor**: A combination of specified values or previously decoded data elements (e.g., sample value or motion vector) used in the *decoding process* of subsequent data elements.

3.111 **profile**: A specified subset of the syntax of this Specification.

3.112 **quadtree**: A *tree* in which a parent node can be split into four child nodes, each of which could become the parent node for another split into four child nodes.

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

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

3.115 **random access decodable leading (RADL) picture**: A *coded picture* for which each *VCL NAL unit* has nal_unit_type equal to RADL_NUT.

NOTE – All RADL pictures are leading pictures. A RADL picture with nuh_layer_id equal to layerId is not used as a reference picture for the decoding process of any picture with nuh_layer_id equal to layerId that follows, in output order, the IRAP picture associated with the RADL picture. When sps_field_seq_flag is equal to 0, all RADL pictures, when present, precede, in decoding order, all non-leading pictures of the same associated IRAP picture.

3.116 **random access decodable leading (RADL) PU**: A *PU* in which the *coded picture* is a *RADL picture*.

3.117 **random access decodable leading (RADL) subpicture**: A *subpicture* for which each *VCL NAL unit* has nal_unit_type equal to RADL_NUT.

3.118 **random access skipped leading (RASL) picture**: A *coded picture* for which there is at least one *VCL NAL unit* with nal_unit_type equal to RASL_NUT and other VCL NAL units all have nal_unit_type equal to RASL_NUT or RADL_NUT.

NOTE – All RASL pictures are leading pictures of an associated CRA picture. When the associated CRA picture has NoOutputBeforeRecoveryFlag equal to 1, the RASL picture is not output and might not be correctly decodable, as the RASL picture could contain references to pictures that are not present in the bitstream. RASL pictures are not used as reference pictures for the decoding process of non-RASL pictures in the same layer, except that a RADL subpicture, when present, in a RASL picture in the same layer could be used for inter prediction of the collocated RADL subpicture in a RADL picture that is associated with the same CRA picture as the RASL picture. When sps_field_seq_flag is equal to 0, all RASL pictures, when present, precede, in decoding order, all non-leading pictures of the same associated CRA picture.

3.119 **random access skipped leading (RASL) PU**: A *PU* in which the *coded picture* is a *RASL picture*.

3.120 **random access skipped leading (RASL) subpicture**: A *subpicture* for which each *VCL NAL unit* has nal_unit_type equal to RASL_NUT.

3.121 **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.

3.122 **raw byte sequence payload (RBSP)**: A *syntax structure* containing an integer number of *bytes* that is encapsulated in a *NAL unit* and is either empty or has the form of a *string of data bits* containing *syntax elements* followed by an *RBSP stop bit* and zero or more subsequent bits equal to 0.

3.123 **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*, for which the location of the end 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*.

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

3.125 **reference picture**: A *picture* that is a *short-term reference picture*, a *long-term reference picture*, or an *inter-layer reference picture*.

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

3.126 **reference picture list (RPL)**: A list of *reference pictures* that is used for *inter prediction* of a *P* or *B slice*.

NOTE – Two RPLs, RPL 0 and RPL 1, are generated for each slice of a picture. The set of unique pictures referred to by all entries in the two RPLs associated with a picture consists of all reference pictures that could be used for inter prediction of the associated picture or any picture following the associated picture in decoding order. For the decoding process of a P slice, only RPL 0 is used for inter prediction. For the decoding process of a B slice, both RPL 0 and RPL 1 are used for inter prediction. For decoding the slice data of an I slice, no RPL is used for for inter prediction.

3.127 **reference picture list 0**: The *reference picture list* used for *inter prediction* of a *P slice* or the first of the two *reference picture lists* used for *inter prediction* of a *B slice*.

3.128 **reference picture list 1**: The second *reference picture list* used for *inter prediction* of a *B slice*.

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

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

3.131 **scaling list**: A list that associates each *frequency index* with a scale factor for the *scaling* process.

3.132 **scaling list APS**: An *APS* with syntax elements used to construct the *scaling lists*.

3.133 **sequence parameter set (SPS)**: A *syntax structure* containing *syntax elements* that apply to zero or more entire *CLVSs* as determined by the content of a *syntax element* found in the *PPS* referred to by a *syntax element* found in each *picture header.*

3.134 **short-term reference picture (STRP)**: A *picture* with nuh_layer_id equal to the nuh_layer_id of the current *picture* and marked as "used for short-term reference".

3.135 **slice**: An integer number of complete *tiles* or an integer number of consecutive complete *CTU* rows within a *tile* of a *picture* that are exclusively contained in a single *NAL unit*.

3.136 **slice header**: A part of a coded *slice* containing the data elements pertaining to all *tiles* or *CTU rows* within a *tile* represented in the *slice*.

3.137 **source**: A term used to describe the video material or some of its attributes before encoding.

3.138 **start code prefix**: A unique sequence of three *bytes* equal to 0x000001 embedded in the *byte stream* as a prefix to each *NAL unit*.

NOTE – 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.

3.139 **step-wise temporal sublayer access (STSA) picture**: A *coded picture* for which each *VCL NAL unit* has nal_unit_type equal to STSA_NUT.

NOTE – An STSA picture does not use pictures in the same layer and with the same TemporalId as the STSA picture for inter prediction reference. Pictures following an STSA picture in decoding order in the same layer and with the same TemporalId as the STSA picture do not use pictures prior to the STSA picture in decoding order in the same layer and with the same TemporalId as the STSA picture for inter prediction reference. An STSA picture enables up-switching, at the STSA picture, to the sublayer containing the STSA picture, from the immediately lower sublayer of the same layer when the coded picture does not belong to the lowest sublayer. STSA pictures in an independent layer always have TemporalId greater than 0.

3.140 **step-wise temporal sublayer access (STSA) PU**: A *PU* in which the *coded picture* is an *STSA picture*.

3.141 **step-wise temporal sublayer access (STSA) subpicture**: A *subpicture* for which each *VCL NAL unit* has nal_unit_type equal to STSA_NUT.

3.142 **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*, where 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.

3.143 **sub-bitstream extraction process**: A specified process by which *NAL units* in a *bitstream* that do not belong to a target set, determined by a target OLS index and a target highest TemporalId, are removed from the *bitstream*, with the output sub-bitstream consisting of the NAL units in the *bitstream* that belong to the target set.

3.144 **sublayer**: A temporal scalable layer of a temporal scalable *bitstream*, consisting of *VCL NAL units* with a particular value of the TemporalId variable and the associated *non-VCL NAL units*.

3.145 **sublayer representation**: A subset of the *bitstream* consisting of *NAL units* of a particular *sublayer* and the lower *sublayers*.

3.146 **subpicture:** A rectangular region of one or more *slices* within a *picture*.

3.147 **subpicture-level slice index**: An index, defined when pps_rect_slice_flag is equal to 1, of a slice to the list of slices in a subpicture in the order as they are signalled in the PPS.

3.148 **supplemental enhancement information (SEI) message**: A *syntax structure* with specified semantics that conveys a particular type of information that assists in processes related to decoding, display or other purposes but is not needed by the *decoding process* in order to determine the values of the samples in *decoded pictures*.

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

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

3.151 **ternary split**: A split of a rectangular MxN *block* of samples into three *blocks* where a vertical split results in a first (M / 4)xN *block*, a second (M / 2)xN *block*, a third (M / 4)xN *block*, and a horizontal split results in a first Mx(N / 4) *block*, a second Mx(N / 2) *block*, a third Mx(N / 4) *block*.

3.152 **tier**: A specified category of *level* constraints imposed on values of the *syntax elements* in the *bitstream*, where the *level* constraints are nested within a *tier* and a *decoder* conforming to a certain *tier* and *level* would be capable of decoding all *bitstreams* that conform to the same *tier* or the lower *tier* of that *level* or any *level* below it.

3.153 **tile**: A rectangular region of *CTUs* within a particular *tile column* and a particular *tile row* in a *picture*.

3.154 **tile column**: A rectangular region of *CTUs* having a height equal to the height of the *picture* and a width specified by *syntax elements* in the *picture parameter set*.

3.155 **tile row**: A rectangular region of *CTUs* having a height specified by *syntax elements* in the *picture parameter set* and a width equal to the width of the *picture*.

3.156 **tile scan**: A specific sequential ordering of *CTUs partitioning* a *picture* in which the *CTUs* are ordered consecutively in *CTU raster scan* in a *tile* whereas *tiles* in a *picture* are ordered consecutively in a *raster scan* of the *tiles* of the *picture*.

3.157 **trailing picture**: A *picture* for which each *VCL NAL unit* has nal_unit_type equal to TRAIL_NUT.

NOTE – Trailing pictures associated with an IRAP or GDR picture also follow the IRAP or GDR picture in decoding order. Pictures that follow the associated IRAP picture in output order and precede the associated IRAP picture in decoding order are not allowed.

3.158 **trailing subpicture**: A *subpicture* for which each *VCL NAL unit* has nal_unit_type equal to TRAIL_NUT.

NOTE – Trailing subpictures associated with an IRAP or GDR subpicture also follow the IRAP or GDR subpicture in decoding order. Subpictures that follow the associated IRAP subpicture in output order and precede the associated IRAP subpicture in decoding order are not allowed.

3.159 **transform**: A part of the *decoding process* by which a *block* of *transform coefficients* is converted to a *block* of spatial-domain values.

3.160 **transform block**: A rectangular MxN *block* of samples resulting from a *transform* in the *decoding process*.

3.161 **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 a *transform* in the *decoding process*.

3.162 **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.

3.163 **transform unit (TU)**: A *transform block* of *luma* samples and two corresponding *transform blocks* of *chroma* samples of a *picture* when using a single *coding unit tree* for *luma* and *chroma*; or, a *transform block* of *luma* samples or two *transform blocks* of *chroma* samples when using two separate *coding unit trees* for *luma* and *chroma*, and *syntax structures* used to transform the *transform block* samples.

3.164 **tree**: A tree is a finite set of nodes with a unique root node.

3.165 **video coding layer (VCL) NAL unit**: A collective term for *coded slice NAL units* and the subset of *NAL units* that have *reserved* values of nal_unit_type that are classified as VCL NAL units in this Specification.

4 Abbreviations

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

ACT	Adaptive Colour Transform
ALF	Adaptive Loop Filter
AMVR	Adaptive Motion Vector Resolution
APS	Adaptation Parameter Set
AU	Access Unit
AUD	Access Unit Delimiter
AVC	Advanced Video Coding (Rec. ITU-T H.264 | ISO/IEC 14496-10)
B	Bi-predictive
BCW	Bi-prediction with CU-level Weights
BDOF	Bi-Directional Optical Flow
BDPCM	Block-based Delta Pulse Code Modulation
BP	Buffering Period
CABAC	Context-based Adaptive Binary Arithmetic Coding
CB	Coding Block
CBR	Constant Bit Rate

CCALF	Cross-Component Adaptive Loop Filter
CPB	Coded Picture Buffer
CRA	Clean Random Access
CRC	Cyclic Redundancy Check
CREI	Constrained RASL Encoding Indication
CTB	Coding Tree Block
CTU	Coding Tree Unit
CU	Coding Unit
CVS	Coded Video Sequence
DPB	Decoded Picture Buffer
DCI	Decoding Capability Information
DRAP	Dependent Random Access Point
DU	Decoding Unit
DUI	Decoding Unit Information
EDRAP	Extended Dependent Random Access Point
EG	Exponential-Golomb
EGk	k-th order Exponential-Golomb
EOB	End Of Bitstream
EOS	End Of Sequence
FD	Filler Data
FIFO	First-In, First-Out
FL	Fixed-Length
GBR	Green, Blue, and Red
GCI	General Constraints Information
GDR	Gradual Decoding Refresh
GPM	Geometric Partitioning Mode
HEVC	High Efficiency Video Coding (Rec. ITU-T H.265 | ISO/IEC 23008-2)
HRD	Hypothetical Reference Decoder
HSS	Hypothetical Stream Scheduler
I	Intra
IBC	Intra Block Copy
IDR	Instantaneous Decoding Refresh
ILRP	Inter-Layer Reference Picture
IRAP	Intra Random Access Point
LFNST	Low Frequency Non-Separable Transform
LPS	Least Probable Symbol
LSB	Least Significant Bit
LTRP	Long-Term Reference Picture
LMCS	Luma Mapping with Chroma Scaling
MIP	Matrix-based Intra Prediction
MPS	Most Probable Symbol
MSB	Most Significant Bit
MTS	Multiple Transform Selection
MVP	Motion Vector Prediction
NAL	Network Abstraction Layer
OLS	Output Layer Set

OP	Operation Point
OPI	Operating Point Information
P	Predictive
PH	Picture Header
POC	Picture Order Count
PPS	Picture Parameter Set
PROF	Prediction Refinement with Optical Flow
PT	Picture Timing
PU	Picture Unit
QP	Quantization Parameter
RADL	Random Access Decodable Leading (picture)
RASL	Random Access Skipped Leading (picture)
RBSP	Raw Byte Sequence Payload
RGB	Red, Green, and Blue
RPL	Reference Picture List
SAO	Sample Adaptive Offset
SEI	Supplemental Enhancement Information
SH	Slice Header
SLI	Subpicture Level Information
SODB	String Of Data Bits
SPS	Sequence Parameter Set
STRP	Short-Term Reference Picture
STSA	Step-wise Temporal Sublayer Access
TR	Truncated Rice
VBR	Variable Bit Rate
VCL	Video Coding Layer
VPS	Video Parameter Set
VSEI	Versatile Supplemental Enhancement Information (Rec. ITU-T H.274 | ISO/IEC 23002-7)
VUI	Video Usability Information
VVC	Versatile Video Coding (Rec. ITU-T H.266 | ISO/IEC 23090-3)

5 Conventions

5.1 General

The term "this Specification" is used to refer to this Recommendation | International Standard.

The word "shall" is used to express mandatory requirements for conformance to this Specification. When used to express a mandatory constraint on the values of syntax elements or the values of variables derived from these syntax elements, it is the responsibility of the encoder to ensure that the constraint is fulfilled.

The word "may" is used to refer to behaviour that is allowed, but not necessarily required.

The word "should" is 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 Specification.

Content of this Specification that is identified as "informative" does not establish any mandatory requirements for conformance to this Specification and is thus not considered an integral part of this Specification. Informative remarks in the text are, in some cases, set apart and prefixed with the word "note" or "NOTE".

The word "reserved" is used to specify that some values of a particular syntax element are for future use by ITU-T | ISO/IEC and shall not be used in syntax structures conforming to this version of this Specification, but could potentially be used in syntax structures conforming to future versions of this Specification by ITU-T | ISO/IEC.

The word "unspecified" is used to describe some values of a particular syntax element to indicate that the values have no specified meaning in this Specification and are not expected to have a specified meaning in the future as an integral part of future versions of this Specification.

NOTE – The mathematical operators used in this Specification are similar to those used in the C programming language. However, the results of integer division and arithmetic shift operations are defined more precisely, and additional operations are defined, such as exponentiation and real-valued division. Numbering and counting conventions generally begin from 0, e.g., "the first" is equivalent to the 0-th, "the second" is equivalent to the 1-st.

5.2 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
	exponentiation
x^y	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.
÷	division in mathematical equations where no truncation or rounding is intended
$\frac{x}{y}$	division in mathematical equations where no truncation or rounding is intended
$\sum_{i=x}^{y} f(i)$	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

5.3 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, evaluates to the value of y; otherwise, evaluates to the value of z

When evaluating a logical expression, the value 0 is interpreted as FALSE and any numerical value not equal to 0 is interpreted as TRUE. The result of any logical expression that evaluates as FALSE is the value 0, and the result of any logical expression that evaluates as TRUE is the value 1.

5.4 Relational operators

>	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.

5.5 Bit-wise operators

&	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 non-negative integer values of y. Bits shifted into the most significant bits (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 non-negative integer values of y. Bits shifted into the least significant bits (LSBs) as a result of the left shift have a value equal to 0.

5.6 Assignment operators

= assignment operator

++ increment, i.e., *x*++ is equivalent to *x* = *x* + 1; when used in an array index, evaluates to the value of the variable prior to the increment operation

− − decrement, i.e., *x*− − is equivalent to *x* = *x* − 1; when used in an array index, evaluates to the value of the variable prior to the decrement operation

+= increment by amount specified, i.e., x += 3 is equivalent to x = x + 3, and x += (−3) is equivalent to x = x + (−3)

−= decrement by amount specified, i.e., x −= 3 is equivalent to x = x − 3, and x −= (−3) is equivalent to x = x − (−3)

5.7 Range notation

x = y..z x takes on integer values starting from y to z, inclusive, with x, y, and z being integer numbers and z being greater than or equal to y.

5.8 Mathematical functions

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

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

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

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

$$\text{ClipH}(\,v, w, x\,) = \begin{cases} x + v & ; \quad x < 0 \\ x - v & ; \quad x > w - 1 \\ x & ; \quad \text{otherwise} \end{cases} \tag{5}$$

Floor(x) largest integer less than or equal to x. (6)

Log2(x) base-2 logarithm of x. (7)

$$\text{Min}(x, y) = \begin{cases} x & ; \quad x <= y \\ y & ; \quad x > y \end{cases} \tag{8}$$

$$\text{Max}(x, y) = \begin{cases} x & ; \quad x >= y \\ y & ; \quad x < y \end{cases} \tag{9}$$

$$\text{Round}(x) = \text{Sign}(x) * \text{Floor}(\text{Abs}(x) + 0.5) \tag{10}$$

$$\text{Sign}(x) = \begin{cases} 1 & ; \quad x > 0 \\ 0 & ; \quad x == 0 \\ -1 & ; \quad x < 0 \end{cases} \tag{11}$$

$$\text{Sqrt}(x) \quad \text{square root of } x \tag{12}$$

$$\text{Swap}(x, y) = (y, x) \tag{13}$$

5.9 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 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 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", "x −= y"

5.10 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), and one descriptor 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 and semantics use the values of other variables derived from the values of syntax elements. 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 could, in some cases, 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 clause 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 could 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 clause 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 clause 5.8) 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).

A one-dimensional array is referred to as a list. A two-dimensional array is referred to as a matrix. Arrays can either be syntax elements or variables. 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 could be denoted either as s[x][y] or as s_{yx}. A single column of a matrix could be referred to as a list and denoted by omission of the row index. Thus, the column of a matrix s at horizontal position x could be referred to as the list s[x].

A specification of values of the entries in rows and columns of an array could be denoted by { {...} {...} }, where each inner pair of brackets specifies the values of the elements within a row in increasing column order and the rows are ordered in increasing row order. Thus, setting a matrix s equal to { { 1 6 } { 4 9 }} specifies that s[0][0] is set equal to 1, s[1][0] is set equal to 6, s[0][1] is set equal to 4, and s[1][1] is set equal to 9.

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", is used in some cases 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.

5.11 Text description of logical operations

In the text, a statement of logical operations as would be described mathematically in the following form:

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

is typically 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 mathematically in the following form:

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

is typically 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 one or more 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 mathematically in the following form:

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

is typically described in the following manner:

When condition 0, statement 0

When condition 1, statement 1

5.12 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 might also have a lower case variable explicitly specified as 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 coding block is sometimes referred to by the variable name having a value equal to the address of the specific coding block.

6 Bitstream and picture formats, partitionings, scanning processes and neighbouring relationships

6.1 Bitstream formats

This clause specifies the relationship between the network abstraction layer (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 Specification. The byte stream format is specified in Annex B.

6.2 Source, decoded and output picture formats

This clause specifies the relationship between source and decoded pictures that is given via the bitstream.

The video source that is represented by the bitstream is a sequence of pictures in decoding order.

The source and decoded pictures are each comprised of one or three sample arrays:

– Luma (Y) only (monochrome).
– Luma and two chroma (e.g., YCbCr or YCgCo).
– Green, blue, and red (GBR, also known as RGB).
– Arrays representing other unspecified monochrome or tri-stimulus colour samplings (for example, YZX, also known as XYZ).

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 VUI parameters as specified in Rec. ITU-T H.274 | ISO/IEC 23002-7.

NOTE 1 – The colour representation indications specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 are aligned with those specified in Rec. ITU-T H.273 | ISO/IEC 23091-2. Further information on the usage of these colour representation indications is available in ITU-T H-Suppl. 19 | ISO/IEC TR 23091-4.

The variables SubWidthC and SubHeightC are specified in Table 2, depending on the chroma format sampling structure, which is specified through sps_chroma_format_idc.

Table 2 – SubWidthC and SubHeightC values derived from sps_chroma_format_idc

sps_chroma_format_idc	Chroma format	SubWidthC	SubHeightC
0	Monochrome	1	1
1	4:2:0	2	2
2	4:2:2	2	1
3	4:4:4	1	1

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, each of the two chroma arrays has the same height and width as the luma array.

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 16, inclusive.

When the value of sps_chroma_format_idc is equal to 1, the nominal vertical and horizontal relative locations of luma and chroma samples in pictures are shown in Figure 1. Alternative chroma sample relative locations can be indicated in VUI parameters as specified in Rec. ITU-T H.274 | ISO/IEC 23002-7.

NOTE 2 – The chroma sample location indications specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 for use when sps_chroma_format_idc is equal to 1 are aligned with those specified in Rec. ITU-T H.273 | ISO/IEC 23091-2. Further information on the usage of these chroma sample location indications is available in ITU-T H-Suppl. 19 | ISO/IEC TR 23091-4.

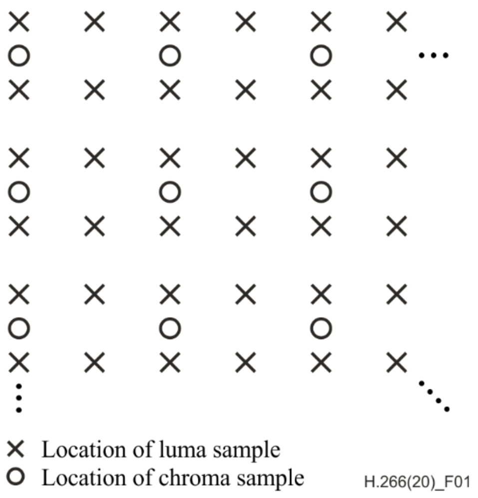

Figure 1 – Nominal vertical and horizontal locations of 4:2:0 luma and chroma samples in a picture

When the value of sps_chroma_format_idc is equal to 2, the chroma samples are co-sited with the corresponding luma samples and the nominal locations in a picture are as shown in Figure 2.

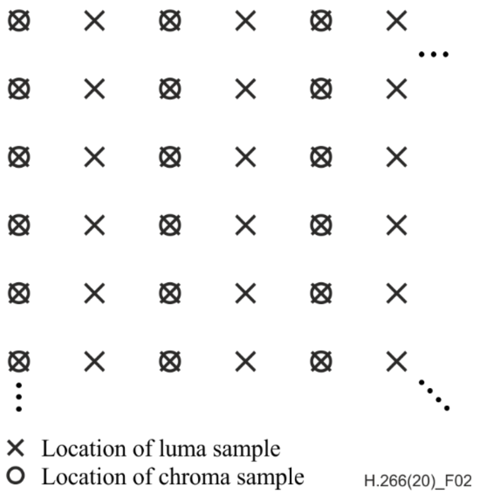

Figure 2 – Nominal vertical and horizontal locations of 4:2:2 luma and chroma samples in a picture

When the value of sps_chroma_format_idc is equal to 3, all array samples are co-sited for all cases of pictures and the nominal locations in a picture are as shown in Figure 3.

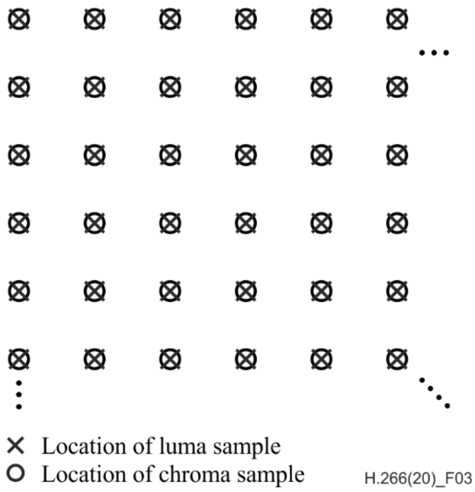

Figure 3 – Nominal vertical and horizontal locations of 4:4:4 luma and chroma samples in a picture

6.3 Partitioning of pictures, subpictures, slices, tiles, and CTUs

6.3.1 Partitioning of pictures into subpictures, slices, and tiles

This clause specifies how a picture is partitioned into subpictures, slices, and tiles.

A picture is divided into one or more tile rows and one or more tile columns. A tile is a sequence of CTUs that covers a rectangular region of a picture. The CTUs in a tile are scanned in raster scan order within that tile.

A slice consists of an integer number of complete tiles or an integer number of consecutive complete CTU rows within a tile of a picture. Consequently, each vertical slice boundary is always also a vertical tile boundary. It is possible that a horizontal boundary of a slice is not a tile boundary but consists of horizontal CTU boundaries within a tile; this occurs when a tile is split into multiple rectangular slices, each of which consists of an integer number of consecutive complete CTU rows within the tile.

Two modes of slices are supported, namely the raster-scan slice mode and the rectangular slice mode. In the raster-scan slice mode, a slice contains a sequence of complete tiles in a tile raster scan of a picture. In the rectangular slice mode, a slice contains either a number of complete tiles that collectively form a rectangular region of the picture or a number of consecutive complete CTU rows of one tile that collectively form a rectangular region of the picture. Tiles within a rectangular slice are scanned in tile raster scan order within the rectangular region corresponding to that slice.

A subpicture contains one or more slices that collectively cover a rectangular region of a picture. Consequently, each subpicture boundary is also always a slice boundary, and each vertical subpicture boundary is always also a vertical tile boundary.

One or both of the following conditions shall be fulfilled for each subpicture and tile:

– All CTUs in a subpicture belong to the same tile.

– All CTUs in a tile belong to the same subpicture.

Figure 4 shows an example of raster-scan slice partitioning of a picture, where the picture is divided into 12 tiles and 3 raster-scan slices.

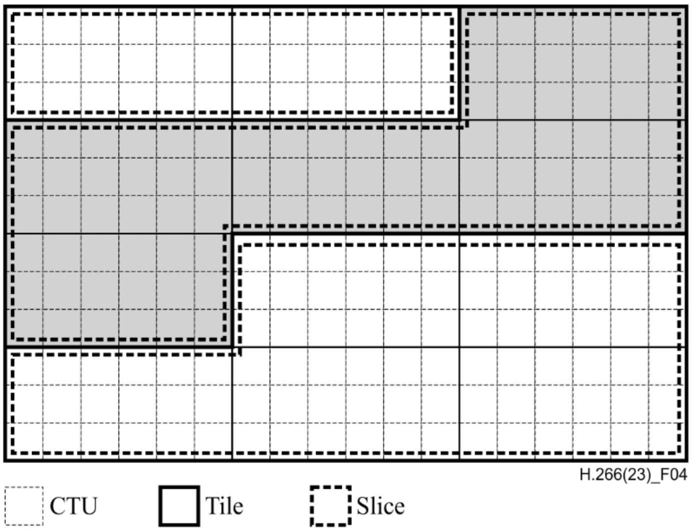

Figure 4 – A picture with 18 by 12 luma CTUs that is partitioned into 12 tiles and 3 raster-scan slices (informative)

Figure 5 shows an example of rectangular slice partitioning of a picture, where the picture is divided into 24 tiles (6 tile columns and 4 tile rows) and 9 rectangular slices.

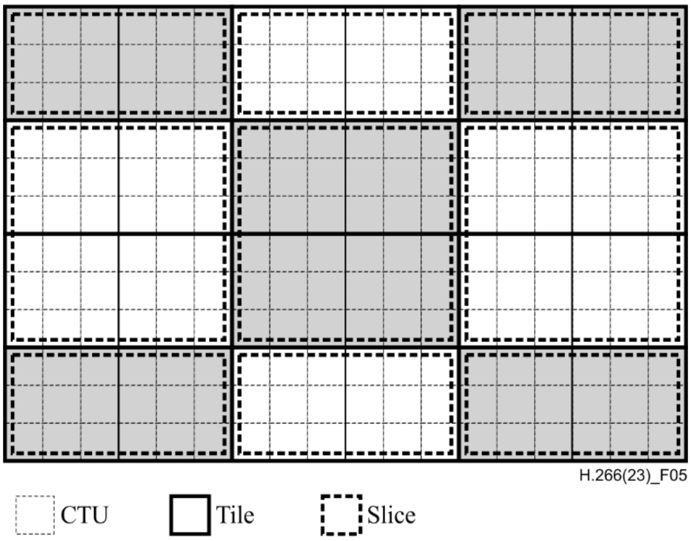

Figure 5 – A picture with 18 by 12 luma CTUs that is partitioned into 24 tiles and 9 rectangular slices (informative)

Figure 6 shows an example of a picture partitioned into tiles and rectangular slices, where the picture is divided into 4 tiles (2 tile columns and 2 tile rows) and 4 rectangular slices.

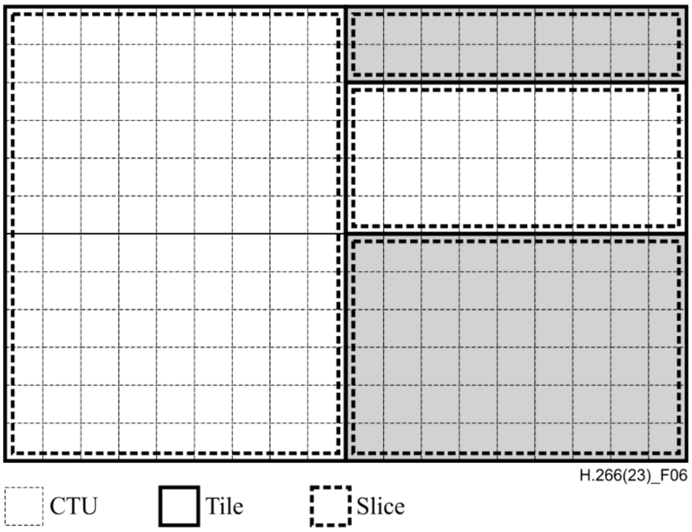

Figure 6 – A picture that is partitioned into 4 tiles and 4 rectangular slices (informative)

Figure 7 shows an example of subpicture partitioning of a picture, where a picture is partitioned into 18 tiles, 12 tiles on the left-hand side each covering one slice of 4 by 4 CTUs and 6 tiles on the right-hand side each covering 2 vertically-stacked slices of 2 by 2 CTUs, altogether resulting in 24 slices and 24 subpictures of varying dimensions (each slice is a subpicture).

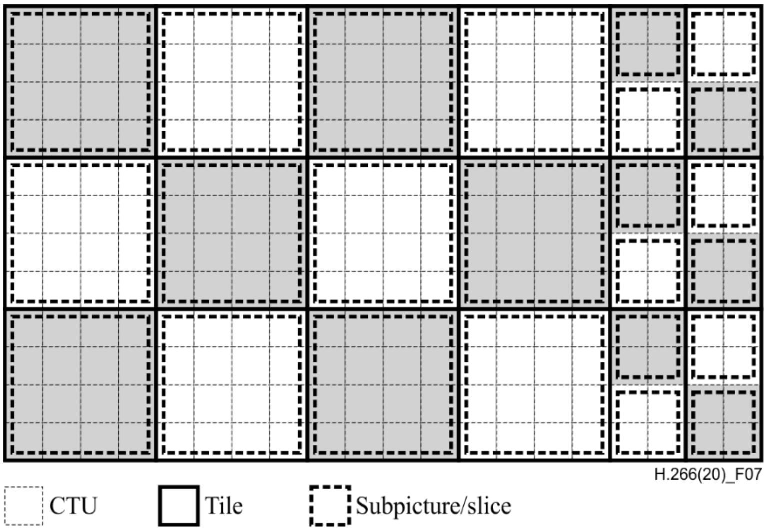

Figure 7 – A picture that is partitioned into 18 tiles, 24 slices and 24 subpictures (informative)

6.3.2 Block, quadtree and multi-type tree structures

The samples are processed in units of CTBs. The array size for each luma CTB in both width and height is CtbSizeY in units of samples. The width and height of the array for each chroma CTB are CtbWidthC and CtbHeightC, respectively, in units of samples. When the component width is not an integer multiple of the CTB size, the CTBs at the right component boundary are incomplete. When the component height is not an integer multiple of the CTB size, the CTBs at the bottom component boundary are incomplete.

Each CTB is assigned a partition signalling to identify the block sizes for intra or inter prediction and for transform coding. The partitioning is a recursive quadtree partitioning with a nested recursive multi-type tree partitioning. The root of the quadtree is associated with the CTB. The quadtree is split until a leaf is reached, which is referred to as the quadtree leaf. The root of the multi-type tree is associated with the quadtree leaf. The multi-type tree is split using horizontal or vertical binary splits or horizontal or vertical ternary splits until a leaf is reached, which is associated with the coding block.

The coding block is the root node of the transform tree. The transform tree specifies the position and size of transform blocks. The splitting information for luma and chroma might or might not be identical for the transform tree.

The blocks and associated syntax structures are grouped into "unit" structures as follows:

- One transform block (monochrome picture) or three transform blocks (luma and chroma components of a picture in 4:2:0, 4:2:2 or 4:4:4 colour format) and the associated transform syntax structures units are associated with a transform unit.
- One coding block (monochrome picture) or three coding blocks (luma and chroma), the associated coding syntax structures and the associated transform units are associated with a coding unit.
- One CTB (monochrome picture) or three CTBs (luma and chroma), the associated coding tree syntax structures and the associated coding units are associated with a CTU.

6.3.3 Spatial or component-wise partitionings

The following divisions of processing elements of this Specification form spatial or component-wise partitioning:

- division of each picture into components;
- division of each component into CTBs;
- division of each picture into subpictures;
- division of each picture into tile columns;
- division of each picture into tile rows;
- division of each tile column into tiles;
- division of each tile row into tiles;
- division of each tile into CTUs;
- division of each picture into slices;
- division of each subpicture into slices;
- division of each slice into CTUs;
- division of each CTU into CTBs;
- division of each CTB into coding blocks, except that the CTBs are incomplete at the right component boundary when the component width is not an integer multiple of the CTB size and the CTBs are incomplete at the bottom component boundary when the component height is not an integer multiple of the CTB size;
- division of each CTU into coding units, except that the CTUs are incomplete at the right picture boundary when the picture width in luma samples is not an integer multiple of the luma CTB size and the CTUs are incomplete at the bottom picture boundary when the picture height in luma samples is not an integer multiple of the luma CTB size;
- division of each coding unit into transform units;
- division of each coding unit into coding blocks;
- division of each coding block into transform blocks;
- division of each transform unit into transform blocks.

For each of these divisions of an entity A into entities B specified as being a partitioning, it is requirement of bitstream conformance that the union of the entities B resulted from the partitioning of the entity A shall cover exactly the entity A with no overlaps, no gaps, and no additions.

For example, corresponding to the division of each picture into subpictures being a partitioning, it is requirement of bitstream conformance that the union of the subpictures resulted from the partitioning of a picture shall cover exactly the picture, with no overlaps, no gaps, and no CTUs in the union that are outside the picture.

6.4 Availability processes

6.4.1 Allowed quad split process

Inputs to this process are:

- a coding block size cbSize in luma samples,
- a multi-type tree depth mttDepth,
- a variable treeType specifying whether a single tree (SINGLE_TREE) or a dual tree is used to partition the coding tree node and, when a dual tree is used, whether the luma (DUAL_TREE_LUMA) or chroma components (DUAL_TREE_CHROMA) are currently processed,
- a variable modeType specifying whether intra (MODE_INTRA), IBC (MODE_IBC), palette (MODE_PLT), and inter coding modes can be used (MODE_TYPE_ALL), or whether only intra, IBC and palette coding modes can be used (MODE_TYPE_INTRA), or whether only inter coding modes can be used (MODE_TYPE_INTER) for coding units inside the coding tree node.

Output of this process is the variable allowSplitQt.

The variable allowSplitQt is derived as follows:

- If one or more of the following conditions are true, allowSplitQt is set equal to FALSE:
  - treeType is equal to SINGLE_TREE or DUAL_TREE_LUMA and cbSize is less than or equal to MinQtSizeY;
  - treeType is equal to DUAL_TREE_CHROMA and cbSize is less than or equal to MinQtSizeC;
  - mttDepth is not equal to 0;
  - treeType is equal to DUAL_TREE_CHROMA and (cbSize / SubWidthC) is less than or equal to 4;
  - treeType is equal to DUAL_TREE_CHROMA and modeType is equal to MODE_TYPE_INTRA;
- Otherwise, allowSplitQt is set equal to TRUE.

6.4.2 Allowed binary split process

Inputs to this process are:

- a binary split mode btSplit,
- a coding block width cbWidth in luma samples,
- a coding block height cbHeight in luma samples,
- a location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture,
- a multi-type tree depth mttDepth,
- a maximum multi-type tree depth with offset maxMttDepth,
- a maximum binary tree size maxBtSize,
- a minimium quadtree size minQtSize,
- a partition index partIdx,
- a variable treeType specifying whether a single tree (SINGLE_TREE) or a dual tree is used to partition the coding tree node and, when a dual tree is used, whether the luma (DUAL_TREE_LUMA) or chroma components (DUAL_TREE_CHROMA) are currently processed,
- a variable modeType specifying whether intra (MODE_INTRA), IBC (MODE_IBC), palette (MODE_PLT), and inter coding modes can be used (MODE_TYPE_ALL), or whether only intra, IBC and palette coding modes can be used (MODE_TYPE_INTRA), or whether only inter coding modes can be used (MODE_TYPE_INTER) for coding units inside the coding tree node.

Output of this process is the variable allowBtSplit.

Table 3 – Specification of parallelTtSplit and cbSize based on btSplit

	btSplit = = SPLIT_BT_VER	**btSplit = = SPLIT_BT_HOR**
parallelTtSplit	SPLIT_TT_VER	SPLIT_TT_HOR
cbSize	cbWidth	cbHeight

The variables parallelTtSplit and cbSize are derived as specified in Table 3.

The variable allowBtSplit is derived as follows:

– If one or more of the following conditions are true, allowBtSplit is set equal to FALSE:
  – cbSize is less than or equal to MinBtSizeY;
  – cbWidth is greater than maxBtSize;
  – cbHeight is greater than maxBtSize;
  – mttDepth is greater than or equal to maxMttDepth;
  – treeType is equal to DUAL_TREE_CHROMA and (cbWidth / SubWidthC) * (cbHeight / SubHeightC) is less than or equal to 16;
  – treeType is equal to DUAL_TREE_CHROMA and (cbWidth / SubWidthC) is equal to 4 and btSplit is equal to SPLIT_BT_VER;
  – treeType is equal to DUAL_TREE_CHROMA and modeType is equal to MODE_TYPE_INTRA;
  – cbWidth * cbHeight is equal to 32 and modeType is equal to MODE_TYPE_INTER;
– Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE:
  – btSplit is equal to SPLIT_BT_VER;
  – y0 + cbHeight is greater than pps_pic_height_in_luma_samples;
– Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE:
  – btSplit is equal to SPLIT_BT_VER;
  – cbHeight is greater than 64;
  – x0 + cbWidth is greater than pps_pic_width_in_luma_samples;
– Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE:
  – btSplit is equal to SPLIT_BT_HOR;
  – cbWidth is greater than 64;
  – y0 + cbHeight is greater than pps_pic_height_in_luma_samples;
– Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE:
  – x0 + cbWidth is greater than pps_pic_width_in_luma_samples;
  – y0 + cbHeight is greater than pps_pic_height_in_luma_samples;
  – cbWidth is greater than minQtSize;
– Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE:
  – btSplit is equal to SPLIT_BT_HOR;
  – x0 + cbWidth is greater than pps_pic_width_in_luma_samples;
  – y0 + cbHeight is less than or equal to pps_pic_height_in_luma_samples;
– Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE:
  – mttDepth is greater than 0;
  – partIdx is equal to 1;

– MttSplitMode[x0][y0][mttDepth − 1] is equal to parallelTtSplit;

– Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE:

  – btSplit is equal to SPLIT_BT_VER;

  – cbWidth is less than or equal to 64;

  – cbHeight is greater than 64;

– Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE:

  – btSplit is equal to SPLIT_BT_HOR;

  – cbWidth is greater than 64;

  – cbHeight is less than or equal to 64;

– Otherwise, allowBtSplit is set equal to TRUE.

6.4.3 Allowed ternary split process

Inputs to this process are:

– a ternary split mode ttSplit,

– a coding block width cbWidth in luma samples,

– a coding block height cbHeight in luma samples,

– a location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture,

– a multi-type tree depth mttDepth,

– a maximum multi-type tree depth with offset maxMttDepth,

– a maximum ternary tree size maxTtSize,

– a variable treeType specifying whether a single tree (SINGLE_TREE) or a dual tree is used to partition the coding tree node and, when a dual tree is used, whether the luma (DUAL_TREE_LUMA) or chroma components (DUAL_TREE_CHROMA) are currently processed,

– a variable modeType specifying whether intra (MODE_INTRA), IBC (MODE_IBC), palette (MODE_PLT), and inter coding modes can be used (MODE_TYPE_ALL), or whether only intra, IBC and palette coding modes can be used (MODE_TYPE_INTRA), or whether only inter coding modes can be used (MODE_TYPE_INTER) for coding units inside the coding tree node.

Output of this process is the variable allowTtSplit.

Table 4 – Specification of cbSize based on ttSplit

	ttSplit = = SPLIT_TT_VER	**ttSplit = = SPLIT_TT_HOR**
cbSize	cbWidth	cbHeight

The variable cbSize is derived as specified in Table 4.

The variable allowTtSplit is derived as follows:

– If one or more of the following conditions are true, allowTtSplit is set equal to FALSE:

  – cbSize is less than or equal to 2 * MinTtSizeY;

  – cbWidth is greater than Min(64, maxTtSize);

  – cbHeight is greater than Min(64, maxTtSize);

  – mttDepth is greater than or equal to maxMttDepth;

  – x0 + cbWidth is greater than pps_pic_width_in_luma_samples;

  – y0 + cbHeight is greater than pps_pic_height_in_luma_samples;

- treeType is equal to DUAL_TREE_CHROMA and (cbWidth / SubWidthC) * (cbHeight / SubHeightC) is less than or equal to 32;
- treeType is equal to DUAL_TREE_CHROMA and (cbWidth / SubWidthC) is equal to 8 and ttSplit is equal to SPLIT_TT_VER;
- treeType is equal to DUAL_TREE_CHROMA and modeType is equal to MODE_TYPE_INTRA;
- cbWidth * cbHeight is equal to 64 and modeType is equal to MODE_TYPE_INTER;

– Otherwise, allowTtSplit is set equal to TRUE.

6.4.4 Derivation process for neighbouring block availability

Inputs to this process are:

– the luma location (xCurr, yCurr) of the top-left sample of the current block relative to the top-left luma sample of the current picture,

– the luma location (xNbY, yNbY) covered by a neighbouring block relative to the top-left luma sample of the current picture,

– the variable checkPredModeY specifying whether availability depends on the prediction mode,

– the variable cIdx specifying the colour component of the current block.

Output of this process is the availability of the neighbouring block covering the location (xNbY, yNbY), denoted as availableN.

The neighbouring block availability availableN is derived as follows:

– If one or more of the following conditions are true, availableN is set equal to FALSE:
  - xNbY is less than 0;
  - yNbY is less than 0;
  - xNbY is greater than or equal to pps_pic_width_in_luma_samples;
  - yNbY is greater than or equal to pps_pic_height_in_luma_samples;
  - (xNbY >> CtbLog2SizeY) is greater than (xCurr >> CtbLog2SizeY) and (yNbY >> CtbLog2SizeY) is greater than or equal to (yCurr >> CtbLog2SizeY);
  - (yNbY >> CtbLog2SizeY) is greater than or equal to (yCurr >> CtbLog2SizeY) + 1;
  - IsAvailable[cIdx][xNbY][yNbY] is equal to FALSE;
  - The neighbouring block is contained in a different slice than the current block;
  - The neighbouring block is contained in a different tile than the current block;
  - sps_entropy_coding_sync_enabled_flag is equal to 1 and (xNbY >> CtbLog2SizeY) is greater than or equal to (xCurr >> CtbLog2SizeY) + 1;

– Otherwise, availableN is set equal to TRUE.

When all of the following conditions are true, availableN is set equal to FALSE:

– checkPredModeY is equal to TRUE;

– CuPredMode[0][xNbY][yNbY] is not equal to CuPredMode[0][xCurr][yCurr].

6.5 Scanning processes

6.5.1 CTB raster scanning, tile scanning, and subpicture scanning processes

The variable NumTileColumns, specifying the number of tile columns, and the list ColWidthVal[i] for i ranging from 0 to NumTileColumns − 1, inclusive, specifying the width of the i-th tile column in units of CTBs, are derived as follows:

```
remainingWidthInCtbsY = PicWidthInCtbsY
for( i = 0; i <= pps_num_exp_tile_columns_minus1; i++ ) {
    ColWidthVal[ i ] = pps_tile_column_width_minus1[ i ] + 1
    remainingWidthInCtbsY −= ColWidthVal[ i ]
```

```
}
uniformTileColWidth = pps_tile_column_width_minus1[ pps_num_exp_tile_columns_minus1 ] + 1        (14)
while( remainingWidthInCtbsY  >=  uniformTileColWidth ) {
    ColWidthVal[ i++ ] = uniformTileColWidth
    remainingWidthInCtbsY  −=  uniformTileColWidth
}
if( remainingWidthInCtbsY > 0 )
    ColWidthVal[ i++ ] = remainingWidthInCtbsY
NumTileColumns = i
```

The variable NumTileRows, specifying the number of tile rows, and the list RowHeightVal[j] for j ranging from 0 to NumTileRows − 1, inclusive, specifying the height of the j-th tile row in units of CTBs, are derived as follows:

```
remainingHeightInCtbsY = PicHeightInCtbsY
for( j = 0; j  <=  pps_num_exp_tile_rows_minus1; j++ ) {
    RowHeightVal[ j ] = pps_tile_row_height_minus1[ j ] + 1
    remainingHeightInCtbsY  −=  RowHeightVal[ j ]
}
uniformTileRowHeight = pps_tile_row_height_minus1[ pps_num_exp_tile_rows_minus1 ] + 1        (15)
while( remainingHeightInCtbsY  >=  uniformTileRowHeight ) {
    RowHeightVal[ j++ ] = uniformTileRowHeight
    remainingHeightInCtbsY  −=  uniformTileRowHeight
}
if( remainingHeightInCtbsY > 0 )
    RowHeightVal[ j++ ] = remainingHeightInCtbsY
NumTileRows = j
```

The variable NumTilesInPic is set equal to NumTileColumns * NumTileRows.

The list TileColBdVal[i] for i ranging from 0 to NumTileColumns, inclusive, specifying the location of the i-th tile column boundary in units of CTBs, is derived as follows:

```
for( TileColBdVal[ 0 ] = 0, i = 0; i < NumTileColumns; i++ )
    TileColBdVal[ i + 1 ] = TileColBdVal[ i ] + ColWidthVal[ i ]        (16)
```

NOTE 1 – The size of the array TileColBdVal[ ] in this derivation is one greater than the actual number of tile columns.

The list TileRowBdVal[j] for j ranging from 0 to NumTileRows, inclusive, specifying the location of the j-th tile row boundary in units of CTBs, is derived as follows:

```
for( TileRowBdVal[ 0 ] = 0, j = 0; j < NumTileRows; j++ )
    TileRowBdVal[ j + 1 ] = TileRowBdVal[ j ] + RowHeightVal[ j ]        (17)
```

NOTE 2 – The size of the array TileRowBdVal[ ] in this derivation is one greater than the actual number of tile rows.

The lists CtbToTileColBd[ctbAddrX] and ctbToTileColIdx[ctbAddrX] for ctbAddrX ranging from 0 to PicWidthInCtbsY, inclusive, specifying the conversion from a horizontal CTB address to a left tile column boundary in units of CTBs and to a tile column index, respectively, are derived as follows:

```
tileX = 0
for( ctbAddrX = 0; ctbAddrX  <=  PicWidthInCtbsY; ctbAddrX++ ) {
    if( ctbAddrX  = =  TileColBdVal[ tileX + 1 ] )        (18)
        tileX++
    CtbToTileColBd[ ctbAddrX ] = TileColBdVal[ tileX ]
    ctbToTileColIdx[ ctbAddrX ] = tileX
}
```

NOTE 3 – The sizes of the arrays CtbToTileColBd[ ] and ctbToTileColIdx[ ] in this derivation are one greater than the actual picture width in CTBs.

The lists CtbToTileRowBd[ctbAddrY] and ctbToTileRowIdx[ctbAddrY] for ctbAddrY ranging from 0 to PicHeightInCtbsY, inclusive, specifying the conversion from a vertical CTB address to a top tile column boundary in units of CTBs and to a tile row index, respectively, are derived as follows:

```
tileY = 0
for( ctbAddrY = 0; ctbAddrY  <=  PicHeightInCtbsY; ctbAddrY++ ) {
```

```
        if( ctbAddrY  = =  TileRowBdVal[ tileY + 1 ] )                                                           (19)
            tileY++
        CtbToTileRowBd[ ctbAddrY ] = TileRowBdVal[ tileY ]
        ctbToTileRowIdx[ ctbAddrY ] = tileY
}
```

NOTE 4 – The sizes of the arrays CtbToTileRowBd[ ] and ctbToTileRowIdx[ ] in this derivation are one greater than the actual picture height in CTBs.

The lists SubpicWidthInTiles[i] and SubpicHeightInTiles[i], for i ranging from 0 to sps_num_subpics_minus1, inclusive, specifying the width and the height of the i-th subpicture in tile columns and rows, respectively, and the list subpicHeightLessThanOneTileFlag[i], for i ranging from 0 to sps_num_subpics_minus1, inclusive, specifying whether the height of the i-th subpicture is less than one tile row, are derived as follows:

```
for( i = 0; i  <=  sps_num_subpics_minus1; i++ ) {
    leftX = sps_subpic_ctu_top_left_x[ i ]
    rightX = leftX + sps_subpic_width_minus1[ i ]
    SubpicWidthInTiles[ i ] = ctbToTileColIdx[ rightX ] + 1 − ctbToTileColIdx[ leftX ]                   (20)
    topY = sps_subpic_ctu_top_left_y[ i ]
    bottomY = topY + sps_subpic_height_minus1[ i ]
    SubpicHeightInTiles[ i ] = ctbToTileRowIdx[ bottomY ] + 1 − ctbToTileRowIdx[ topY ]
    if( SubpicHeightInTiles[ i ]  = =  1  &&
            sps_subpic_height_minus1[ i ] + 1 < RowHeightVal[ ctbToTileRowIdx[ topY ] ] )
        subpicHeightLessThanOneTileFlag[ i ] = 1
    else
        subpicHeightLessThanOneTileFlag[ i ] = 0
}
```

NOTE 5 – When a tile is partitioned into multiple rectangular slices and only a subset of the rectangular slices of the tile is included in the i-th subpicture, the tile is counted as one tile in the value of SubpicHeightInTiles[i].

When pps_rect_slice_flag is equal to 1, the list NumCtusInSlice[i] for i ranging from 0 to pps_num_slices_in_pic_minus1, inclusive, specifying the number of CTUs in the i-th slice, the list SliceTopLeftTileIdx[i] for i ranging from 0 to pps_num_slices_in_pic_minus1, inclusive, specifying the tile index of the tile containing the first CTU in the slice, and the matrix CtbAddrInSlice[i][j] for i ranging from 0 to pps_num_slices_in_pic_minus1, inclusive, and j ranging from 0 to NumCtusInSlice[i] − 1, inclusive, specifying the picture raster scan address of the j-th CTB within the i-th slice, and the variable NumSlicesInTile[i], specifying the number of slices in the tile containing the i-th slice, are derived as follows:

```
if( pps_single_slice_per_subpic_flag ) {
     if( !sps_subpic_info_present_flag ) /* There is no subpicture info and only one slice in a picture. */
          for( j = 0; j < NumTileRows; j++ )
              for( i = 0; i < NumTileColumns; i++ )
                   AddCtbsToSlice( 0, TileColBdVal[ i ], TileColBdVal[ i + 1 ], TileRowBdVal[ j ],
                            TileRowBdVal[ j + 1 ] )
     else {
          for( i = 0; i  <=   sps_num_subpics_minus1; i++ ) {
              NumCtusInSlice[ i ] = 0
              if( subpicHeightLessThanOneTileFlag[ i ] ) /* The slice consists of a set of CTU rows in a tile. */
                   AddCtbsToSlice( i, sps_subpic_ctu_top_left_x[ i ],
                       sps_subpic_ctu_top_left_x[ i ] + sps_subpic_width_minus1[ i ] + 1,
                       sps_subpic_ctu_top_left_y[ i ],
                       sps_subpic_ctu_top_left_y[ i ] + sps_subpic_height_minus1[ i ] + 1 )
              else { /* The slice consists of a number of complete tiles covering a rectangular region. */
                   tileX = ctbToTileColIdx[ sps_subpic_ctu_top_left_x[ i ] ]
                   tileY = ctbToTileRowIdx[ sps_subpic_ctu_top_left_y[ i ] ]
                   for( j = 0; j < SubpicHeightInTiles[ i ]; j++ )
                       for( k = 0; k < SubpicWidthInTiles[ i ]; k++ )
                            AddCtbsToSlice( i, TileColBdVal[ tileX + k ], TileColBdVal[ tileX + k + 1 ],
                                     TileRowBdVal[ tileY + j ], TileRowBdVal[ tileY + j + 1 ] )
              }
          }
     }
} else {
```

```
    tileIdx = 0
    for( i = 0; i  <=  pps_num_slices_in_pic_minus1; i++ )
        NumCtusInSlice[ i ] = 0
    for( i = 0; i  <=  pps_num_slices_in_pic_minus1; i++ ) {
        SliceTopLeftTileIdx[ i ] = tileIdx
        tileX = tileIdx % NumTileColumns
        tileY = tileIdx / NumTileColumns
        if( i < pps_num_slices_in_pic_minus1 ) {
            sliceWidthInTiles[ i ] = pps_slice_width_in_tiles_minus1[ i ] + 1
            sliceHeightInTiles[ i ] = pps_slice_height_in_tiles_minus1[ i ] + 1
        } else {
            sliceWidthInTiles[ i ] = NumTileColumns − tileX
            sliceHeightInTiles[ i ] = NumTileRows − tileY
            NumSlicesInTile[ i ] = 1
        }
        if( sliceWidthInTiles[ i ]  = =  1  &&  sliceHeightInTiles[ i ]  = =  1 ) {                    (21)
            if( pps_num_exp_slices_in_tile[ i ]  = = 0 ) {
                NumSlicesInTile[ i ] = 1
                sliceHeightInCtus[ i ] = RowHeightVal[ SliceTopLeftTileIdx[ i ] / NumTileColumns ]
            } else {
                remainingHeightInCtbsY = RowHeightVal[ SliceTopLeftTileIdx[ i ] / NumTileColumns ]
                for( j = 0; j < pps_num_exp_slices_in_tile[ i ]; j++ ) {
                    sliceHeightInCtus[ i + j ] = pps_exp_slice_height_in_ctus_minus1[ i ][ j ] + 1
                    remainingHeightInCtbsY  −=  sliceHeightInCtus[ i + j ]
                }
                uniformSliceHeight = sliceHeightInCtus[ i + j − 1 ]
                while( remainingHeightInCtbsY  >=  uniformSliceHeight ) {
                    sliceHeightInCtus[ i + j ] = uniformSliceHeight
                    remainingHeightInCtbsY  −=  uniformSliceHeight
                    j++
                }
                if( remainingHeightInCtbsY > 0 ) {
                    sliceHeightInCtus[ i + j ] = remainingHeightInCtbsY
                    j++
                }
                NumSlicesInTile[ i ] = j
            }
            ctbY = TileRowBdVal[ tileY ]
            for( j = 0; j < NumSlicesInTile[ i ]; j++ ) {
                AddCtbsToSlice( i + j, TileColBdVal[ tileX ], TileColBdVal[ tileX + 1 ],
                        ctbY, ctbY + sliceHeightInCtus[ i + j ] )
                ctbY  +=  sliceHeightInCtus[ i + j ]
                sliceWidthInTiles[ i + j ] = 1
                sliceHeightInTiles[ i + j ] = 1
            }
            i  +=  NumSlicesInTile[ i ] − 1
        } else
            for( j = 0; j < sliceHeightInTiles[ i ]; j++ )
                for( k = 0; k < sliceWidthInTiles[ i ]; k++ )
                    AddCtbsToSlice( i, TileColBdVal[ tileX + k ], TileColBdVal[ tileX + k + 1 ],
                        TileRowBdVal[ tileY + j ], TileRowBdVal[ tileY + j + 1 ] )
        if( i < pps_num_slices_in_pic_minus1 ) {
            if( pps_tile_idx_delta_present_flag )
                tileIdx  +=  pps_tile_idx_delta_val[ i ]
            else {
                tileIdx  +=  sliceWidthInTiles[ i ]
                if( tileIdx % NumTileColumns  = =  0 )
                    tileIdx  +=  ( sliceHeightInTiles[ i ] − 1 ) * NumTileColumns
            }
        }
    }
}
```

Where the function AddCtbsToSlice(sliceIdx, startX, stopX, startY, stopY) is specified as follows:

```
for( ctbY = startY; ctbY < stopY; ctbY++ )
    for( ctbX = startX; ctbX < stopX; ctbX++ ) {
        CtbAddrInSlice[ sliceIdx ][ NumCtusInSlice[ sliceIdx ] ] = ctbY * PicWidthInCtbsY + ctbX          (22)
        NumCtusInSlice[ sliceIdx ]++
    }
```

It is a requirement of bitstream conformance that the values of NumCtusInSlice[i] for i ranging from 0 to pps_num_slices_in_pic_minus1, inclusive, shall be greater than 0. Additionally, it is a requirement of bitstream conformance that the matrix CtbAddrInSlice[i][j] for i ranging from 0 to pps_num_slices_in_pic_minus1, inclusive, and j ranging from 0 to NumCtusInSlice[i] − 1, inclusive, shall include each of all CTB addresses in the range of 0 to PicSizeInCtbsY − 1, inclusive, once and only once.

The lists NumSlicesInSubpic[i], SubpicLevelSliceIdx[j], and SubpicIdxForSlice[j], specifying the number of slices in the i-th subpicture, the subpicture-level slice index of the slice with picture-level slice index j, and the subpicture index of the slice with picture-level slice index j, respectively, are derived as follows:

```
for( i = 0; i  <=  sps_num_subpics_minus1; i++ ) {
    NumSlicesInSubpic[ i ] = 0
    for( j = 0; j  <=  pps_num_slices_in_pic_minus1; j++ ) {
        posX = CtbAddrInSlice[ j ][ 0 ] % PicWidthInCtbsY
        posY = CtbAddrInSlice[ j ][ 0 ] / PicWidthInCtbsY
        if( ( posX  >=  sps_subpic_ctu_top_left_x[ i ] )  &&                                        (23)
                ( posX < sps_subpic_ctu_top_left_x[ i ] + sps_subpic_width_minus1[ i ] + 1 )  &&
                ( posY  >=  sps_subpic_ctu_top_left_y[ i ] )  &&
                ( posY < sps_subpic_ctu_top_left_y[ i ] + sps_subpic_height_minus1[ i ] + 1 ) ) {
            SubpicIdxForSlice[ j ] = i
            SubpicLevelSliceIdx[ j ] = NumSlicesInSubpic[ i ]
            NumSlicesInSubpic[ i ]++
        }
    }
}
```

6.5.2 Up-right diagonal scan order array initialization process

Input to this process is a block width blkWidth and a block height blkHeight.

Output of this process is the array diagScan[sPos][sComp]. The array index sPos specify the scan position ranging from 0 to (blkWidth * blkHeight) − 1. The array index sComp equal to 0 specifies the horizontal component and the array index sComp equal to 1 specifies the vertical component. Depending on the value of blkWidth and blkHeight, the array diagScan is derived as follows:

```
i = 0
x = 0
y = 0
stopLoop = FALSE
while( !stopLoop ) {
    while( y  >=  0 ) {
        if( x < blkWidth  &&  y < blkHeight ) {                                                      (24)
            diagScan[ i ][ 0 ] = x
            diagScan[ i ][ 1 ] = y
            i++
        }
        y− −
        x++
    }
    y = x
    x = 0
    if( i  >=  blkWidth * blkHeight )
        stopLoop = TRUE
}
```

6.5.3 Horizontal and vertical traverse scan order array initialization process

Input to this process is a block width blkWidth and a block height blkHeight.

Outputs of this process are the arrays hTravScan[sPos][sComp] and vTravScan[sPos][sComp]. The array hTravScan represents the horizontal traverse scan order and the array vTravScan represents the vertical traverse scan order. The array index sPos specifies the scan position ranging from 0 to (blkWidth * blkHeight) − 1, inclusive. The array index sComp equal to 0 specifies the horizontal component and the array index sComp equal to 1 specifies the vertical component. Depending on the value of blkWidth and blkHeight, the array hTravScan anfd vTravScan are derived as follows:

```
i = 0
for( y = 0; y  < blkHeight; y++ )
    if( y % 2  = =  0 )
        for( x = 0; x < blkWidth; x++ ) {
            hTravScan[ i ][ 0 ] = x
            hTravScan[ i ][ 1 ] = y
            i++
        }
    else                                                          (25)
        for( x = blkWidth − 1; x  >=  0; x− − ) {
            hTravScan[ i ][ 0 ] = x
            hTravScan[ i ][ 1 ] = y
            i++
        }
```

```
i = 0
for( x = 0; x  < blkWidth; x++ )
    if( x % 2  = =  0 )
        for( y = 0; y < blkHeight; y++ ) {
            vTravScan[ i ][ 0 ] = x
            vTravScan[ i ][ 1 ] = y
            i++
        }
    else                                                          (26)
        for( y = blkHeight − 1; y  >=  0; y− − ) {
            vTravScan[ i ][ 0 ] = x
            vTravScan[ i ][ 1 ] = y
            i++
        }
```

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 might be specified, either directly or indirectly, in other clauses.

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

The following table lists examples of the syntax specification format. 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.

	Descriptor
/* A statement can be a syntax element with an associated 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	ue(k)
conditioning statement	
/* A group of statements enclosed in curly brackets is a compound statement and is treated functionally as a single statement. */	

	Descriptor
{	
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 and descriptors

The functions presented in this clause are used in the specification of the syntax. These functions are expressed in terms of the value of a bitstream pointer that indicates 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 a byte-aligned position, i.e., the current position is an integer multiple of 8 bits from the position of the first bit in the bitstream, 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_data_in_payload( ) is specified as follows:

– If byte_aligned( ) is equal to TRUE and the current position in the sei_payload( ) or vui_payload( ) syntax structure is 8 * payloadSize bits from the beginning of the syntax structure, the return value of more_data_in_payload( ) is equal to FALSE.

– Otherwise, the return value of more_data_in_payload( ) is equal to TRUE.

more_rbsp_data( ) is specified as follows:

– If there is no more data in the raw byte sequence payload (RBSP), the return value of more_rbsp_data( ) is equal to FALSE.

– Otherwise, the RBSP data are 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 format as specified in Annex B and fewer than n bits remain within the byte stream, next_bits(n) returns a value of 0.

payload_extension_present( ) is specified as follows:

– If the current position in the sei_payload( ) or vui_payload( ) syntax structure is not the position of the last (least significant, right-most) bit that is equal to 1 that is less than 8 * payloadSize bits from the beginning of the syntax structure (i.e., the position of the sei_payload_bit_equal_to_one or vui_payload_bit_equal_to_one syntax element), the return value of payload_extension_present( ) is equal to TRUE.

– Otherwise, the return value of payload_extension_present( ) is equal to FALSE.

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.

The following descriptors specify the parsing process of each syntax element:

– ae(v): context-adaptive arithmetic entropy-coded syntax element. The parsing process for this descriptor is specified in clause 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).

– 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.

– se(v): signed integer 0-th order Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in clause 9.2 with the order k equal to 0.

– 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 0-th order Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in clause 9.2 with the order k equal to 0.

7.3 Syntax in tabular form

7.3.1 NAL unit syntax

7.3.1.1 General NAL unit syntax

nal_unit(NumBytesInNalUnit) {	**Descriptor**
nal_unit_header( )	
NumBytesInRbsp = 0	
for(i = 2; i < NumBytesInNalUnit; i++)	
if(i + 2 < NumBytesInNalUnit && next_bits(24) = = 0x000003) {	
rbsp_byte[NumBytesInRbsp++]	b(8)
rbsp_byte[NumBytesInRbsp++]	b(8)
i += 2	
emulation_prevention_three_byte /* equal to 0x03 */	f(8)
} else	
rbsp_byte[NumBytesInRbsp++]	b(8)
}	

7.3.1.2 NAL unit header syntax

nal_unit_header( ) {	**Descriptor**
forbidden_zero_bit	f(1)
nuh_reserved_zero_bit	u(1)
nuh_layer_id	u(6)
nal_unit_type	u(5)
nuh_temporal_id_plus1	u(3)
}	

7.3.2 Raw byte sequence payloads, trailing bits and byte alignment syntax

7.3.2.1 Decoding capability information RBSP syntax

decoding_capability_information_rbsp( ) {	**Descriptor**
dci_reserved_zero_4bits	u(4)
dci_num_ptls_minus1	u(4)
for(i = 0; i <= dci_num_ptls_minus1; i++)	
profile_tier_level(1, 0)	
dci_extension_flag	u(1)
if(dci_extension_flag)	
while(more_rbsp_data( ))	
dci_extension_data_flag	u(1)
rbsp_trailing_bits( )	
}	

7.3.2.2 Operating point information RBSP syntax

operating_point_information_rbsp( ) {	**Descriptor**
opi_ols_info_present_flag	u(1)
opi_htid_info_present_flag	u(1)
if(opi_ols_info_present_flag)	
opi_ols_idx	ue(v)
if(opi_htid_info_present_flag)	
opi_htid_plus1	u(3)
opi_extension_flag	u(1)
if(opi_extension_flag)	
while(more_rbsp_data( ))	
opi_extension_data_flag	u(1)
rbsp_trailing_bits( )	
}	

7.3.2.3 Video parameter set RBSP syntax

video_parameter_set_rbsp( ) {	**Descriptor**
vps_video_parameter_set_id	u(4)
vps_max_layers_minus1	u(6)
vps_max_sublayers_minus1	u(3)
if(vps_max_layers_minus1 > 0 && vps_max_sublayers_minus1 > 0)	
vps_default_ptl_dpb_hrd_max_tid_flag	u(1)
if(vps_max_layers_minus1 > 0)	
vps_all_independent_layers_flag	u(1)
for(i = 0; i <= vps_max_layers_minus1; i++) {	
vps_layer_id[i]	u(6)
if(i > 0 && !vps_all_independent_layers_flag) {	
vps_independent_layer_flag[i]	u(1)
if(!vps_independent_layer_flag[i]) {	
vps_max_tid_ref_present_flag[i]	u(1)
for(j = 0; j < i; j++) {	
vps_direct_ref_layer_flag[i][j]	u(1)
if(vps_max_tid_ref_present_flag[i] && vps_direct_ref_layer_flag[i][j])	
vps_max_tid_il_ref_pics_plus1[i][j]	u(3)
}	
}	
}	
}	
if(vps_max_layers_minus1 > 0) {	
if(vps_all_independent_layers_flag)	
vps_each_layer_is_an_ols_flag	u(1)
if(!vps_each_layer_is_an_ols_flag) {	
if(!vps_all_independent_layers_flag)	
vps_ols_mode_idc	u(2)
if(vps_ols_mode_idc = = 2) {	

vps_num_output_layer_sets_minus2	u(8)
for(i = 1; i <= vps_num_output_layer_sets_minus2 + 1; i ++)	
for(j = 0; j <= vps_max_layers_minus1; j++)	
vps_ols_output_layer_flag[i][j]	u(1)
}	
}	
vps_num_ptls_minus1	u(8)
}	
for(i = 0; i <= vps_num_ptls_minus1; i++) {	
if(i > 0)	
vps_pt_present_flag[i]	u(1)
if(!vps_default_ptl_dpb_hrd_max_tid_flag)	
vps_ptl_max_tid[i]	u(3)
}	
while(!byte_aligned( ))	
vps_ptl_alignment_zero_bit /* equal to 0 */	f(1)
for(i = 0; i <= vps_num_ptls_minus1; i++)	
profile_tier_level(vps_pt_present_flag[i], vps_ptl_max_tid[i])	
for(i = 0; i < TotalNumOlss; i++)	
if(vps_num_ptls_minus1 > 0 && vps_num_ptls_minus1 + 1 != TotalNumOlss)	
vps_ols_ptl_idx[i]	u(8)
if(!vps_each_layer_is_an_ols_flag) {	
vps_num_dpb_params_minus1	ue(v)
if(vps_max_sublayers_minus1 > 0)	
vps_sublayer_dpb_params_present_flag	u(1)
for(i = 0; i < VpsNumDpbParams; i++) {	
if(!vps_default_ptl_dpb_hrd_max_tid_flag)	
vps_dpb_max_tid[i]	u(3)
dpb_parameters(vps_dpb_max_tid[i], vps_sublayer_dpb_params_present_flag)	
}	
for(i = 0; i < NumMultiLayerOlss; i++) {	
vps_ols_dpb_pic_width[i]	ue(v)
vps_ols_dpb_pic_height[i]	ue(v)
vps_ols_dpb_chroma_format[i]	u(2)
vps_ols_dpb_bitdepth_minus8[i]	ue(v)
if(VpsNumDpbParams > 1 && VpsNumDpbParams != NumMultiLayerOlss)	
vps_ols_dpb_params_idx[i]	ue(v)
}	
vps_timing_hrd_params_present_flag	u(1)
if(vps_timing_hrd_params_present_flag) {	
general_timing_hrd_parameters( )	
if(vps_max_sublayers_minus1 > 0)	
vps_sublayer_cpb_params_present_flag	u(1)
vps_num_ols_timing_hrd_params_minus1	ue(v)
for(i = 0; i <= vps_num_ols_timing_hrd_params_minus1; i++) {	
if(!vps_default_ptl_dpb_hrd_max_tid_flag)	
vps_hrd_max_tid[i]	u(3)

firstSubLayer = vps_sublayer_cpb_params_present_flag ? 0 : vps_hrd_max_tid[i]	
ols_timing_hrd_parameters(firstSubLayer, vps_hrd_max_tid[i])	
}	
if(vps_num_ols_timing_hrd_params_minus1 > 0 && vps_num_ols_timing_hrd_params_minus1 + 1 != NumMultiLayerOlss)	
for(i = 0; i < NumMultiLayerOlss; i++)	
vps_ols_timing_hrd_idx[i]	ue(v)
}	
}	
vps_extension_flag	u(1)
if(vps_extension_flag)	
while(more_rbsp_data( ))	
vps_extension_data_flag	u(1)
rbsp_trailing_bits( )	
}	

7.3.2.4 Sequence parameter set RBSP syntax

seq_parameter_set_rbsp( ) {	**Descriptor**
sps_seq_parameter_set_id	u(4)
sps_video_parameter_set_id	u(4)
sps_max_sublayers_minus1	u(3)
sps_chroma_format_idc	u(2)
sps_log2_ctu_size_minus5	u(2)
sps_ptl_dpb_hrd_params_present_flag	u(1)
if(sps_ptl_dpb_hrd_params_present_flag)	
profile_tier_level(1, sps_max_sublayers_minus1)	
sps_gdr_enabled_flag	u(1)
sps_ref_pic_resampling_enabled_flag	u(1)
if(sps_ref_pic_resampling_enabled_flag)	
sps_res_change_in_clvs_allowed_flag	u(1)
sps_pic_width_max_in_luma_samples	ue(v)
sps_pic_height_max_in_luma_samples	ue(v)
sps_conformance_window_flag	u(1)
if(sps_conformance_window_flag) {	
sps_conf_win_left_offset	ue(v)
sps_conf_win_right_offset	ue(v)
sps_conf_win_top_offset	ue(v)
sps_conf_win_bottom_offset	ue(v)
}	
sps_subpic_info_present_flag	u(1)
if(sps_subpic_info_present_flag) {	
sps_num_subpics_minus1	ue(v)
if(sps_num_subpics_minus1 > 0) {	
sps_independent_subpics_flag	u(1)
sps_subpic_same_size_flag	u(1)
}	

for(i = 0; sps_num_subpics_minus1 > 0 && i <= sps_num_subpics_minus1; i++) {	
if(!sps_subpic_same_size_flag || i = = 0) {	
if(i > 0 && sps_pic_width_max_in_luma_samples > CtbSizeY)	
sps_subpic_ctu_top_left_x[i]	u(v)
if(i > 0 && sps_pic_height_max_in_luma_samples > CtbSizeY)	
sps_subpic_ctu_top_left_y[i]	u(v)
if(i < sps_num_subpics_minus1 && sps_pic_width_max_in_luma_samples > CtbSizeY)	
sps_subpic_width_minus1[i]	u(v)
if(i < sps_num_subpics_minus1 && sps_pic_height_max_in_luma_samples > CtbSizeY)	
sps_subpic_height_minus1[i]	u(v)
}	
if(!sps_independent_subpics_flag) {	
sps_subpic_treated_as_pic_flag[i]	u(1)
sps_loop_filter_across_subpic_enabled_flag[i]	u(1)
}	
}	
sps_subpic_id_len_minus1	ue(v)
sps_subpic_id_mapping_explicitly_signalled_flag	u(1)
if(sps_subpic_id_mapping_explicitly_signalled_flag) {	
sps_subpic_id_mapping_present_flag	u(1)
if(sps_subpic_id_mapping_present_flag)	
for(i = 0; i <= sps_num_subpics_minus1; i++)	
sps_subpic_id[i]	u(v)
}	
}	
sps_bitdepth_minus8	ue(v)
sps_entropy_coding_sync_enabled_flag	u(1)
sps_entry_point_offsets_present_flag	u(1)
sps_log2_max_pic_order_cnt_lsb_minus4	u(4)
sps_poc_msb_cycle_flag	u(1)
if(sps_poc_msb_cycle_flag)	
sps_poc_msb_cycle_len_minus1	ue(v)
sps_num_extra_ph_bytes	u(2)
for(i = 0; i < (sps_num_extra_ph_bytes * 8); i++)	
sps_extra_ph_bit_present_flag[i]	u(1)
sps_num_extra_sh_bytes	u(2)
for(i = 0; i < (sps_num_extra_sh_bytes * 8); i++)	
sps_extra_sh_bit_present_flag[i]	u(1)
if(sps_ptl_dpb_hrd_params_present_flag) {	
if(sps_max_sublayers_minus1 > 0)	
sps_sublayer_dpb_params_flag	u(1)
dpb_parameters(sps_max_sublayers_minus1, sps_sublayer_dpb_params_flag)	
}	
sps_log2_min_luma_coding_block_size_minus2	ue(v)
sps_partition_constraints_override_enabled_flag	u(1)
sps_log2_diff_min_qt_min_cb_intra_slice_luma	ue(v)
sps_max_mtt_hierarchy_depth_intra_slice_luma	ue(v)

if(sps_max_mtt_hierarchy_depth_intra_slice_luma != 0) {	
sps_log2_diff_max_bt_min_qt_intra_slice_luma	ue(v)
sps_log2_diff_max_tt_min_qt_intra_slice_luma	ue(v)
}	
if(sps_chroma_format_idc != 0)	
sps_qtbtt_dual_tree_intra_flag	u(1)
if(sps_qtbtt_dual_tree_intra_flag) {	
sps_log2_diff_min_qt_min_cb_intra_slice_chroma	ue(v)
sps_max_mtt_hierarchy_depth_intra_slice_chroma	ue(v)
if(sps_max_mtt_hierarchy_depth_intra_slice_chroma != 0) {	
sps_log2_diff_max_bt_min_qt_intra_slice_chroma	ue(v)
sps_log2_diff_max_tt_min_qt_intra_slice_chroma	ue(v)
}	
}	
sps_log2_diff_min_qt_min_cb_inter_slice	ue(v)
sps_max_mtt_hierarchy_depth_inter_slice	ue(v)
if(sps_max_mtt_hierarchy_depth_inter_slice != 0) {	
sps_log2_diff_max_bt_min_qt_inter_slice	ue(v)
sps_log2_diff_max_tt_min_qt_inter_slice	ue(v)
}	
if(CtbSizeY > 32)	
sps_max_luma_transform_size_64_flag	u(1)
sps_transform_skip_enabled_flag	u(1)
if(sps_transform_skip_enabled_flag) {	
sps_log2_transform_skip_max_size_minus2	ue(v)
sps_bdpcm_enabled_flag	u(1)
}	
sps_mts_enabled_flag	u(1)
if(sps_mts_enabled_flag) {	
sps_explicit_mts_intra_enabled_flag	u(1)
sps_explicit_mts_inter_enabled_flag	u(1)
}	
sps_lfnst_enabled_flag	u(1)
if(sps_chroma_format_idc != 0) {	
sps_joint_cbcr_enabled_flag	u(1)
sps_same_qp_table_for_chroma_flag	u(1)
numQpTables = sps_same_qp_table_for_chroma_flag ? 1 : (sps_joint_cbcr_enabled_flag ? 3 : 2)	
for(i = 0; i < numQpTables; i++) {	
sps_qp_table_start_minus26[i]	se(v)
sps_num_points_in_qp_table_minus1[i]	ue(v)
for(j = 0; j <= sps_num_points_in_qp_table_minus1[i]; j++) {	
sps_delta_qp_in_val_minus1[i][j]	ue(v)
sps_delta_qp_diff_val[i][j]	ue(v)
}	
}	
}	
sps_sao_enabled_flag	u(1)

sps_alf_enabled_flag	u(1)
if(sps_alf_enabled_flag && sps_chroma_format_idc != 0)	
sps_ccalf_enabled_flag	u(1)
sps_lmcs_enabled_flag	u(1)
sps_weighted_pred_flag	u(1)
sps_weighted_bipred_flag	u(1)
sps_long_term_ref_pics_flag	u(1)
if(sps_video_parameter_set_id > 0)	
sps_inter_layer_prediction_enabled_flag	u(1)
sps_idr_rpl_present_flag	u(1)
sps_rpl1_same_as_rpl0_flag	u(1)
for(i = 0; i < (sps_rpl1_same_as_rpl0_flag ? 1 : 2); i++) {	
sps_num_ref_pic_lists[i]	ue(v)
for(j = 0; j < sps_num_ref_pic_lists[i]; j++)	
ref_pic_list_struct(i, j)	
}	
sps_ref_wraparound_enabled_flag	u(1)
sps_temporal_mvp_enabled_flag	u(1)
if(sps_temporal_mvp_enabled_flag)	
sps_sbtmvp_enabled_flag	u(1)
sps_amvr_enabled_flag	u(1)
sps_bdof_enabled_flag	u(1)
if(sps_bdof_enabled_flag)	
sps_bdof_control_present_in_ph_flag	u(1)
sps_smvd_enabled_flag	u(1)
sps_dmvr_enabled_flag	u(1)
if(sps_dmvr_enabled_flag)	
sps_dmvr_control_present_in_ph_flag	u(1)
sps_mmvd_enabled_flag	u(1)
if(sps_mmvd_enabled_flag)	
sps_mmvd_fullpel_only_enabled_flag	u(1)
sps_six_minus_max_num_merge_cand	ue(v)
sps_sbt_enabled_flag	u(1)
sps_affine_enabled_flag	u(1)
if(sps_affine_enabled_flag) {	
sps_five_minus_max_num_subblock_merge_cand	ue(v)
sps_6param_affine_enabled_flag	u(1)
if(sps_amvr_enabled_flag)	
sps_affine_amvr_enabled_flag	u(1)
sps_affine_prof_enabled_flag	u(1)
if(sps_affine_prof_enabled_flag)	
sps_prof_control_present_in_ph_flag	u(1)
}	
sps_bcw_enabled_flag	u(1)
sps_ciip_enabled_flag	u(1)
if(MaxNumMergeCand >= 2) {	
sps_gpm_enabled_flag	u(1)
if(sps_gpm_enabled_flag && MaxNumMergeCand >= 3)	

sps_max_num_merge_cand_minus_max_num_gpm_cand	ue(v)
}	
sps_log2_parallel_merge_level_minus2	ue(v)
sps_isp_enabled_flag	u(1)
sps_mrl_enabled_flag	u(1)
sps_mip_enabled_flag	u(1)
if(sps_chroma_format_idc != 0)	
sps_cclm_enabled_flag	u(1)
if(sps_chroma_format_idc = = 1) {	
sps_chroma_horizontal_collocated_flag	u(1)
sps_chroma_vertical_collocated_flag	u(1)
}	
sps_palette_enabled_flag	u(1)
if(sps_chroma_format_idc = = 3 && !sps_max_luma_transform_size_64_flag)	
sps_act_enabled_flag	u(1)
if(sps_transform_skip_enabled_flag || sps_palette_enabled_flag)	
sps_min_qp_prime_ts	ue(v)
sps_ibc_enabled_flag	u(1)
if(sps_ibc_enabled_flag)	
sps_six_minus_max_num_ibc_merge_cand	ue(v)
sps_ladf_enabled_flag	u(1)
if(sps_ladf_enabled_flag) {	
sps_num_ladf_intervals_minus2	u(2)
sps_ladf_lowest_interval_qp_offset	se(v)
for(i = 0; i < sps_num_ladf_intervals_minus2 + 1; i++) {	
sps_ladf_qp_offset[i]	se(v)
sps_ladf_delta_threshold_minus1[i]	ue(v)
}	
}	
sps_explicit_scaling_list_enabled_flag	u(1)
if(sps_lfnst_enabled_flag && sps_explicit_scaling_list_enabled_flag)	
sps_scaling_matrix_for_lfnst_disabled_flag	u(1)
if(sps_act_enabled_flag && sps_explicit_scaling_list_enabled_flag)	
sps_scaling_matrix_for_alternative_colour_space_disabled_flag	u(1)
if(sps_scaling_matrix_for_alternative_colour_space_disabled_flag)	
sps_scaling_matrix_designated_colour_space_flag	u(1)
sps_dep_quant_enabled_flag	u(1)
sps_sign_data_hiding_enabled_flag	u(1)
sps_virtual_boundaries_enabled_flag	u(1)
if(sps_virtual_boundaries_enabled_flag) {	
sps_virtual_boundaries_present_flag	u(1)
if(sps_virtual_boundaries_present_flag) {	
sps_num_ver_virtual_boundaries	ue(v)
for(i = 0; i < sps_num_ver_virtual_boundaries; i++)	
sps_virtual_boundary_pos_x_minus1[i]	ue(v)
sps_num_hor_virtual_boundaries	ue(v)
for(i = 0; i < sps_num_hor_virtual_boundaries; i++)	
sps_virtual_boundary_pos_y_minus1[i]	ue(v)

}	
}	
if(sps_ptl_dpb_hrd_params_present_flag) {	
sps_timing_hrd_params_present_flag	u(1)
if(sps_timing_hrd_params_present_flag) {	
general_timing_hrd_parameters( )	
if(sps_max_sublayers_minus1 > 0)	
sps_sublayer_cpb_params_present_flag	u(1)
firstSubLayer = sps_sublayer_cpb_params_present_flag ? 0 : sps_max_sublayers_minus1	
ols_timing_hrd_parameters(firstSubLayer, sps_max_sublayers_minus1)	
}	
}	
sps_field_seq_flag	u(1)
sps_vui_parameters_present_flag	u(1)
if(sps_vui_parameters_present_flag) {	
sps_vui_payload_size_minus1	ue(v)
while(!byte_aligned( ))	
sps_vui_alignment_zero_bit	f(1)
vui_payload(sps_vui_payload_size_minus1 + 1)	
}	
sps_extension_flag	u(1)
if(sps_extension_flag) {	
sps_range_extension_flag	u(1)
sps_extension_7bits	u(7)
if(sps_range_extension_flag)	
sps_range_extension( )	
}	
if(sps_extension_7bits)	
while(more_rbsp_data( ))	
sps_extension_data_flag	u(1)
rbsp_trailing_bits( )	
}	

7.3.2.5 Picture parameter set RBSP syntax

pic_parameter_set_rbsp( ) {	**Descriptor**
pps_pic_parameter_set_id	u(6)
pps_seq_parameter_set_id	u(4)
pps_mixed_nalu_types_in_pic_flag	u(1)
pps_pic_width_in_luma_samples	ue(v)
pps_pic_height_in_luma_samples	ue(v)
pps_conformance_window_flag	u(1)
if(pps_conformance_window_flag) {	
pps_conf_win_left_offset	ue(v)
pps_conf_win_right_offset	ue(v)
pps_conf_win_top_offset	ue(v)

pps_conf_win_bottom_offset	ue(v)
}	
pps_scaling_window_explicit_signalling_flag	u(1)
if(pps_scaling_window_explicit_signalling_flag) {	
pps_scaling_win_left_offset	se(v)
pps_scaling_win_right_offset	se(v)
pps_scaling_win_top_offset	se(v)
pps_scaling_win_bottom_offset	se(v)
}	
pps_output_flag_present_flag	u(1)
pps_no_pic_partition_flag	u(1)
pps_subpic_id_mapping_present_flag	u(1)
if(pps_subpic_id_mapping_present_flag) {	
if(!pps_no_pic_partition_flag)	
pps_num_subpics_minus1	ue(v)
pps_subpic_id_len_minus1	ue(v)
for(i = 0; i <= pps_num_subpics_minus1; i++)	
pps_subpic_id[i]	u(v)
}	
if(!pps_no_pic_partition_flag) {	
pps_log2_ctu_size_minus5	u(2)
pps_num_exp_tile_columns_minus1	ue(v)
pps_num_exp_tile_rows_minus1	ue(v)
for(i = 0; i <= pps_num_exp_tile_columns_minus1; i++)	
pps_tile_column_width_minus1[i]	ue(v)
for(i = 0; i <= pps_num_exp_tile_rows_minus1; i++)	
pps_tile_row_height_minus1[i]	ue(v)
if(NumTilesInPic > 1) {	
pps_loop_filter_across_tiles_enabled_flag	u(1)
pps_rect_slice_flag	u(1)
}	
if(pps_rect_slice_flag)	
pps_single_slice_per_subpic_flag	u(1)
if(pps_rect_slice_flag && !pps_single_slice_per_subpic_flag) {	
pps_num_slices_in_pic_minus1	ue(v)
if(pps_num_slices_in_pic_minus1 > 1)	
pps_tile_idx_delta_present_flag	u(1)
for(i = 0; i < pps_num_slices_in_pic_minus1; i++) {	
if(SliceTopLeftTileIdx[i] % NumTileColumns != NumTileColumns − 1)	
pps_slice_width_in_tiles_minus1[i]	ue(v)

if(SliceTopLeftTileIdx[i] / NumTileColumns != NumTileRows − 1 &&
            (pps_tile_idx_delta_present_flag | |
            SliceTopLeftTileIdx[i] % NumTileColumns = = 0))	

pps_slice_height_in_tiles_minus1[i]	ue(v)

if(pps_slice_width_in_tiles_minus1[i] = = 0 &&
            pps_slice_height_in_tiles_minus1[i] = = 0 &&
            RowHeightVal[SliceTopLeftTileIdx[i] / NumTileColumns] > 1) {	

pps_num_exp_slices_in_tile[i]	ue(v)
for(j = 0; j < pps_num_exp_slices_in_tile[i]; j++)	

pps_exp_slice_height_in_ctus_minus1[i][j]	ue(v)
i += NumSlicesInTile[i] − 1	
}	
if(pps_tile_idx_delta_present_flag && i < pps_num_slices_in_pic_minus1)	
pps_tile_idx_delta_val[i]	se(v)
}	
}	
if(!pps_rect_slice_flag || pps_single_slice_per_subpic_flag || pps_num_slices_in_pic_minus1 > 0)	
pps_loop_filter_across_slices_enabled_flag	u(1)
}	
pps_cabac_init_present_flag	u(1)
for(i = 0; i < 2; i++)	
pps_num_ref_idx_default_active_minus1[i]	ue(v)
pps_rpl1_idx_present_flag	u(1)
pps_weighted_pred_flag	u(1)
pps_weighted_bipred_flag	u(1)
pps_ref_wraparound_enabled_flag	u(1)
if(pps_ref_wraparound_enabled_flag)	
pps_pic_width_minus_wraparound_offset	ue(v)
pps_init_qp_minus26	se(v)
pps_cu_qp_delta_enabled_flag	u(1)
pps_chroma_tool_offsets_present_flag	u(1)
if(pps_chroma_tool_offsets_present_flag) {	
pps_cb_qp_offset	se(v)
pps_cr_qp_offset	se(v)
pps_joint_cbcr_qp_offset_present_flag	u(1)
if(pps_joint_cbcr_qp_offset_present_flag)	
pps_joint_cbcr_qp_offset_value	se(v)
pps_slice_chroma_qp_offsets_present_flag	u(1)
pps_cu_chroma_qp_offset_list_enabled_flag	u(1)
if(pps_cu_chroma_qp_offset_list_enabled_flag) {	
pps_chroma_qp_offset_list_len_minus1	ue(v)
for(i = 0; i <= pps_chroma_qp_offset_list_len_minus1; i++) {	
pps_cb_qp_offset_list[i]	se(v)
pps_cr_qp_offset_list[i]	se(v)
if(pps_joint_cbcr_qp_offset_present_flag)	
pps_joint_cbcr_qp_offset_list[i]	se(v)
}	
}	
}	
pps_deblocking_filter_control_present_flag	u(1)
if(pps_deblocking_filter_control_present_flag) {	
pps_deblocking_filter_override_enabled_flag	u(1)
pps_deblocking_filter_disabled_flag	u(1)
if(!pps_no_pic_partition_flag && pps_deblocking_filter_override_enabled_flag)	
pps_dbf_info_in_ph_flag	u(1)
if(!pps_deblocking_filter_disabled_flag) {	

pps_luma_beta_offset_div2	se(v)
pps_luma_tc_offset_div2	se(v)
if(pps_chroma_tool_offsets_present_flag) {	
pps_cb_beta_offset_div2	se(v)
pps_cb_tc_offset_div2	se(v)
pps_cr_beta_offset_div2	se(v)
pps_cr_tc_offset_div2	se(v)
}	
}	
}	
if(!pps_no_pic_partition_flag) {	
pps_rpl_info_in_ph_flag	u(1)
pps_sao_info_in_ph_flag	u(1)
pps_alf_info_in_ph_flag	u(1)
if((pps_weighted_pred_flag || pps_weighted_bipred_flag) && pps_rpl_info_in_ph_flag)	
pps_wp_info_in_ph_flag	u(1)
pps_qp_delta_info_in_ph_flag	u(1)
}	
pps_picture_header_extension_present_flag	u(1)
pps_slice_header_extension_present_flag	u(1)
pps_extension_flag	u(1)
if(pps_extension_flag)	
while(more_rbsp_data( ))	
pps_extension_data_flag	u(1)
rbsp_trailing_bits( )	
}	

7.3.2.6 Adaptation parameter set RBSP syntax

adaptation_parameter_set_rbsp( ) {	**Descriptor**
aps_params_type	u(3)
aps_adaptation_parameter_set_id	u(5)
aps_chroma_present_flag	u(1)
if(aps_params_type = = ALF_APS)	
alf_data( )	
else if(aps_params_type = = LMCS_APS)	
lmcs_data( )	
else if(aps_params_type = = SCALING_APS)	
scaling_list_data( )	
aps_extension_flag	u(1)
if(aps_extension_flag)	
while(more_rbsp_data( ))	
aps_extension_data_flag	u(1)
rbsp_trailing_bits( )	
}	

7.3.2.7 Picture header RBSP syntax

picture_header_rbsp( ) {	**Descriptor**
picture_header_structure( )	
rbsp_trailing_bits( )	
}	

7.3.2.8 Picture header structure syntax

picture_header_structure( ) {	**Descriptor**
ph_gdr_or_irap_pic_flag	u(1)
ph_non_ref_pic_flag	u(1)
if(ph_gdr_or_irap_pic_flag)	
ph_gdr_pic_flag	u(1)
ph_inter_slice_allowed_flag	u(1)
if(ph_inter_slice_allowed_flag)	
ph_intra_slice_allowed_flag	u(1)
ph_pic_parameter_set_id	ue(v)
ph_pic_order_cnt_lsb	u(v)
if(ph_gdr_pic_flag)	
ph_recovery_poc_cnt	ue(v)
for(i = 0; i < NumExtraPhBits; i++)	
ph_extra_bit[i]	u(1)
if(sps_poc_msb_cycle_flag) {	
ph_poc_msb_cycle_present_flag	u(1)
if(ph_poc_msb_cycle_present_flag)	
ph_poc_msb_cycle_val	u(v)
}	
if(sps_alf_enabled_flag && pps_alf_info_in_ph_flag) {	
ph_alf_enabled_flag	u(1)
if(ph_alf_enabled_flag) {	
ph_num_alf_aps_ids_luma	u(3)
for(i = 0; i < ph_num_alf_aps_ids_luma; i++)	
ph_alf_aps_id_luma[i]	u(3)
if(sps_chroma_format_idc != 0) {	
ph_alf_cb_enabled_flag	u(1)
ph_alf_cr_enabled_flag	u(1)
}	
if(ph_alf_cb_enabled_flag || ph_alf_cr_enabled_flag)	
ph_alf_aps_id_chroma	u(3)
if(sps_ccalf_enabled_flag) {	
ph_alf_cc_cb_enabled_flag	u(1)
if(ph_alf_cc_cb_enabled_flag)	
ph_alf_cc_cb_aps_id	u(3)
ph_alf_cc_cr_enabled_flag	u(1)
if(ph_alf_cc_cr_enabled_flag)	
ph_alf_cc_cr_aps_id	u(3)

}	
}	
}	
if(sps_lmcs_enabled_flag) {	
ph_lmcs_enabled_flag	u(1)
if(ph_lmcs_enabled_flag) {	
ph_lmcs_aps_id	u(2)
if(sps_chroma_format_idc != 0)	
ph_chroma_residual_scale_flag	u(1)
}	
}	
if(sps_explicit_scaling_list_enabled_flag) {	
ph_explicit_scaling_list_enabled_flag	u(1)
if(ph_explicit_scaling_list_enabled_flag)	
ph_scaling_list_aps_id	u(3)
}	
if(sps_virtual_boundaries_enabled_flag && !sps_virtual_boundaries_present_flag) {	
ph_virtual_boundaries_present_flag	u(1)
if(ph_virtual_boundaries_present_flag) {	
ph_num_ver_virtual_boundaries	ue(v)
for(i = 0; i < ph_num_ver_virtual_boundaries; i++)	
ph_virtual_boundary_pos_x_minus1[i]	ue(v)
ph_num_hor_virtual_boundaries	ue(v)
for(i = 0; i < ph_num_hor_virtual_boundaries; i++)	
ph_virtual_boundary_pos_y_minus1[i]	ue(v)
}	
}	
if(pps_output_flag_present_flag && !ph_non_ref_pic_flag)	
ph_pic_output_flag	u(1)
if(pps_rpl_info_in_ph_flag)	
ref_pic_lists( )	
if(sps_partition_constraints_override_enabled_flag)	
ph_partition_constraints_override_flag	u(1)
if(ph_intra_slice_allowed_flag) {	
if(ph_partition_constraints_override_flag) {	
ph_log2_diff_min_qt_min_cb_intra_slice_luma	ue(v)
ph_max_mtt_hierarchy_depth_intra_slice_luma	ue(v)
if(ph_max_mtt_hierarchy_depth_intra_slice_luma != 0) {	
ph_log2_diff_max_bt_min_qt_intra_slice_luma	ue(v)
ph_log2_diff_max_tt_min_qt_intra_slice_luma	ue(v)
}	
if(sps_qtbtt_dual_tree_intra_flag) {	
ph_log2_diff_min_qt_min_cb_intra_slice_chroma	ue(v)
ph_max_mtt_hierarchy_depth_intra_slice_chroma	ue(v)
if(ph_max_mtt_hierarchy_depth_intra_slice_chroma != 0) {	
ph_log2_diff_max_bt_min_qt_intra_slice_chroma	ue(v)
ph_log2_diff_max_tt_min_qt_intra_slice_chroma	ue(v)
}	

}	
}	
if(pps_cu_qp_delta_enabled_flag)	
ph_cu_qp_delta_subdiv_intra_slice	ue(v)
if(pps_cu_chroma_qp_offset_list_enabled_flag)	
ph_cu_chroma_qp_offset_subdiv_intra_slice	ue(v)
}	
if(ph_inter_slice_allowed_flag) {	
if(ph_partition_constraints_override_flag) {	
ph_log2_diff_min_qt_min_cb_inter_slice	ue(v)
ph_max_mtt_hierarchy_depth_inter_slice	ue(v)
if(ph_max_mtt_hierarchy_depth_inter_slice != 0) {	
ph_log2_diff_max_bt_min_qt_inter_slice	ue(v)
ph_log2_diff_max_tt_min_qt_inter_slice	ue(v)
}	
}	
if(pps_cu_qp_delta_enabled_flag)	
ph_cu_qp_delta_subdiv_inter_slice	ue(v)
if(pps_cu_chroma_qp_offset_list_enabled_flag)	
ph_cu_chroma_qp_offset_subdiv_inter_slice	ue(v)
if(sps_temporal_mvp_enabled_flag) {	
ph_temporal_mvp_enabled_flag	u(1)
if(ph_temporal_mvp_enabled_flag && pps_rpl_info_in_ph_flag) {	
if(num_ref_entries[1][RplsIdx[1]] > 0)	
ph_collocated_from_l0_flag	u(1)
if((ph_collocated_from_l0_flag && num_ref_entries[0][RplsIdx[0]] > 1) || (!ph_collocated_from_l0_flag && num_ref_entries[1][RplsIdx[1]] > 1))	
ph_collocated_ref_idx	ue(v)
}	
}	
if(sps_mmvd_fullpel_only_enabled_flag)	
ph_mmvd_fullpel_only_flag	u(1)
presenceFlag = 0	
if(!pps_rpl_info_in_ph_flag) /* This condition is intentionally not merged into the next, to avoid possible interpretation of RplsIdx[i] not having a specified value. */	
presenceFlag = 1	
else if(num_ref_entries[1][RplsIdx[1]] > 0)	
presenceFlag = 1	
if(presenceFlag) {	
ph_mvd_l1_zero_flag	u(1)
if(sps_bdof_control_present_in_ph_flag)	
ph_bdof_disabled_flag	u(1)
if(sps_dmvr_control_present_in_ph_flag)	
ph_dmvr_disabled_flag	u(1)
}	
if(sps_prof_control_present_in_ph_flag)	
ph_prof_disabled_flag	u(1)

if((pps_weighted_pred_flag || pps_weighted_bipred_flag) && pps_wp_info_in_ph_flag)	
pred_weight_table( )	
}	
if(pps_qp_delta_info_in_ph_flag)	
ph_qp_delta	se(v)
if(sps_joint_cbcr_enabled_flag)	
ph_joint_cbcr_sign_flag	u(1)
if(sps_sao_enabled_flag && pps_sao_info_in_ph_flag) {	
ph_sao_luma_enabled_flag	u(1)
if(sps_chroma_format_idc != 0)	
ph_sao_chroma_enabled_flag	u(1)
}	
if(pps_dbf_info_in_ph_flag) {	
ph_deblocking_params_present_flag	u(1)
if(ph_deblocking_params_present_flag) {	
if(!pps_deblocking_filter_disabled_flag)	
ph_deblocking_filter_disabled_flag	u(1)
if(!ph_deblocking_filter_disabled_flag) {	
ph_luma_beta_offset_div2	se(v)
ph_luma_tc_offset_div2	se(v)
if(pps_chroma_tool_offsets_present_flag) {	
ph_cb_beta_offset_div2	se(v)
ph_cb_tc_offset_div2	se(v)
ph_cr_beta_offset_div2	se(v)
ph_cr_tc_offset_div2	se(v)
}	
}	
}	
}	
if(pps_picture_header_extension_present_flag) {	
ph_extension_length	ue(v)
for(i = 0; i < ph_extension_length; i++)	
ph_extension_data_byte[i]	u(8)
}	
}	

7.3.2.9 Supplemental enhancement information RBSP syntax

sei_rbsp( ) {	**Descriptor**
do	
sei_message( )	
while(more_rbsp_data( ))	
rbsp_trailing_bits( )	
}	

7.3.2.10 AU delimiter RBSP syntax

access_unit_delimiter_rbsp( ) {	**Descriptor**
aud_irap_or_gdr_flag	u(1)
aud_pic_type	u(3)
rbsp_trailing_bits( )	
}	

7.3.2.11 End of sequence RBSP syntax

end_of_seq_rbsp( ) {	**Descriptor**
}	

7.3.2.12 End of bitstream RBSP syntax

end_of_bitstream_rbsp( ) {	**Descriptor**
}	

7.3.2.13 Filler data RBSP syntax

filler_data_rbsp( ) {	**Descriptor**
while(next_bits(8) = = 0xFF)	
fd_ff_byte /* equal to 0xFF */	f(8)
rbsp_trailing_bits( )	
}	

7.3.2.14 Slice layer RBSP syntax

slice_layer_rbsp( ) {	**Descriptor**
slice_header( )	
slice_data( )	
rbsp_slice_trailing_bits( )	
}	

7.3.2.15 RBSP slice trailing bits syntax

rbsp_slice_trailing_bits( ) {	**Descriptor**
rbsp_trailing_bits( )	
while(more_rbsp_trailing_data( ))	
rbsp_cabac_zero_word /* equal to 0x0000 */	f(16)
}	

7.3.2.16 RBSP trailing bits syntax

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

7.3.2.17 Byte alignment syntax

byte_alignment( ) {	**Descriptor**
byte_alignment_bit_equal_to_one /* equal to 1 */	f(1)
while(!byte_aligned( ))	
byte_alignment_bit_equal_to_zero /* equal to 0 */	f(1)
}	

7.3.2.18 Adaptive loop filter data syntax

alf_data( ) {	**Descriptor**
alf_luma_filter_signal_flag	u(1)
if(aps_chroma_present_flag) {	
alf_chroma_filter_signal_flag	u(1)
alf_cc_cb_filter_signal_flag	u(1)
alf_cc_cr_filter_signal_flag	u(1)
}	
if(alf_luma_filter_signal_flag) {	
alf_luma_clip_flag	u(1)
alf_luma_num_filters_signalled_minus1	ue(v)
if(alf_luma_num_filters_signalled_minus1 > 0)	
for(filtIdx = 0; filtIdx < NumAlfFilters; filtIdx++)	
alf_luma_coeff_delta_idx[filtIdx]	u(v)
for(sfIdx = 0; sfIdx <= alf_luma_num_filters_signalled_minus1; sfIdx++)	
for(j = 0; j < 12; j++) {	
alf_luma_coeff_abs[sfIdx][j]	ue(v)
if(alf_luma_coeff_abs[sfIdx][j])	
alf_luma_coeff_sign[sfIdx][j]	u(1)
}	
if(alf_luma_clip_flag)	
for(sfIdx = 0; sfIdx <= alf_luma_num_filters_signalled_minus1; sfIdx++)	
for(j = 0; j < 12; j++)	
alf_luma_clip_idx[sfIdx][j]	u(2)
}	
if(alf_chroma_filter_signal_flag) {	
alf_chroma_clip_flag	u(1)
alf_chroma_num_alt_filters_minus1	ue(v)
for(altIdx = 0; altIdx <= alf_chroma_num_alt_filters_minus1; altIdx++) {	

for(j = 0; j < 6; j++) {	
alf_chroma_coeff_abs[altIdx][j]	ue(v)
if(alf_chroma_coeff_abs[altIdx][j] > 0)	
alf_chroma_coeff_sign[altIdx][j]	u(1)
}	
if(alf_chroma_clip_flag)	
for(j = 0; j < 6; j++)	
alf_chroma_clip_idx[altIdx][j]	u(2)
}	
}	
if(alf_cc_cb_filter_signal_flag) {	
alf_cc_cb_filters_signalled_minus1	ue(v)
for(k = 0; k < alf_cc_cb_filters_signalled_minus1 + 1; k++) {	
for(j = 0; j < 7; j++) {	
alf_cc_cb_mapped_coeff_abs[k][j]	u(3)
if(alf_cc_cb_mapped_coeff_abs[k][j])	
alf_cc_cb_coeff_sign[k][j]	u(1)
}	
}	
}	
if(alf_cc_cr_filter_signal_flag) {	
alf_cc_cr_filters_signalled_minus1	ue(v)
for(k = 0; k < alf_cc_cr_filters_signalled_minus1 + 1; k++) {	
for(j = 0; j < 7; j++) {	
alf_cc_cr_mapped_coeff_abs[k][j]	u(3)
if(alf_cc_cr_mapped_coeff_abs[k][j])	
alf_cc_cr_coeff_sign[k][j]	u(1)
}	
}	
}	
}	

7.3.2.19 Luma mapping with chroma scaling data syntax

lmcs_data( ) {	**Descriptor**
lmcs_min_bin_idx	ue(v)
lmcs_delta_max_bin_idx	ue(v)
lmcs_delta_cw_prec_minus1	ue(v)
for(i = lmcs_min_bin_idx; i <= LmcsMaxBinIdx; i++) {	
lmcs_delta_abs_cw[i]	u(v)
if(lmcs_delta_abs_cw[i] > 0)	
lmcs_delta_sign_cw_flag[i]	u(1)
}	
if(aps_chroma_present_flag) {	
lmcs_delta_abs_crs	u(3)

if(lmcs_delta_abs_crs > 0)	
lmcs_delta_sign_crs_flag	u(1)
}	
}	

7.3.2.20 Scaling list data syntax

scaling_list_data( ) {	**Descriptor**
for(id = 0; id < 28; id ++) {	
matrixSize = id < 2 ? 2 : (id < 8 ? 4 : 8)	
if(aps_chroma_present_flag || id % 3 = = 2 || id = = 27) {	
scaling_list_copy_mode_flag[id]	u(1)
if(!scaling_list_copy_mode_flag[id])	
scaling_list_pred_mode_flag[id]	u(1)
if((scaling_list_copy_mode_flag[id] || scaling_list_pred_mode_flag[id]) && id != 0 && id != 2 && id != 8)	
scaling_list_pred_id_delta[id]	ue(v)
if(!scaling_list_copy_mode_flag[id]) {	
nextCoef = 0	
if(id > 13) {	
scaling_list_dc_coef[id − 14]	se(v)
nextCoef += scaling_list_dc_coef[id − 14]	
}	
for(i = 0; i < matrixSize * matrixSize; i++) {	
x = DiagScanOrder[3][3][i][0]	
y = DiagScanOrder[3][3][i][1]	
if(!(id > 25 && x >= 4 && y >= 4)) {	
scaling_list_delta_coef[id][i]	se(v)
nextCoef += scaling_list_delta_coef[id][i]	
}	
ScalingList[id][i] = nextCoef	
}	
}	
}	
}	
}	

7.3.2.21 VUI payload syntax

vui_payload(payloadSize) {	**Descriptor**
VuiExtensionBitsPresentFlag = 0	
vui_parameters(payloadSize) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
if(VuiExtensionBitsPresentFlag || more_data_in_payload( )) {	
if(payload_extension_present( ))	
vui_reserved_payload_extension_data	u(v)
vui_payload_bit_equal_to_one /* equal to 1 */	f(1)
while(!byte_aligned( ))	

vui_payload_bit_equal_to_zero /* equal to 0 */	f(1)
}	
}	

7.3.2.22 Sequence parameter set range extension syntax

sps_range_extension( ) {	**Descriptor**
sps_extended_precision_flag	u(1)
if(sps_transform_skip_enabled_flag)	
sps_ts_residual_coding_rice_present_in_sh_flag	u(1)
sps_rrc_rice_extension_flag	u(1)
sps_persistent_rice_adaptation_enabled_flag	u(1)
sps_reverse_last_sig_coeff_enabled_flag	u(1)
}	

7.3.3 Profile, tier, and level syntax

7.3.3.1 General profile, tier, and level syntax

profile_tier_level(profileTierPresentFlag, MaxNumSubLayersMinus1) {	**Descriptor**
if(profileTierPresentFlag) {	
general_profile_idc	u(7)
general_tier_flag	u(1)
}	
general_level_idc	u(8)
ptl_frame_only_constraint_flag	u(1)
ptl_multilayer_enabled_flag	u(1)
if(profileTierPresentFlag)	
general_constraints_info( )	
for(i = MaxNumSubLayersMinus1 − 1; i >= 0; i− −)	
ptl_sublayer_level_present_flag[i]	u(1)
while(!byte_aligned( ))	
ptl_reserved_zero_bit	u(1)
for(i = MaxNumSubLayersMinus1 − 1; i >= 0; i− −)	
if(ptl_sublayer_level_present_flag[i])	
sublayer_level_idc[i]	u(8)
if(profileTierPresentFlag) {	
ptl_num_sub_profiles	u(8)
for(i = 0; i < ptl_num_sub_profiles; i++)	
general_sub_profile_idc[i]	u(32)
}	
}	

7.3.3.2 General constraints information syntax

general_constraints_info( ) {	**Descriptor**
gci_present_flag	u(1)
if(gci_present_flag) {	
/* general */	
gci_intra_only_constraint_flag	u(1)
gci_all_layers_independent_constraint_flag	u(1)
gci_one_au_only_constraint_flag	u(1)
/* picture format */	
gci_sixteen_minus_max_bitdepth_constraint_idc	u(4)
gci_three_minus_max_chroma_format_constraint_idc	u(2)
/* NAL unit type related */	
gci_no_mixed_nalu_types_in_pic_constraint_flag	u(1)
gci_no_trail_constraint_flag	u(1)
gci_no_stsa_constraint_flag	u(1)
gci_no_rasl_constraint_flag	u(1)
gci_no_radl_constraint_flag	u(1)
gci_no_idr_constraint_flag	u(1)
gci_no_cra_constraint_flag	u(1)
gci_no_gdr_constraint_flag	u(1)
gci_no_aps_constraint_flag	u(1)
gci_no_idr_rpl_constraint_flag	u(1)
/* tile, slice, subpicture partitioning */	
gci_one_tile_per_pic_constraint_flag	u(1)
gci_pic_header_in_slice_header_constraint_flag	u(1)
gci_one_slice_per_pic_constraint_flag	u(1)
gci_no_rectangular_slice_constraint_flag	u(1)
gci_one_slice_per_subpic_constraint_flag	u(1)
gci_no_subpic_info_constraint_flag	u(1)
/* CTU and block partitioning */	
gci_three_minus_max_log2_ctu_size_constraint_idc	u(2)
gci_no_partition_constraints_override_constraint_flag	u(1)
gci_no_mtt_constraint_flag	u(1)
gci_no_qtbtt_dual_tree_intra_constraint_flag	u(1)
/* intra */	
gci_no_palette_constraint_flag	u(1)
gci_no_ibc_constraint_flag	u(1)
gci_no_isp_constraint_flag	u(1)
gci_no_mrl_constraint_flag	u(1)
gci_no_mip_constraint_flag	u(1)
gci_no_cclm_constraint_flag	u(1)
/* inter */	
gci_no_ref_pic_resampling_constraint_flag	u(1)
gci_no_res_change_in_clvs_constraint_flag	u(1)
gci_no_weighted_prediction_constraint_flag	u(1)
gci_no_ref_wraparound_constraint_flag	u(1)
gci_no_temporal_mvp_constraint_flag	u(1)

gci_no_sbtmvp_constraint_flag	u(1)
gci_no_amvr_constraint_flag	u(1)
gci_no_bdof_constraint_flag	u(1)
gci_no_smvd_constraint_flag	u(1)
gci_no_dmvr_constraint_flag	u(1)
gci_no_mmvd_constraint_flag	u(1)
gci_no_affine_motion_constraint_flag	u(1)
gci_no_prof_constraint_flag	u(1)
gci_no_bcw_constraint_flag	u(1)
gci_no_ciip_constraint_flag	u(1)
gci_no_gpm_constraint_flag	u(1)
/* transform, quantization, residual */	
gci_no_luma_transform_size_64_constraint_flag	u(1)
gci_no_transform_skip_constraint_flag	u(1)
gci_no_bdpcm_constraint_flag	u(1)
gci_no_mts_constraint_flag	u(1)
gci_no_lfnst_constraint_flag	u(1)
gci_no_joint_cbcr_constraint_flag	u(1)
gci_no_sbt_constraint_flag	u(1)
gci_no_act_constraint_flag	u(1)
gci_no_explicit_scaling_list_constraint_flag	u(1)
gci_no_dep_quant_constraint_flag	u(1)
gci_no_sign_data_hiding_constraint_flag	u(1)
gci_no_cu_qp_delta_constraint_flag	u(1)
gci_no_chroma_qp_offset_constraint_flag	u(1)
/* loop filter */	
gci_no_sao_constraint_flag	u(1)
gci_no_alf_constraint_flag	u(1)
gci_no_ccalf_constraint_flag	u(1)
gci_no_lmcs_constraint_flag	u(1)
gci_no_ladf_constraint_flag	u(1)
gci_no_virtual_boundaries_constraint_flag	u(1)
gci_num_additional_bits	u(8)
if(gci_num_additional_bits > 5) {	
gci_all_rap_pictures_constraint_flag	u(1)
gci_no_extended_precision_processing_constraint_flag	u(1)
gci_no_ts_residual_coding_rice_constraint_flag	u(1)
gci_no_rrc_rice_extension_constraint_flag	u(1)
gci_no_persistent_rice_adaptation_constraint_flag	u(1)
gci_no_reverse_last_sig_coeff_constraint_flag	u(1)
numAdditionalBitsUsed = 6	
} else	
numAdditionalBitsUsed = 0	
for(i = 0; i < gci_num_additional_bits − numAdditionalBitsUsed; i++)	
gci_reserved_bit[i]	u(1)
}	
while(!byte_aligned( ))	
gci_alignment_zero_bit	f(1)

}	

7.3.4 DPB parameters syntax

dpb_parameters(MaxSubLayersMinus1, subLayerInfoFlag) {	**Descriptor**
for(i = (subLayerInfoFlag ? 0 : MaxSubLayersMinus1); i <= MaxSubLayersMinus1; i++) {	
dpb_max_dec_pic_buffering_minus1[i]	ue(v)
dpb_max_num_reorder_pics[i]	ue(v)
dpb_max_latency_increase_plus1[i]	ue(v)
}	
}	

7.3.5 Timing and HRD parameters syntax

7.3.5.1 General timing and HRD parameters syntax

general_timing_hrd_parameters( ) {	**Descriptor**
num_units_in_tick	u(32)
time_scale	u(32)
general_nal_hrd_params_present_flag	u(1)
general_vcl_hrd_params_present_flag	u(1)
if(general_nal_hrd_params_present_flag || general_vcl_hrd_params_present_flag) {	
general_same_pic_timing_in_all_ols_flag	u(1)
general_du_hrd_params_present_flag	u(1)
if(general_du_hrd_params_present_flag)	
tick_divisor_minus2	u(8)
bit_rate_scale	u(4)
cpb_size_scale	u(4)
if(general_du_hrd_params_present_flag)	
cpb_size_du_scale	u(4)
hrd_cpb_cnt_minus1	ue(v)
}	
}	

7.3.5.2 OLS timing and HRD parameters syntax

ols_timing_hrd_parameters(firstSubLayer, MaxSubLayersVal) {	**Descriptor**
for(i = firstSubLayer; i <= MaxSubLayersVal; i++) {	
fixed_pic_rate_general_flag[i]	u(1)
if(!fixed_pic_rate_general_flag[i])	
fixed_pic_rate_within_cvs_flag[i]	u(1)
if(fixed_pic_rate_within_cvs_flag[i])	
elemental_duration_in_tc_minus1[i]	ue(v)
else if((general_nal_hrd_params_present_flag || general_vcl_hrd_params_present_flag) && hrd_cpb_cnt_minus1 = = 0)	
low_delay_hrd_flag[i]	u(1)

if(general_nal_hrd_params_present_flag)	
sublayer_hrd_parameters(i)	
if(general_vcl_hrd_params_present_flag)	
sublayer_hrd_parameters(i)	
}	
}	

7.3.5.3 Sublayer HRD parameters syntax

sublayer_hrd_parameters(subLayerId) {	**Descriptor**
for(j = 0; j <= hrd_cpb_cnt_minus1; j++) {	
bit_rate_value_minus1[subLayerId][j]	ue(v)
cpb_size_value_minus1[subLayerId][j]	ue(v)
if(general_du_hrd_params_present_flag) {	
cpb_size_du_value_minus1[subLayerId][j]	ue(v)
bit_rate_du_value_minus1[subLayerId][j]	ue(v)
}	
cbr_flag[subLayerId][j]	u(1)
}	
}	

7.3.6 Supplemental enhancement information message syntax

sei_message( ) {	**Descriptor**
payloadType = 0	
do {	
payload_type_byte	u(8)
payloadType += payload_type_byte	
} while(payload_type_byte = = 0xFF)	
payloadSize = 0	
do {	
payload_size_byte	u(8)
payloadSize += payload_size_byte	
} while(payload_size_byte = = 0xFF)	
sei_payload(payloadType, payloadSize)	
}	

7.3.7 Slice header syntax

slice_header( ) {	**Descriptor**
sh_picture_header_in_slice_header_flag	u(1)
if(sh_picture_header_in_slice_header_flag)	
picture_header_structure( )	
if(sps_subpic_info_present_flag)	
sh_subpic_id	u(v)

if((pps_rect_slice_flag && NumSlicesInSubpic[CurrSubpicIdx] > 1) || (!pps_rect_slice_flag && NumTilesInPic > 1))	
sh_slice_address	u(v)
for(i = 0; i < NumExtraShBits; i++)	
sh_extra_bit[i]	u(1)
if(!pps_rect_slice_flag && NumTilesInPic − sh_slice_address > 1)	
sh_num_tiles_in_slice_minus1	ue(v)
if(ph_inter_slice_allowed_flag)	
sh_slice_type	ue(v)
if(nal_unit_type = = IDR_W_RADL || nal_unit_type = = IDR_N_LP || nal_unit_type = = CRA_NUT || nal_unit_type = = GDR_NUT)	
sh_no_output_of_prior_pics_flag	u(1)
if(sps_alf_enabled_flag && !pps_alf_info_in_ph_flag) {	
sh_alf_enabled_flag	u(1)
if(sh_alf_enabled_flag) {	
sh_num_alf_aps_ids_luma	u(3)
for(i = 0; i < sh_num_alf_aps_ids_luma; i++)	
sh_alf_aps_id_luma[i]	u(3)
if(sps_chroma_format_idc != 0) {	
sh_alf_cb_enabled_flag	u(1)
sh_alf_cr_enabled_flag	u(1)
}	
if(sh_alf_cb_enabled_flag || sh_alf_cr_enabled_flag)	
sh_alf_aps_id_chroma	u(3)
if(sps_ccalf_enabled_flag) {	
sh_alf_cc_cb_enabled_flag	u(1)
if(sh_alf_cc_cb_enabled_flag)	
sh_alf_cc_cb_aps_id	u(3)
sh_alf_cc_cr_enabled_flag	u(1)
if(sh_alf_cc_cr_enabled_flag)	
sh_alf_cc_cr_aps_id	u(3)
}	
}	
}	
if(ph_lmcs_enabled_flag && !sh_picture_header_in_slice_header_flag)	
sh_lmcs_used_flag	u(1)
if(ph_explicit_scaling_list_enabled_flag && !sh_picture_header_in_slice_header_flag)	
sh_explicit_scaling_list_used_flag	u(1)
if(!pps_rpl_info_in_ph_flag && ((nal_unit_type != IDR_W_RADL && nal_unit_type != IDR_N_LP) || sps_idr_rpl_present_flag))	
ref_pic_lists( )	
if((sh_slice_type != I && num_ref_entries[0][RplsIdx[0]] > 1) || (sh_slice_type = = B && num_ref_entries[1][RplsIdx[1]] > 1)) {	
sh_num_ref_idx_active_override_flag	u(1)
if(sh_num_ref_idx_active_override_flag)	
for(i = 0; i < (sh_slice_type = = B ? 2: 1); i++)	
if(num_ref_entries[i][RplsIdx[i]] > 1)	
sh_num_ref_idx_active_minus1[i]	ue(v)
}	

if(sh_slice_type != I) {	
if(pps_cabac_init_present_flag)	
sh_cabac_init_flag	u(1)
if(ph_temporal_mvp_enabled_flag && !pps_rpl_info_in_ph_flag) {	
if(sh_slice_type = = B)	
sh_collocated_from_l0_flag	u(1)
if((sh_collocated_from_l0_flag && NumRefIdxActive[0] > 1) || (! sh_collocated_from_l0_flag && NumRefIdxActive[1] > 1))	
sh_collocated_ref_idx	ue(v)
}	
if(!pps_wp_info_in_ph_flag && ((pps_weighted_pred_flag && sh_slice_type = = P) || (pps_weighted_bipred_flag && sh_slice_type = = B)))	
pred_weight_table( )	
}	
if(!pps_qp_delta_info_in_ph_flag)	
sh_qp_delta	se(v)
if(pps_slice_chroma_qp_offsets_present_flag) {	
sh_cb_qp_offset	se(v)
sh_cr_qp_offset	se(v)
if(sps_joint_cbcr_enabled_flag)	
sh_joint_cbcr_qp_offset	se(v)
}	
if(pps_cu_chroma_qp_offset_list_enabled_flag)	
sh_cu_chroma_qp_offset_enabled_flag	u(1)
if(sps_sao_enabled_flag && !pps_sao_info_in_ph_flag) {	
sh_sao_luma_used_flag	u(1)
if(sps_chroma_format_idc != 0)	
sh_sao_chroma_used_flag	u(1)
}	
if(pps_deblocking_filter_override_enabled_flag && !pps_dbf_info_in_ph_flag)	
sh_deblocking_params_present_flag	u(1)
if(sh_deblocking_params_present_flag) {	
if(!pps_deblocking_filter_disabled_flag)	
sh_deblocking_filter_disabled_flag	u(1)
if(!sh_deblocking_filter_disabled_flag) {	
sh_luma_beta_offset_div2	se(v)
sh_luma_tc_offset_div2	se(v)
if(pps_chroma_tool_offsets_present_flag) {	
sh_cb_beta_offset_div2	se(v)
sh_cb_tc_offset_div2	se(v)
sh_cr_beta_offset_div2	se(v)
sh_cr_tc_offset_div2	se(v)
}	
}	
}	
if(sps_dep_quant_enabled_flag)	
sh_dep_quant_used_flag	u(1)
if(sps_sign_data_hiding_enabled_flag && !sh_dep_quant_used_flag)	

sh_sign_data_hiding_used_flag	u(1)
if(sps_transform_skip_enabled_flag && !sh_dep_quant_used_flag && !sh_sign_data_hiding_used_flag)	
sh_ts_residual_coding_disabled_flag	u(1)
if(!sh_ts_residual_coding_disabled_flag && sps_ts_residual_coding_rice_present_in_sh_flag)	
sh_ts_residual_coding_rice_idx_minus1	u(3)
if(sps_reverse_last_sig_coeff_enabled_flag)	
sh_reverse_last_sig_coeff_flag	u(1)
if(pps_slice_header_extension_present_flag) {	
sh_slice_header_extension_length	ue(v)
for(i = 0; i < sh_slice_header_extension_length; i++)	
sh_slice_header_extension_data_byte[i]	u(8)
}	
if(NumEntryPoints > 0) {	
sh_entry_offset_len_minus1	ue(v)
for(i = 0; i < NumEntryPoints; i++)	
sh_entry_point_offset_minus1[i]	u(v)
}	
byte_alignment( )	
}	

7.3.8 Weighted prediction parameters syntax

pred_weight_table( ) {	**Descriptor**
luma_log2_weight_denom	ue(v)
if(sps_chroma_format_idc != 0)	
delta_chroma_log2_weight_denom	se(v)
if(pps_wp_info_in_ph_flag)	
num_l0_weights	ue(v)
for(i = 0; i < NumWeightsL0; i++)	
luma_weight_l0_flag[i]	u(1)
if(sps_chroma_format_idc != 0)	
for(i = 0; i < NumWeightsL0; i++)	
chroma_weight_l0_flag[i]	u(1)
for(i = 0; i < NumWeightsL0; i++) {	
if(luma_weight_l0_flag[i]) {	
delta_luma_weight_l0[i]	se(v)
luma_offset_l0[i]	se(v)
}	
if(chroma_weight_l0_flag[i])	
for(j = 0; j < 2; j++) {	
delta_chroma_weight_l0[i][j]	se(v)
delta_chroma_offset_l0[i][j]	se(v)
}	
}	
if(pps_weighted_bipred_flag && pps_wp_info_in_ph_flag && num_ref_entries[1][RplsIdx[1]] > 0)	

num_l1_weights	ue(v)
for(i = 0; i < NumWeightsL1; i++)	
luma_weight_l1_flag[i]	u(1)
if(sps_chroma_format_idc != 0)	
for(i = 0; i < NumWeightsL1; i++)	
chroma_weight_l1_flag[i]	u(1)
for(i = 0; i < NumWeightsL1; i++) {	
if(luma_weight_l1_flag[i]) {	
delta_luma_weight_l1[i]	se(v)
luma_offset_l1[i]	se(v)
}	
if(chroma_weight_l1_flag[i])	
for(j = 0; j < 2; j++) {	
delta_chroma_weight_l1[i][j]	se(v)
delta_chroma_offset_l1[i][j]	se(v)
}	
}	
}	

7.3.9 Reference picture lists syntax

ref_pic_lists( ) {	**Descriptor**
for(i = 0; i < 2; i++) {	
if(sps_num_ref_pic_lists[i] > 0 && (i = = 0 || (i = = 1 && pps_rpl1_idx_present_flag)))	
rpl_sps_flag[i]	u(1)
if(rpl_sps_flag[i]) {	
if(sps_num_ref_pic_lists[i] > 1 && (i = = 0 || (i = = 1 && pps_rpl1_idx_present_flag)))	
rpl_idx[i]	u(v)
} else	
ref_pic_list_struct(i, sps_num_ref_pic_lists[i])	
for(j = 0; j < NumLtrpEntries[i][RplsIdx[i]]; j++) {	
if(ltrp_in_header_flag[i][RplsIdx[i]])	
poc_lsb_lt[i][j]	u(v)
delta_poc_msb_cycle_present_flag[i][j]	u(1)
if(delta_poc_msb_cycle_present_flag[i][j])	
delta_poc_msb_cycle_lt[i][j]	ue(v)
}	
}	
}	

7.3.10 Reference picture list structure syntax

ref_pic_list_struct(listIdx, rplsIdx) {	**Descriptor**
num_ref_entries[listIdx][rplsIdx]	ue(v)

if(sps_long_term_ref_pics_flag && rplsIdx < sps_num_ref_pic_lists[listIdx] && num_ref_entries[listIdx][rplsIdx] > 0)	
ltrp_in_header_flag[listIdx][rplsIdx]	u(1)
for(i = 0, j = 0; i < num_ref_entries[listIdx][rplsIdx]; i++) {	
if(sps_inter_layer_prediction_enabled_flag)	
inter_layer_ref_pic_flag[listIdx][rplsIdx][i]	u(1)
if(!inter_layer_ref_pic_flag[listIdx][rplsIdx][i]) {	
if(sps_long_term_ref_pics_flag)	
st_ref_pic_flag[listIdx][rplsIdx][i]	u(1)
if(st_ref_pic_flag[listIdx][rplsIdx][i]) {	
abs_delta_poc_st[listIdx][rplsIdx][i]	ue(v)
if(AbsDeltaPocSt[listIdx][rplsIdx][i] > 0)	
strp_entry_sign_flag[listIdx][rplsIdx][i]	u(1)
} else if(!ltrp_in_header_flag[listIdx][rplsIdx])	
rpls_poc_lsb_lt[listIdx][rplsIdx][j++]	u(v)
} else	
ilrp_idx[listIdx][rplsIdx][i]	ue(v)
}	
}	

7.3.11 Slice data syntax

7.3.11.1 General slice data syntax

slice_data( ) {	**Descriptor**
FirstCtbRowInSlice = 1	
for(i = 0; i < NumCtusInCurrSlice; i++) {	
CtbAddrInRs = CtbAddrInCurrSlice[i]	
CtbAddrX = (CtbAddrInRs % PicWidthInCtbsY)	
CtbAddrY = (CtbAddrInRs / PicWidthInCtbsY)	
if(CtbAddrX = = CtbToTileColBd[CtbAddrX]) {	
NumHmvpCand = 0	
NumHmvpIbcCand = 0	
ResetIbcBuf = 1	
}	
coding_tree_unit( )	
if(i = = NumCtusInCurrSlice − 1)	
end_of_slice_one_bit /* equal to 1 */	ae(v)
else if(CtbAddrX = = CtbToTileColBd[CtbAddrX + 1] − 1) {	
if(CtbAddrY = = CtbToTileRowBd[CtbAddrY + 1] − 1) {	
end_of_tile_one_bit /* equal to 1 */	ae(v)
byte_alignment( )	
} else if(sps_entropy_coding_sync_enabled_flag) {	
end_of_subset_one_bit /* equal to 1 */	ae(v)
byte_alignment( )	
}	
FirstCtbRowInSlice = 0	
}	

}	
}	

7.3.11.2 Coding tree unit syntax

coding_tree_unit( ) {	**Descriptor**
xCtb = CtbAddrX << CtbLog2SizeY	
yCtb = CtbAddrY << CtbLog2SizeY	
if(sh_sao_luma_used_flag || sh_sao_chroma_used_flag)	
sao(CtbAddrX, CtbAddrY)	
if(sh_alf_enabled_flag){	
alf_ctb_flag[0][CtbAddrX][CtbAddrY]	ae(v)
if(alf_ctb_flag[0][CtbAddrX][CtbAddrY]) {	
if(sh_num_alf_aps_ids_luma > 0)	
alf_use_aps_flag	ae(v)
if(alf_use_aps_flag) {	
if(sh_num_alf_aps_ids_luma > 1)	
alf_luma_prev_filter_idx	ae(v)
} else	
alf_luma_fixed_filter_idx	ae(v)
}	
if(sh_alf_cb_enabled_flag) {	
alf_ctb_flag[1][CtbAddrX][CtbAddrY]	ae(v)
if(alf_ctb_flag[1][CtbAddrX][CtbAddrY] && alf_chroma_num_alt_filters_minus1 > 0)	
alf_ctb_filter_alt_idx[0][CtbAddrX][CtbAddrY]	ae(v)
}	
if(sh_alf_cr_enabled_flag) {	
alf_ctb_flag[2][CtbAddrX][CtbAddrY]	ae(v)
if(alf_ctb_flag[2][CtbAddrX][CtbAddrY] && alf_chroma_num_alt_filters_minus1 > 0)	
alf_ctb_filter_alt_idx[1][CtbAddrX][CtbAddrY]	ae(v)
}	
}	
if(sh_alf_cc_cb_enabled_flag)	
alf_ctb_cc_cb_idc[CtbAddrX][CtbAddrY]	ae(v)
if(sh_alf_cc_cr_enabled_flag)	
alf_ctb_cc_cr_idc[CtbAddrX][CtbAddrY]	ae(v)
if(sh_slice_type = = I && sps_qtbtt_dual_tree_intra_flag)	
dual_tree_implicit_qt_split(xCtb, yCtb, CtbSizeY, 0)	
else	
coding_tree(xCtb, yCtb, CtbSizeY, CtbSizeY, 1, 1, 0, 0, 0, 0, 0, SINGLE_TREE, MODE_TYPE_ALL)	
}	

dual_tree_implicit_qt_split(x0, y0, cbSize, cqtDepth) {	**Descriptor**
cbSubdiv = 2 * cqtDepth	
if(cbSize > 64) {	

if(pps_cu_qp_delta_enabled_flag && cbSubdiv <= CuQpDeltaSubdiv) {	
IsCuQpDeltaCoded = 0	
CuQpDeltaVal = 0	
CuQgTopLeftX = x0	
CuQgTopLeftY = y0	
}	
if(sh_cu_chroma_qp_offset_enabled_flag && cbSubdiv <= CuChromaQpOffsetSubdiv) {	
IsCuChromaQpOffsetCoded = 0	
CuQpOffset_{Cb} = 0	
CuQpOffset_{Cr} = 0	
CuQpOffset_{CbCr} = 0	
}	
x1 = x0 + (cbSize / 2)	
y1 = y0 + (cbSize / 2)	
dual_tree_implicit_qt_split(x0, y0, cbSize / 2, cqtDepth + 1)	
if(x1 < pps_pic_width_in_luma_samples)	
dual_tree_implicit_qt_split(x1, y0, cbSize / 2, cqtDepth + 1)	
if(y1 < pps_pic_height_in_luma_samples)	
dual_tree_implicit_qt_split(x0, y1, cbSize / 2, cqtDepth + 1)	
if(x1 < pps_pic_width_in_luma_samples && y1 < pps_pic_height_in_luma_samples)	
dual_tree_implicit_qt_split(x1, y1, cbSize / 2, cqtDepth + 1)	
} else {	
coding_tree(x0, y0, cbSize, cbSize, 1, 0, cbSubdiv, cqtDepth, 0, 0, 0, DUAL_TREE_LUMA, MODE_TYPE_ALL)	
coding_tree(x0, y0, cbSize, cbSize, 0, 1, cbSubdiv, cqtDepth, 0, 0, 0, DUAL_TREE_CHROMA, MODE_TYPE_ALL)	
}	
}	

7.3.11.3 Sample adaptive offset syntax

sao(rx, ry) {	**Descriptor**
if(rx > 0) {	
leftCtbAvailable = rx != CtbToTileColBd[rx]	
if(leftCtbAvailable)	
sao_merge_left_flag	ae(v)
}	
if(ry > 0 && !sao_merge_left_flag) {	
upCtbAvailable = ry != CtbToTileRowBd[ry] && !FirstCtbRowInSlice	
if(upCtbAvailable)	
sao_merge_up_flag	ae(v)
}	
if(!sao_merge_up_flag && !sao_merge_left_flag)	
for(cIdx = 0; cIdx < (sps_chroma_format_idc != 0 ? 3 : 1); cIdx++)	
if((sh_sao_luma_used_flag && cIdx = = 0) || (sh_sao_chroma_used_flag && cIdx > 0)) {	
if(cIdx = = 0)	

sao_type_idx_luma	ae(v)
else if(cIdx = = 1)	
sao_type_idx_chroma	ae(v)
if(SaoTypeIdx[cIdx][rx][ry] != 0) {	
for(i = 0; i < 4; i++)	
sao_offset_abs[cIdx][rx][ry][i]	ae(v)
if(SaoTypeIdx[cIdx][rx][ry] = = 1) {	
for(i = 0; i < 4; i++)	
if(sao_offset_abs[cIdx][rx][ry][i] != 0)	
sao_offset_sign_flag[cIdx][rx][ry][i]	ae(v)
sao_band_position[cIdx][rx][ry]	ae(v)
} else {	
if(cIdx = = 0)	
sao_eo_class_luma	ae(v)
if(cIdx = = 1)	
sao_eo_class_chroma	ae(v)
}	
}	
}	
}	

7.3.11.4 Coding tree syntax

coding_tree(x0, y0, cbWidth, cbHeight, qgOnY, qgOnC, cbSubdiv, cqtDepth, mttDepth, depthOffset, partIdx, treeTypeCurr, modeTypeCurr) {	**Descriptor**
if((allowSplitBtVer || allowSplitBtHor || allowSplitTtVer || allowSplitTtHor || allowSplitQt) && (x0 + cbWidth <= pps_pic_width_in_luma_samples) && (y0 + cbHeight <= pps_pic_height_in_luma_samples))	
split_cu_flag	ae(v)
if(pps_cu_qp_delta_enabled_flag && qgOnY && cbSubdiv <= CuQpDeltaSubdiv) {	
IsCuQpDeltaCoded = 0	
CuQpDeltaVal = 0	
CuQgTopLeftX = x0	
CuQgTopLeftY = y0	
}	
if(sh_cu_chroma_qp_offset_enabled_flag && qgOnC && cbSubdiv <= CuChromaQpOffsetSubdiv) {	
IsCuChromaQpOffsetCoded = 0	
$\text{CuQpOffset}_{\text{Cb}}$ = 0	
$\text{CuQpOffset}_{\text{Cr}}$ = 0	
$\text{CuQpOffset}_{\text{CbCr}}$ = 0	
}	
if(split_cu_flag) {	
if((allowSplitBtVer || allowSplitBtHor || allowSplitTtVer || allowSplitTtHor) && allowSplitQt)	
split_qt_flag	ae(v)
if(!split_qt_flag) {	
if((allowSplitBtHor || allowSplitTtHor) && (allowSplitBtVer || allowSplitTtVer))	

mtt_split_cu_vertical_flag	ae(v)
if((allowSplitBtVer && allowSplitTtVer && mtt_split_cu_vertical_flag) || (allowSplitBtHor && allowSplitTtHor && !mtt_split_cu_vertical_flag))	
mtt_split_cu_binary_flag	ae(v)
}	
if(ModeTypeCondition = = 1)	
modeType = MODE_TYPE_INTRA	
else if(ModeTypeCondition = = 2) {	
non_inter_flag	ae(v)
modeType = non_inter_flag ? MODE_TYPE_INTRA : MODE_TYPE_INTER	
} else	
modeType = modeTypeCurr	
treeType = (modeType = = MODE_TYPE_INTRA) ? DUAL_TREE_LUMA : treeTypeCurr	
if(!split_qt_flag) {	
if(MttSplitMode[x0][y0][mttDepth] = = SPLIT_BT_VER) {	
depthOffset += (x0 + cbWidth > pps_pic_width_in_luma_samples) ? 1 : 0	
x1 = x0 + (cbWidth / 2)	
coding_tree(x0, y0, cbWidth / 2, cbHeight, qgOnY, qgOnC, cbSubdiv + 1, cqtDepth, mttDepth + 1, depthOffset, 0, treeType, modeType)	
if(x1 < pps_pic_width_in_luma_samples)	
coding_tree(x1, y0, cbWidth / 2, cbHeight, qgOnY, qgOnC, cbSubdiv + 1, cqtDepth, mttDepth + 1, depthOffset, 1, treeType, modeType)	
} else if(MttSplitMode[x0][y0][mttDepth] = = SPLIT_BT_HOR) {	
depthOffset += (y0 + cbHeight > pps_pic_height_in_luma_samples) ? 1 : 0	
y1 = y0 + (cbHeight / 2)	
coding_tree(x0, y0, cbWidth, cbHeight / 2, qgOnY, qgOnC, cbSubdiv + 1, cqtDepth, mttDepth + 1, depthOffset, 0, treeType, modeType)	
if(y1 < pps_pic_height_in_luma_samples)	
coding_tree(x0, y1, cbWidth, cbHeight / 2, qgOnY, qgOnC, cbSubdiv + 1, cqtDepth, mttDepth + 1, depthOffset, 1, treeType, modeType)	
} else if(MttSplitMode[x0][y0][mttDepth] = = SPLIT_TT_VER) {	
x1 = x0 + (cbWidth / 4)	
x2 = x0 + (3 * cbWidth / 4)	
qgNextOnY = qgOnY && (cbSubdiv + 2 <= CuQpDeltaSubdiv)	
qgNextOnC = qgOnC && (cbSubdiv + 2 <= CuChromaQpOffsetSubdiv)	
coding_tree(x0, y0, cbWidth / 4, cbHeight, qgNextOnY, qgNextOnC, cbSubdiv + 2, cqtDepth, mttDepth + 1, depthOffset, 0, treeType, modeType)	
coding_tree(x1, y0, cbWidth / 2, cbHeight, qgNextOnY, qgNextOnC, cbSubdiv + 1, cqtDepth, mttDepth + 1, depthOffset, 1, treeType, modeType)	
coding_tree(x2, y0, cbWidth / 4, cbHeight, qgNextOnY, qgNextOnC, cbSubdiv + 2, cqtDepth, mttDepth + 1, depthOffset, 2, treeType, modeType)	
} else { /* SPLIT_TT_HOR */	
y1 = y0 + (cbHeight / 4)	
y2 = y0 + (3 * cbHeight / 4)	
qgNextOnY = qgOnY && (cbSubdiv + 2 <= CuQpDeltaSubdiv)	
qgNextOnC = qgOnC && (cbSubdiv + 2 <= CuChromaQpOffsetSubdiv)	
coding_tree(x0, y0, cbWidth, cbHeight / 4, qgNextOnY, qgNextOnC, cbSubdiv + 2, cqtDepth, mttDepth + 1, depthOffset, 0, treeType, modeType)	
coding_tree(x0, y1, cbWidth, cbHeight / 2, qgNextOnY, qgNextOnC, cbSubdiv + 1, cqtDepth, mttDepth + 1, depthOffset, 1, treeType, modeType)	

coding_tree(x0, y2, cbWidth, cbHeight / 4, qgNextOnY, qgNextOnC, cbSubdiv + 2, cqtDepth, mttDepth + 1, depthOffset, 2, treeType, modeType)	
}	
} else {	
x1 = x0 + (cbWidth / 2)	
y1 = y0 + (cbHeight / 2)	
coding_tree(x0, y0, cbWidth / 2, cbHeight / 2, qgOnY, qgOnC, cbSubdiv + 2, cqtDepth + 1, 0, 0, 0, treeType, modeType)	
if(x1 < pps_pic_width_in_luma_samples)	
coding_tree(x1, y0, cbWidth / 2, cbHeight / 2, qgOnY, qgOnC, cbSubdiv + 2, cqtDepth + 1, 0, 0, 1, treeType, modeType)	
if(y1 < pps_pic_height_in_luma_samples)	
coding_tree(x0, y1, cbWidth / 2, cbHeight / 2, qgOnY, qgOnC, cbSubdiv + 2, cqtDepth + 1, 0, 0, 2, treeType, modeType)	
if(y1 < pps_pic_height_in_luma_samples && x1 < pps_pic_width_in_luma_samples)	
coding_tree(x1, y1, cbWidth / 2, cbHeight / 2, qgOnY, qgOnC, cbSubdiv + 2, cqtDepth + 1, 0, 0, 3, treeType, modeType)	
}	
if(modeTypeCurr = = MODE_TYPE_ALL && modeType = = MODE_TYPE_INTRA)	
coding_tree(x0, y0, cbWidth, cbHeight, 0, qgOnC, cbSubdiv, cqtDepth, mttDepth, 0, 0, DUAL_TREE_CHROMA, modeType)	
} else	
coding_unit(x0, y0, cbWidth, cbHeight, cqtDepth, treeTypeCurr, modeTypeCurr)	
}	

7.3.11.5 Coding unit syntax

coding_unit(x0, y0, cbWidth, cbHeight, cqtDepth, treeType, modeType) {	**Descriptor**
if(sh_slice_type = = I && (cbWidth > 64 || cbHeight > 64))	
modeType = MODE_TYPE_INTRA	
chType = treeType = = DUAL_TREE_CHROMA ? 1 : 0	
if(sh_slice_type != I || sps_ibc_enabled_flag) {	
if(treeType != DUAL_TREE_CHROMA && ((!(cbWidth = = 4 && cbHeight = = 4) && modeType != MODE_TYPE_INTRA) || (sps_ibc_enabled_flag && cbWidth <= 64 && cbHeight <= 64)))	
cu_skip_flag[x0][y0]	ae(v)
if(cu_skip_flag[x0][y0] = = 0 && sh_slice_type != I && !(cbWidth = = 4 && cbHeight = = 4) && modeType = = MODE_TYPE_ALL)	
pred_mode_flag	ae(v)
if(((sh_slice_type = = I && cu_skip_flag[x0][y0] = =0) || (sh_slice_type != I && (CuPredMode[chType][x0][y0] != MODE_INTRA || (((cbWidth = = 4 && cbHeight = = 4) || modeType = = MODE_TYPE_INTRA) && cu_skip_flag[x0][y0] = = 0)))) && cbWidth <= 64 && cbHeight <= 64 && modeType != MODE_TYPE_INTER && sps_ibc_enabled_flag && treeType != DUAL_TREE_CHROMA)	
pred_mode_ibc_flag	ae(v)
}	

if(CuPredMode[chType][x0][y0] = = MODE_INTRA && sps_palette_enabled_flag && cbWidth <= 64 && cbHeight <= 64 && cu_skip_flag[x0][y0] = = 0 && modeType != MODE_TYPE_INTER && ((cbWidth * cbHeight) > (treeType != DUAL_TREE_CHROMA ? 16 : 16 * SubWidthC * SubHeightC)) && (modeType != MODE_TYPE_INTRA || treeType != DUAL_TREE_CHROMA))	
pred_mode_plt_flag	ae(v)
if(CuPredMode[chType][x0][y0] = = MODE_INTRA && sps_act_enabled_flag && treeType = = SINGLE_TREE)	
cu_act_enabled_flag	ae(v)
if(CuPredMode[chType][x0][y0] = = MODE_INTRA || CuPredMode[chType][x0][y0] = = MODE_PLT) {	
if(treeType = = SINGLE_TREE || treeType = = DUAL_TREE_LUMA) {	
if(pred_mode_plt_flag)	
palette_coding(x0, y0, cbWidth, cbHeight, treeType)	
else {	
if(sps_bdpcm_enabled_flag && cbWidth <= MaxTsSize && cbHeight <= MaxTsSize)	
intra_bdpcm_luma_flag	ae(v)
if(intra_bdpcm_luma_flag)	
intra_bdpcm_luma_dir_flag	ae(v)
else {	
if(sps_mip_enabled_flag)	
intra_mip_flag	ae(v)
if(intra_mip_flag) {	
intra_mip_transposed_flag[x0][y0]	ae(v)
intra_mip_mode[x0][y0]	ae(v)
} else {	
if(sps_mrl_enabled_flag && ((y0 % CtbSizeY) > 0))	
intra_luma_ref_idx	ae(v)
if(sps_isp_enabled_flag && intra_luma_ref_idx = = 0 && (cbWidth <= MaxTbSizeY && cbHeight <= MaxTbSizeY) && (cbWidth * cbHeight > MinTbSizeY * MinTbSizeY) && !cu_act_enabled_flag)	
intra_subpartitions_mode_flag	ae(v)
if(intra_subpartitions_mode_flag = = 1)	
intra_subpartitions_split_flag	ae(v)
if(intra_luma_ref_idx = = 0)	
intra_luma_mpm_flag[x0][y0]	ae(v)
if(intra_luma_mpm_flag[x0][y0]) {	
if(intra_luma_ref_idx = = 0)	
intra_luma_not_planar_flag[x0][y0]	ae(v)
if(intra_luma_not_planar_flag[x0][y0])	
intra_luma_mpm_idx[x0][y0]	ae(v)
} else	
intra_luma_mpm_remainder[x0][y0]	ae(v)
}	
}	
}	
}	

if((treeType = = SINGLE_TREE || treeType = = DUAL_TREE_CHROMA) && sps_chroma_format_idc != 0) {	
if(pred_mode_plt_flag && treeType = = DUAL_TREE_CHROMA)	
palette_coding(x0, y0, cbWidth / SubWidthC, cbHeight / SubHeightC, treeType)	
else if(!pred_mode_plt_flag) {	
if(!cu_act_enabled_flag) {	
if(cbWidth / SubWidthC <= MaxTsSize && cbHeight / SubHeightC <= MaxTsSize && sps_bdpcm_enabled_flag)	
intra_bdpcm_chroma_flag	ae(v)
if(intra_bdpcm_chroma_flag)	
intra_bdpcm_chroma_dir_flag	ae(v)
else {	
if(CclmEnabled)	
cclm_mode_flag	ae(v)
if(cclm_mode_flag)	
cclm_mode_idx	ae(v)
else	
intra_chroma_pred_mode	ae(v)
}	
}	
}	
}	
} else if(treeType != DUAL_TREE_CHROMA) { /* MODE_INTER or MODE_IBC */	
if(cu_skip_flag[x0][y0] = = 0)	
general_merge_flag[x0][y0]	ae(v)
if(general_merge_flag[x0][y0])	
merge_data(x0, y0, cbWidth, cbHeight, chType)	
else if(CuPredMode[chType][x0][y0] = = MODE_IBC) {	
mvd_coding(x0, y0, 0, 0)	
if(MaxNumIbcMergeCand > 1)	
mvp_l0_flag[x0][y0]	ae(v)
if(sps_amvr_enabled_flag && (MvdL0[x0][y0][0] != 0 || MvdL0[x0][y0][1] != 0))	
amvr_precision_idx[x0][y0]	ae(v)
} else {	
if(sh_slice_type = = B)	
inter_pred_idc[x0][y0]	ae(v)
if(sps_affine_enabled_flag && cbWidth >= 16 && cbHeight >= 16) {	
inter_affine_flag[x0][y0]	ae(v)
if(sps_6param_affine_enabled_flag && inter_affine_flag[x0][y0])	
cu_affine_type_flag[x0][y0]	ae(v)
}	
if(sps_smvd_enabled_flag && !ph_mvd_l1_zero_flag && inter_pred_idc[x0][y0] = = PRED_BI && !inter_affine_flag[x0][y0] && RefIdxSymL0 > −1 && RefIdxSymL1 > −1)	
sym_mvd_flag[x0][y0]	ae(v)
if(inter_pred_idc[x0][y0] != PRED_L1) {	
if(NumRefIdxActive[0] > 1 && !sym_mvd_flag[x0][y0])	
ref_idx_l0[x0][y0]	ae(v)

<table>
<tr><td>mvd_coding(x0, y0, 0, 0)</td><td></td></tr>
<tr><td>if(MotionModelIdc[x0][y0] > 0)</td><td></td></tr>
<tr><td>mvd_coding(x0, y0, 0, 1)</td><td></td></tr>
<tr><td>if(MotionModelIdc[x0][y0] > 1)</td><td></td></tr>
<tr><td>mvd_coding(x0, y0, 0, 2)</td><td></td></tr>
<tr><td>mvp_l0_flag[x0][y0]</td><td>ae(v)</td></tr>
<tr><td>} else {</td><td></td></tr>
<tr><td>MvdL0[x0][y0][0] = 0</td><td></td></tr>
<tr><td>MvdL0[x0][y0][1] = 0</td><td></td></tr>
<tr><td>}</td><td></td></tr>
<tr><td>if(inter_pred_idc[x0][y0] != PRED_L0) {</td><td></td></tr>
<tr><td>if(NumRefIdxActive[1] > 1 && !sym_mvd_flag[x0][y0])</td><td></td></tr>
<tr><td>ref_idx_l1[x0][y0]</td><td>ae(v)</td></tr>
<tr><td>if(ph_mvd_l1_zero_flag && inter_pred_idc[x0][y0] = = PRED_BI) {</td><td></td></tr>
<tr><td>MvdL1[x0][y0][0] = 0</td><td></td></tr>
<tr><td>MvdL1[x0][y0][1] = 0</td><td></td></tr>
<tr><td>MvdCpL1[x0][y0][0][0] = 0</td><td></td></tr>
<tr><td>MvdCpL1[x0][y0][0][1] = 0</td><td></td></tr>
<tr><td>MvdCpL1[x0][y0][1][0] = 0</td><td></td></tr>
<tr><td>MvdCpL1[x0][y0][1][1] = 0</td><td></td></tr>
<tr><td>MvdCpL1[x0][y0][2][0] = 0</td><td></td></tr>
<tr><td>MvdCpL1[x0][y0][2][1] = 0</td><td></td></tr>
<tr><td>} else {</td><td></td></tr>
<tr><td>if(sym_mvd_flag[x0][y0]) {</td><td></td></tr>
<tr><td>MvdL1[x0][y0][0] = −MvdL0[x0][y0][0]</td><td></td></tr>
<tr><td>MvdL1[x0][y0][1] = −MvdL0[x0][y0][1]</td><td></td></tr>
<tr><td>} else</td><td></td></tr>
<tr><td>mvd_coding(x0, y0, 1, 0)</td><td></td></tr>
<tr><td>if(MotionModelIdc[x0][y0] > 0)</td><td></td></tr>
<tr><td>mvd_coding(x0, y0, 1, 1)</td><td></td></tr>
<tr><td>if(MotionModelIdc[x0][y0] > 1)</td><td></td></tr>
<tr><td>mvd_coding(x0, y0, 1, 2)</td><td></td></tr>
<tr><td>}</td><td></td></tr>
<tr><td>mvp_l1_flag[x0][y0]</td><td>ae(v)</td></tr>
<tr><td>} else {</td><td></td></tr>
<tr><td>MvdL1[x0][y0][0] = 0</td><td></td></tr>
<tr><td>MvdL1[x0][y0][1] = 0</td><td></td></tr>
<tr><td>}</td><td></td></tr>
<tr><td>if((sps_amvr_enabled_flag && inter_affine_flag[x0][y0] = = 0 &&
(MvdL0[x0][y0][0] != 0 || MvdL0[x0][y0][1] != 0 ||
MvdL1[x0][y0][0] != 0 || MvdL1[x0][y0][1] != 0)) ||
(sps_affine_amvr_enabled_flag && inter_affine_flag[x0][y0] = = 1 &&
(MvdCpL0[x0][y0][0][0] != 0 || MvdCpL0[x0][y0][0][1] != 0 ||
MvdCpL1[x0][y0][0][0] != 0 || MvdCpL1[x0][y0][0][1] != 0 ||
MvdCpL0[x0][y0][1][0] != 0 || MvdCpL0[x0][y0][1][1] != 0 ||
MvdCpL1[x0][y0][1][0] != 0 || MvdCpL1[x0][y0][1][1] != 0 ||
MvdCpL0[x0][y0][2][0] != 0 || MvdCpL0[x0][y0][2][1] != 0 ||
MvdCpL1[x0][y0][2][0] != 0 || MvdCpL1[x0][y0][2][1] != 0))) {</td><td></td></tr>
<tr><td>amvr_flag[x0][y0]</td><td>ae(v)</td></tr>
<tr><td>if(amvr_flag[x0][y0])</td><td></td></tr>
</table>

amvr_precision_idx[x0][y0]	ae(v)
}	
if(sps_bcw_enabled_flag && inter_pred_idc[x0][y0] = = PRED_BI && luma_weight_l0_flag[ref_idx_l0 [x0][y0]] = = 0 && luma_weight_l1_flag[ref_idx_l1 [x0][y0]] = = 0 && chroma_weight_l0_flag[ref_idx_l0 [x0][y0]] = = 0 && chroma_weight_l1_flag[ref_idx_l1 [x0][y0]] = = 0 && cbWidth * cbHeight >= 256)	
bcw_idx[x0][y0]	ae(v)
}	
}	
if(CuPredMode[chType][x0][y0] != MODE_INTRA && !pred_mode_plt_flag && general_merge_flag[x0][y0] = = 0)	
cu_coded_flag	ae(v)
if(cu_coded_flag) {	

if(CuPredMode[chType][x0][y0] = = MODE_INTER &&
sps_sbt_enabled_flag && !ciip_flag[x0][y0] &&
cbWidth <= MaxTbSizeY && cbHeight <= MaxTbSizeY) {	

allowSbtVerH = cbWidth >= 8	
allowSbtVerQ = cbWidth >= 16	
allowSbtHorH = cbHeight >= 8	
allowSbtHorQ = cbHeight >= 16	
if(allowSbtVerH || allowSbtHorH)	
cu_sbt_flag	ae(v)
if(cu_sbt_flag) {	
if((allowSbtVerH || allowSbtHorH) && (allowSbtVerQ || allowSbtHorQ))	
cu_sbt_quad_flag	ae(v)
if((cu_sbt_quad_flag && allowSbtVerQ && allowSbtHorQ) || (!cu_sbt_quad_flag && allowSbtVerH && allowSbtHorH))	
cu_sbt_horizontal_flag	ae(v)
cu_sbt_pos_flag	ae(v)
}	
}	
if(sps_act_enabled_flag && CuPredMode[chType][x0][y0] != MODE_INTRA && treeType = = SINGLE_TREE)	
cu_act_enabled_flag	ae(v)
LfnstDcOnly = 1	
LfnstZeroOutSigCoeffFlag = 1	
MtsDcOnly = 1	
MtsZeroOutSigCoeffFlag = 1	
transform_tree(x0, y0, cbWidth, cbHeight, treeType, chType)	
lfnstWidth = (treeType = = DUAL_TREE_CHROMA) ? cbWidth / SubWidthC : ((IntraSubPartitionsSplitType = = ISP_VER_SPLIT) ? cbWidth / NumIntraSubPartitions : cbWidth)	
lfnstHeight = (treeType = = DUAL_TREE_CHROMA) ? cbHeight / SubHeightC : ((IntraSubPartitionsSplitType = = ISP_HOR_SPLIT) ? cbHeight / NumIntraSubPartitions : cbHeight)	
lfnstNotTsFlag = (treeType = = DUAL_TREE_CHROMA || !tu_y_coded_flag[x0][y0] || transform_skip_flag[x0][y0][0] = = 0) && (treeType = = DUAL_TREE_LUMA || ((!tu_cb_coded_flag[x0][y0] || transform_skip_flag[x0][y0][1] = = 0) && (!tu_cr_coded_flag[x0][y0] || transform_skip_flag[x0][y0][2] = = 0)))	

if(Min(lfnstWidth, lfnstHeight) >= 4 && sps_lfnst_enabled_flag = = 1 &&
CuPredMode[chType][x0][y0] = = MODE_INTRA && lfnstNotTsFlag = = 1
&& (treeType = = DUAL_TREE_CHROMA || !IntraMipFlag[x0][y0] ||
Min(lfnstWidth, lfnstHeight) >= 16) &&
Max(cbWidth, cbHeight) <= MaxTbSizeY) {	

if((IntraSubPartitionsSplitType != ISP_NO_SPLIT || LfnstDcOnly = = 0) && LfnstZeroOutSigCoeffFlag = = 1)	
lfnst_idx	ae(v)
}	

if(treeType != DUAL_TREE_CHROMA && lfnst_idx = = 0 &&
transform_skip_flag[x0][y0][0] = = 0 && Max(cbWidth, cbHeight) <= 32
&& IntraSubPartitionsSplitType = = ISP_NO_SPLIT && cu_sbt_flag = = 0
&& MtsZeroOutSigCoeffFlag = = 1 && MtsDcOnly = = 0) {	

if(((CuPredMode[chType][x0][y0] = = MODE_INTER && sps_explicit_mts_inter_enabled_flag) || (CuPredMode[chType][x0][y0] = = MODE_INTRA && sps_explicit_mts_intra_enabled_flag)))	
mts_idx	ae(v)
}	
}	
}	

7.3.11.6 Palette coding syntax

palette_coding(x0, y0, cbWidth, cbHeight, treeType) {	**Descriptor**
startComp = (treeType = = DUAL_TREE_CHROMA) ? 1 : 0	
numComps = (treeType = = SINGLE_TREE) ? (sps_chroma_format_idc = = 0 ? 1 : 3) : (treeType = = DUAL_TREE_CHROMA) ? 2 : 1	
maxNumPaletteEntries = (treeType = = SINGLE_TREE) ? 31 : 15	
palettePredictionFinished = 0	
NumPredictedPaletteEntries = 0	

for(predictorEntryIdx = 0; predictorEntryIdx < PredictorPaletteSize[startComp] &&
!palettePredictionFinished &&
NumPredictedPaletteEntries < maxNumPaletteEntries; predictorEntryIdx++) {	

palette_predictor_run	ae(v)
if(palette_predictor_run != 1) {	
if(palette_predictor_run > 1)	
predictorEntryIdx += palette_predictor_run − 1	
PalettePredictorEntryReuseFlags[predictorEntryIdx] = 1	
NumPredictedPaletteEntries++	
} else	
palettePredictionFinished = 1	
}	
if(NumPredictedPaletteEntries < maxNumPaletteEntries)	
num_signalled_palette_entries	ae(v)
for(cIdx = startComp; cIdx < (startComp + numComps); cIdx++)	
for(i = 0; i < num_signalled_palette_entries; i++)	
new_palette_entries[cIdx][i]	ae(v)
if(CurrentPaletteSize[startComp] > 0)	
palette_escape_val_present_flag	ae(v)
if(MaxPaletteIndex > 0) {	

adjust = 0	
palette_transpose_flag	ae(v)
}	
if(treeType != DUAL_TREE_CHROMA && palette_escape_val_present_flag)	
if(pps_cu_qp_delta_enabled_flag && !IsCuQpDeltaCoded) {	
cu_qp_delta_abs	ae(v)
if(cu_qp_delta_abs)	
cu_qp_delta_sign_flag	ae(v)
}	
if(treeType != DUAL_TREE_LUMA && palette_escape_val_present_flag)	
if(sh_cu_chroma_qp_offset_enabled_flag && !IsCuChromaQpOffsetCoded) {	
cu_chroma_qp_offset_flag	ae(v)
if(cu_chroma_qp_offset_flag && pps_chroma_qp_offset_list_len_minus1 > 0)	
cu_chroma_qp_offset_idx	ae(v)
}	
PreviousRunPosition = 0	
PreviousRunType = 0	
for(subSetId = 0; subSetId <= (cbWidth * cbHeight − 1) / 16; subSetId++) {	
minSubPos = subSetId * 16	
if(minSubPos + 16 > cbWidth * cbHeight)	
maxSubPos = cbWidth * cbHeight	
else	
maxSubPos = minSubPos + 16	
RunCopyMap[x0][y0] = 0	
PaletteScanPos = minSubPos	
log2CbWidth = Log2(cbWidth)	
log2CbHeight = Log2(cbHeight)	
while(PaletteScanPos < maxSubPos) {	
xC = x0 + TraverseScanOrder[log2CbWidth][log2CbHeight][PaletteScanPos][0]	
yC = y0 + TraverseScanOrder[log2CbWidth][log2CbHeight][PaletteScanPos][1]	
if(PaletteScanPos > 0) {	
xcPrev = x0 + TraverseScanOrder[log2CbWidth][log2CbHeight][PaletteScanPos − 1][0]	
ycPrev = y0 + TraverseScanOrder[log2CbWidth][log2CbHeight][PaletteScanPos − 1][1]	
}	
if(MaxPaletteIndex > 0 && PaletteScanPos > 0) {	
run_copy_flag	ae(v)
RunCopyMap[xC][yC] = run_copy_flag	
}	
CopyAboveIndicesFlag[xC][yC] = 0	
if(MaxPaletteIndex > 0 && !RunCopyMap[xC][yC]) {	

if(((!palette_transpose_flag && yC > y0) || (palette_transpose_flag &&
xC > x0)) && CopyAboveIndicesFlag[xcPrev][ycPrev] = = 0 &&
PaletteScanPos > 0) {	

copy_above_palette_indices_flag	ae(v)
CopyAboveIndicesFlag[xC][yC] = copy_above_palette_indices_flag	
}	
PreviousRunType = CopyAboveIndicesFlag[xC][yC]	

PreviousRunPosition = PaletteScanPos	
} else if(PaletteScanPos > 0)	
CopyAboveIndicesFlag[xC][yC] = CopyAboveIndicesFlag[xcPrev][ycPrev]	
PaletteScanPos ++	
}	
PaletteScanPos = minSubPos	
while(PaletteScanPos < maxSubPos) {	
xC = x0 + TraverseScanOrder[log2CbWidth][log2CbHeight][PaletteScanPos][0]	
yC = y0 + TraverseScanOrder[log2CbWidth][log2CbHeight][PaletteScanPos][1]	
if(PaletteScanPos > 0) {	
xcPrev =x0 + TraverseScanOrder[log2CbWidth][log2CbHeight][PaletteScanPos − 1][0]	
ycPrev = y0 + TraverseScanOrder[log2CbWidth][log2CbHeight][PaletteScanPos − 1][1]	
}	
if(MaxPaletteIndex > 0 && !RunCopyMap[xC][yC] && CopyAboveIndicesFlag[xC][yC] = = 0) {	
if(MaxPaletteIndex − adjust > 0)	
palette_idx_idc	ae(v)
adjust = 1	
}	
if(!RunCopyMap[xC][yC] && CopyAboveIndicesFlag[xC][yC] = = 0)	
CurrPaletteIndex = palette_idx_idc	
if(CopyAboveIndicesFlag[xC][yC] = = 0)	
PaletteIndexMap[xC][yC] = CurrPaletteIndex	
else if(!palette_transpose_flag)	
PaletteIndexMap[xC][yC] = PaletteIndexMap[xC][yC − 1]	
else	
PaletteIndexMap[xC][yC] = PaletteIndexMap[xC − 1][yC]	
PaletteScanPos ++	
}	
if(palette_escape_val_present_flag) {	
for(cIdx = startComp; cIdx < (startComp + numComps); cIdx++) {	
for(sPos = minSubPos; sPos < maxSubPos; sPos++) {	
xC = x0 + TraverseScanOrder[log2CbWidth][log2CbHeight][sPos][0]	
yC = y0 + TraverseScanOrder[log2CbWidth][log2CbHeight][sPos][1]	
if(!(treeType = = SINGLE_TREE && cIdx != 0 && (xC % SubWidthC != 0 || yC % SubHeightC != 0))) {	
if(PaletteIndexMap[xC][yC] = = MaxPaletteIndex) {	
palette_escape_val	ae(v)
PaletteEscapeVal[cIdx][xC][yC] = palette_escape_val	
}	
}	
}	
}	
}	
}	
}	

7.3.11.7 Merge data syntax

merge_data(x0, y0, cbWidth, cbHeight, chType) {	**Descriptor**
if(CuPredMode[chType][x0][y0] = = MODE_IBC) {	
if(MaxNumIbcMergeCand > 1)	
merge_idx[x0][y0]	ae(v)
} else {	
if(MaxNumSubblockMergeCand > 0 && cbWidth >= 8 && cbHeight >= 8)	
merge_subblock_flag[x0][y0]	ae(v)
if(merge_subblock_flag[x0][y0] = = 1) {	
if(MaxNumSubblockMergeCand > 1)	
merge_subblock_idx[x0][y0]	ae(v)
} else {	
if(cbWidth < 128 && cbHeight < 128 && ((sps_ciip_enabled_flag && cu_skip_flag[x0][y0] = = 0 && (cbWidth * cbHeight) >= 64) || (sps_gpm_enabled_flag && sh_slice_type = = B && cbWidth >= 8 && cbHeight >= 8 && cbWidth < (8 * cbHeight) && cbHeight < (8 * cbWidth))))	
regular_merge_flag[x0][y0]	ae(v)
if(regular_merge_flag[x0][y0] = = 1) {	
if(sps_mmvd_enabled_flag)	
mmvd_merge_flag[x0][y0]	ae(v)
if(mmvd_merge_flag[x0][y0] = = 1) {	
if(MaxNumMergeCand > 1)	
mmvd_cand_flag[x0][y0]	ae(v)
mmvd_distance_idx[x0][y0]	ae(v)
mmvd_direction_idx[x0][y0]	ae(v)
} else if(MaxNumMergeCand > 1)	
merge_idx[x0][y0]	ae(v)
} else {	
if(sps_ciip_enabled_flag && sps_gpm_enabled_flag && sh_slice_type = = B && cu_skip_flag[x0][y0] = = 0 && cbWidth >= 8 && cbHeight >= 8 && cbWidth < (8 * cbHeight) && cbHeight < (8 * cbWidth) && cbWidth < 128 && cbHeight < 128)	
ciip_flag[x0][y0]	ae(v)
if(ciip_flag[x0][y0] && MaxNumMergeCand > 1)	
merge_idx[x0][y0]	ae(v)
if(!ciip_flag[x0][y0]) {	
merge_gpm_partition_idx[x0][y0]	ae(v)
merge_gpm_idx0[x0][y0]	ae(v)
if(MaxNumGpmMergeCand > 2)	
merge_gpm_idx1[x0][y0]	ae(v)
}	
}	
}	
}	
}	

7.3.11.8 Motion vector difference syntax

mvd_coding(x0, y0, refList, cpIdx) {	**Descriptor**
abs_mvd_greater0_flag[0]	ae(v)
abs_mvd_greater0_flag[1]	ae(v)
if(abs_mvd_greater0_flag[0])	
abs_mvd_greater1_flag[0]	ae(v)
if(abs_mvd_greater0_flag[1])	
abs_mvd_greater1_flag[1]	ae(v)
if(abs_mvd_greater0_flag[0]) {	
if(abs_mvd_greater1_flag[0])	
abs_mvd_minus2[0]	ae(v)
mvd_sign_flag[0]	ae(v)
}	
if(abs_mvd_greater0_flag[1]) {	
if(abs_mvd_greater1_flag[1])	
abs_mvd_minus2[1]	ae(v)
mvd_sign_flag[1]	ae(v)
}	
}	

7.3.11.9 Transform tree syntax

transform_tree(x0, y0, tbWidth, tbHeight, treeType, chType) {	**Descriptor**
InferTuCbfLuma = 1	
if(IntraSubPartitionsSplitType = = ISP_NO_SPLIT && !cu_sbt_flag) {	
if(tbWidth > MaxTbSizeY || tbHeight > MaxTbSizeY) {	
verSplitFirst = (tbWidth > MaxTbSizeY && tbWidth > tbHeight) ? 1 : 0	
trafoWidth = verSplitFirst ? (tbWidth / 2) : tbWidth	
trafoHeight = !verSplitFirst ? (tbHeight / 2) : tbHeight	
transform_tree(x0, y0, trafoWidth, trafoHeight, treeType, chType)	
if(verSplitFirst)	
transform_tree(x0 + trafoWidth, y0, trafoWidth, trafoHeight, treeType, chType)	
else	
transform_tree(x0, y0 + trafoHeight, trafoWidth, trafoHeight, treeType, chType)	
} else {	
transform_unit(x0, y0, tbWidth, tbHeight, treeType, 0, chType)	
}	
} else if(cu_sbt_flag) {	
if(!cu_sbt_horizontal_flag) {	
trafoWidth = tbWidth * SbtNumFourthsTb0 / 4	
transform_unit(x0, y0, trafoWidth, tbHeight, treeType, 0, 0)	
transform_unit(x0 + trafoWidth, y0, tbWidth − trafoWidth, tbHeight, treeType, 1, 0)	
} else {	
trafoHeight = tbHeight * SbtNumFourthsTb0 / 4	
transform_unit(x0, y0, tbWidth, trafoHeight, treeType, 0, 0)	
transform_unit(x0, y0 + trafoHeight, tbWidth, tbHeight − trafoHeight, treeType, 1, 0)	

}	
} else if(IntraSubPartitionsSplitType = = ISP_HOR_SPLIT) {	
trafoHeight = tbHeight / NumIntraSubPartitions	
for(partIdx = 0; partIdx < NumIntraSubPartitions; partIdx++)	
transform_unit(x0, y0 + trafoHeight * partIdx, tbWidth, trafoHeight, treeType, partIdx, 0)	
} else if(IntraSubPartitionsSplitType = = ISP_VER_SPLIT) {	
trafoWidth = tbWidth / NumIntraSubPartitions	
for(partIdx = 0; partIdx < NumIntraSubPartitions; partIdx++)	
transform_unit(x0 + trafoWidth * partIdx, y0, trafoWidth, tbHeight, treeType, partIdx, 0)	
}	
}	

7.3.11.10 Transform unit syntax

transform_unit(x0, y0, tbWidth, tbHeight, treeType, subTuIndex, chType) {	**Descriptor**
if(IntraSubPartitionsSplitType != ISP_NO_SPLIT && treeType = = SINGLE_TREE && subTuIndex = = NumIntraSubPartitions − 1) {	
xC = CbPosX[chType][x0][y0]	
yC = CbPosY[chType][x0][y0]	
wC = CbWidth[chType][x0][y0] / SubWidthC	
hC = CbHeight[chType][x0][y0] / SubHeightC	
} else {	
xC = x0	
yC = y0	
wC = tbWidth / SubWidthC	
hC = tbHeight / SubHeightC	
}	
chromaAvailable = treeType != DUAL_TREE_LUMA && sps_chroma_format_idc != 0 && (IntraSubPartitionsSplitType = = ISP_NO_SPLIT || (IntraSubPartitionsSplitType != ISP_NO_SPLIT && subTuIndex = = NumIntraSubPartitions − 1))	
if((treeType = = SINGLE_TREE || treeType = = DUAL_TREE_CHROMA) && sps_chroma_format_idc != 0 && ((IntraSubPartitionsSplitType = = ISP_NO_SPLIT && !(cu_sbt_flag && ((subTuIndex = = 0 && cu_sbt_pos_flag) || (subTuIndex = = 1 && !cu_sbt_pos_flag)))) || (IntraSubPartitionsSplitType != ISP_NO_SPLIT && (subTuIndex = = NumIntraSubPartitions − 1)))) {	
tu_cb_coded_flag[xC][yC]	ae(v)
tu_cr_coded_flag[xC][yC]	ae(v)
}	
if(treeType = = SINGLE_TREE || treeType = = DUAL_TREE_LUMA) {	

if((IntraSubPartitionsSplitType = = ISP_NO_SPLIT && !(cu_sbt_flag && ((subTuIndex = = 0 && cu_sbt_pos_flag) || (subTuIndex = = 1 && !cu_sbt_pos_flag))) && ((CuPredMode[chType][x0][y0] = = MODE_INTRA && !cu_act_enabled_flag[x0][y0]) || (chromaAvailable && (tu_cb_coded_flag[xC][yC] || tu_cr_coded_flag[xC][yC])) || CbWidth[chType][x0][y0] > MaxTbSizeY || CbHeight[chType][x0][y0] > MaxTbSizeY)) || (IntraSubPartitionsSplitType != ISP_NO_SPLIT && (subTuIndex < NumIntraSubPartitions − 1 || !InferTuCbfLuma)))	
tu_y_coded_flag[x0][y0]	ae(v)
if(IntraSubPartitionsSplitType != ISP_NO_SPLIT)	
InferTuCbfLuma = InferTuCbfLuma && !tu_y_coded_flag[x0][y0]	
}	

if((CbWidth[chType][x0][y0] > 64 || CbHeight[chType][x0][y0] > 64 ||
tu_y_coded_flag[x0][y0] || (chromaAvailable &&
(tu_cb_coded_flag[xC][yC] || tu_cr_coded_flag[xC][yC]))) &&
treeType != DUAL_TREE_CHROMA &&
pps_cu_qp_delta_enabled_flag && !IsCuQpDeltaCoded) {	

cu_qp_delta_abs	ae(v)
if(cu_qp_delta_abs)	
cu_qp_delta_sign_flag	ae(v)
}	

if((CbWidth[chType][x0][y0] > 64 || CbHeight[chType][x0][y0] > 64 ||
(chromaAvailable && (tu_cb_coded_flag[xC][yC] ||
tu_cr_coded_flag[xC][yC]))) &&
treeType != DUAL_TREE_LUMA && sh_cu_chroma_qp_offset_enabled_flag &&
!IsCuChromaQpOffsetCoded) {	

cu_chroma_qp_offset_flag	ae(v)
if(cu_chroma_qp_offset_flag && pps_chroma_qp_offset_list_len_minus1 > 0)	
cu_chroma_qp_offset_idx	ae(v)
}	
if(sps_joint_cbcr_enabled_flag && ((CuPredMode[chType][x0][y0] = = MODE_INTRA && (tu_cb_coded_flag[xC][yC] || tu_cr_coded_flag[xC][yC])) || (tu_cb_coded_flag[xC][yC] && tu_cr_coded_flag[xC][yC])) && chromaAvailable)	
tu_joint_cbcr_residual_flag[xC][yC]	ae(v)
if(tu_y_coded_flag[x0][y0] && treeType != DUAL_TREE_CHROMA) {	
if(sps_transform_skip_enabled_flag && !BdpcmFlag[x0][y0][0] && tbWidth <= MaxTsSize && tbHeight <= MaxTsSize && (IntraSubPartitionsSplitType = = ISP_NO_SPLIT) && !cu_sbt_flag)	
transform_skip_flag[x0][y0][0]	ae(v)
if(!transform_skip_flag[x0][y0][0] || sh_ts_residual_coding_disabled_flag)	
residual_coding(x0, y0, Log2(tbWidth), Log2(tbHeight), 0)	
else	
residual_ts_coding(x0, y0, Log2(tbWidth), Log2(tbHeight), 0)	
}	
if(tu_cb_coded_flag[xC][yC] && treeType != DUAL_TREE_LUMA) {	
if(sps_transform_skip_enabled_flag && !BdpcmFlag[x0][y0][1] && wC <= MaxTsSize && hC <= MaxTsSize && !cu_sbt_flag)	
transform_skip_flag[xC][yC][1]	ae(v)
if(!transform_skip_flag[xC][yC][1] || sh_ts_residual_coding_disabled_flag)	
residual_coding(xC, yC, Log2(wC), Log2(hC), 1)	
else	

residual_ts_coding(xC, yC, Log2(wC), Log2(hC), 1)	
}	
if(tu_cr_coded_flag[xC][yC] && treeType != DUAL_TREE_LUMA && !(tu_cb_coded_flag[xC][yC] && tu_joint_cbcr_residual_flag[xC][yC])) {	
if(sps_transform_skip_enabled_flag && !BdpcmFlag[x0][y0][2] && wC <= MaxTsSize && hC <= MaxTsSize && !cu_sbt_flag)	
transform_skip_flag[xC][yC][2]	ae(v)
if(!transform_skip_flag[xC][yC][2] || sh_ts_residual_coding_disabled_flag)	
residual_coding(xC, yC, Log2(wC), Log2(hC), 2)	
else	
residual_ts_coding(xC, yC, Log2(wC), Log2(hC), 2)	
}	
}	

7.3.11.11 Residual coding syntax

residual_coding(x0, y0, log2TbWidth, log2TbHeight, cIdx) {	**Descriptor**
if(sps_mts_enabled_flag && cu_sbt_flag && cIdx = = 0 && log2TbWidth = = 5 && log2TbHeight < 6)	
Log2ZoTbWidth = 4	
else	
Log2ZoTbWidth = Min(log2TbWidth, 5)	
if(sps_mts_enabled_flag && cu_sbt_flag && cIdx = = 0 && log2TbWidth < 6 && log2TbHeight = = 5)	
Log2ZoTbHeight = 4	
else	
Log2ZoTbHeight = Min(log2TbHeight, 5)	
Log2FullTbWidth = log2TbWidth	
Log2FullTbHeight = log2TbHeight	
if(log2TbWidth > 0)	
last_sig_coeff_x_prefix	ae(v)
if(log2TbHeight > 0)	
last_sig_coeff_y_prefix	ae(v)
if(last_sig_coeff_x_prefix > 3)	
last_sig_coeff_x_suffix	ae(v)
if(last_sig_coeff_y_prefix > 3)	
last_sig_coeff_y_suffix	ae(v)
remBinsPass1 = ((1 << (Log2ZoTbWidth + Log2ZoTbHeight)) * 7) >> 2	
log2SbW = (Min(Log2ZoTbWidth, Log2ZoTbHeight) < 2 ? 1 : 2)	
log2SbH = log2SbW	
if(Log2ZoTbWidth + Log2ZoTbHeight > 3)	
if(Log2ZoTbWidth < 2) {	
log2SbW = Log2ZoTbWidth	
log2SbH = 4 − log2SbW	
} else if(Log2ZoTbHeight < 2) {	
log2SbH = Log2ZoTbHeight	
log2SbW = 4 − log2SbH	
}	
numSbCoeff = 1 << (log2SbW + log2SbH)	

lastScanPos = numSbCoeff	
lastSubBlock = (1 << (Log2ZoTbWidth + Log2ZoTbHeight − (log2SbW + log2SbH))) − 1	
HistValue = sps_persistent_rice_adaptation_enabled_flag ? (1 << StatCoeff[cIdx]) : 0	
updateHist = sps_persistent_rice_adaptation_enabled_flag ? 1 : 0	
do {	
if(lastScanPos = = 0) {	
lastScanPos = numSbCoeff	
lastSubBlock− −	
}	
lastScanPos− −	
xS = DiagScanOrder[Log2ZoTbWidth − log2SbW][Log2ZoTbHeight − log2SbH] [lastSubBlock][0]	
yS = DiagScanOrder[Log2ZoTbWidth − log2SbW][Log2ZoTbHeight − log2SbH] [lastSubBlock][1]	
xC = (xS << log2SbW) + DiagScanOrder[log2SbW][log2SbH][lastScanPos][0]	
yC = (yS << log2SbH) + DiagScanOrder[log2SbW][log2SbH][lastScanPos][1]	
} while((xC != LastSignificantCoeffX) || (yC != LastSignificantCoeffY))	
if(lastSubBlock = = 0 && Log2ZoTbWidth >= 2 && Log2ZoTbHeight >= 2 && !transform_skip_flag[x0][y0][cIdx] && lastScanPos > 0)	
LfnstDcOnly = 0	
if((lastSubBlock > 0 && Log2ZoTbWidth >= 2 && Log2ZoTbHeight >= 2) || (lastScanPos > 7 && (Log2ZoTbWidth = = 2 || Log2ZoTbWidth = = 3) && Log2ZoTbWidth = = Log2ZoTbHeight))	
LfnstZeroOutSigCoeffFlag = 0	
if((lastSubBlock > 0 || lastScanPos > 0) && cIdx = = 0)	
MtsDcOnly = 0	
QState = 0	
for(i = lastSubBlock; i >= 0; i− −) {	
startQStateSb = QState	
xS = DiagScanOrder[Log2ZoTbWidth − log2SbW][Log2ZoTbHeight − log2SbH] [i][0]	
yS = DiagScanOrder[Log2ZoTbWidth − log2SbW][Log2ZoTbHeight − log2SbH] [i][1]	
inferSbDcSigCoeffFlag = 0	
if(i < lastSubBlock && i > 0) {	
sb_coded_flag[xS][yS]	ae(v)
inferSbDcSigCoeffFlag = 1	
}	
if(sb_coded_flag[xS][yS] && (xS > 3 || yS > 3) && cIdx = = 0)	
MtsZeroOutSigCoeffFlag = 0	
firstSigScanPosSb = numSbCoeff	
lastSigScanPosSb = −1	
firstPosMode0 = (i = = lastSubBlock ? lastScanPos : numSbCoeff − 1)	
firstPosMode1 = firstPosMode0	
for(n = firstPosMode0; n >= 0 && remBinsPass1 >= 4; n− −) {	
xC = (xS << log2SbW) + DiagScanOrder[log2SbW][log2SbH][n][0]	
yC = (yS << log2SbH) + DiagScanOrder[log2SbW][log2SbH][n][1]	
if(sb_coded_flag[xS][yS] && (n > 0 || !inferSbDcSigCoeffFlag) && (xC != LastSignificantCoeffX || yC != LastSignificantCoeffY)) {	

sig_coeff_flag[xC][yC]	ae(v)
remBinsPass1− −	
if(sig_coeff_flag[xC][yC])	
inferSbDcSigCoeffFlag = 0	
}	
if(sig_coeff_flag[xC][yC]) {	
abs_level_gtx_flag[n][0]	ae(v)
remBinsPass1− −	
if(abs_level_gtx_flag[n][0]) {	
par_level_flag[n]	ae(v)
remBinsPass1− −	
abs_level_gtx_flag[n][1]	ae(v)
remBinsPass1− −	
}	
if(lastSigScanPosSb = = −1)	
lastSigScanPosSb = n	
firstSigScanPosSb = n	
}	
AbsLevelPass1[xC][yC] = sig_coeff_flag[xC][yC] + par_level_flag[n] + abs_level_gtx_flag[n][0] + 2 * abs_level_gtx_flag[n][1]	
if(sh_dep_quant_used_flag)	
QState = QStateTransTable[QState][AbsLevelPass1[xC][yC] & 1]	
firstPosMode1 = n − 1	
}	
for(n = firstPosMode0; n > firstPosMode1; n− −) {	
xC = (xS << log2SbW) + DiagScanOrder[log2SbW][log2SbH][n][0]	
yC = (yS << log2SbH) + DiagScanOrder[log2SbW][log2SbH][n][1]	
if(abs_level_gtx_flag[n][1]) {	
abs_remainder[n]	ae(v)
if(updateHist && abs_remainder[n] > 0) {	
StatCoeff[cIdx] = (StatCoeff[cIdx] + Floor(Log2(abs_remainder[n])) + 2) >> 1	
updateHist = 0	
}	
}	
AbsLevel[xC][yC] = AbsLevelPass1[xC][yC] +2 * abs_remainder[n]	
}	
for(n = firstPosMode1; n >= 0; n− −) {	
xC = (xS << log2SbW) + DiagScanOrder[log2SbW][log2SbH][n][0]	
yC = (yS << log2SbH) + DiagScanOrder[log2SbW][log2SbH][n][1]	
if(sb_coded_flag[xS][yS]) {	
dec_abs_level[n]	ae(v)
if(updateHist && dec_abs_level[n] > 0) {	
StatCoeff[cIdx] = (StatCoeff[cIdx] + Floor(Log2(dec_abs_level[n]))) >> 1	
updateHist = 0	
}	
}	
if(AbsLevel[xC][yC] > 0) {	

if(lastSigScanPosSb = = −1)	
lastSigScanPosSb = n	
firstSigScanPosSb = n	
}	
if(sh_dep_quant_used_flag)	
QState = QStateTransTable[QState][AbsLevel[xC][yC] & 1]	
}	
signHiddenFlag = sh_sign_data_hiding_used_flag && (lastSigScanPosSb − firstSigScanPosSb > 3 ? 1 : 0)	
for(n = numSbCoeff − 1; n >= 0; n− −) {	
xC = (xS << log2SbW) + DiagScanOrder[log2SbW][log2SbH][n][0]	
yC = (yS << log2SbH) + DiagScanOrder[log2SbW][log2SbH][n][1]	
if((AbsLevel[xC][yC] > 0) && (!signHiddenFlag || (n != firstSigScanPosSb)))	
coeff_sign_flag[n]	ae(v)
}	
if(sh_dep_quant_used_flag) {	
QState = startQStateSb	
for(n = numSbCoeff − 1; n >= 0; n− −) {	
xC = (xS << log2SbW) + DiagScanOrder[log2SbW][log2SbH][n][0]	
yC = (yS << log2SbH) + DiagScanOrder[log2SbW][log2SbH][n][1]	
if(AbsLevel[xC][yC] > 0)	
TransCoeffLevel[x0][y0][cIdx][xC][yC] = (2 * AbsLevel[xC][yC] − (QState > 1 ? 1 : 0)) * (1 − 2 * coeff_sign_flag[n])	
QState = QStateTransTable[QState][AbsLevel[xC][yC] & 1]	
}	
} else {	
sumAbsLevel = 0	
for(n = numSbCoeff − 1; n >= 0; n− −) {	
xC = (xS << log2SbW) + DiagScanOrder[log2SbW][log2SbH][n][0]	
yC = (yS << log2SbH) + DiagScanOrder[log2SbW][log2SbH][n][1]	
if(AbsLevel[xC][yC] > 0) {	
TransCoeffLevel[x0][y0][cIdx][xC][yC] = AbsLevel[xC][yC] * (1 − 2 * coeff_sign_flag[n])	
if(signHiddenFlag) {	
sumAbsLevel += AbsLevel[xC][yC]	
if(n = = firstSigScanPosSb && sumAbsLevel % 2 = = 1)	
TransCoeffLevel[x0][y0][cIdx][xC][yC] = −TransCoeffLevel[x0][y0][cIdx][xC][yC]	
}	
}	
}	
}	
}	
}	

residual_ts_coding(x0, y0, log2TbWidth, log2TbHeight, cIdx) {	**Descriptor**
log2SbW = (Min(log2TbWidth, log2TbHeight) < 2 ? 1 : 2)	

log2SbH = log2SbW	
if(log2TbWidth + log2TbHeight > 3)	
if(log2TbWidth < 2) {	
log2SbW = log2TbWidth	
log2SbH = 4 − log2SbW	
} else if(log2TbHeight < 2) {	
log2SbH = log2TbHeight	
log2SbW = 4 − log2SbH	
}	
numSbCoeff = 1 << (log2SbW + log2SbH)	
lastSubBlock = (1 << (log2TbWidth + log2TbHeight − (log2SbW + log2SbH))) − 1	
inferSbCbf = 1	
RemCcbs = ((1 << (log2TbWidth + log2TbHeight)) * 7) >> 2	
Log2ZoTbWidth = log2TbWidth	
Log2ZoTbHeight = log2TbHeight	
for(i =0; i <= lastSubBlock; i++) {	
xS = DiagScanOrder[log2TbWidth − log2SbW][log2TbHeight − log2SbH][i][0]	
yS = DiagScanOrder[log2TbWidth − log2SbW][log2TbHeight − log2SbH][i][1]	
if(i != lastSubBlock || !inferSbCbf)	
sb_coded_flag[xS][yS]	ae(v)
if(sb_coded_flag[xS][yS] && i < lastSubBlock)	
inferSbCbf = 0	
/* First scan pass */	
inferSbSigCoeffFlag = 1	
lastScanPosPass1 = −1	
for(n = 0; n <= numSbCoeff − 1 && RemCcbs >= 4; n++) {	
xC = (xS << log2SbW) + DiagScanOrder[log2SbW][log2SbH][n][0]	
yC = (yS << log2SbH) + DiagScanOrder[log2SbW][log2SbH][n][1]	
lastScanPosPass1 = n	
if(sb_coded_flag[xS][yS] && (n != numSbCoeff − 1 || !inferSbSigCoeffFlag)) {	
sig_coeff_flag[xC][yC]	ae(v)
RemCcbs− −	
if(sig_coeff_flag[xC][yC])	
inferSbSigCoeffFlag = 0	
}	
CoeffSignLevel[xC][yC] = 0	
if(sig_coeff_flag[xC][yC]) {	
coeff_sign_flag[n]	ae(v)
RemCcbs− −	
CoeffSignLevel[xC][yC] = (coeff_sign_flag[n] > 0 ? −1 : 1)	
abs_level_gtx_flag[n][0]	ae(v)
RemCcbs− −	
if(abs_level_gtx_flag[n][0]) {	
par_level_flag[n]	ae(v)
RemCcbs− −	
}	
}	

AbsLevelPass1[xC][yC] = sig_coeff_flag[xC][yC] + par_level_flag[n] + abs_level_gtx_flag[n][0]	
}	
/* Greater than X scan pass (numGtXFlags=5) */	
lastScanPosPass2 = −1	
for(n = 0; n <= numSbCoeff − 1 && RemCcbs >= 4; n++) {	
xC = (xS << log2SbW) + DiagScanOrder[log2SbW][log2SbH][n][0]	
yC = (yS << log2SbH) + DiagScanOrder[log2SbW][log2SbH][n][1]	
AbsLevelPass2[xC][yC] = AbsLevelPass1[xC][yC]	
for(j = 1; j < 5; j++) {	
if(abs_level_gtx_flag[n][j − 1]) {	
abs_level_gtx_flag[n][j]	ae(v)
RemCcbs− −	
}	
AbsLevelPass2[xC][yC] += 2 * abs_level_gtx_flag[n][j]	
}	
lastScanPosPass2 = n	
}	
/* remainder scan pass */	
for(n = 0; n <= numSbCoeff − 1; n++) {	
xC = (xS << log2SbW) + DiagScanOrder[log2SbW][log2SbH][n][0]	
yC = (yS << log2SbH) + DiagScanOrder[log2SbW][log2SbH][n][1]	
if((n <= lastScanPosPass2 && AbsLevelPass2[xC][yC] >= 10) || (n > lastScanPosPass2 && n <= lastScanPosPass1 && AbsLevelPass1[xC][yC] >= 2) || (n > lastScanPosPass1 && sb_coded_flag[xS][yS]))	
abs_remainder[n]	ae(v)
if(n <= lastScanPosPass2)	
AbsLevel[xC][yC] = AbsLevelPass2[xC][yC] + 2 * abs_remainder[n]	
else if(n <= lastScanPosPass1)	
AbsLevel[xC][yC] = AbsLevelPass1[xC][yC] + 2 * abs_remainder[n]	
else { /* bypass */	
AbsLevel[xC][yC] = abs_remainder[n]	
if(abs_remainder[n])	
coeff_sign_flag[n]	ae(v)
}	
if(BdpcmFlag[x0][y0][cIdx] = = 0 && n <= lastScanPosPass1) {	
absLeftCoeff = xC > 0 ? AbsLevel[xC − 1][yC] : 0	
absAboveCoeff = yC > 0 ? AbsLevel[xC][yC − 1] : 0	
predCoeff = Max(absLeftCoeff, absAboveCoeff)	
if(AbsLevel[xC][yC] = = 1 && predCoeff > 0)	
AbsLevel[xC][yC] = predCoeff	
else if(AbsLevel[xC][yC] > 0 && AbsLevel[xC][yC] <= predCoeff)	
AbsLevel[xC][yC]− −	
}	
TransCoeffLevel[x0][y0][cIdx][xC][yC] = (1 − 2 * coeff_sign_flag[n]) * AbsLevel[xC][yC]	
}	
}	

}	

7.4 Semantics

7.4.1 General

Semantics associated with the syntax structures and with the syntax elements within these structures are specified in clause 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 Specification.

7.4.2 NAL unit semantics

7.4.2.1 General NAL unit semantics

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 could be specified outside of this Specification.

NOTE 1 – The video coding layer (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 specified in Annex B.

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 a string of data bits (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 first (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) The rbsp_trailing_bits( ) syntax structure is present after the SODB as follows:

i) The first (most significant, left-most) bits of the final RBSP byte contain the remaining bits of the SODB (if any).

ii) The next bit consists of a single bit equal to 1 (i.e., rbsp_stop_one_bit).

iii) When the rbsp_stop_one_bit is not the last bit of a byte-aligned byte, one or more zero-valued bits (i.e., instances of rbsp_alignment_zero_bit) are present to result in byte alignment.

3) One or more rbsp_cabac_zero_word 16-bit syntax elements equal to 0x0000 could 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 are 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 is as specified in Table 5.

NOTE 2 – When the boundaries of the RBSP are known, the decoder could 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.

7.4.2.2 NAL unit header semantics

forbidden_zero_bit shall be equal to 0.

nuh_reserved_zero_bit shall be equal to 0. The value 1 of nuh_reserved_zero_bit could be specified in the future by ITU-T | ISO/IEC. Although the value of nuh_reserved_zero_bit is required to be equal to 0 in this version of this Specification, decoders conforming to this version of this Specification shall also allow the value of nuh_reserved_zero_bit equal to 1 to appear in the syntax and shall ignore (i.e., remove from the bitstream and discard) NAL units with nuh_reserved_zero_bit equal to 1.

nuh_layer_id specifies the identifier of the layer to which a VCL NAL unit belongs or the identifier of a layer to which a non-VCL NAL unit applies. The value of nuh_layer_id shall be in the range of 0 to 55, inclusive. Other values for nuh_layer_id are reserved for future use by ITU-T | ISO/IEC. Although the value of nuh_layer_id is required to be the range of 0 to 55, inclusive, in this version of this Specification, decoders conforming to this version of this Specification shall also allow the value of nuh_layer_id to be greater than 55 to appear in the syntax and shall ignore (i.e., remove from the bitstream and discard) NAL units with nuh_layer_id greater than 55.

The value of nuh_layer_id shall be the same for all VCL NAL units of a coded picture. The value of nuh_layer_id of a coded picture or a PU is the value of the nuh_layer_id of the VCL NAL units of the coded picture or the PU.

When nal_unit_type is equal to PH_NUT, or FD_NUT, nuh_layer_id shall be equal to the nuh_layer_id of associated VCL NAL unit.

When nal_unit_type is equal to EOS_NUT, nuh_layer_id shall be equal to one of the nuh_layer_id values of the layers present in the CVS.

NOTE 1 – The value of nuh_layer_id for DCI, OPI, VPS, AUD, and EOB NAL units is not constrained.

nal_unit_type specifies the NAL unit type, i.e., the type of RBSP data structure contained in the NAL unit as specified in Table 5.

NAL units that have nal_unit_type in the range of UNSPEC_28..UNSPEC_31, inclusive, for which semantics are not specified, shall not affect the decoding process specified in this Specification.

NOTE 2 – NAL unit types in the range of UNSPEC_28..UNSPEC_31 could be used as determined by the application. No decoding process for these values of nal_unit_type is specified in this Specification. Since different applications might use these NAL unit types for different purposes, particular care is expected to be exercised in the design of encoders that generate NAL units with these nal_unit_type values, and in the design of decoders that interpret the content of NAL units with these nal_unit_type values. This Specification does not define any management for these values. These nal_unit_type values might only be suitable for use in contexts in which "collisions" of usage (i.e., different definitions of the meaning of the NAL unit content for the same nal_unit_type value) are unimportant, or not possible, or are managed – e.g., defined or managed in the controlling application or transport specification, or by controlling the environment in which bitstreams are distributed.

For purposes other than determining the amount of data in the DUs of the bitstream (as specified in Annex C), 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 Specification.

Table 5 – NAL unit type codes and NAL unit type classes

nal_unit_type	Name of nal_unit_type	Content of NAL unit and RBSP syntax structure	NAL unit type class
0	TRAIL_NUT	Coded slice of a trailing picture or subpicture* slice_layer_rbsp( )	VCL
1	STSA_NUT	Coded slice of an STSA picture or subpicture* slice_layer_rbsp( )	VCL
2	RADL_NUT	Coded slice of a RADL picture or subpicture* slice_layer_rbsp( )	VCL

Table 5 – NAL unit type codes and NAL unit type classes

nal_unit_type	Name of nal_unit_type	Content of NAL unit and RBSP syntax structure	NAL unit type class
3	RASL_NUT	Coded slice of a RASL picture or subpicture* slice_layer_rbsp( )	VCL
4..6	RSV_VCL_4.. RSV_VCL_6	Reserved non-IRAP VCL NAL unit types	VCL
7 8	IDR_W_RADL IDR_N_LP	Coded slice of an IDR picture or subpicture* slice_layer_rbsp( )	VCL
9	CRA_NUT	Coded slice of a CRA picture or subpicture* slice_layer_rbsp( )	VCL
10	GDR_NUT	Coded slice of a GDR picture or subpicture* slice_layer_rbsp( )	VCL
11	RSV_IRAP_11	Reserved IRAP VCL NAL unit type	VCL
12	OPI_NUT	Operating point information operating_point_information_rbsp( )	non-VCL
13	DCI_NUT	Decoding capability information decoding_capability_information_rbsp( )	non-VCL
14	VPS_NUT	Video parameter set video_parameter_set_rbsp( )	non-VCL
15	SPS_NUT	Sequence parameter set seq_parameter_set_rbsp( )	non-VCL
16	PPS_NUT	Picture parameter set pic_parameter_set_rbsp( )	non-VCL
17 18	PREFIX_APS_NUT SUFFIX_APS_NUT	Adaptation parameter set adaptation_parameter_set_rbsp( )	non-VCL
19	PH_NUT	Picture header picture_header_rbsp( )	non-VCL
20	AUD_NUT	AU delimiter access_unit_delimiter_rbsp( )	non-VCL
21	EOS_NUT	End of sequence end_of_seq_rbsp( )	non-VCL
22	EOB_NUT	End of bitstream end_of_bitstream_rbsp( )	non-VCL
23 24	PREFIX_SEI_NUT SUFFIX_SEI_NUT	Supplemental enhancement information sei_rbsp( )	non-VCL
25	FD_NUT	Filler data filler_data_rbsp( )	non-VCL
26 27	RSV_NVCL_26 RSV_NVCL_27	Reserved non-VCL NAL unit types	non-VCL
28..31	UNSPEC_28.. UNSPEC_31	Unspecified non-VCL NAL unit types	non-VCL
* indicates a property of a picture when pps_mixed_nalu_types_in_pic_flag is equal to 0 and a property of the subpicture when pps_mixed_nalu_types_in_pic_flag is equal to 1.			

NOTE 4 – A clean random access (CRA) picture could have associated RASL or RADL pictures present in the bitstream.

NOTE 5 – An instantaneous decoding refresh (IDR) picture having nal_unit_type equal to IDR_N_LP does not have associated leading pictures present in the bitstream. An IDR picture having nal_unit_type equal to IDR_W_RADL does not have associated RASL pictures present in the bitstream, but could have associated RADL pictures in the bitstream.

The value of nal_unit_type shall be the same for all VCL NAL units of a subpicture. A subpicture is referred to as having the same NAL unit type as the VCL NAL units of the subpicture.

For VCL NAL units of any particular picture, the following applies:

– If pps_mixed_nalu_types_in_pic_flag is equal to 0, the value of nal_unit_type shall be the same for all VCL NAL units of a picture, and a picture or a PU is referred to as having the same NAL unit type as the coded slice NAL units of the picture or PU.

– Otherwise (pps_mixed_nalu_types_in_pic_flag is equal to 1), all of the following constraints apply:

  – The picture shall have at least two subpictures.

  – VCL NAL units of the picture shall have two or more different nal_unit_type values.

  – There shall be no VCL NAL unit of the picture that has nal_unit_type equal to GDR_NUT.

  – When a VCL NAL unit of the picture has nal_unit_type equal to nalUnitTypeA that is equal to IDR_W_RADL, IDR_N_LP, or CRA_NUT, other VCL NAL units of the picture shall all have nal_unit_type equal to nalUnitTypeA or TRAIL_NUT.

The value of nal_unit_type shall be the same for all pictures in an IRAP or GDR AU.

When sps_video_parameter_set_id is greater than 0, vps_max_tid_il_ref_pics_plus1[i][j] is equal to 0 for j equal to GeneralLayerIdx[nuh_layer_id] and any value of i in the range of j + 1 to vps_max_layers_minus1, inclusive, and pps_mixed_nalu_types_in_pic_flag is equal to 1, the value of nal_unit_type shall not be equal to IDR_W_RADL, IDR_N_LP, or CRA_NUT.

It is a requirement of bitstream conformance that the following constraints apply:

– When a picture is a leading picture of an IRAP picture, it shall be a RADL or RASL picture.

– When a subpicture is a leading subpicture of an IRAP subpicture, it shall be a RADL or RASL subpicture.

– When a picture is not a leading picture of an IRAP picture, it shall not be a RADL or RASL picture.

– When a subpicture is not a leading subpicture of an IRAP subpicture, it shall not be a RADL or RASL subpicture.

– No RASL pictures shall be present in the bitstream that are associated with an IDR picture.

– No RASL subpictures shall be present in the bitstream that are associated with an IDR subpicture.

– No RADL pictures shall be present in the bitstream that are associated with an IDR picture having nal_unit_type equal to IDR_N_LP.

  NOTE 6 – It is possible to perform random access at the position of an IRAP AU by discarding all PUs before the IRAP AU (and to correctly decode the non-RASL pictures in the IRAP AU and all the subsequent AUs in decoding order), provided each parameter set is available (either in the bitstream or by external means not specified in this Specification) when it is referenced.

– No RADL subpictures shall be present in the bitstream that are associated with an IDR subpicture having nal_unit_type equal to IDR_N_LP.

– Any picture, with nuh_layer_id equal to a particular value layerId, that precedes an IRAP picture with nuh_layer_id equal to layerId in decoding order shall precede the IRAP picture in output order and shall precede any RADL picture associated with the IRAP picture in output order.

– Any subpicture, with nuh_layer_id equal to a particular value layerId and subpicture index equal to a particular value subpicIdx, that precedes, in decoding order, an IRAP subpicture with nuh_layer_id equal to layerId and subpicture index equal to subpicIdx shall precede, in output order, the IRAP subpicture and all its associated RADL subpictures.

– Any picture, with nuh_layer_id equal to a particular value layerId, that precedes a recovery point picture with nuh_layer_id equal to layerId in decoding order shall precede the recovery point picture in output order.

– Any subpicture, with nuh_layer_id equal to a particular value layerId and subpicture index equal to a particular value subpicIdx, that precedes, in decoding order, a subpicture with nuh_layer_id equal to layerId and subpicture index equal to subpicIdx in a recovery point picture shall precede that subpicture in the recovery point picture in output order.

– Any RASL picture associated with a CRA picture shall precede any RADL picture associated with the CRA picture in output order.

– Any RASL subpicture associated with a CRA subpicture shall precede any RADL subpicture associated with the CRA subpicture in output order.

– Any RASL picture, with nuh_layer_id equal to a particular value layerId, associated with a CRA picture shall follow, in output order, any IRAP or GDR picture with nuh_layer_id equal to layerId that precedes the CRA picture in decoding order.

– Any RASL subpicture, with nuh_layer_id equal to a particular value layerId and subpicture index equal to a particular value subpicIdx, associated with a CRA subpicture shall follow, in output order, any IRAP or GDR subpicture, with nuh_layer_id equal to layerId and subpicture index equal to subpicIdx, that precedes the CRA subpicture in decoding order.

– If sps_field_seq_flag is equal to 0, the following applies: when the current picture, with nuh_layer_id equal to a particular value layerId, is a leading picture associated with an IRAP picture, it shall precede, in decoding order, all non-leading pictures that are associated with the same IRAP picture. Otherwise (sps_field_seq_flag is equal to 1), let picA and picB be the first and the last leading pictures, in decoding order, associated with an IRAP picture, respectively, there shall be at most one non-leading picture with nuh_layer_id equal to layerId preceding picA in decoding order, and there shall be no non-leading picture with nuh_layer_id equal to layerId between picA and picB in decoding order.

– If sps_field_seq_flag is equal to 0, the following applies: when the current subpicture, with nuh_layer_id equal to a particular value layerId and subpicture index equal to a particular value subpicIdx, is a leading subpicture associated with an IRAP subpicture, it shall precede, in decoding order, all non-leading subpictures that are associated with the same IRAP subpicture. Otherwise (sps_field_seq_flag is equal to 1), let subpicA and subpicB be the first and the last leading subpictures, in decoding order, associated with an IRAP subpicture, respectively, there shall be at most one non-leading subpicture with nuh_layer_id equal to layerId and subpicture index equal to subpicIdx preceding subpicA in decoding order, and there shall be no non-leading picture with nuh_layer_id equal to layerId and subpicture index equal to subpicIdx between picA and picB in decoding order.

nuh_temporal_id_plus1 minus 1 specifies a temporal identifier for the NAL unit.

The value of nuh_temporal_id_plus1 shall not be equal to 0.

The variable TemporalId is derived as follows:

$$\text{TemporalId} = \text{nuh\_temporal\_id\_plus1} - 1 \quad (27)$$

When nal_unit_type is in the range of IDR_W_RADL to RSV_IRAP_11, inclusive, TemporalId shall be equal to 0.

When nal_unit_type is equal to STSA_NUT and vps_independent_layer_flag[GeneralLayerIdx[nuh_layer_id]] is equal to 1, TemporalId shall be greater than 0.

The value of TemporalId shall be the same for all VCL NAL units of an AU. The value of TemporalId of a coded picture, a PU, or an AU is the value of the TemporalId of the VCL NAL units of the coded picture, PU, or AU. The value of TemporalId of a sublayer representation is the greatest value of TemporalId of all VCL NAL units in the sublayer representation.

The value of TemporalId for non-VCL NAL units is constrained as follows:

– If nal_unit_type is equal to DCI_NUT, OPI_NUT, VPS_NUT, or SPS_NUT, TemporalId shall be equal to 0 and the TemporalId of the AU containing the NAL unit shall be equal to 0.

– Otherwise, if nal_unit_type is equal to PH_NUT, TemporalId shall be equal to the TemporalId of the PU containing the NAL unit.

– Otherwise, if nal_unit_type is equal to EOS_NUT or EOB_NUT, TemporalId shall be equal to 0.

– Otherwise, if nal_unit_type is equal to AUD_NUT, FD_NUT, PREFIX_SEI_NUT, or SUFFIX_SEI_NUT, TemporalId shall be equal to the TemporalId of the AU containing the NAL unit.

– Otherwise, when nal_unit_type is equal to PPS_NUT, PREFIX_APS_NUT, or SUFFIX_APS_NUT, TemporalId shall be greater than or equal to the TemporalId of the PU containing the NAL unit.

NOTE 7 – When the NAL unit is a non-VCL NAL unit, the value of TemporalId is equal to the minimum value of the TemporalId values of all AUs to which the non-VCL NAL unit applies. When nal_unit_type is equal to PPS_NUT, PREFIX_APS_NUT, or SUFFIX_APS_NUT, TemporalId could be greater than or equal to the TemporalId of the containing AU, as all PPSs and APSs could be included in the beginning of the bitstream (e.g., when they are transported out-of-band, and the receiver places them at the beginning of the bitstream), wherein the first coded picture has TemporalId equal to 0.

7.4.2.3 Encapsulation of an SODB within an RBSP (informative)

This clause does not form an integral part of this Specification.

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 described 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 greater than that of the SODB under some circumstances (using one or more rbsp_cabac_zero_word syntax elements).

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

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

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

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

   '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 rbsp_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 of the form '00000000 00000000 000000xx'.
2. The resulting sequence of bytes is then prefixed with the NAL unit header, within which the nal_unit_type indicates the type of RBSP data structure in the NAL unit.

This procedure results in the construction of the entire content of the NAL unit that follows the NAL unit header.

This process can allow any SODB to be represented in a NAL unit while ensuring both of the following:

– 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.

7.4.2.4 Order of NAL units in the bitstream

7.4.2.4.1 General

This clause 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 clauses 7.3 and D.2 specifies the decoding order of syntax elements. When the VUI parameters or any SEI message specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 is included in a NAL unit specified in this Specification, the syntax of the VUI parameters or the SEI message specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 specifies the decoding order of those syntax elements. Decoders shall be capable of receiving NAL units and their syntax elements in decoding order.

7.4.2.4.2 Order of AUs and their association to CVSs

A bitstream consists of one or more CVSs.

A CVS consists of one or more AUs. The order of PUs and their association to AUs are described in clause 7.4.2.4.3.

The first AU of a CVS is a CVSS AU, wherein each present PU is a CLVSS PU, which is either an IRAP PU with NoOutputBeforeRecoveryFlag equal to 1 or a GDR PU with NoOutputBeforeRecoveryFlag equal to 1.

Each CVSS AU shall have a PU for each of the layers present in the CVS and each picture in an AU in a CVS shall have nuh_layer_id equal to the nuh_layer_id of one of the pictures present in the first AU of the CVS.

7.4.2.4.3 Order of PUs and their association to AUs

An AU consists of one or more PUs in increasing order of nuh_layer_id. The order NAL units and coded pictures and their association to PUs are described in clause 7.4.2.4.4.

There can be at most one AUD NAL unit in an AU. When an AUD NAL unit is present in an AU, it shall be the first NAL unit of the AU, and consequently, it is the first NAL unit of the first PU of the AU.

When vps_max_layers_minus1 is greater than 0, there shall be one and only one AUD NAL unit in each IRAP or GDR AU.

There can be at most one OPI NAL unit in an AU. When an OPI NAL unit is present in an AU, it shall be the first NAL unit following the AUD NAL unit, if any, and otherwise shall be the first NAL unit of the AU.

There can be at most one EOB NAL unit in an AU.

When an EOB NAL unit is present in an AU, it shall be the last NAL unit of the AU, and consequently, it is the last NAL unit of the last PU of the AU.

A VCL NAL unit is the first VCL NAL unit of an AU (and consequently the PU containing the VCL NAL unit is the first PU of the AU) when the VCL NAL unit is the first VCL NAL unit of a picture, determined as specified in clause 7.4.2.4.4, and one or more of the following conditions are true:

– The value of nuh_layer_id of the VCL NAL unit is less than or equal to the nuh_layer_id of the previous picture in decoding order.
– The value of ph_pic_order_cnt_lsb of the VCL NAL unit differs from the ph_pic_order_cnt_lsb of the previous picture in decoding order.
– PicOrderCntVal derived for the VCL NAL unit differs from the PicOrderCntVal of the previous picture in decoding order.

Let firstVclNalUnitInAu be the first VCL NAL unit of an AU. The first of any of the following NAL units preceding firstVclNalUnitInAu and succeeding the last VCL NAL unit preceding firstVclNalUnitInAu, if any, specifies the start of a new AU:

– AUD NAL unit (when present),
– OPI NAL unit (when present),
– DCI NAL unit (when present),
– VPS NAL unit (when present),
– SPS NAL unit (when present),
– PPS NAL unit (when present),
– Prefix APS NAL unit (when present),
– PH NAL unit (when present),
– Prefix SEI NAL unit (when present),
– NAL unit with nal_unit_type equal to RSV_NVCL_26 (when present),
– NAL unit with nal_unit_type in the range of UNSPEC28..UNSPEC29 (when present).

NOTE – The first NAL unit preceding firstVclNalUnitInAu and succeeding the last VCL NAL unit preceding firstVclNalUnitInAu, if any, is one of these types of NAL units.

It is a requirement of bitstream conformance that, when present, the next PU of a particular layer after an EOS NAL unit that belongs to the same layer shall be an IRAP or GDR PU.

7.4.2.4.4 Order of NAL units and coded pictures and their association to PUs

A PU consists of zero or one PH NAL unit, one coded picture, which comprises of one or more VCL NAL units, and zero or more other non-VCL NAL units. The association of VCL NAL units to coded pictures is described in clause 7.4.2.4.5.

When a picture consists of more than one VCL NAL unit, a PH NAL unit shall be present in the PU.

When a VCL NAL unit has sh_picture_header_in_slice_header_flag equal to 1 or is the first VCL NAL unit that follows a PH NAL unit, the VCL NAL unit is the first VCL NAL unit of a picture.

The order of the non-VCL NAL units (other than the AUD, OPI, and EOB NAL units) within a PU shall obey the following constraints:

– When a PH NAL unit is present in a PU, it shall precede the first VCL NAL unit of the PU.
– When any DCI NAL units, VPS NAL units, SPS NAL units, PPS NAL units, prefix SEI NAL units, NAL units with nal_unit_type equal to RSV_NVCL_26, or NAL units with nal_unit_type in the range of UNSPEC_28..UNSPEC_29 are present in a PU, they shall not follow the last VCL NAL unit of the PU.

– When any DCI NAL units, VPS NAL units, SPS NAL units, or PPS NAL units are present in a PU, they shall precede the PH NAL unit (when present) of the PU and shall precede the first VCL NAL unit of the PU.

– NAL units having nal_unit_type equal to SUFFIX_SEI_NUT, FD_NUT, or RSV_NVCL_27, or in the range of UNSPEC_30..UNSPEC_31 in a PU shall not precede the first VCL NAL unit of the PU.

– When any prefix APS NAL units are present in a PU, they shall precede the first VCL NAL unit of the PU.

– When any suffix APS NAL units are present in a PU, they shall follow the last VCL NAL unit of the PU.

– When an EOS NAL unit is present in a PU, it shall be the last NAL unit among all NAL units within the PU other than other EOS NAL units (when present) or an EOB NAL unit (when present).

7.4.2.4.5 Order of VCL NAL units and their association to coded pictures

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

– For any two coded slice NAL units A and B of a coded picture, let subpicIdxA and subpicIdxB be their subpicture level values, and sliceAddrA and sliceAddrB be their sh_slice_address values.

– When either of the following conditions is true, coded slice NAL unit A shall precede coded slice NAL unit B:

  – subpicIdxA is less than subpicIdxB.

  – subpicIdxA is equal to subpicIdxB and sliceAddrA is less than sliceAddrB.

7.4.3 Raw byte sequence payloads, trailing bits and byte alignment semantics

7.4.3.1 Decoding capability information RBSP semantics

A DCI RBSP could be made available to the decoder, through either being present in the bitstream, included in at least the first AU of the bitstream, or provided through external means.

NOTE 1 – The information contained in the DCI RBSP is not necessary for operation of the decoding process specified in clauses 2 through 9 of this Specification.

When present, all DCI NAL units in a bitstream shall have the same content.

dci_reserved_zero_4bits shall be equal to 0 in bitstreams conforming to this version of this Specification. The values greater than 0 for dci_reserved_zero_4bits are reserved for future use by ITU-T | ISO/IEC. Decoders shall also allow values greater than 0 for dci_reserved_zero_4bits to appear in the bitstream and shall ignore the value of dci_reserved_zero_4bits.

dci_num_ptls_minus1 plus 1 specifies the number of profile_tier_level( ) syntax structures in the DCI NAL unit. The value of dci_num_ptls_minus1 shall be in the range of 0 to 14, inclusive. The value 15 for dci_num_ptls_minus1 is reserved for future use by ITU-T | ISO/IEC.

It is a requirement of bitstream conformance that each OLS in a CVS in the bitstream shall conform to at least one of the profile_tier_level( ) syntax structures in the DCI NAL unit.

NOTE 2 – The DCI NAL unit could include PTL information, possibly carried in multiple profile_tier_level( ) syntax structures, that applies collectively to multiple OLSs, and does not need to include PTL information for each of the OLSs individually.

dci_extension_flag equal to 0 specifies that no dci_extension_data_flag syntax elements are present in the DCI RBSP syntax structure. dci_extension_flag equal to 1 specifies that dci_extension_data_flag syntax elements might be present in the DCI RBSP syntax structure. dci_extension_flag shall be equal to 0 in bitstreams conforming to this version of this Specification. However, some use of dci_extension_flag equal to 1 could be specified in some future version of this Specification, and decoders conforming to this version of this Specification shall also allow the value of dci_extension_flag equal to 1 to appear in the syntax.

dci_extension_data_flag could have any value. Its presence and value do not affect the decoding process specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore the dci_extension_data_flag syntax elements.

7.4.3.2 Operating point information RBSP semantics

An OPI NAL unit provides information of the operation point at which the decoder is operating.

All OPI NAL units in a CVS shall have the same content. An OPI NAL unit shall not be present in an AU that does not contain any VCL NAL unit with nal_unit_type in the range of IDR_W_RADL to RSV_IRAP_11, inclusive.

opi_ols_info_present_flag equal to 0 specifies that opi_ols_idx is not present in the OPI NAL unit. opi_ols_info_present_flag equal to 1 specifies that opi_ols_idx is present in the OPI NAL unit.

opi_htid_info_present_flag equal to 0 specifies that opi_htid_plus1 is not present in the OPI NAL unit. opi_htid_info_present_flag equal to 1 specifies that opi_htid_plus1 is present in the OPI NAL unit.

opi_ols_idx specifies that the current CVS and the next CVSs in decoding order up to and not including the next CVS for which opi_ols_idx is provided in an OPI NAL unit do not contain any other layers than those included in the OLS with OLS index equal to opi_ols_idx.

opi_htid_plus1 equal to 0 specifies that all the pictures in the current CVS and the next CVSs in decoding order up to and not including the next CVS for which opi_htid_plus1 is provided in an OPI NAL unit are IRAP pictures or GDR pictures with ph_recovery_poc_cnt equal to 0. opi_htid_plus1 greater than 0 specifies that all the pictures in the current CVS and the next CVSs in decoding order up to and not including the next CVS for which opi_htid_plus1 is provided in an OPI NAL unit have TemporalId less than opi_htid_plus1.

opi_extension_flag equal to 0 specifies that no opi_extension_data_flag syntax elements are present in the OPI RBSP syntax structure. opi_extension_flag equal to 1 specifies that opi_extension_data_flag syntax elements might be present in the OPI RBSP syntax structure. opi_extension_flag shall be equal to 0 in bitstreams conforming to this version of this Specification. However, some use of opi_extension_flag equal to 1 could be specified in some future version of this Specification, and decoders conforming to this version of this Specification shall also allow the value of opi_extension_flag equal to 1 to appear in the syntax.

One or more of opi_htid_info_present_flag, opi_ols_info_present_flag, and opi_extension_flag shall be equal to 1.

opi_extension_data_flag could have any value. Its presence and value do not affect the decoding process specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore the opi_extension_data_flag syntax elements.

7.4.3.3 Video parameter set RBSP semantics

A VPS RBSP shall be available to the decoding process, by inclusion in at least one AU with TemporalId equal to 0 or provided through external means, prior to it being referenced by either of the following:

– a PH NAL unit having a ph_pic_parameter_set_id that refers to a PPS with pps_seq_parameter_set_id equal to the value of sps_seq_parameter_set_id in an SPS NAL unit with sps_video_parameter_set_id equal to the value of vps_video_parameter_set_id in the VPS RBSP,

– a coded slice NAL unit having sh_picture_header_in_slice_header_flag equal to 1 with a ph_pic_parameter_set_id that refers to a PPS with pps_seq_parameter_set_id equal to the value of sps_seq_parameter_set_id in an SPS NAL unit with sps_video_parameter_set_id equal to the value of vps_video_parameter_set_id in the VPS RBSP.

Such a PH NAL unit or coded slice NAL unit references the previous VPS RBSP in decoding order (relative to the position of the PH NAL unit or coded slice NAL unit in decoding order) with that value of vps_video_parameter_set_id.

All VPS NAL units with a particular value of vps_video_parameter_set_id in a CVS shall have the same content.

vps_video_parameter_set_id provides an identifier for the VPS for reference by other syntax elements. The value of vps_video_parameter_set_id shall be greater than 0.

VPS NAL units, regardless of the nuh_layer_id values, share the same value space of vps_video_parameter_set_id.

vps_max_layers_minus1 plus 1 specifies the number of layers specified by the VPS, which is the maximum allowed number of layers in each CVS referring to the VPS.

vps_max_sublayers_minus1 plus 1 specifies the maximum number of temporal sublayers that may be present in a layer specified by the VPS. The value of vps_max_sublayers_minus1 shall be in the range of 0 to 6, inclusive.

vps_default_ptl_dpb_hrd_max_tid_flag equal to 1 specifies that the syntax elements vps_ptl_max_tid[i], vps_dpb_max_tid[i], and vps_hrd_max_tid[i] are not present and are inferred to be equal to the default value vps_max_sublayers_minus1. vps_default_ptl_dpb_hrd_max_tid_flag equal to 0 specifies that the syntax elements vps_ptl_max_tid[i], vps_dpb_max_tid[i], and vps_hrd_max_tid[i] are present. When not present, the value of vps_default_ptl_dpb_hrd_max_tid_flag is inferred to be equal to 1.

vps_all_independent_layers_flag equal to 1 specifies that all layers specified by the VPS are independently coded without using inter-layer prediction. vps_all_independent_layers_flag equal to 0 specifies that one or more of the layers specified by the VPS might use inter-layer prediction. When not present, the value of vps_all_independent_layers_flag is inferred to be equal to 1.

vps_layer_id[i] specifies the nuh_layer_id value of the i-th layer. For any two non-negative integer values of m and n, when m is less than n, the value of vps_layer_id[m] shall be less than vps_layer_id[n].

vps_independent_layer_flag[i] equal to 1 specifies that the layer with index i does not use inter-layer prediction. vps_independent_layer_flag[i] equal to 0 specifies that the layer with index i might use inter-layer prediction and the syntax elements vps_direct_ref_layer_flag[i][j] for j in the range of 0 to i − 1, inclusive, are present in the VPS. When not present, the value of vps_independent_layer_flag[i] is inferred to be equal to 1.

vps_max_tid_ref_present_flag[i] equal to 1 specifies that the syntax element vps_max_tid_il_ref_pics_plus1[i][j] could be present. vps_max_tid_ref_present_flag[i] equal to 0 specifies that the syntax element vps_max_tid_il_ref_pics_plus1[i][j] is not present.

vps_direct_ref_layer_flag[i][j] equal to 0 specifies that the layer with index j is not a direct reference layer for the layer with index i. vps_direct_ref_layer_flag [i][j] equal to 1 specifies that the layer with index j is a direct reference layer for the layer with index i. When vps_direct_ref_layer_flag[i][j] is not present for i and j in the range of 0 to vps_max_layers_minus1, inclusive, it is inferred to be equal to 0. When vps_independent_layer_flag[i] is equal to 0, there shall be at least one value of j in the range of 0 to i − 1, inclusive, such that the value of vps_direct_ref_layer_flag[i][j] is equal to 1.

The variables NumDirectRefLayers[i], DirectRefLayerIdx[i][d], NumRefLayers[i], ReferenceLayerIdx[i][r], and LayerUsedAsRefLayerFlag[j] are derived as follows:

```
for( i = 0; i  <=  vps_max_layers_minus1; i++ ) {
    for( j = 0; j  <=  vps_max_layers_minus1; j++ ) {
        dependencyFlag[ i ][ j ] = vps_direct_ref_layer_flag[ i ][ j ]
        for( k = 0; k < i; k++ )
            if( vps_direct_ref_layer_flag[ i ][ k ]  &&  dependencyFlag[ k ][ j ] )
                dependencyFlag[ i ][ j ] = 1
    }
    LayerUsedAsRefLayerFlag[ i ] = 0
}
for( i = 0; i  <=  vps_max_layers_minus1; i++ ) {
    for( j = 0, d = 0, r = 0; j  <=  vps_max_layers_minus1; j++ ) {                    (28)
        if( vps_direct_ref_layer_flag[ i ][ j ] ) {
            DirectRefLayerIdx[ i ][ d++ ] = j
            LayerUsedAsRefLayerFlag[ j ] = 1
        }
        if( dependencyFlag[ i ][ j ] )
            ReferenceLayerIdx[ i ][ r++ ] = j
    }
    NumDirectRefLayers[ i ] = d
    NumRefLayers[ i ] = r
}
```

The variable GeneralLayerIdx[i], specifying the layer index of the layer with nuh_layer_id equal to vps_layer_id[i], is derived as follows:

```
for( i = 0; i  <=  vps_max_layers_minus1; i++ )                                        (29)
    GeneralLayerIdx[ vps_layer_id[ i ] ] = i
```

For any two different values of i and j, both in the range of 0 to vps_max_layers_minus1, inclusive, when dependencyFlag[i][j] equal to 1, it is a requirement of bitstream conformance that the values of sps_chroma_format_idc and sps_bitdepth_minus8 that apply to the i-th layer shall be equal to the values of sps_chroma_format_idc and sps_bitdepth_minus8, respectively, that apply to the j-th layer.

vps_max_tid_il_ref_pics_plus1[i][j] equal to 0 specifies that the pictures of the j-th layer that are neither IRAP pictures nor GDR pictures with ph_recovery_poc_cnt equal to 0 are not used as ILRPs for decoding of pictures of the i-th layer. vps_max_tid_il_ref_pics_plus1[i][j] greater than 0 specifies that, for decoding pictures of the i-th layer, no picture from the j-th layer with TemporalId greater than vps_max_tid_il_ref_pics_plus1[i][j] − 1 is used as ILRP and no APS with nuh_layer_id equal to vps_layer_id[j] and TemporalId greater than vps_max_tid_il_ref_pics_plus1[i][j] − 1 is referenced. When not present, the value of vps_max_tid_il_ref_pics_plus1[i][j] is inferred to be equal to vps_max_sublayers_minus1 + 1.

vps_each_layer_is_an_ols_flag equal to 1 specifies that each OLS specified by the VPS contains only one layer and each layer specified by the VPS is an OLS with the single included layer being the only output layer. vps_each_layer_is_an_ols_flag equal to 0 specifies that at least one OLS specified by the VPS contains more than one layer. If vps_max_layers_minus1 is equal to 0, the value of vps_each_layer_is_an_ols_flag is inferred to be equal to 1. Otherwise, when vps_all_independent_layers_flag is equal to 0, the value of vps_each_layer_is_an_ols_flag is inferred to be equal to 0.

vps_ols_mode_idc equal to 0 specifies that the total number of OLSs specified by the VPS is equal to vps_max_layers_minus1 + 1, the i-th OLS includes the layers with layer indices from 0 to i, inclusive, and for each OLS only the highest layer in the OLS is an output layer.

vps_ols_mode_idc equal to 1 specifies that the total number of OLSs specified by the VPS is equal to vps_max_layers_minus1 + 1, the i-th OLS includes the layers with layer indices from 0 to i, inclusive, and for each OLS all layers in the OLS are output layers.

vps_ols_mode_idc equal to 2 specifies that the total number of OLSs specified by the VPS is explicitly signalled and for each OLS the output layers are explicitly signalled and other layers are the layers that are direct or indirect reference layers of the output layers of the OLS.

The value of vps_ols_mode_idc shall be in the range of 0 to 2, inclusive. The value 3 of vps_ols_mode_idc is reserved for future use by ITU-T | ISO/IEC. Decoders conforming to this version of this Specification shall ignore the OLSs with vps_ols_mode_idc equal to 3.

When vps_all_independent_layers_flag is equal to 1 and vps_each_layer_is_an_ols_flag is equal to 0, the value of vps_ols_mode_idc is inferred to be equal to 2.

vps_num_output_layer_sets_minus2 plus 2 specifies the total number of OLSs specified by the VPS when vps_ols_mode_idc is equal to 2.

The variable olsModeIdc is derived as follows:

```
if( !vps_each_layer_is_an_ols_flag )
    olsModeIdc = vps_ols_mode_idc                                        (30)
else
    olsModeIdc = 4
```

The variable TotalNumOlss, specifying the total number of OLSs specified by the VPS, is derived as follows:

```
if( olsModeIdc = = 4 | | olsModeIdc = = 0 | | olsModeIdc = = 1 )
    TotalNumOlss = vps_max_layers_minus1 + 1                             (31)
else if( olsModeIdc = = 2 )
    TotalNumOlss = vps_num_output_layer_sets_minus2 + 2
```

vps_ols_output_layer_flag[i][j] equal to 1 specifies that the layer with nuh_layer_id equal to vps_layer_id[j] is an output layer of the i-th OLS when vps_ols_mode_idc is equal to 2. vps_ols_output_layer_flag[i][j] equal to 0 specifies that the layer with nuh_layer_id equal to vps_layer_id[j] is not an output layer of the i-th OLS when vps_ols_mode_idc is equal to 2.

The variable NumOutputLayersInOls[i], specifying the number of output layers in the i-th OLS, the variable NumSubLayersInLayerInOLS[i][j], specifying the number of sublayers in the j-th layer in the i-th OLS, the variable OutputLayerIdInOls[i][j], specifying the nuh_layer_id value of the j-th output layer in the i-th OLS, and the variable LayerUsedAsOutputLayerFlag[k], specifying whether the k-th layer is used as an output layer in at least one OLS, are derived as follows:

```
NumOutputLayersInOls[ 0 ] = 1
OutputLayerIdInOls[ 0 ][ 0 ] = vps_layer_id[ 0 ]
NumSubLayersInLayerInOLS[ 0 ][ 0 ] = vps_ptl_max_tid[ vps_ols_ptl_idx[ 0 ] ] + 1
LayerUsedAsOutputLayerFlag[ 0 ] = 1
for( i = 1; i <= vps_max_layers_minus1; i++ ) {
   if( olsModeIdc = = 4 | | olsModeIdc < 2 )
      LayerUsedAsOutputLayerFlag[ i ] = 1
   else if( vps_ols_mode_idc = = 2 )
      LayerUsedAsOutputLayerFlag[ i ] = 0
}
for( i = 1; i < TotalNumOlss; i++ )
   if( olsModeIdc = = 4 | | olsModeIdc = = 0 ) {
      NumOutputLayersInOls[ i ] = 1
      OutputLayerIdInOls[ i ][ 0 ] = vps_layer_id[ i ]
      if( vps_each_layer_is_an_ols_flag )
         NumSubLayersInLayerInOLS[ i ][ 0 ] = vps_ptl_max_tid[ vps_ols_ptl_idx[ i ] ] + 1
      else {
         NumSubLayersInLayerInOLS[ i ][ i ] = vps_ptl_max_tid[ vps_ols_ptl_idx[ i ] ] + 1
         for( k = i − 1; k >= 0; k− − ) {
            NumSubLayersInLayerInOLS[ i ][ k ] = 0
```

```
                for( m = k + 1; m  <=  i; m++ ) {
                    maxSublayerNeeded = Min( NumSubLayersInLayerInOLS[ i ][ m ],
                            vps_max_tid_il_ref_pics_plus1[ m ][ k ] )
                    if( vps_direct_ref_layer_flag[ m ][ k ]  &&
                            NumSubLayersInLayerInOLS[ i ][ k ] < maxSublayerNeeded )
                        NumSubLayersInLayerInOLS[ i ][ k ] = maxSublayerNeeded
                }
            }
        }
    } else if( vps_ols_mode_idc  = =  1 ) {
        NumOutputLayersInOls[ i ] = i + 1
        for( j = 0; j < NumOutputLayersInOls[ i ]; j++ ) {
            OutputLayerIdInOls[ i ][ j ] = vps_layer_id[ j ]
            NumSubLayersInLayerInOLS[ i ][ j ] = vps_ptl_max_tid[ vps_ols_ptl_idx[ i ] ] + 1
        }
    } else if( vps_ols_mode_idc  = =  2 ) {
        for( j = 0; j  <=  vps_max_layers_minus1; j++ ) {
            layerIncludedInOlsFlag[ i ][ j ] = 0
            NumSubLayersInLayerInOLS[ i ][ j ] = 0
        }
        highestIncludedLayer = 0
        for( k = 0, j = 0; k  <=  vps_max_layers_minus1; k++ )                          (32)
            if( vps_ols_output_layer_flag[ i ][ k ] ) {
                layerIncludedInOlsFlag[ i ][ k ] = 1
                highestIncludedLayer = k
                LayerUsedAsOutputLayerFlag[ k ] = 1
                OutputLayerIdx[ i ][ j ] = k
                OutputLayerIdInOls[ i ][ j++ ] = vps_layer_id[ k ]
                NumSubLayersInLayerInOLS[ i ][ k ] = vps_ptl_max_tid[ vps_ols_ptl_idx[ i ] ] + 1
            }
        NumOutputLayersInOls[ i ] = j
        for( j = 0; j < NumOutputLayersInOls[ i ]; j++ ) {
            idx = OutputLayerIdx[ i ][ j ]
            for( k = 0; k < NumRefLayers[ idx ]; k++ ) {
                if (!layerIncludedInOlsFlag[ i ][ ReferenceLayerIdx[ idx ][ k ] ] )
                layerIncludedInOlsFlag[ i ][ ReferenceLayerIdx[ idx ][ k ] ] = 1
            }
        }
        for( k = highestIncludedLayer − 1; k  >=  0; k− − )
            if( layerIncludedInOlsFlag[ i ][ k ]  &&  !vps_ols_output_layer_flag[ i ][ k ] )
                for( m = k + 1; m  <=  highestIncludedLayer; m++ ) {
                    maxSublayerNeeded = Min( NumSubLayersInLayerInOLS[ i ][ m ],
                            vps_max_tid_il_ref_pics_plus1[ m ][ k ] )
                    if( vps_direct_ref_layer_flag[ m ][ k ]  &&  layerIncludedInOlsFlag[ i ][ m ]  &&
                            NumSubLayersInLayerInOLS[ i ][ k ] < maxSublayerNeeded )
                        NumSubLayersInLayerInOLS[ i ][ k ] = maxSublayerNeeded
                }
    }
```

For each value of i in the range of 0 to vps_max_layers_minus1, inclusive, the values of LayerUsedAsRefLayerFlag[i] and LayerUsedAsOutputLayerFlag[i] shall not both be equal to 0. In other words, there shall be no layer that is neither an output layer of at least one OLS nor a direct reference layer of any other layer.

For each OLS, there shall be at least one layer that is an output layer. In other words, for any value of i in the range of 0 to TotalNumOlss − 1, inclusive, the value of NumOutputLayersInOls[i] shall be greater than or equal to 1.

The variable NumLayersInOls[i], specifying the number of layers in the i-th OLS, the variable LayerIdInOls[i][j], specifying the nuh_layer_id value of the j-th layer in the i-th OLS, the variable NumMultiLayerOlss, specifying the number of multi-layer OLSs (i.e., OLSs that contain more than one layer), and the variable MultiLayerOlsIdx[i], specifying the index to the list of multi-layer OLSs for the i-th OLS when NumLayersInOls[i] is greater than 0, are derived as follows:

```
NumLayersInOls[ 0 ] = 1
LayerIdInOls[ 0 ][ 0 ] = vps_layer_id[ 0 ]
```

```
NumMultiLayerOlss = 0
for( i = 1; i < TotalNumOlss; i++ ) {
    if( vps_each_layer_is_an_ols_flag ) {
        NumLayersInOls[ i ] = 1
        LayerIdInOls[ i ][ 0 ] = vps_layer_id[ i ]                                    (33)
    } else if( vps_ols_mode_idc  = =  0  | |  vps_ols_mode_idc  = =  1 ) {
        NumLayersInOls[ i ] = i + 1
        for( j = 0; j < NumLayersInOls[ i ]; j++ )
            LayerIdInOls[ i ][ j ] = vps_layer_id[ j ]
    } else if( vps_ols_mode_idc  = =  2 ) {
        for( k = 0, j = 0; k  <=  vps_max_layers_minus1; k++ )
            if( layerIncludedInOlsFlag[ i ][ k ] )
                LayerIdInOls[ i ][ j++ ] = vps_layer_id[ k ]
        NumLayersInOls[ i ] = j
    }
    if( NumLayersInOls[ i ] > 1 ) {
        MultiLayerOlsIdx[ i ] = NumMultiLayerOlss
        NumMultiLayerOlss++
    }
}
```

NOTE 1 – The 0-th OLS contains only the lowest layer (i.e., the layer with nuh_layer_id equal to vps_layer_id[0]) and for the 0-th OLS the only included layer is output.

The lowest layer in each OLS shall be an independent layer. In other words, for each i in the range of 0 to TotalNumOlss − 1, inclusive, the value of vps_independent_layer_flag[GeneralLayerIdx[LayerIdInOls[i][0]]] shall be equal to 1.

Each layer shall be included in at least one OLS specified by the VPS. In other words, for each layer with a particular value of nuh_layer_id nuhLayerId equal to one of vps_layer_id[k] for k in the range of 0 to vps_max_layers_minus1, inclusive, there shall be at least one pair of values of i and j, where i is in the range of 0 to TotalNumOlss − 1, inclusive, and j is in the range of NumLayersInOls[i] − 1, inclusive, such that the value of LayerIdInOls[i][j] is equal to nuhLayerId.

vps_num_ptls_minus1 plus 1 specifies the number of profile_tier_level( ) syntax structures in the VPS. The value of vps_num_ptls_minus1 shall be less than TotalNumOlss. When not present, the value of vps_num_ptls_minus1 is inferred to be equal to 0.

vps_pt_present_flag[i] equal to 1 specifies that profile, tier, and general constraints information are present in the i-th profile_tier_level( ) syntax structure in the VPS. vps_pt_present_flag[i] equal to 0 specifies that profile, tier, and general constraints information are not present in the i-th profile_tier_level( ) syntax structure in the VPS. The value of vps_pt_present_flag[0] is inferred to be equal to 1. When vps_pt_present_flag[i] is equal to 0, the profile, tier, and general constraints information for the i-th profile_tier_level( ) syntax structure in the VPS are inferred to be the same as that for the (i − 1)-th profile_tier_level( ) syntax structure in the VPS.

vps_ptl_max_tid[i] specifies the TemporalId of the highest sublayer representation for which the level information is present in the i-th profile_tier_level( ) syntax structure in the VPS and the TemporalId of the highest sublayer representation that is present in the OLSs with OLS index olsIdx such that vps_ols_ptl_idx[olsIdx] is equal to i. The value of vps_ptl_max_tid[i] shall be in the range of 0 to vps_max_sublayers_minus1, inclusive. When vps_default_ptl_dpb_hrd_max_tid_flag is equal to 1, the value of vps_ptl_max_tid[i] is inferred to be equal to vps_max_sublayers_minus1.

vps_ptl_alignment_zero_bit shall be equal to 0.

vps_ols_ptl_idx[i] specifies the index, to the list of profile_tier_level( ) syntax structures in the VPS, of the profile_tier_level( ) syntax structure that applies to the i-th OLS. When present, the value of vps_ols_ptl_idx[i] shall be in the range of 0 to vps_num_ptls_minus1, inclusive.

When not present, the value of vps_ols_ptl_idx[i] is inferred as follows:

– If vps_num_ptls_minus1 is equal to 0, the value of vps_ols_ptl_idx[i] is inferred to be equal to 0.
– Otherwise (vps_num_ptls_minus1 is greater than 0 and vps_num_ptls_minus1 + 1 is equal to TotalNumOlss), the value of vps_ols_ptl_idx[i] is inferred to be equal to i.

When NumLayersInOls[i] is equal to 1, the profile_tier_level( ) syntax structure that applies to the i-th OLS is also present in the SPS referred to by the layer in the i-th OLS. It is a requirement of bitstream conformance that, when NumLayersInOls[i] is equal to 1, the profile_tier_level( ) syntax structures signalled in the VPS and in the SPS for the i-th OLS shall be identical.

Each profile_tier_level( ) syntax structure in the VPS shall be referred to by at least one value of vps_ols_ptl_idx[i] for i in the range of 0 to TotalNumOlss − 1, inclusive.

vps_num_dpb_params_minus1 plus 1, when present, specifies the number of dpb_parameters( ) syntax strutcures in the VPS. The value of vps_num_dpb_params_minus1 shall be in the range of 0 to NumMultiLayerOlss − 1, inclusive.

The variable VpsNumDpbParams, specifying the number of dpb_parameters( ) syntax strutcures in the VPS, is derived as follows:

```
if( vps_each_layer_is_an_ols_flag )
    VpsNumDpbParams = 0                                   (34)
else
    VpsNumDpbParams = vps_num_dpb_params_minus1 + 1
```

vps_sublayer_dpb_params_present_flag is used to control the presence of dpb_max_dec_pic_buffering_minus1[j], dpb_max_num_reorder_pics[j], and dpb_max_latency_increase_plus1[j] syntax elements in the dpb_parameters( ) syntax strucures in the VPS for j in range from 0 to vps_dpb_max_tid[i] − 1, inclusive, when vps_dpb_max_tid[i] is greater than 0. When not present, the value of vps_sub_dpb_params_info_present_flag is inferred to be equal to 0.

vps_dpb_max_tid[i] specifies the TemporalId of the highest sublayer representation for which the DPB parameters could be present in the i-th dpb_parameters( ) syntax strutcure in the VPS. The value of vps_dpb_max_tid[i] shall be in the range of 0 to vps_max_sublayers_minus1, inclusive. When not present, the value of vps_dpb_max_tid[i] is inferred to be equal to vps_max_sublayers_minus1.

The value of vps_dpb_max_tid[vps_ols_dpb_params_idx[m]] shall be greater than or equal to vps_ptl_max_tid[vps_ols_ptl_idx[n]] for each m-th multi-layer OLS for m from 0 to NumMultiLayerOlss − 1, inclusive, and n being the OLS index of the m-th multi-layer OLS among all OLSs.

vps_ols_dpb_pic_width[i] specifies the width, in units of luma samples, of each picture storage buffer for the i-th multi-layer OLS.

vps_ols_dpb_pic_height[i] specifies the height, in units of luma samples, of each picture storage buffer for the i-th multi-layer OLS.

vps_ols_dpb_chroma_format[i] specifies the greatest allowed value of sps_chroma_format_idc for all SPSs that are referred to by CLVSs in the CVS for the i-th multi-layer OLS.

vps_ols_dpb_bitdepth_minus8[i] specifies the greatest allowed value of sps_bitdepth_minus8 for all SPSs that are referred to by CLVSs in the CVS for the i-th multi-layer OLS. The value of vps_ols_dpb_bitdepth_minus8[i] shall be in the range of 0 to 8, inclusive.

NOTE 2 – For decoding the i-th multi-layer OLS, the decoder could safely allocate memory for the DPB according to the values of the syntax elements vps_ols_dpb_pic_width[i], vps_ols_dpb_pic_height[i], vps_ols_dpb_chroma_format[i], and vps_ols_dpb_bitdepth_minus8[i].

vps_ols_dpb_params_idx[i] specifies the index, to the list of dpb_parameters( ) syntax structures in the VPS, of the dpb_parameters( ) syntax structure that applies to the i-th multi-layer OLS. When present, the value of vps_ols_dpb_params_idx[i] shall be in the range of 0 to VpsNumDpbParams − 1, inclusive.

When vps_ols_dpb_params_idx[i] is not present, it is inferred as follows:

– If VpsNumDpbParams is equal to 1, the value of vps_ols_dpb_params_idx[i] to be equal to 0.
– Otherwise (VpsNumDpbParams is greater than 1 and equal to NumMultiLayerOlss), the value of vps_ols_dpb_params_idx[i] is inferred to be equal to i.

For a single-layer OLS, the applicable dpb_parameters( ) syntax structure is present in the SPS referred to by the layer in the OLS.

Each dpb_parameters( ) syntax structure in the VPS shall be referred to by at least one value of vps_ols_dpb_params_idx[i] for i in the range of 0 to NumMultiLayerOlss − 1, inclusive.

vps_timing_hrd_params_present_flag equal to 1 specifies that the VPS contains a general_timing_hrd_parameters( ) syntax structure and other HRD parameters. vps_timing_hrd_params_present_flag equal to 0 specifies that the VPS does not contain a general_timing_hrd_parameters( ) syntax structure or other HRD parameters.

When NumLayersInOls[i] is equal to 1, the general_timing_hrd_parameters( ) syntax structure and the ols_timing_hrd_parameters( ) syntax structure that apply to the i-th OLS are present in the SPS referred to by the layer in the i-th OLS.

vps_sublayer_cpb_params_present_flag equal to 1 specifies that the i-th ols_timing_hrd_parameters( ) syntax structure in the VPS contains HRD parameters for the sublayer representations with TemporalId in the range of 0 to vps_hrd_max_tid[i], inclusive. vps_sublayer_cpb_params_present_flag equal to 0 specifies that the i-th ols_timing_hrd_parameters( ) syntax structure in the VPS contains HRD parameters for the sublayer representation with TemporalId equal to vps_hrd_max_tid[i] only. When vps_max_sublayers_minus1 is equal to 0, the value of vps_sublayer_cpb_params_present_flag is inferred to be equal to 0.

When vps_sublayer_cpb_params_present_flag is equal to 0, the HRD parameters for the sublayer representations with TemporalId in the range of 0 to vps_hrd_max_tid[i] − 1, inclusive, are inferred to be the same as that for the sublayer representation with TemporalId equal to vps_hrd_max_tid[i]. These include the HRD parameters starting from the fixed_pic_rate_general_flag[i] syntax element till the sublayer_hrd_parameters(i) syntax structure immediately under the condition "if(general_vcl_hrd_params_present_flag)" in the ols_timing_hrd_parameters syntax structure.

vps_num_ols_timing_hrd_params_minus1 plus 1 specifies the number of ols_timing_hrd_parameters( ) syntax structures present in the VPS when vps_timing_hrd_params_present_flag is equal to 1. The value of vps_num_ols_timing_hrd_params_minus1 shall be in the range of 0 to NumMultiLayerOlss − 1, inclusive.

vps_hrd_max_tid[i] specifies the TemporalId of the highest sublayer representation for which the HRD parameters are contained in the i-th ols_timing_hrd_parameters( ) syntax structure. The value of vps_hrd_max_tid[i] shall be in the range of 0 to vps_max_sublayers_minus1, inclusive. When not present, the value of vps_hrd_max_tid[i] is inferred to be equal to vps_max_sublayers_minus1.

The value of vps_hrd_max_tid[vps_ols_timing_hrd_idx[m]] shall be greater than or equal to vps_ptl_max_tid[vps_ols_ptl_idx[n]] for each m-th multi-layer OLS for m from 0 to NumMultiLayerOlss − 1, inclusive, and n being the OLS index of the m-th multi-layer OLS among all OLSs.

vps_ols_timing_hrd_idx[i] specifies the index, to the list of ols_timing_hrd_parameters( ) syntax structures in the VPS, of the ols_timing_hrd_parameters( ) syntax structure that applies to the i-th multi-layer OLS. The value of vps_ols_timing_hrd_idx[i] shall be in the range of 0 to vps_num_ols_timing_hrd_params_minus1, inclusive.

When vps_ols_timing_hrd_idx[i] is not present, it is inferred as follows:

– If vps_num_ols_timing_hrd_params_minus1 is equal to 0, the value of vps_ols_timing_hrd_idx[[i] is inferred to be equal to 0.
– Otherwise (vps_num_ols_timing_hrd_params_minus1 + 1 is greater than 1 and equal to NumMultiLayerOlss), the value of vps_ols_timing_hrd_idx[i] is inferred to be equal to i.

For a single-layer OLS, the applicable ols_timing_hrd_parameters( ) syntax structure is present in the SPS referred to by the layer in the OLS.

Each ols_timing_hrd_parameters( ) syntax structure in the VPS shall be referred to by at least one value of vps_ols_timing_hrd_idx[i] for i in the range of 1 to NumMultiLayerOlss − 1, inclusive.

vps_extension_flag equal to 0 specifies that no vps_extension_data_flag syntax elements are present in the VPS RBSP syntax structure. vps_extension_flag equal to 1 specifies that vps_extension_data_flag syntax elements might be present in the VPS RBSP syntax structure. vps_extension_flag shall be equal to 0 in bitstreams conforming to this version of this Specification. However, some use of vps_extension_flag equal to 1 could be specified in some future version of this Specification, and decoders conforming to this version of this Specification shall also allow the value of vps_extension_flag equal to 1 to appear in the syntax.

vps_extension_data_flag could have any value. Its presence and value do not affect the decoding process specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore the vps_extension_data_flag syntax elements.

7.4.3.4 Sequence parameter set RBSP semantics

An SPS RBSP shall be available to the decoding process, by inclusion in at least one AU with TemporalId equal to 0 or provided through external means, prior to it being referenced by either of the following:

– a PH NAL unit having a ph_pic_parameter_set_id that refers to a PPS with pps_seq_parameter_set_id equal to the value of sps_seq_parameter_set_id in the SPS RBSP,
– a coded slice NAL unit having sh_picture_header_in_slice_header_flag equal to 1 with a ph_pic_parameter_set_id that refers to a PPS with pps_seq_parameter_set_id equal to the value of sps_seq_parameter_set_id in the SPS RBSP.

Such a PH NAL unit or coded slice NAL unit references the previous SPS RBSP in decoding order (relative to the position of the PH NAL unit or coded slice NAL unit in decoding order) with that value of sps_seq_parameter_set_id.

All SPS NAL units with a particular value of sps_seq_parameter_set_id in a CVS shall have the same content.

sps_seq_parameter_set_id provides an identifier for the SPS for reference by other syntax elements.

SPS NAL units, regardless of the nuh_layer_id values, share the same value space of sps_seq_parameter_set_id.

Let spsLayerId be the value of the nuh_layer_id of a particular SPS NAL unit, and vclLayerId be the value of the nuh_layer_id of a particular VCL NAL unit. The particular VCL NAL unit shall not refer to the particular SPS NAL unit unless spsLayerId is less than or equal to vclLayerId and all OLSs specified by the VPS that contain the layer with nuh_layer_id equal to vclLayerId also contain the layer with nuh_layer_id equal to spsLayerId.

NOTE 1 – In a CVS that contains only one layer, the nuh_layer_id of referenced SPSs is equal to the nuh_layer_id of the VCL NAL units.

sps_video_parameter_set_id, when greater than 0, specifies the value of vps_video_parameter_set_id for the VPS referred to by the SPS.

When sps_video_parameter_set_id is equal to 0, the following applies:

– The SPS does not refer to a VPS, and no VPS is referred to when decoding each CLVS referring to the SPS.
– The value of vps_max_layers_minus1 is inferred to be equal to 0.
– The CVS shall contain only one layer (i.e., all VCL NAL unit in the CVS shall have the same value of nuh_layer_id).
– The value of GeneralLayerIdx[nuh_layer_id] is set equal to 0.
– The value of vps_independent_layer_flag[GeneralLayerIdx[nuh_layer_id]] is inferred to be equal to 1.
– The value of TotalNumOlss is set equal to 1, the value of NumLayersInOls[0] is set equal to 1, and value of vps_layer_id[0] is inferred to be equal to the value of nuh_layer_id of all the VCL NAL units, and the value of LayerIdInOls[0][0] is set equal to vps_layer_id[0].

  NOTE 2 – When sps_video_parameter_set_id is equal to 0, the phrase "layers specified by the VPS" used in the specification refers to the only present layer that has nuh_layer_id equal to vps_layer_id[0], and the phrase "OLSs specified by the VPS" used in the specification refers to the only present OLS that has OLS index equal to 0 and LayerIdInOls[0][0] equal to vps_layer_id[0].

When vps_independent_layer_flag[GeneralLayerIdx[nuh_layer_id]] is equal to 1, the SPS referred to by a CLVS with a particular nuh_layer_id value nuhLayerId shall have nuh_layer_id equal to nuhLayerId.

The value of sps_video_parameter_set_id shall be the same in all SPSs that are referred to by CLVSs in a CVS.

sps_max_sublayers_minus1 plus 1 specifies the maximum number of temporal sublayers that could be present in each CLVS referring to the SPS.

If sps_video_parameter_set_id is greater than 0, the value of sps_max_sublayers_minus1 shall be in the range of 0 to vps_max_sublayers_minus1, inclusive.

Otherwise (sps_video_parameter_set_id is equal to 0), the following applies:

– The value of sps_max_sublayers_minus1 shall be in the range of 0 to 6, inclusive.
– The value of vps_max_sublayers_minus1 is inferred to be equal to sps_max_sublayers_minus1.
– The value of NumSubLayersInLayerInOLS[0][0] is inferred to be equal to sps_max_sublayers_minus1 + 1.
– The value of vps_ols_ptl_idx[0] is inferred to be equal to 0, and the value of vps_ptl_max_tid[vps_ols_ptl_idx[0]], i.e., vps_ptl_max_tid[0], is inferred to be equal to sps_max_sublayers_minus1.

sps_chroma_format_idc specifies the chroma sampling relative to the luma sampling as specified in clause 6.2.

When sps_video_parameter_set_id is greater than 0 and the SPS is referenced by a layer that is included in the i-th multi-layer OLS specified by the VPS for any i in the range of 0 to NumMultiLayerOlss − 1, inclusive, it is a requirement of bitstream conformance that the value of sps_chroma_format_idc shall be less than or equal to the value of vps_ols_dpb_chroma_format[i].

sps_log2_ctu_size_minus5 plus 5 specifies the luma coding tree block size of each CTU. The value of sps_log2_ctu_size_minus5 shall be in the range of 0 to 2, inclusive. The value 3 for sps_log2_ctu_size_minus5 is reserved for future use by ITU-T | ISO/IEC. Decoders conforming to this version of this Specification shall ignore the CLVSs with sps_log2_ctu_size_minus5 equal to 3.

The variables CtbLog2SizeY and CtbSizeY are derived as follows:

$$\text{CtbLog2SizeY} = \text{sps\_log2\_ctu\_size\_minus5} + 5 \quad (35)$$

$$\text{CtbSizeY} = 1 \ll \text{CtbLog2SizeY} \quad (36)$$

sps_ptl_dpb_hrd_params_present_flag equal to 1 specifies that a profile_tier_level( ) syntax structure and a dpb_parameters( ) syntax structure are present in the SPS, and a general_timing_hrd_parameters( ) syntax structure and an ols_timing_hrd_parameters( ) syntax structure could also be present in the SPS. sps_ptl_dpb_hrd_params_present_flag equal to 0 specifies that none of these four syntax structures is present in the SPS.

When sps_video_parameter_set_id is greater than 0 and there is an OLS that contains only one layer with nuh_layer_id equal to the nuh_layer_id of the SPS, or when sps_video_parameter_set_id is equal to 0, the value of sps_ptl_dpb_hrd_params_present_flag shall be equal to 1.

sps_gdr_enabled_flag equal to 1 specifies that GDR pictures are enabled and could be present in the CLVS. sps_gdr_enabled_flag equal to 0 specifies that GDR pictures are disabled and not present in the CLVS.

sps_ref_pic_resampling_enabled_flag equal to 1 specifies that reference picture resampling is enabled and a current picture referring to the SPS might have slices that refer to a reference picture in an active entry of an RPL that has one or more of the following seven parameters different than that of the current picture: 1) pps_pic_width_in_luma_samples, 2) pps_pic_height_in_luma_samples, 3) pps_scaling_win_left_offset, 4) pps_scaling_win_right_offset, 5) pps_scaling_win_top_offset, 6) pps_scaling_win_bottom_offset, and 7) sps_num_subpics_minus1. sps_ref_pic_resampling_enabled_flag equal to 0 specifies that reference picture resampling is disabled and no current picture referring to the SPS has slices that refer to a reference picture in an active entry of an RPL that has one or more of these seven parameters different than that of the current picture.

NOTE 3 – When sps_ref_pic_resampling_enabled_flag is equal to 1, for a current picture the reference picture that has one or more of these seven parameters different than that of the current picture could either belong to the same layer or a different layer than the layer containing the current picture.

sps_res_change_in_clvs_allowed_flag equal to 1 specifies that the picture spatial resolution might change within a CLVS referring to the SPS. sps_res_change_in_clvs_allowed_flag equal to 0 specifies that the picture spatial resolution does not change within any CLVS referring to the SPS. When not present, the value of sps_res_change_in_clvs_allowed_flag is inferred to be equal to 0.

sps_pic_width_max_in_luma_samples specifies the maximum width, in units of luma samples, of each decoded picture referring to the SPS. sps_pic_width_max_in_luma_samples shall not be equal to 0 and shall be an integer multiple of Max(8, MinCbSizeY).

When sps_video_parameter_set_id is greater than 0 and the SPS is referenced by a layer that is included in the i-th multi-layer OLS specified by the VPS for any i in the range of 0 to NumMultiLayerOlss − 1, inclusive, it is a requirement of bitstream conformance that the value of sps_pic_width_max_in_luma_samples shall be less than or equal to the value of vps_ols_dpb_pic_width[i].

sps_pic_height_max_in_luma_samples specifies the maximum height, in units of luma samples, of each decoded picture referring to the SPS. sps_pic_height_max_in_luma_samples shall not be equal to 0 and shall be an integer multiple of Max(8, MinCbSizeY).

When sps_video_parameter_set_id is greater than 0 and the SPS is referenced by a layer that is included in the i-th multi-layer OLS specified by the VPS for any i in the range of 0 to NumMultiLayerOlss − 1, inclusive, it is a requirement of bitstream conformance that the value of sps_pic_height_max_in_luma_samples shall be less than or equal to the value of vps_ols_dpb_pic_height[i].

sps_conformance_window_flag equal to 1 indicates that the conformance cropping window offset parameters follow next in the SPS. sps_conformance_window_flag equal to 0 indicates that the conformance cropping window offset parameters are not present in the SPS.

sps_conf_win_left_offset, **sps_conf_win_right_offset**, **sps_conf_win_top_offset**, and **sps_conf_win_bottom_offset** specify the cropping window that is applied to pictures with pps_pic_width_in_luma_samples equal to sps_pic_width_max_in_luma_samples and pps_pic_height_in_luma_samples equal to sps_pic_height_max_in_luma_samples. When sps_conformance_window_flag is equal to 0, the values of sps_conf_win_left_offset, sps_conf_win_right_offset, sps_conf_win_top_offset, and sps_conf_win_bottom_offset are inferred to be equal to 0.

The conformance cropping window contains the luma samples with horizontal picture coordinates from SubWidthC * sps_conf_win_left_offset to sps_pic_width_max_in_luma_samples − (SubWidthC * sps_conf_win_right_offset + 1) and vertical picture coordinates from SubHeightC * sps_conf_win_top_offset to sps_pic_height_max_in_luma_samples − (SubHeightC * sps_conf_win_bottom_offset + 1), inclusive.

The value of SubWidthC * (sps_conf_win_left_offset + sps_conf_win_right_offset) shall be less than sps_pic_width_max_in_luma_samples, and the value of SubHeightC * (sps_conf_win_top_offset + sps_conf_win_bottom_offset) shall be less than sps_pic_height_max_in_luma_samples.

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

NOTE 4 – The conformance cropping window offset parameters are only applied at the output. All internal decoding processes are applied to the uncropped picture size.

sps_subpic_info_present_flag equal to 1 specifies that subpicture information is present for the CLVS and there might be one or more than one subpicture in each picture of the CLVS. sps_subpic_info_present_flag equal to 0 specifies that subpicture information is not present for the CLVS and there is only one subpicture in each picture of the CLVS.

When sps_res_change_in_clvs_allowed_flag is equal to 1, the value of sps_subpic_info_present_flag shall be equal to 0.

NOTE 5 – When a bitstream is the result of a subpicture sub-bitstream extraction process and contains only a subset of the subpictures of the input bitstream to the subpicture sub-bitstream extraction process, it might be required to set the value of sps_subpic_info_present_flag equal to 1 in the RBSP of the SPSs.

sps_num_subpics_minus1 plus 1 specifies the number of subpictures in each picture in the CLVS. The value of sps_num_subpics_minus1 shall be in the range of 0 to MaxSlicesPerAu − 1, inclusive, where MaxSlicesPerAu is specified in Annex A. When not present, the value of sps_num_subpics_minus1 is inferred to be equal to 0.

sps_independent_subpics_flag equal to 1 specifies that all subpicture boundaries in the CLVS are treated as picture boundaries and there is no loop filtering across the subpicture boundaries. sps_independent_subpics_flag equal to 0 does not impose such a constraint. When not present, the value of sps_independent_subpics_flag is inferred to be equal to 1.

sps_subpic_same_size_flag equal to 1 specifies that all subpictures in the CLVS have the same width specified by sps_subpic_width_minus1[0] and the same height specified by sps_subpic_height_minus1[0]. sps_subpic_same_size_flag equal to 0 does not impose such a constraint. When not present, the value of sps_subpic_same_size_flag is inferred to be equal to 0.

Let the variable tmpWidthVal be set equal to (sps_pic_width_max_in_luma_samples + CtbSizeY − 1) / CtbSizeY, and the variable tmpHeightVal be set equal to (sps_pic_height_max_in_luma_samples + CtbSizeY − 1) / CtbSizeY.

sps_subpic_ctu_top_left_x[i] specifies horizontal position of top-left CTU of i-th subpicture in unit of CtbSizeY. The length of the syntax element is Ceil(Log2(tmpWidthVal)) bits.

When not present, the value of sps_subpic_ctu_top_left_x[i] is inferred as follows:

– If sps_subpic_same_size_flag is equal to 0 or i is equal to 0, the value of sps_subpic_ctu_top_left_x[i] is inferred to be equal to 0.

– Otherwise, the value of sps_subpic_ctu_top_left_x[i] is inferred to be equal to (i % numSubpicCols) * (sps_subpic_width_minus1[0] + 1).

When sps_subpic_same_size_flag is equal to 1, the variable numSubpicCols, specifying the number of subpicture columns in each picture in the CLVS, is derived as follows:

$$\text{numSubpicCols} = \text{tmpWidthVal} \,/\, (\ \text{sps\_subpic\_width\_minus1}[\ 0\ ] + 1\ ) \quad (37)$$

When sps_subpic_same_size_flag is equal to 1, the value of numSubpicCols * tmpHeightVal / (sps_subpic_height_minus1[0] + 1) − 1 shall be equal to sps_num_subpics_minus1.

sps_subpic_ctu_top_left_y[i] specifies vertical position of top-left CTU of i-th subpicture in unit of CtbSizeY. The length of the syntax element is Ceil(Log2(tmpHeightVal)) bits.

When not present, the value of sps_subpic_ctu_top_left_y[i] is inferred as follows:

– If sps_subpic_same_size_flag is equal to 0 or i is equal to 0, the value of sps_subpic_ctu_top_left_y[i] is inferred to be equal to 0.

– Otherwise, the value of sps_subpic_ctu_top_left_y[i] is inferred to be equal to (i / numSubpicCols)* (sps_subpic_height_minus1[0] + 1).

sps_subpic_width_minus1[i] plus 1 specifies the width of the i-th subpicture in units of CtbSizeY. The length of the syntax element is Ceil(Log2(tmpWidthVal)) bits.

When not present, the value of sps_subpic_width_minus1[i] is inferred as follows:

– If sps_subpic_same_size_flag is equal to 0 or i is equal to 0, the value of sps_subpic_width_minus1[i] is inferred to be equal to tmpWidthVal − sps_subpic_ctu_top_left_x[i] − 1.

– Otherwise, the value of sps_subpic_width_minus1[i] is inferred to be equal to sps_subpic_width_minus1[0].

When sps_subpic_same_size_flag is equal to 1, the value of tmpWidthVal % (sps_subpic_width_minus1[0] + 1) shall be equal to 0.

sps_subpic_height_minus1[i] plus 1 specifies the height of the i-th subpicture in units of CtbSizeY. The length of the syntax element is Ceil(Log2(tmpHeightVal)) bits.

When not present, the value of sps_subpic_height_minus1[i] is inferred as follows:

– If sps_subpic_same_size_flag is equal to 0 or i is equal to 0, the value of sps_subpic_height_minus1[i] is inferred to be equal to tmpHeightVal − sps_subpic_ctu_top_left_y[i] − 1.
– Otherwise, the value of sps_subpic_height_minus1[i] is inferred to be equal to sps_subpic_height_minus1[0].

When sps_subpic_same_size_flag is equal to 1, the value of tmpHeightVal % (sps_subpic_height_minus1[0] + 1) shall be equal to 0.

It is a requirement of bitstream conformance that the shapes of the subpictures shall be such that each subpicture, when decoded, shall have its entire left boundary and entire top boundary consisting of picture boundaries or consisting of boundaries of previously decoded subpictures.

For each subpicture with subpicture index i in the range of 0 to sps_num_subpics_minus1, inclusive, it is a requirement of bitstream conformance that all of the following conditions are true:

– The value of (sps_subpic_ctu_top_left_x[i] * CtbSizeY) shall be less than (sps_pic_width_max_in_luma_samples − sps_conf_win_right_offset * SubWidthC).
– The value of ((sps_subpic_ctu_top_left_x[i] + sps_subpic_width_minus1[i] + 1) * CtbSizeY) shall be greater than (sps_conf_win_left_offset * SubWidthC).
– The value of (sps_subpic_ctu_top_left_y[i] * CtbSizeY) shall be less than (sps_pic_height_max_in_luma_samples − sps_conf_win_bottom_offset * SubHeightC).
– The value of ((sps_subpic_ctu_top_left_y[i] + sps_subpic_height_minus1[i] + 1) * CtbSizeY) shall be greater than (sps_conf_win_top_offset * SubHeightC).

sps_subpic_treated_as_pic_flag[i] equal to 1 specifies that the i-th subpicture of each coded picture in the CLVS is treated as a picture in the decoding process excluding in-loop filtering operations. sps_subpic_treated_as_pic_flag[i] equal to 0 specifies that the i-th subpicture of each coded picture in the CLVS is not treated as a picture in the decoding process excluding in-loop filtering operations. When not present, the value of sps_subpic_treated_as_pic_flag[i] is inferred to be equal to 1.

sps_loop_filter_across_subpic_enabled_flag[i] equal to 1 specifies that in-loop filtering operations across subpicture boundaries is enabled and might be performed across the boundaries of the i-th subpicture in each coded picture in the CLVS. sps_loop_filter_across_subpic_enabled_flag[i] equal to 0 specifies that in-loop filtering operations across subpicture boundaries is disabled and are not performed across the boundaries of the i-th subpicture in each coded picture in the CLVS. When not present, the value of sps_loop_filter_across_subpic_enabled_pic_flag[i] is inferred to be equal to 0.

sps_subpic_id_len_minus1 plus 1 specifies the number of bits used to represent the syntax element sps_subpic_id[i], the syntax elements pps_subpic_id[i], when present, and the syntax element sh_subpic_id, when present. The value of sps_subpic_id_len_minus1 shall be in the range of 0 to 15, inclusive. The value of 1 << (sps_subpic_id_len_minus1 + 1) shall be greater than or equal to sps_num_subpics_minus1 + 1.

sps_subpic_id_mapping_explicitly_signalled_flag equal to 1 specifies that the subpicture ID mapping is explicitly signalled, either in the SPS or in the PPSs referred to by coded pictures of the CLVS. sps_subpic_id_mapping_explicitly_signalled_flag equal to 0 specifies that the subpicture ID mapping is not explicitly signalled for the CLVS. When not present, the value of sps_subpic_id_mapping_explicitly_signalled_flag is inferred to be equal to 0.

sps_subpic_id_mapping_present_flag equal to 1 specifies that the subpicture ID mapping is signalled in the SPS when sps_subpic_id_mapping_explicitly_signalled_flag is equal to 1. sps_subpic_id_mapping_present_flag equal to 0 specifies that subpicture ID mapping is signalled in the PPSs referred to by coded pictures of the CLVS when sps_subpic_id_mapping_explicitly_signalled_flag is equal to 1.

sps_subpic_id[i] specifies the subpicture ID of the i-th subpicture. The length of the sps_subpic_id[i] syntax element is sps_subpic_id_len_minus1 + 1 bits.

sps_bitdepth_minus8 specifies the bit depth of the samples of the luma and chroma arrays, BitDepth, and the value of the luma and chroma quantization parameter range offset, QpBdOffset, as follows:

$$\text{BitDepth} = 8 + \text{sps\_bitdepth\_minus8} \quad (38)$$

$$\text{QpBdOffset} = 6 * \text{sps\_bitdepth\_minus8} \quad (39)$$

sps_bitdepth_minus8 shall be in the range of 0 to 8, inclusive.

When sps_video_parameter_set_id is greater than 0 and the SPS is referenced by a layer that is included in the i-th multi-layer OLS specified by the VPS for any i in the range of 0 to NumMultiLayerOlss − 1, inclusive, it is a requirement of bitstream conformance that the value of sps_bitdepth_minus8 shall be less than or equal to the value of vps_ols_dpb_bitdepth_minus8[i].

sps_entropy_coding_sync_enabled_flag equal to 1 specifies that a specific synchronization process for context variables is invoked before decoding the CTU that includes the first CTB of a row of CTBs in each tile in each picture referring to the SPS, and a specific storage process for context variables is invoked after decoding the CTU that includes the first CTB of a row of CTBs in each tile in each picture referring to the SPS. sps_entropy_coding_sync_enabled_flag equal to 0 specifies that no specific synchronization process for context variables is required to be invoked before decoding the CTU that includes the first CTB of a row of CTBs in each tile in each picture referring to the SPS, and no specific storage process for context variables is required to be invoked after decoding the CTU that includes the first CTB of a row of CTBs in each tile in each picture referring to the SPS.

NOTE 6 – When sps_entropy_coding_sync_enabled_flag is equal to 1, the so-called wavefront parallel processing (WPP) is enabled.

sps_entry_point_offsets_present_flag equal to 1 specifies that signalling for entry point offsets for tiles or tile-specific CTU rows could be present in the slice headers of pictures referring to the SPS. sps_entry_point_offsets_present_flag equal to 0 specifies that signalling for entry point offsets for tiles or tile-specific CTU rows are not present in the slice headers of pictures referring to the SPS.

sps_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 follows:

$$\text{MaxPicOrderCntLsb} = 2^{(\text{ sps\_log2\_max\_pic\_order\_cnt\_lsb\_minus4} + 4)} \quad (40)$$

The value of sps_log2_max_pic_order_cnt_lsb_minus4 shall be in the range of 0 to 12, inclusive.

sps_poc_msb_cycle_flag equal to 1 specifies that the ph_poc_msb_cycle_present_flag syntax element is present in PH syntax structures referring to the SPS. sps_poc_msb_cycle_flag equal to 0 specifies that the ph_poc_msb_cycle_present_flag syntax element is not present in PH syntax structures referring to the SPS.

sps_poc_msb_cycle_len_minus1 plus 1 specifies the length, in bits, of the ph_poc_msb_cycle_val syntax elements, when present in PH syntax structures referring to the SPS. The value of sps_poc_msb_cycle_len_minus1 shall be in the range of 0 to 32 − sps_log2_max_pic_order_cnt_lsb_minus4 − 5, inclusive.

sps_num_extra_ph_bytes specifies the number of bytes of extra bits in the PH syntax structure for coded pictures referring to the SPS. The value of sps_num_extra_ph_bytes shall be equal to 0 in bitstreams conforming to this version of this Specification. Although the value of sps_num_extra_ph_bytes is required to be equal to 0 in this version of this Specification, decoders conforming to this version of this Specification shall also allow the value of sps_num_extra_ph_bytes equal to 1 or 2 to appear in the syntax.

sps_extra_ph_bit_present_flag[i] equal to 1 specifies that the i-th extra bit is present in PH syntax structures referring to the SPS. sps_extra_ph_bit_present_flag[i] equal to 0 specifies that the i-th extra bit is not present in PH syntax structures referring to the SPS.

The variable NumExtraPhBits is derived as follows:

```
NumExtraPhBits = 0
for( i = 0; i < ( sps_num_extra_ph_bytes * 8 ); i++ )
    if( sps_extra_ph_bit_present_flag[ i ] )                    (41)
        NumExtraPhBits++
```

sps_num_extra_sh_bytes specifies the number of bytes of extra bits in the slice headers for coded pictures referring to the SPS. The value of sps_num_extra_sh_bytes shall be equal to 0 in bitstreams conforming to this version of this Specification. Although the value of sps_num_extra_sh_bytes is required to be equal to 0 in this version of this Specification, decoders conforming to this version of this Specification shall also allow the value of sps_num_extra_sh_bytes equal to 1 or 2 to appear in the syntax.

sps_extra_sh_bit_present_flag[i] equal to 1 specifies that the i-th extra bit is present in the slice headers of pictures referring to the SPS. sps_extra_sh_bit_present_flag[i] equal to 0 specifies that the i-th extra bit is not present in the slice headers of pictures referring to the SPS.

The variable NumExtraShBits is derived as follows:

```
NumExtraShBits = 0
for( i = 0; i < ( sps_num_extra_sh_bytes * 8 ); i++ )
    if( sps_extra_sh_bit_present_flag[ i ] )                    (42)
        NumExtraShBits++
```

sps_sublayer_dpb_params_flag is used to control the presence of dpb_max_dec_pic_buffering_minus1[i], dpb_max_num_reorder_pics[i], and dpb_max_latency_increase_plus1[i] syntax elements in the dpb_parameters( ) syntax strucure in the SPS for i in range from 0 to sps_max_sublayers_minus1 − 1, inclusive, when sps_max_sublayers_minus1 is greater than 0. When not present, the value of sps_sublayer_dpb_params_flag is inferred to be equal to 0.

sps_log2_min_luma_coding_block_size_minus2 plus 2 specifies the minimum luma coding block size. The value range of sps_log2_min_luma_coding_block_size_minus2 shall be in the range of 0 to Min(4, sps_log2_ctu_size_minus5 + 3), inclusive.

The variables MinCbLog2SizeY, MinCbSizeY, IbcBufWidthY, IbcBufWidthC and Vsize are derived as follows:

MinCbLog2SizeY = sps_log2_min_luma_coding_block_size_minus2 + 2 (43)

MinCbSizeY = 1 << MinCbLog2SizeY (44)

IbcBufWidthY = 256 * 128 / CtbSizeY (45)

IbcBufWidthC = IbcBufWidthY / SubWidthC (46)

VSize = Min(64, CtbSizeY) (47)

The value of MinCbSizeY shall be less than or equal to VSize.

The variables CtbWidthC and CtbHeightC, which specify the width and height, respectively, of the array for each chroma CTB, are derived as follows:

– If sps_chroma_format_idc is equal to 0 (monochrome), CtbWidthC and CtbHeightC are both set equal to 0.
– Otherwise, CtbWidthC and CtbHeightC are derived as follows:

CtbWidthC = CtbSizeY / SubWidthC (48)

CtbHeightC = CtbSizeY / SubHeightC (49)

For log2BlockWidth ranging from 0 to 4 and for log2BlockHeight ranging from 0 to 4, inclusive, the up-right diagonal scan order array initialization process as specified in clause 6.5.2 is invoked with 1 << log2BlockWidth and 1 << log2BlockHeight as inputs, and the output is assigned to DiagScanOrder[log2BlockWidth][log2BlockHeight].

For log2BlockWidth ranging from 0 to 6 and for log2BlockHeight ranging from 0 to 6, inclusive, the horizontal and vertical traverse scan order array initialization process as specified in clause 6.5.3 is invoked with 1 << log2BlockWidth and 1 << log2BlockHeight as inputs, and the output is assigned to HorTravScanOrder[log2BlockWidth][log2BlockHeight] and VerTravScanOrder[log2BlockWidth][log2BlockHeight].

sps_partition_constraints_override_enabled_flag equal to 1 specifies the presence of ph_partition_constraints_override_flag in PH syntax structures referring to the SPS. sps_partition_constraints_override_enabled_flag equal to 0 specifies the absence of ph_partition_constraints_override_flag in PH syntax structures referring to the SPS.

sps_log2_diff_min_qt_min_cb_intra_slice_luma specifies the default difference between the base 2 logarithm of the minimum size in luma samples of a luma leaf block resulting from quadtree splitting of a CTU and the base 2 logarithm of the minimum coding block size in luma samples for luma CUs in slices with sh_slice_type equal to 2 (I) referring to the SPS. When sps_partition_constraints_override_enabled_flag is equal to 1, the default difference can be overridden by ph_log2_diff_min_qt_min_cb_intra_slice_luma present in PH syntax structures referring to the SPS. The value of sps_log2_diff_min_qt_min_cb_intra_slice_luma shall be in the range of 0 to Min(6, CtbLog2SizeY) − MinCbLog2SizeY, inclusive. The base 2 logarithm of the minimum size in luma samples of a luma leaf block resulting from quadtree splitting of a CTU is derived as follows:

MinQtLog2SizeIntraY = sps_log2_diff_min_qt_min_cb_intra_slice_luma + MinCbLog2SizeY (50)

sps_max_mtt_hierarchy_depth_intra_slice_luma specifies the default maximum hierarchy depth for coding units resulting from multi-type tree splitting of a quadtree leaf in slices with sh_slice_type equal to 2 (I) referring to the SPS. When sps_partition_constraints_override_enabled_flag is equal to 1, the default maximum hierarchy depth can be overridden by ph_max_mtt_hierarchy_depth_intra_slice_luma present in PH syntax structures referring to the SPS. The value of sps_max_mtt_hierarchy_depth_intra_slice_luma shall be in the range of 0 to 2*(CtbLog2SizeY − MinCbLog2SizeY), inclusive.

sps_log2_diff_max_bt_min_qt_intra_slice_luma specifies the default difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a luma coding block that can be split using a binary split and the base 2 logarithm of the minimum size (width or height) in luma samples of a luma leaf block resulting from quadtree splitting of a CTU in slices with sh_slice_type equal to 2 (I) referring to the SPS. When sps_partition_constraints_override_enabled_flag is equal to 1, the default difference can be overridden by ph_log2_diff_max_bt_min_qt_luma present in PH syntax structures referring to the SPS. The value of sps_log2_diff_max_bt_min_qt_intra_slice_luma shall be in the range of 0 to CtbLog2SizeY − MinQtLog2SizeIntraY, inclusive. When sps_log2_diff_max_bt_min_qt_intra_slice_luma is not present, the value of sps_log2_diff_max_bt_min_qt_intra_slice_luma is inferred to be equal to 0.

sps_log2_diff_max_tt_min_qt_intra_slice_luma specifies the default difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a luma coding block that can be split using a ternary split and the base 2 logarithm of the minimum size (width or height) in luma samples of a luma leaf block resulting from quadtree splitting of a CTU in slices with sh_slice_type equal to 2 (I) referring to the SPS. When sps_partition_constraints_override_enabled_flag is equal to 1, the default difference can be overridden by ph_log2_diff_max_tt_min_qt_luma present in PH syntax structures referring to the SPS. The value of sps_log2_diff_max_tt_min_qt_intra_slice_luma shall be in the range of 0 to Min(6, CtbLog2SizeY) − MinQtLog2SizeIntraY, inclusive. When sps_log2_diff_max_tt_min_qt_intra_slice_luma is not present, the value of sps_log2_diff_max_tt_min_qt_intra_slice_luma is inferred to be equal to 0.

sps_qtbtt_dual_tree_intra_flag equal to 1 specifies that, for I slices, each CTU is split into coding units with 64×64 luma samples using an implicit quadtree split, and these coding units are the root of two separate coding_tree syntax structure for luma and chroma. sps_qtbtt_dual_tree_intra_flag equal to 0 specifies separate coding_tree syntax structure is not used for I slices. When sps_qtbtt_dual_tree_intra_flag is not present, it is inferred to be equal to 0. When sps_log2_diff_max_bt_min_qt_intra_slice_luma is greater than Min(6, CtbLog2SizeY) − MinQtLog2SizeIntraY, the value of sps_qtbtt_dual_tree_intra_flag shall be equal to 0.

sps_log2_diff_min_qt_min_cb_intra_slice_chroma specifies the default difference between the base 2 logarithm of the minimum size in luma samples of a chroma leaf block resulting from quadtree splitting of a chroma CTU with treeType equal to DUAL_TREE_CHROMA and the base 2 logarithm of the minimum coding block size in luma samples for chroma CUs with treeType equal to DUAL_TREE_CHROMA in slices with sh_slice_type equal to 2 (I) referring to the SPS. When sps_partition_constraints_override_enabled_flag is equal to 1, the default difference can be overridden by ph_log2_diff_min_qt_min_cb_chroma present in PH syntax structures referring to the SPS. The value of sps_log2_diff_min_qt_min_cb_intra_slice_chroma shall be in the range of 0 to Min(6, CtbLog2SizeY) − MinCbLog2SizeY, inclusive. When not present, the value of sps_log2_diff_min_qt_min_cb_intra_slice_chroma is inferred to be equal to 0. The base 2 logarithm of the minimum size in luma samples of a chroma leaf block resulting from quadtree splitting of a CTU with treeType equal to DUAL_TREE_CHROMA is derived as follows:

$$\text{MinQtLog2SizeIntraC} = \text{sps\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_chroma} + \text{MinCbLog2SizeY} \quad (51)$$

sps_max_mtt_hierarchy_depth_intra_slice_chroma specifies the default maximum hierarchy depth for chroma coding units resulting from multi-type tree splitting of a chroma quadtree leaf with treeType equal to DUAL_TREE_CHROMA in slices with sh_slice_type equal to 2 (I) referring to the SPS. When sps_partition_constraints_override_enabled_flag is equal to 1, the default maximum hierarchy depth can be overridden by ph_max_mtt_hierarchy_depth_chroma present in PH syntax structures referring to the SPS. The value of sps_max_mtt_hierarchy_depth_intra_slice_chroma shall be in the range of 0 to 2*(CtbLog2SizeY − MinCbLog2SizeY), inclusive. When not present, the value of sps_max_mtt_hierarchy_depth_intra_slice_chroma is inferred to be equal to 0.

sps_log2_diff_max_bt_min_qt_intra_slice_chroma specifies the default difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a chroma coding block that can be split using a binary split and the base 2 logarithm of the minimum size (width or height) in luma samples of a chroma leaf block resulting from quadtree splitting of a chroma CTU with treeType equal to DUAL_TREE_CHROMA in slices with sh_slice_type equal to 2 (I) referring to the SPS. When sps_partition_constraints_override_enabled_flag is equal to 1, the default difference can be overridden by ph_log2_diff_max_bt_min_qt_chroma present in PH syntax structures referring to the SPS. The value of sps_log2_diff_max_bt_min_qt_intra_slice_chroma shall be in the range of 0 to Min(6, CtbLog2SizeY) − MinQtLog2SizeIntraC, inclusive. When sps_log2_diff_max_bt_min_qt_intra_slice_chroma is not present, the value of sps_log2_diff_max_bt_min_qt_intra_slice_chroma is inferred to be equal to 0.

sps_log2_diff_max_tt_min_qt_intra_slice_chroma specifies the default difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a chroma coding block that can be split using a ternary split and the base 2 logarithm of the minimum size (width or height) in luma samples of a chroma leaf block resulting from quadtree splitting of a chroma CTU with treeType equal to DUAL_TREE_CHROMA in slices with sh_slice_type equal to 2 (I) referring to the SPS. When sps_partition_constraints_override_enabled_flag is equal to 1, the default difference can be overridden by ph_log2_diff_max_tt_min_qt_chroma present in PH syntax structures referring to the SPS. The value of sps_log2_diff_max_tt_min_qt_intra_slice_chroma shall be in the range of 0 to

Min(6, CtbLog2SizeY) − MinQtLog2SizeIntraC, inclusive. When sps_log2_diff_max_tt_min_qt_intra_slice_chroma is not present, the value of sps_log2_diff_max_tt_min_qt_intra_slice_chroma is inferred to be equal to 0.

sps_log2_diff_min_qt_min_cb_inter_slice specifies the default difference between the base 2 logarithm of the minimum size in luma samples of a luma leaf block resulting from quadtree splitting of a CTU and the base 2 logarithm of the minimum luma coding block size in luma samples for luma CUs in slices with sh_slice_type equal to 0 (B) or 1 (P) referring to the SPS. When sps_partition_constraints_override_enabled_flag is equal to 1, the default difference can be overridden by ph_log2_diff_min_qt_min_cb_inter_slice present in PH syntax structures referring to the SPS. The value of sps_log2_diff_min_qt_min_cb_inter_slice shall be in the range of 0 to Min(6, CtbLog2SizeY) − MinCbLog2SizeY, inclusive. The base 2 logarithm of the minimum size in luma samples of a luma leaf block resulting from quadtree splitting of a CTU is derived as follows:

$$\text{MinQtLog2SizeInterY} = \text{sps\_log2\_diff\_min\_qt\_min\_cb\_inter\_slice} + \text{MinCbLog2SizeY} \quad (52)$$

sps_max_mtt_hierarchy_depth_inter_slice specifies the default maximum hierarchy depth for coding units resulting from multi-type tree splitting of a quadtree leaf in slices with sh_slice_type equal to 0 (B) or 1 (P) referring to the SPS. When sps_partition_constraints_override_enabled_flag is equal to 1, the default maximum hierarchy depth can be overridden by ph_max_mtt_hierarchy_depth_inter_slice present in PH syntax structures referring to the SPS. The value of sps_max_mtt_hierarchy_depth_inter_slice shall be in the range of 0 to 2*(CtbLog2SizeY − MinCbLog2SizeY), inclusive.

sps_log2_diff_max_bt_min_qt_inter_slice specifies the default difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a luma coding block that can be split using a binary split and the base 2 logarithm of the minimum size (width or height) in luma samples of a luma leaf block resulting from quadtree splitting of a CTU in slices with sh_slice_type equal to 0 (B) or 1 (P) referring to the SPS. When sps_partition_constraints_override_enabled_flag is equal to 1, the default difference can be overridden by ph_log2_diff_max_bt_min_qt_luma present in PH syntax structures referring to the SPS. The value of sps_log2_diff_max_bt_min_qt_inter_slice shall be in the range of 0 to CtbLog2SizeY − MinQtLog2SizeInterY, inclusive. When sps_log2_diff_max_bt_min_qt_inter_slice is not present, the value of sps_log2_diff_max_bt_min_qt_inter_slice is inferred to be equal to 0.

sps_log2_diff_max_tt_min_qt_inter_slice specifies the default difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a luma coding block that can be split using a ternary split and the base 2 logarithm of the minimum size (width or height) in luma samples of a luma leaf block resulting from quadtree splitting of a CTU in slices with sh_slice_type equal to 0 (B) or 1 (P) referring to the SPS. When sps_partition_constraints_override_enabled_flag is equal to 1, the default difference can be overridden by ph_log2_diff_max_tt_min_qt_luma present in PH syntax structures referring to the SPS. The value of sps_log2_diff_max_tt_min_qt_inter_slice shall be in the range of 0 to Min(6, CtbLog2SizeY) − MinQtLog2SizeInterY, inclusive. When sps_log2_diff_max_tt_min_qt_inter_slice is not present, the value of sps_log2_diff_max_tt_min_qt_inter_slice is inferred to be equal to 0.

sps_max_luma_transform_size_64_flag equal to 1 specifies that the maximum transform size in luma samples is equal to 64. sps_max_luma_transform_size_64_flag equal to 0 specifies that the maximum transform size in luma samples is equal to 32. When not present, the value of sps_max_luma_transform_size_64_flag is inferred to be equal to 0.

The variables MinTbLog2SizeY, MaxTbLog2SizeY, MinTbSizeY, and MaxTbSizeY are derived as follows:

$$\text{MinTbLog2SizeY} = 2 \quad (53)$$

$$\text{MaxTbLog2SizeY} = \text{sps\_max\_luma\_transform\_size\_64\_flag} \ ?\ 6 : 5 \quad (54)$$

$$\text{MinTbSizeY} = 1 \ll \text{MinTbLog2SizeY} \quad (55)$$

$$\text{MaxTbSizeY} = 1 \ll \text{MaxTbLog2SizeY} \quad (56)$$

sps_transform_skip_enabled_flag equal to 1 specifies that transform_skip_flag could be present in the transform unit syntax. sps_transform_skip_enabled_flag equal to 0 specifies that transform_skip_flag is not present in the transform unit syntax.

sps_log2_transform_skip_max_size_minus2 specifies the maximum block size used for transform skip, and shall be in the range of 0 to 3, inclusive.

The variable MaxTsSize is set equal to 1 << (sps_log2_transform_skip_max_size_minus2 + 2).

sps_bdpcm_enabled_flag equal to 1 specifies that intra_bdpcm_luma_flag and intra_bdpcm_chroma_flag could be present in the coding unit syntax for intra coding units. sps_bdpcm_enabled_flag equal to 0 specifies that intra_bdpcm_luma_flag and intra_bdpcm_chroma_flag are not present in the coding unit syntax for intra coding units. When not present, the value of sps_bdpcm_enabled_flag is inferred to be equal to 0.

sps_mts_enabled_flag equal to 1 specifies that sps_explicit_mts_intra_enabled_flag and sps_explicit_mts_inter_enabled_flag are present in the SPS. sps_mts_enabled_flag equal to 0 specifies that sps_explicit_mts_intra_enabled_flag and sps_explicit_mts_inter_enabled_flag are not present in the SPS.

sps_explicit_mts_intra_enabled_flag equal to 1 specifies that mts_idx could be present in the intra coding unit syntax of the CLVS. sps_explicit_mts_intra_enabled_flag equal to 0 specifies that mts_idx is not present in the intra coding unit syntax of the CLVS. When not present, the value of sps_explicit_mts_intra_enabled_flag is inferred to be equal to 0.

sps_explicit_mts_inter_enabled_flag equal to 1 specifies that mts_idx could be present in the inter coding unit syntax of the CLVS. sps_explicit_mts_inter_enabled_flag equal to 0 specifies that mts_idx is not present in the inter coding unit syntax of the CLVS. When not present, the value of sps_explicit_mts_inter_enabled_flag is inferred to be equal to 0.

sps_lfnst_enabled_flag equal to 1 specifies that lfnst_idx could be present in intra coding unit syntax. sps_lfnst_enabled_flag equal to 0 specifies that lfnst_idx is not present in intra coding unit syntax.

sps_joint_cbcr_enabled_flag equal to 1 specifies that the joint coding of chroma residuals is enabled for the CLVS. sps_joint_cbcr_enabled_flag equal to 0 specifies that the joint coding of chroma residuals is disabled for the CLVS. When not present, the value of sps_joint_cbcr_enabled_flag is inferred to be equal to 0.

sps_same_qp_table_for_chroma_flag equal to 1 specifies that only one chroma QP mapping table is signalled and this table applies to Cb and Cr residuals and additionally to joint Cb-Cr residuals when sps_joint_cbcr_enabled_flag is equal to 1. sps_same_qp_table_for_chroma_flag equal to 0 specifies that chroma QP mapping tables, two for Cb and Cr, and one additional for joint Cb-Cr when sps_joint_cbcr_enabled_flag is equal to 1, are signalled in the SPS. When not present, the value of sps_same_qp_table_for_chroma_flag is inferred to be equal to 1.

sps_qp_table_start_minus26[i] plus 26 specifies the starting luma and chroma QP used to describe the i-th chroma QP mapping table. The value of sps_qp_table_start_minus26[i] shall be in the range of −26 − QpBdOffset to 36 inclusive. When not present, the value of sps_qp_table_start_minus26[i] is inferred to be equal to 0.

sps_num_points_in_qp_table_minus1[i] plus 1 specifies the number of points used to describe the i-th chroma QP mapping table. The value of sps_num_points_in_qp_table_minus1[i] shall be in the range of 0 to 36 − sps_qp_table_start_minus26[i], inclusive. When not present, the value of sps_num_points_in_qp_table_minus1[0] is inferred to be equal to 0.

sps_delta_qp_in_val_minus1[i][j] specifies a delta value used to derive the input coordinate of the j-th pivot point of the i-th chroma QP mapping table. When not present, the value of sps_delta_qp_in_val_minus1[0][j] is inferred to be equal to 0.

sps_delta_qp_diff_val[i][j] specifies a delta value used to derive the output coordinate of the j-th pivot point of the i-th chroma QP mapping table.

The i-th chroma QP mapping table ChromaQpTable[i] for i = 0..numQpTables − 1 is derived as follows:

```
qpInVal[ i ][ 0 ] = sps_qp_table_start_minus26[ i ] + 26
qpOutVal[ i ][ 0 ] = qpInVal[ i ][ 0 ]
for( j = 0; j  <=  sps_num_points_in_qp_table_minus1[ i ]; j++ ) {
  qpInVal[ i ][ j + 1 ] = qpInVal[ i ][ j ] + sps_delta_qp_in_val_minus1[ i ][ j ] + 1
  qpOutVal[ i ][ j + 1 ] = qpOutVal[ i ][ j ] +
        ( sps_delta_qp_in_val_minus1[ i ][ j ] ^ sps_delta_qp_diff_val[ i ][ j ] )
}
ChromaQpTable[ i ][ qpInVal[ i ][ 0 ] ] = qpOutVal[ i ][ 0 ]
for( k = qpInVal[ i ][ 0 ] − 1; k  >=  −QpBdOffset; k − − )
  ChromaQpTable[ i ][ k ] = Clip3( −QpBdOffset, 63, ChromaQpTable[ i ][ k + 1 ] − 1 )                    (57)
for( j = 0; j  <=  sps_num_points_in_qp_table_minus1[ i ]; j++ ) {
  sh = ( sps_delta_qp_in_val_minus1[ i ][ j ] + 1 )  >>  1
  for( k = qpInVal[ i ][ j ] + 1, m = 1; k  <=  qpInVal[ i ][ j + 1 ]; k++, m++ )
        ChromaQpTable[ i ][ k ] = ChromaQpTable[ i ][ qpInVal[ i ][ j ] ] +
              ( ( qpOutVal[ i ][ j + 1 ] − qpOutVal[ i ][ j ] ) * m + sh ) /
( sps_delta_qp_in_val_minus1[ i ][ j ] + 1 )
}
for( k = qpInVal[ i ][ sps_num_points_in_qp_table_minus1[ i ] + 1 ] + 1; k  <=  63; k++ )
  ChromaQpTable[ i ][ k ] = Clip3( −QpBdOffset, 63, ChromaQpTable[ i ][ k − 1 ] + 1 )
```

When sps_same_qp_table_for_chroma_flag is equal to 1, ChromaQpTable[1][k] and ChromaQpTable[2][k] are set equal to ChromaQpTable[0][k] for k in the range of −QpBdOffset to 63, inclusive.

It is a requirement of bitstream conformance that the values of qpInVal[i][j] and qpOutVal[i][j] shall be in the range of −QpBdOffset to 63, inclusive for i in the range of 0 to numQpTables − 1, inclusive, and j in the range of 0 to sps_num_points_in_qp_table_minus1[i] + 1, inclusive.

sps_sao_enabled_flag equal to 1 specifies that SAO is enabled for the CLVS. sps_sao_enabled_flag equal to 0 specifies that SAO is disabled for the CLVS.

sps_alf_enabled_flag equal to 1 specifies that ALF is enabled for the CLVS. sps_alf_enabled_flag equal to 0 specifies that ALF is disabled for the CLVS.

sps_ccalf_enabled_flag equal to 1 specifies that CCALF is enabled for the CLVS. sps_ccalf_enabled_flag equal to 0 specifies that CCALF is disabled for the CLVS. When not present, the value of sps_ccalf_enabled_flag is inferred to be equal to 0.

sps_lmcs_enabled_flag equal to 1 specifies that LMCS is enabled for the CLVS. sps_lmcs_enabled_flag equal to 0 specifies that LMCS is disabled for the CLVS.

sps_weighted_pred_flag equal to 1 specifies that weighted prediction might be applied to P slices referring to the SPS. sps_weighted_pred_flag equal to 0 specifies that weighted prediction is not applied to P slices referring to the SPS.

sps_weighted_bipred_flag equal to 1 specifies that explicit weighted prediction might be applied to B slices referring to the SPS. sps_weighted_bipred_flag equal to 0 specifies that explicit weighted prediction is not applied to B slices referring to the SPS.

sps_long_term_ref_pics_flag equal to 0 specifies that no LTRP is used for inter prediction of any coded picture in the CLVS. sps_long_term_ref_pics_flag equal to 1 specifies that LTRPs might be used for inter prediction of one or more coded pictures in the CLVS.

sps_inter_layer_prediction_enabled_flag equal to 1 specifies that inter-layer prediction is enabled for the CLVS and ILRPs might be used for inter prediction of one or more coded pictures in the CLVS. sps_inter_layer_prediction_enabled_flag equal to 0 specifies that inter-layer prediction is disabled for the CLVS and no ILRP is used for inter prediction of any coded picture in the CLVS. When sps_video_parameter_set_id is equal to 0, the value of sps_inter_layer_prediction_enabled_flag is inferred to be equal to 0. When vps_independent_layer_flag[GeneralLayerIdx[nuh_layer_id]] is equal to 1, the value of sps_inter_layer_prediction_enabled_flag shall be equal to 0.

sps_idr_rpl_present_flag equal to 1 specifies that RPL syntax elements could be present in slice headers of slices with nal_unit_type equal to IDR_N_LP or IDR_W_RADL. sps_idr_rpl_present_flag equal to 0 specifies that RPL syntax elements are not present in slice headers of slices with nal_unit_type equal to IDR_N_LP or IDR_W_RADL.

sps_rpl1_same_as_rpl0_flag equal to 1 specifies that the syntax element sps_num_ref_pic_lists[1] and the syntax structure ref_pic_list_struct(1, rplsIdx) are not present and the following applies:

– The value of sps_num_ref_pic_lists[1] is inferred to be equal to the value of sps_num_ref_pic_lists[0].
– The value of each of syntax elements in ref_pic_list_struct(1, rplsIdx) is inferred to be equal to the value of corresponding syntax element in ref_pic_list_struct(0, rplsIdx) for rplsIdx ranging from 0 to sps_num_ref_pic_lists[0] − 1.

sps_num_ref_pic_lists[i] specifies the number of the ref_pic_list_struct(listIdx, rplsIdx) syntax structures with listIdx equal to i included in the SPS. The value of sps_num_ref_pic_lists[i] shall be in the range of 0 to 64, inclusive.

NOTE 7 – For each value of listIdx (equal to 0 or 1), a decoder could allocate memory for a total number of sps_num_ref_pic_lists[i] + 1 ref_pic_list_struct(listIdx, rplsIdx) syntax structures since there could be one ref_pic_list_struct(listIdx, rplsIdx) syntax structure directly signalled in the picture headers or slice headers of a current picture.

sps_ref_wraparound_enabled_flag equal to 1 specifies that horizontal wrap-around motion compensation is enabled for the CLVS. sps_ref_wraparound_enabled_flag equal to 0 specifies that horizontal wrap-around motion compensation is disabled for the CLVS.

It is a requirement of bitstream conformance that, when there is one or more values of i in the range of 0 to sps_num_subpics_minus1, inclusive, for which sps_subpic_treated_as_pic_flag[i] is equal to 1 and sps_subpic_width_minus1[i] plus 1 is not equal to (sps_pic_width_max_in_luma_samples + CtbSizeY − 1) >> CtbLog2SizeY), the value of sps_ref_wraparound_enabled_flag shall be equal to 0.

sps_temporal_mvp_enabled_flag equal to 1 specifies that temporal motion vector predictors are enabled for the CLVS. sps_temporal_mvp_enabled_flag equal to 0 specifies that temporal motion vector predictors are disabled for the CLVS.

sps_sbtmvp_enabled_flag equal to 1 specifies that subblock-based temporal motion vector predictors are enabled and might be used in decoding of pictures with all slices having sh_slice_type not equal to I in the CLVS.

sps_sbtmvp_enabled_flag equal to 0 specifies that subblock-based temporal motion vector predictors are disabled and not used in decoding of pictures in the CLVS. When sps_sbtmvp_enabled_flag is not present, it is inferred to be equal to 0.

sps_amvr_enabled_flag equal to 1 specifies that adaptive motion vector difference resolution is enabled for the CVLS. amvr_enabled_flag equal to 0 specifies that adaptive motion vector difference resolution is disabled for the CLVS.

sps_bdof_enabled_flag equal to 1 specifies that the bi-directional optical flow inter prediction is enabled for the CLVS. sps_bdof_enabled_flag equal to 0 specifies that the bi-directional optical flow inter prediction is disabled for the CLVS.

sps_bdof_control_present_in_ph_flag equal to 1 specifies that ph_bdof_disabled_flag could be present in PH syntax structures referring to the SPS. sps_bdof_control_present_in_ph_flag equal to 0 specifies that ph_bdof_disabled_flag is not present in PH syntax structures referring to the SPS. When not present, the value of sps_bdof_control_present_in_ph_flag is inferred to be equal to 0.

sps_smvd_enabled_flag equal to 1 specifies that symmetric motion vector difference is enabled for the CLVS. sps_smvd_enabled_flag equal to 0 specifies that symmetric motion vector difference is disabled for the CLVS.

sps_dmvr_enabled_flag equal to 1 specifies that decoder motion vector refinement based inter bi-prediction is enabled for the CLVS. sps_dmvr_enabled_flag equal to 0 specifies that decoder motion vector refinement based inter bi-prediction is disabled for the CLVS.

sps_dmvr_control_present_in_ph_flag equal to 1 specifies that ph_dmvr_disabled_flag could be present in PH syntax structures referring to the SPS. sps_dmvr_control_present_in_ph_flag equal to 0 specifies that ph_dmvr_disabled_flag is not present in PH syntax structures referring to the SPS. When not present, the value of sps_dmvr_control_present_in_ph_flag is inferred to be equal to 0.

sps_mmvd_enabled_flag equal to 1 specifies that merge mode with motion vector difference is enabled for the CLVS. sps_mmvd_enabled_flag equal to 0 specifies that merge mode with motion vector difference is disabled for the CLVS.

sps_mmvd_fullpel_only_enabled_flag equal to 1 specifies that the merge mode with motion vector difference using only integer sample precision is enabled for the CLVS. sps_mmvd_fullpel_enabled_only_flag equal to 0 specifies that the merge mode with motion vector difference using only integer sample precision is disabled for the CLVS. When not present, the value of sps_mmvd_fullpel_only_enabled_flag is inferred to be equal to 0.

sps_six_minus_max_num_merge_cand specifies the maximum number of merging motion vector prediction (MVP) candidates supported in the SPS subtracted from 6. The value of sps_six_minus_max_num_merge_cand shall be in the range of 0 to 5, inclusive.

The maximum number of merging MVP candidates, MaxNumMergeCand, is derived as follows:

$$\text{MaxNumMergeCand} = 6 - \text{sps\_six\_minus\_max\_num\_merge\_cand} \qquad (58)$$

sps_sbt_enabled_flag equal to 1 specifies that subblock transform for inter-predicted CUs is enabled for the CLVS. sps_sbt_enabled_flag equal to 0 specifies that subblock transform for inter-predicted CUs is disabled for the CLVS.

sps_affine_enabled_flag equal to 1 specifies that the affine model based motion compensation is enabled for the CLVS and inter_affine_flag and cu_affine_type_flag could be present in the coding unit syntax of the CLVS. sps_affine_enabled_flag equal to 0 specifies that the affine model based motion compensation is disabled for the CLVS and inter_affine_flag and cu_affine_type_flag are not present in the coding unit syntax of the CLVS.

sps_five_minus_max_num_subblock_merge_cand specifies the maximum number of subblock-based merging motion vector prediction candidates supported in the SPS subtracted from 5. The value of sps_five_minus_max_num_subblock_merge_cand shall be in the range of 0 to 5 − sps_sbtmvp_enabled_flag, inclusive.

sps_6param_affine_enabled_flag equal to 1 specifies that the 6-parameter affine model based motion compensation is enabled for the CLVS. sps_6param_affine_enabled_flag equal to 0 specifies that the 6-parameter affine model based motion compensation is disabled for the CLVS. When not present, the value of sps_6param_affine_enabled_flag is inferred to be equal to 0.

sps_affine_amvr_enabled_flag equal to 1 specifies that adaptive motion vector difference resolution is enabled for the CLVS. sps_affine_amvr_enabled_flag equal to 0 specifies that adaptive motion vector difference resolution is disabled for the CLVS. When not present, the value of sps_affine_amvr_enabled_flag is inferred to be equal to 0.

sps_affine_prof_enabled_flag equal to 1 specifies that the affine motion compensation refined with optical flow is enabled for the CLVS. sps_affine_prof_enabled_flag equal to 0 specifies that the affine motion compensation refined with optical flow is disabled for the CLVS. When not present, the value of sps_affine_prof_enabled_flag is inferred to be equal to 0.

sps_prof_control_present_in_ph_flag equal to 1 specifies that ph_prof_disabled_flag could be present in PH syntax structures referring to the SPS. sps_prof_control_present_in_ph_flag equal to 0 specifies that ph_prof_disabled_flag is not present in PH syntax structures referring to the SPS. When sps_prof_control_present_in_ph_flag is not present, the value of sps_prof_control_present_in_ph_flag is inferred to be equal to 0.

sps_bcw_enabled_flag equal to 1 specifies that bi-prediction with CU weights is enabled for the CLVS and bcw_idx could be present in the coding unit syntax of the CLVS. sps_bcw_enabled_flag equal to 0 specifies that bi-prediction with CU weights is disabled for the CLVS and bcw_idx is not present in the coding unit syntax of the CLVS.

sps_ciip_enabled_flag equal to 1 specifies that ciip_flag could be present in the coding unit syntax for inter coding units. sps_ciip_enabled_flag equal to 0 specifies that ciip_flag is not present in the coding unit syntax for inter coding units.

sps_gpm_enabled_flag equal to 1 specifies that the geometric partition based motion compensation is enabled for the CLVS and merge_gpm_partition_idx, merge_gpm_idx0, and merge_gpm_idx1 could be present in the coding unit syntax of the CLVS. sps_gpm_enabled_flag equal to 0 specifies that the geometric partition based motion compensation is disabled for the CLVS and merge_gpm_partition_idx, merge_gpm_idx0, and merge_gpm_idx1 are not present in the coding unit syntax of the CLVS. When not present, the value of sps_gpm_enabled_flag is inferred to be equal to 0.

sps_max_num_merge_cand_minus_max_num_gpm_cand specifies the maximum number of geometric partitioning merge mode candidates supported in the SPS subtracted from MaxNumMergeCand. The value of sps_max_num_merge_cand_minus_max_num_gpm_cand shall be in the range of 0 to MaxNumMergeCand − 2, inclusive.

The maximum number of geometric partitioning merge mode candidates, MaxNumGpmMergeCand, is derived as follows:

```
if( sps_gpm_enabled_flag  &&  MaxNumMergeCand  >=  3 )
    MaxNumGpmMergeCand = MaxNumMergeCand −
            sps_max_num_merge_cand_minus_max_num_gpm_cand                    (59)
else if( sps_gpm_enabled_flag  &&  MaxNumMergeCand  = =  2 )
    MaxNumGpmMergeCand = 2
else
    MaxNumGpmMergeCand = 0
```

sps_log2_parallel_merge_level_minus2 plus 2 specifies the value of the variable Log2ParMrgLevel, which is used in the derivation process for spatial merging candidates as specified in clause 8.5.2.3, the derivation process for motion vectors and reference indices in subblock merge mode as specified in clause 8.5.5.2, and to control the invocation of the updating process for the history-based motion vector predictor list in clause 8.5.2.1. The value of sps_log2_parallel_merge_level_minus2 shall be in the range of 0 to CtbLog2SizeY − 2, inclusive. The variable Log2ParMrgLevel is derived as follows:

```
Log2ParMrgLevel = sps_log2_parallel_merge_level_minus2 + 2                   (60)
```

sps_isp_enabled_flag equal to 1 specifies that intra prediction with subpartitions is enabled for the CLVS. sps_isp_enabled_flag equal to 0 specifies that intra prediction with subpartitions is disabled for the CLVS.

sps_mrl_enabled_flag equal to 1 specifies that intra prediction with multiple reference lines is enabled for the CLVS. sps_mrl_enabled_flag equal to 0 specifies that intra prediction with multiple reference lines is disabled for the CLVS.

sps_mip_enabled_flag equal to 1 specifies that the matrix-based intra prediction is enabled for the CLVS. sps_mip_enabled_flag equal to 0 specifies that the matrix-based intra prediction is disabled for the CLVS.

sps_cclm_enabled_flag equal to 1 specifies that the cross-component linear model intra prediction from luma component to chroma component is enabled for the CLVS. sps_cclm_enabled_flag equal to 0 specifies that the cross-component linear model intra prediction from luma component to chroma component is disabled for the CLVS. When sps_cclm_enabled_flag is not present, it is inferred to be equal to 0.

sps_chroma_horizontal_collocated_flag equal to 1 specifies that prediction processes operate in a manner designed for chroma sample positions that are not horizontally shifted relative to corresponding luma sample positions. sps_chroma_horizontal_collocated_flag equal to 0 specifies that prediction processes operate in a manner designed for chroma sample positions that are shifted to the right by 0.5 in units of luma samples relative to corresponding luma sample positions. When sps_chroma_horizontal_collocated_flag is not present, it is inferred to be equal to 1.

sps_chroma_vertical_collocated_flag equal to 1 specifies that prediction processes operate in a manner designed for chroma sample positions that are not vertically shifted relative to corresponding luma sample positions. sps_chroma_vertical_collocated_flag equal to 0 specifies that prediction processes operate in a manner designed for chroma sample positions that are shifted downward by 0.5 in units of luma samples relative to corresponding luma sample positions. When sps_chroma_vertical_collocated_flag is not present, it is inferred to be equal to 1.

sps_palette_enabled_flag equal to 1 specifies that the palette prediction mode is enabled for the CLVS. sps_palette_enabled_flag equal to 0 specifies that the palette prediction mode is disabled for the CLVS. When sps_palette_enabled_flag is not present, it is inferred to be equal to 0.

sps_act_enabled_flag equal to 1 specifies that the adaptive colour transform is enabled for the CLVS and the cu_act_enabled_flag could be present in the coding unit syntax of the CLVS. sps_act_enabled_flag equal to 0 speifies that the adaptive colour transform is disabled for the CLVS and cu_act_enabled_flag is not present in the coding unit syntax of the CLVS. When sps_act_enabled_flag is not present, it is inferred to be equal to 0.

sps_min_qp_prime_ts specifies the minimum allowed quantization parameter for transform skip mode as follows:

```
QpPrimeTsMin = 4 + 6 * sps_min_qp_prime_ts                                    (61)
```

The value of sps_min_qp_prime_ts shall be in the range of 0 to 8, inclusive.

sps_ibc_enabled_flag equal to 1 specifies that the IBC prediction mode is enabled for the CLVS. sps_ibc_enabled_flag equal to 0 specifies that the IBC prediction mode is disabled for the CLVS. When sps_ibc_enabled_flag is not present, it is inferred to be equal to 0.

sps_six_minus_max_num_ibc_merge_cand, when sps_ibc_enabled_flag is equal to 1, specifies the maximum number of IBC merging block vector prediction (BVP) candidates supported in the SPS subtracted from 6. The value of sps_six_minus_max_num_ibc_merge_cand shall be in the range of 0 to 5, inclusive.

The maximum number of IBC merging BVP candidates, MaxNumIbcMergeCand, is derived as follows:

```
if( sps_ibc_enabled_flag )
    MaxNumIbcMergeCand = 6 − sps_six_minus_max_num_ibc_merge_cand                (62)
else
    MaxNumIbcMergeCand = 0
```

sps_ladf_enabled_flag equal to 1 specifies that sps_num_ladf_intervals_minus2, sps_ladf_lowest_interval_qp_offset, sps_ladf_qp_offset[i], and sps_ladf_delta_threshold_minus1[i] are present in the SPS. sps_ladf_enabled_flag equal to 0 specifies that sps_num_ladf_intervals_minus2, sps_ladf_lowest_interval_qp_offset, sps_ladf_qp_offset[i], and sps_ladf_delta_threshold_minus1[i] are not present in the SPS.

sps_num_ladf_intervals_minus2 plus 2 specifies the number of sps_ladf_delta_threshold_minus1[i] and sps_ladf_qp_offset[i] syntax elements that are present in the SPS. The value of sps_num_ladf_intervals_minus2 shall be in the range of 0 to 3, inclusive.

sps_ladf_lowest_interval_qp_offset specifies the offset used to derive the variable qP as specified in clause 8.8.3.6.2. The value of sps_ladf_lowest_interval_qp_offset shall be in the range of −63 to 63, inclusive.

sps_ladf_qp_offset[i] specifies the offset array used to derive the variable qP as specified in clause 8.8.3.6.2. The value of sps_ladf_qp_offset[i] shall be in the range of −63 to 63, inclusive.

sps_ladf_delta_threshold_minus1[i] is used to compute the values of SpsLadfIntervalLowerBound[i], which specifies the lower bound of the i-th luma intensity level interval. The value of sps_ladf_delta_threshold_minus1[i] shall be in the range of 0 to $2^{BitDepth} - 3$, inclusive.

The value of SpsLadfIntervalLowerBound[0] is set equal to 0.

For each value of i in the range of 0 to sps_num_ladf_intervals_minus2, inclusive, the variable SpsLadfIntervalLowerBound[i + 1] is derived as follows:

```
SpsLadfIntervalLowerBound[ i + 1 ] = SpsLadfIntervalLowerBound[ i ]            (63)
                                     + sps_ladf_delta_threshold_minus1[ i ] + 1
```

sps_explicit_scaling_list_enabled_flag equal to 1 specifies that the use of an explicit scaling list, which is signalled in a scaling list APS, in the scaling process for transform coefficients when decoding a slice is enabled for the CLVS. sps_explicit_scaling_list_enabled_flag equal to 0 specifies that the use of an explicit scaling list in the scaling process for transform coefficients when decoding a slice is disabled for the CLVS.

sps_scaling_matrix_for_lfnst_disabled_flag equal to 1 specifies that scaling matrices are disabled for blocks coded with LFNST for the CLVS. sps_scaling_matrix_for_lfnst_disabled_flag equal to 0 specifies that the scaling matrices is enabled for blocks coded with LFNST for the CLVS.

sps_scaling_matrix_for_alternative_colour_space_disabled_flag equal to 1 specifies, for the CLVS, that scaling matrices are disabled and not applied to blocks of a coding unit when the decoded residuals of the current coding unit are applied using a colour space conversion. sps_scaling_matrix_for_alternative_colour_space_disabled_flag equal to 0 specifies, for the CLVS, that scaling matrices are enabled and could be applied to blocks of a coding unit when the decoded residuals of the current coding unit are applied using a colour space conversion. When not present, the value of sps_scaling_matrix_for_alternative_colour_space_disabled_flag is inferred to be equal to 0.

sps_scaling_matrix_designated_colour_space_flag equal to 1 specifies that the colour space of the scaling matrices is the colour space that does not use a colour space conversion for the decoded residuals. sps_scaling_matrix_designated_colour_space_flag equal to 0 specifies that the designated colour space of the scaling matrices is the colour space that uses a colour space conversion for the decoded residuals.

sps_dep_quant_enabled_flag equal to 1 specifies that dependent quantization is enabled for the CLVS. sps_dep_quant_enabled_flag equal to 0 specifies that dependent quantization is disabled for the CLVS.

sps_sign_data_hiding_enabled_flag equal to 1 specifies that sign bit hiding is enabled for the CLVS. sps_sign_data_hiding_enabled_flag equal to 0 specifies that sign bit hiding is disabled for the CLVS.

sps_virtual_boundaries_enabled_flag equal to 1 specifies that disabling in-loop filtering across virtual boundaries is enabled for the CLVS. sps_virtual_boundaries_enabled_flag equal to 0 specifies that disabling in-loop filtering across virtual boundaries is disabled for the CLVS. In-loop filtering operations include the deblocking filter, sample adaptive offset filter, and adaptive loop filter operations.

sps_virtual_boundaries_present_flag equal to 1 specifies that information of virtual boundaries is signalled in the SPS. sps_virtual_boundaries_present_flag equal to 0 specifies that information of virtual boundaries is not signalled in the SPS. When there is one or more than one virtual boundaries signalled in the SPS, the in-loop filtering operations are disabled across the virtual boundaries in pictures referring to the SPS. In-loop filtering operations include the deblocking filter, sample adaptive offset filter, and adaptive loop filter operations. When not present, the value of sps_virtual_boundaries_present_flag is inferred to be equal to 0.

When sps_res_change_in_clvs_allowed_flag is equal to 1, the value of sps_virtual_boundaries_present_flag shall be equal to 0.

When sps_subpic_info_present_flag and sps_virtual_boundaries_enabled_flag are both equal to 1, the value of sps_virtual_boundaries_present_flag shall be equal to 1.

sps_num_ver_virtual_boundaries specifies the number of sps_virtual_boundary_pos_x_minus1[i] syntax elements that are present in the SPS. The value of sps_num_ver_virtual_boundaries shall be in the range of 0 to (sps_pic_width_max_in_luma_samples <= 8 ? 0 : 3), inclusive. When sps_num_ver_virtual_boundaries is not present, it is inferred to be equal to 0.

sps_virtual_boundary_pos_x_minus1[i] plus 1 specifies the location of the i-th vertical virtual boundary in units of luma samples divided by 8. The value of sps_virtual_boundary_pos_x_minus1[i] shall be in the range of 0 to Ceil(sps_pic_width_max_in_luma_samples ÷ 8) − 2, inclusive.

sps_num_hor_virtual_boundaries specifies the number of sps_virtual_boundary_pos_y_minus1[i] syntax elements that are present in the SPS. The value of sps_num_hor_virtual_boundaries shall be in the range of 0 to (sps_pic_height_max_in_luma_samples <= 8 ? 0: 3), inclusive. When sps_num_hor_virtual_boundaries is not present, it is inferred to be equal to 0.

When sps_virtual_boundaries_enabled_flag is equal to 1 and sps_virtual_boundaries_present_flag is equal to 1, the sum of sps_num_ver_virtual_boundaries and sps_num_hor_virtual_boundaries shall be greater than 0.

sps_virtual_boundary_pos_y_minus1[i] plus 1 specifies the location of the i-th horizontal virtual boundary in units of luma samples divided by 8. The value of sps_virtual_boundary_pos_y_minus1[i] shall be in the range of 0 to Ceil(sps_pic_height_max_in_luma_samples ÷ 8) − 2, inclusive.

sps_timing_hrd_params_present_flag equal to 1 specifies that the SPS contains a general_timing_hrd_parameters( ) syntax structure and an ols_timing_hrd_parameters( ) syntax structure. sps_timing_hrd_params_present_flag equal to 0 specifies that the SPS does not contain a general_timing_hrd_parameters( ) syntax structure or an ols_timing_hrd_parameters( ) syntax structure.

sps_sublayer_cpb_params_present_flag equal to 1 specifies that the ols_timing_hrd_parameters( ) syntax structure in the SPS includes HRD parameters for sublayer representations with TemporalId in the range of 0 to sps_max_sublayers_minus1, inclusive. sps_sublayer_cpb_params_present_flag equal to 0 specifies that the ols_timing_hrd_parameters( ) syntax structure in the SPS includes HRD parameters for the sublayer representation with TemporalId equal to sps_max_sublayers_minus1 only. When sps_max_sublayers_minus1 is equal to 0, the value of sps_sublayer_cpb_params_present_flag is inferred to be equal to 0.

When sps_sublayer_cpb_params_present_flag is equal to 0, the HRD parameters for the sublayer representations with TemporalId in the range of 0 to sps_max_sublayers_minus1 − 1, inclusive, are inferred to be the same as that for the sublayer representation with TemporalId equal to sps_max_sublayers_minus1. These include the HRD parameters starting from the fixed_pic_rate_general_flag[i] syntax element till the sublayer_hrd_parameters(i) syntax structure immediately under the condition "if(general_vcl_hrd_params_present_flag)" in the ols_timing_hrd_parameters syntax structure.

sps_field_seq_flag equal to 1 indicates that the CLVS conveys pictures that represent fields. sps_field_seq_flag equal to 0 indicates that the CLVS conveys pictures that represent frames.

When sps_field_seq_flag is equal to 1, a frame-field information SEI message shall be present for every coded picture in the CLVS.

NOTE 8 – The specified decoding process does not treat pictures that represent fields or frames differently. A sequence of pictures that represent fields would therefore be coded with the picture dimensions of an individual field. For example, pictures that represent 1080i fields would commonly have cropped output dimensions of 1920x540, while the sequence picture rate would commonly express the rate of the source fields (typically between 50 and 60 Hz), instead of the source frame rate (typically between 25 and 30 Hz).

sps_vui_parameters_present_flag equal to 1 specifies that the syntax structure vui_payload( ) is present in the SPS RBSP syntax structure. sps_vui_parameters_present_flag equal to 0 specifies that the syntax structure vui_payload( ) is not present in the SPS RBSP syntax structure.

When sps_vui_parameters_present_flag is equal to 0, the information conveyed in the vui_payload( ) syntax structure is considered unspecified or determined by the application by external means. See also clause D.11 for further detail on the inferred video usability information.

sps_vui_payload_size_minus1 plus 1 specifies the number of RBSP bytes in the vui_payload( ) syntax structure. The value of sps_vui_payload_size_minus1 shall be in the range of 0 to 1023, inclusive.

NOTE 9 – The SPS NAL unit byte sequence containing the vui_payload( ) syntax structure might include one or more emulation prevention bytes (represented by emulation_prevention_three_byte syntax elements). Since the payload size of the vui_payload( ) syntax structure is specified in RBSP bytes, the quantity of emulation prevention bytes is not included in the size payloadSize of the vui_payload( ) syntax structure.

sps_vui_alignment_zero_bit shall be equal to 0.

sps_extension_flag equal to 1 specifies that the syntax elements sps_range_extension_flag and sps_extension_7bits are present in the SPS RBSP syntax structure. sps_extension_flag equal to 0 specifies that these syntax elements are not present.

sps_range_extension_flag equal to 1 specifies that the sps_range_extension( ) syntax structure is present in the SPS RBSP syntax structure. sps_range_extension_flag equal to 0 specifies that the sps_range_extension( ) syntax structure is not present. When not present, the value of sps_range_extension_flag is inferred to be equal to 0. When BitDepth is less than or equal to 10, the value of sps_range_extension_flag shall be equal to 0.

sps_extension_7bits equal to 0 specifies that no sps_extension_data_flag syntax elements are present in the SPS RBSP syntax structure. When present, sps_extension_7bits shall be equal to 0 in bitstreams conforming to this version of this document. Values of sps_extension_7bits not equal to 0 are reserved for future use by ITU-T | ISO/IEC. Decoders shall also allow values of sps_extension_7bits not equal to 0 to appear in the syntax. When not present, the value of sps_extension_7bits is inferred to be equal to 0.

sps_extension_data_flag may have any value. Its presence and value do not affect the decoding process specified in this version of this document. Decoders conforming to this version of this document shall ignore the sps_extension_data_flag syntax elements, when present.

7.4.3.5 Picture parameter set RBSP semantics

A PPS RBSP shall be available to the decoding process, by inclusion in at least one AU with TemporalId less than or equal to the TemporalId of the PPS NAL unit or provided through external means, prior to it being referenced by either of the following:

– a PH NAL unit having ph_pic_parameter_set_id equal to the value of pps_pic_parameter_set_id in the PPS RBSP,

– a coded slice NAL unit having sh_picture_header_in_slice_header_flag equal to 1 with ph_pic_parameter_set_id equal to the value of pps_pic_parameter_set_id in the PPS RBSP.

Such a PH NAL unit or coded slice NAL unit references the previous PPS RBSP in decoding order (relative to the position of the PH NAL unit or coded slice NAL unit in decoding order) with that value of pps_pic_parameter_set_id.

All PPS NAL units with a particular value of pps_pic_parameter_set_id within a PU shall have the same content.

pps_pic_parameter_set_id identifies the PPS for reference by other syntax elements.

PPS NAL units, regardless of the nuh_layer_id values, share the same value space of pps_pic_parameter_set_id.

Let ppsLayerId be the value of the nuh_layer_id of a particular PPS NAL unit, and vclLayerId be the value of the nuh_layer_id of a particular VCL NAL unit. The particular VCL NAL unit shall not refer to the particular PPS NAL unit unless ppsLayerId is less than or equal to vclLayerId and all OLSs specified by the VPS that contain the layer with nuh_layer_id equal to vclLayerId also contain the layer with nuh_layer_id equal to ppsLayerId.

NOTE 1 – In a CVS that contains only one layer, the nuh_layer_id of referenced PPSs is equal to the nuh_layer_id of the VCL NAL units.

pps_seq_parameter_set_id specifies the value of sps_seq_parameter_set_id for the SPS. The value of pps_seq_parameter_set_id shall be in the range of 0 to 15, inclusive. The value of pps_seq_parameter_set_id shall be the same in all PPSs that are referred to by coded pictures in a CLVS.

pps_mixed_nalu_types_in_pic_flag equal to 1 specifies that each picture referring to the PPS has more than one VCL NAL unit and the VCL NAL units do not have the same value of nal_unit_type. pps_mixed_nalu_types_in_pic_flag equal to 0 specifies that each picture referring to the PPS has one or more VCL NAL units and the VCL NAL units of each picture refering to the PPS have the same value of nal_unit_type.

NOTE 2– pps_mixed_nalu_types_in_pic_flag equal to 1 indicates that pictures referring to the PPS contain slices with different NAL unit types, e.g., coded pictures originating from a subpicture bitstream merging operation for which encoders have to ensure matching bitstream structure and further alignment of parameters of the original bitstreams. One example of such alignments is as follows: When the value of sps_idr_rpl_present_flag is equal to 0 and pps_mixed_nalu_types_in_pic_flag is equal to 1, a picture referring to the PPS might not have slices with nal_unit_type equal to IDR_W_RADL or IDR_N_LP.

pps_pic_width_in_luma_samples specifies the width of each decoded picture referring to the PPS in units of luma samples. pps_pic_width_in_luma_samples shall not be equal to 0, shall be an integer multiple of Max(8, MinCbSizeY), and shall be less than or equal to sps_pic_width_max_in_luma_samples.

When sps_res_change_in_clvs_allowed_flag equal to 0, the value of pps_pic_width_in_luma_samples shall be equal to sps_pic_width_max_in_luma_samples.

When sps_ref_wraparound_enabled_flag is equal to 1, the value of (CtbSizeY / MinCbSizeY + 1) shall be less than or equal to the value of (pps_pic_width_in_luma_samples / MinCbSizeY − 1).

pps_pic_height_in_luma_samples specifies the height of each decoded picture referring to the PPS in units of luma samples. pps_pic_height_in_luma_samples shall not be equal to 0 and shall be an integer multiple of Max(8, MinCbSizeY), and shall be less than or equal to sps_pic_height_max_in_luma_samples.

When sps_res_change_in_clvs_allowed_flag equal to 0, the value of pps_pic_height_in_luma_samples shall be equal to sps_pic_height_max_in_luma_samples.

The variables PicWidthInCtbsY, PicHeightInCtbsY, PicSizeInCtbsY, PicWidthInMinCbsY, PicHeightInMinCbsY, PicSizeInMinCbsY, PicSizeInSamplesY, PicWidthInSamplesC and PicHeightInSamplesC are derived as follows:

$$\text{PicWidthInCtbsY} = \text{Ceil}(\ \text{pps\_pic\_width\_in\_luma\_samples} \div \text{CtbSizeY}\ ) \tag{64}$$

$$\text{PicHeightInCtbsY} = \text{Ceil}(\ \text{pps\_pic\_height\_in\_luma\_samples} \div \text{CtbSizeY}\ ) \tag{65}$$

$$\text{PicSizeInCtbsY} = \text{PicWidthInCtbsY} * \text{PicHeightInCtbsY} \tag{66}$$

$$\text{PicWidthInMinCbsY} = \text{pps\_pic\_width\_in\_luma\_samples} / \text{MinCbSizeY} \tag{67}$$

$$\text{PicHeightInMinCbsY} = \text{pps\_pic\_height\_in\_luma\_samples} / \text{MinCbSizeY} \tag{68}$$

$$\text{PicSizeInMinCbsY} = \text{PicWidthInMinCbsY} * \text{PicHeightInMinCbsY} \tag{69}$$

$$\text{PicSizeInSamplesY} = \text{pps\_pic\_width\_in\_luma\_samples} * \text{pps\_pic\_height\_in\_luma\_samples} \tag{70}$$

$$\text{PicWidthInSamplesC} = \text{pps\_pic\_width\_in\_luma\_samples} / \text{SubWidthC} \tag{71}$$

$$\text{PicHeightInSamplesC} = \text{pps\_pic\_height\_in\_luma\_samples} / \text{SubHeightC} \tag{72}$$

pps_conformance_window_flag equal to 1 specifies that the conformance cropping window offset parameters follow next in the PPS. pps_conformance_window_flag equal to 0 specifies that the conformance cropping window offset parameters are not present in the PPS.

When pps_pic_width_in_luma_samples is equal to sps_pic_width_max_in_luma_samples and pps_pic_height_in_luma_samples is equal to sps_pic_height_max_in_luma_samples, the value of pps_conformance_window_flag shall be equal to 0.

pps_conf_win_left_offset, **pps_conf_win_right_offset**, **pps_conf_win_top_offset**, and **pps_conf_win_bottom_offset** specify the samples of the pictures in the CLVS that are output from the decoding process, in terms of a rectangular region specified in picture coordinates for output.

When pps_conformance_window_flag is equal to 0, the following applies:

– If pps_pic_width_in_luma_samples is equal to sps_pic_width_max_in_luma_samples and pps_pic_height_in_luma_samples is equal to sps_pic_height_max_in_luma_samples, the values of pps_conf_win_left_offset, pps_conf_win_right_offset, pps_conf_win_top_offset, and pps_conf_win_bottom_offset are inferred to be equal to sps_conf_win_left_offset, sps_conf_win_right_offset, sps_conf_win_top_offset, and sps_conf_win_bottom_offset, respectively.

– Otherwise, the values of pps_conf_win_left_offset, pps_conf_win_right_offset, pps_conf_win_top_offset, and pps_conf_win_bottom_offset are inferred to be equal to 0.

The conformance cropping window contains the luma samples with horizontal picture coordinates from SubWidthC * pps_conf_win_left_offset to pps_pic_width_in_luma_samples − (SubWidthC * pps_conf_win_right_offset + 1) and vertical picture coordinates from SubHeightC * pps_conf_win_top_offset to pps_pic_height_in_luma_samples − (SubHeightC * pps_conf_win_bottom_offset + 1), inclusive.

The value of SubWidthC * (pps_conf_win_left_offset + pps_conf_win_right_offset) shall be less than pps_pic_width_in_luma_samples, and the value of SubHeightC * (pps_conf_win_top_offset + pps_conf_win_bottom_offset) shall be less than pps_pic_height_in_luma_samples.

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

NOTE 3 – The conformance cropping window offset parameters are only applied at the output. All internal decoding processes are applied to the uncropped picture size.

Let ppsA and ppsB be any two PPSs referring to the same SPS. It is a requirement of bitstream conformance that, when ppsA and ppsB have the same values of pps_pic_width_in_luma_samples and pps_pic_height_in_luma_samples, respectively, ppsA and ppsB shall have the same values of pps_conf_win_left_offset, pps_conf_win_right_offset, pps_conf_win_top_offset, and pps_conf_win_bottom_offset, respectively.

pps_scaling_window_explicit_signalling_flag equal to 1 specifies that the scaling window offset parameters are present in the PPS. pps_scaling_window_explicit_signalling_flag equal to 0 specifies that the scaling window offset parameters are not present in the PPS. When sps_ref_pic_resampling_enabled_flag is equal to 0, the value of pps_scaling_window_explicit_signalling_flag shall be equal to 0.

pps_scaling_win_left_offset, **pps_scaling_win_right_offset**, **pps_scaling_win_top_offset**, and **pps_scaling_win_bottom_offset** specify the offsets that are applied to the picture size for scaling ratio calculation. When not present, the values of pps_scaling_win_left_offset, pps_scaling_win_right_offset, pps_scaling_win_top_offset, and pps_scaling_win_bottom_offset are inferred to be equal to pps_conf_win_left_offset, pps_conf_win_right_offset, pps_conf_win_top_offset, and pps_conf_win_bottom_offset, respectively.

The values of SubWidthC * pps_scaling_win_left_offset and SubWidthC * pps_scaling_win_right_offset shall both be greater than or equal to −pps_pic_width_in_luma_samples * 15 and less than pps_pic_width_in_luma_samples. The values of SubHeightC * pps_scaling_win_top_offset and SubHeightC *pps_scaling_win_bottom_offset shall both be greater than or equal to −pps_pic_height_in_luma_samples * 15 and less than pps_pic_height_in_luma_samples.

The value of SubWidthC * (pps_scaling_win_left_offset + pps_scaling_win_right_offset) shall be greater than or equal to −pps_pic_width_in_luma_samples * 15 and less than pps_pic_width_in_luma_samples, and the value of SubHeightC * (pps_scaling_win_top_offset + pps_scaling_win_bottom_offset) shall be greater than or equal to −pps_pic_height_in_luma_samples * 15 and less than pps_pic_height_in_luma_samples.

The variables CurrPicScalWinWidthL and CurrPicScalWinHeightL are derived as follows:

CurrPicScalWinWidthL = pps_pic_width_in_luma_samples −
SubWidthC * (pps_scaling_win_right_offset + pps_scaling_win_left_offset) (73)

CurrPicScalWinHeightL = pps_pic_height_in_luma_samples −
SubHeightC * (pps_scaling_win_bottom_offset + pps_scaling_win_top_offset) (74)

Let refPicScalWinWidthL and refPicScalWinHeightL be the CurrPicScalWinWidthL and CurrPicScalWinHeightL, respectively, of a reference picture of a current picture referring to this PPS. It is a requirement of bitstream conformance that all of the following conditions shall be satisfied:

– CurrPicScalWinWidthL * 2 is greater than or equal to refPicScalWinWidthL.
– CurrPicScalWinHeightL * 2 is greater than or equal to refPicScalWinHeightL.
– CurrPicScalWinWidthL is less than or equal to refPicScalWinWidthL * 8.
– CurrPicScalWinHeightL is less than or equal to refPicScalWinHeightL * 8.
– CurrPicScalWinWidthL * sps_pic_width_max_in_luma_samples is greater than or equal to refPicScalWinWidthL * (pps_pic_width_in_luma_samples − Max(8, MinCbSizeY)).
– CurrPicScalWinHeightL * sps_pic_height_max_in_luma_samples is greater than or equal to refPicScalWinHeightL * (pps_pic_height_in_luma_samples − Max(8, MinCbSizeY)).

pps_output_flag_present_flag equal to 1 specifies that the ph_pic_output_flag syntax element could be present in PH syntax structures referring to the PPS. pps_output_flag_present_flag equal to 0 specifies that the ph_pic_output_flag syntax element is not present in PH syntax structures referring to the PPS.

pps_no_pic_partition_flag equal to 1 specifies that no picture partitioning is applied to each picture referring to the PPS. pps_no_pic_partition_flag equal to 0 specifies that each picture referring to the PPS might be partitioned into more than one tile or slice.

When sps_num_subpics_minus1 is greater than 0 or pps_mixed_nalu_types_in_pic_flag is equal to 1, the value of pps_no_pic_partition_flag shall be equal to 0.

pps_subpic_id_mapping_present_flag equal to 1 specifies that the subpicture ID mapping is signalled in the PPS. pps_subpic_id_mapping_present_flag equal to 0 specifies that the subpicture ID mapping is not signalled in the PPS. If sps_subpic_id_mapping_explicitly_signalled_flag is 0 or sps_subpic_id_mapping_present_flag is equal to 1, the value of pps_subpic_id_mapping_present_flag shall be equal to 0. Otherwise (sps_subpic_id_mapping_explicitly_signalled_flag is equal to 1 and sps_subpic_id_mapping_present_flag is equal to 0), the value of pps_subpic_id_mapping_present_flag shall be equal to 1.

pps_num_subpics_minus1 shall be equal to sps_num_subpics_minus1. When pps_no_pic_partition_flag is equal to 1, the value of pps_num_subpics_minus1 is inferred to be equal to 0.

pps_subpic_id_len_minus1 shall be equal to sps_subpic_id_len_minus1.

pps_subpic_id[i] specifies the subpicture ID of the i-th subpicture. The length of the pps_subpic_id[i] syntax element is pps_subpic_id_len_minus1 + 1 bits.

The variable SubpicIdVal[i], for each value of i in the range of 0 to sps_num_subpics_minus1, inclusive, is derived as follows:

```
for( i = 0; i <= sps_num_subpics_minus1; i++ )
  if( sps_subpic_id_mapping_explicitly_signalled_flag )
     SubpicIdVal[ i ] = pps_subpic_id_mapping_present_flag ? pps_subpic_id[ i ] : sps_subpic_id[ i ]     (75)
  else
     SubpicIdVal[ i ] = i
```

It is a requirement of bitstream conformance that, for any two different values of i and j in the range of 0 to sps_num_subpics_minus1, inclusive, SubpicIdVal[i] shall not be equal to SubpicIdVal[j].

pps_log2_ctu_size_minus5 plus 5 specifies the luma coding tree block size of each CTU. pps_log2_ctu_size_minus5 shall be equal to sps_log2_ctu_size_minus5.

pps_num_exp_tile_columns_minus1 plus 1 specifies the number of explicitly provided tile column widths. The value of pps_num_exp_tile_columns_minus1 shall be in the range of 0 to PicWidthInCtbsY − 1, inclusive. When pps_no_pic_partition_flag is equal to 1, the value of pps_num_exp_tile_columns_minus1 is inferred to be equal to 0.

pps_num_exp_tile_rows_minus1 plus 1 specifies the number of explicitly provided tile row heights. The value of pps_num_exp_tile_rows_minus1 shall be in the range of 0 to PicHeightInCtbsY − 1, inclusive. When pps_no_pic_partition_flag is equal to 1, the value of pps_num_exp_tile_rows_minus1 is inferred to be equal to 0.

pps_tile_column_width_minus1[i] plus 1 specifies the width of the i-th tile column in units of CTBs for i in the range of 0 to pps_num_exp_tile_columns_minus1, inclusive. pps_tile_column_width_minus1[pps_num_exp_tile_ columns_minus1] is also used to derive the widths of the tile columns with index greater than pps_num_exp_tile_columns_minus1 as specified in clause 6.5.1. The value of pps_tile_column_width_minus1[i] shall be in the range of 0 to PicWidthInCtbsY − 1, inclusive. When not present, the value of pps_tile_column_width_minus1[0] is inferred to be equal to PicWidthInCtbsY − 1.

pps_tile_row_height_minus1[i] plus 1 specifies the height of the i-th tile row in units of CTBs for i in the range of 0 to pps_num_exp_tile_rows_minus1, inclusive. pps_tile_row_height_minus1[pps_num_exp_tile_rows_minus1] is also used to derive the heights of the tile rows with index greater than pps_num_exp_tile_rows_minus1 as specified in clause 6.5.1. The value of pps_tile_row_height_minus1[i] shall be in the range of 0 to PicHeightInCtbsY − 1, inclusive. When not present, the value of pps_tile_row_height_minus1[0] is inferred to be equal to PicHeightInCtbsY − 1.

pps_loop_filter_across_tiles_enabled_flag equal to 1 specifies that in-loop filtering operations across tile boundaries are enabled for pictures referring to the PPS. pps_loop_filter_across_tiles_enabled_flag equal to 0 specifies that in-loop filtering operations across tile boundaries are disabled for pictures referring to the PPS. The in-loop filtering operations include the deblocking filter, SAO, and ALF operations. When not present, the value of pps_loop_filter_across_tiles_enabled_flag is inferred to be equal to 0.

pps_rect_slice_flag equal to 0 specifies that the raster-scan slice mode is in use for each picture referring to the PPS and the slice layout is not signalled in PPS. pps_rect_slice_flag equal to 1 specifies that the rectangular slice mode is in use for each picture referring to the PPS and the slice layout is signalled in the PPS. When not present, the value of pps_rect_slice_flag is inferred to be equal to 1. When sps_subpic_info_present_flag is equal to 1 or pps_mixed_nalu_types_in_pic_flag is equal to 1, the value of pps_rect_slice_flag shall be equal to 1.

pps_single_slice_per_subpic_flag equal to 1 specifies that each subpicture consists of one and only one rectangular slice. pps_single_slice_per_subpic_flag equal to 0 specifies that each subpicture could consist of one or more rectangular slices. When pps_no_pic_partition_flag is equal to 1, the value of pps_single_slice_per_subpic_flag is inferred to be equal to 1.

NOTE 4 – When there is only one subpicture per picture, pps_single_slice_per_subpic_flag equal to 1 means that there is only one slice per picture.

pps_num_slices_in_pic_minus1 plus 1 specifies the number of rectangular slices in each picture referring to the PPS. The value of pps_num_slices_in_pic_minus1 shall be in the range of 0 to MaxSlicesPerAu − 1, inclusive, where MaxSlicesPerAu is specified in Annex A. When pps_no_pic_partition_flag is equal to 1, the value of pps_num_slices_in_pic_minus1 is inferred to be equal to 0. When pps_single_slice_per_subpic_flag is equal to 1, the value of pps_num_slices_in_pic_minus1 is inferred to be equal to sps_num_subpics_minus1.

pps_tile_idx_delta_present_flag equal to 0 specifies that pps_tile_idx_delta_val[i] syntax elements are not present in the PPS and all pictures referring to the PPS are partitioned into rectangular slice rows and rectangular slice columns in slice raster order. pps_tile_idx_delta_present_flag equal to 1 specifies that pps_tile_idx_delta_val[i] syntax elements could be present in the PPS and all rectangular slices in pictures referring to the PPS are specified in the order indicated by the values of the pps_tile_idx_delta_val[i] in increasing values of i. When not present, the value of pps_tile_idx_delta_present_flag is inferred to be equal to 0.

pps_slice_width_in_tiles_minus1[i] plus 1 specifies the width of the i-th rectangular slice in units of tile columns. The value of pps_slice_width_in_tiles_minus1[i] shall be in the range of 0 to NumTileColumns − 1, inclusive. When not present, the value of pps_slice_width_in_tiles_minus1[i] is inferred to be equal to 0.

pps_slice_height_in_tiles_minus1[i] plus 1 specifies the height of the i-th rectangular slice in units of tile rows when pps_num_exp_slices_in_tile[i] is equal to 0. The value of pps_slice_height_in_tiles_minus1[i] shall be in the range of 0 to NumTileRows − 1, inclusive.

When pps_slice_height_in_tiles_minus1[i] is not present, it is inferred as follows:

– If SliceTopLeftTileIdx[i] / NumTileColumns is equal to NumTileRows − 1, the value of pps_slice_height_in_tiles_minus1[i] is inferred to be equal to 0.

– Otherwise, the value of pps_slice_height_in_tiles_minus1[i] is inferred to be equal to pps_slice_height_in_tiles_minus1[i − 1].

pps_num_exp_slices_in_tile[i] specifies the number of explicitly provided slice heights for the slices in the tile containing the i-th slice (i.e., the tile with tile index equal to SliceTopLeftTileIdx[i]). The value of pps_num_exp_slices_in_tile[i] shall be in the range of 0 to RowHeightVal[SliceTopLeftTileIdx[i] / NumTileColumns] − 1, inclusive. When not present, the value of pps_num_exp_slices_in_tile[i] is inferred to be equal to 0.

NOTE 5 – If pps_num_exp_slices_in_tile[i] is equal to 0, the tile containing the i-th slice is not split into multiple slices. Otherwise (pps_num_exp_slices_in_tile[i] is greater than 0), the tile containing the i-th slice might or might not be split into multiple slices.

pps_exp_slice_height_in_ctus_minus1[i][j] plus 1 specifies the height of the j-th rectangular slice in the tile containing the i-th slice, in units of CTU rows, for j in the range of 0 to pps_num_exp_slices_in_tile[i] − 1, inclusive, when pps_num_exp_slices_in_tile[i] is greater than 0. pps_exp_slice_height_in_ctus_minus1[i][pps_num_exp_slices_in_tile[i]] is also used to derive the heights of the rectangular slices in the tile containing the i-th slice with index greater than pps_num_exp_slices_in_tile[i] − 1 as specified in clause 6.5.1. The value of pps_exp_slice_height_in_ctus_minus1[i][j] shall be in the range of 0 to RowHeightVal[SliceTopLeftTileIdx[i] / NumTileColumns] − 1, inclusive.

pps_tile_idx_delta_val[i] specifies the difference between the tile index of the tile containing the first CTU in the (i + 1)-th rectangular slice and the tile index of the tile containing the first CTU in the i-th rectangular slice. The value of pps_tile_idx_delta_val[i] shall be in the range of −NumTilesInPic + 1 to NumTilesInPic − 1, inclusive. When not present, the value of pps_tile_idx_delta_val[i] is inferred to be equal to 0. When present, the value of pps_tile_idx_delta_val[i] shall not be equal to 0.

When pps_rect_slice_flag is equal to 1, it is a requirement of bitstrream conformance that, for any two slices, with picture-level slice indices idxA and idxB, that belong to the same picture and different subpictures, when SubpicIdxForSlice[idxA] is less than SubpicIdxForSlice[idxB], the value of idxA shall be less than idxB.

pps_loop_filter_across_slices_enabled_flag equal to 1 specifies that in-loop filtering operations across slice boundaries are enabled for pictures referring to the PPS. loop_filter_across_slice_enabled_flag equal to 0 specifies that in-loop filtering operations across slice boundaries are disabled for the PPS. The in-loop filtering operations include the deblocking filter, SAO, and ALF operations. When not present, the value of pps_loop_filter_across_slices_enabled_flag is inferred to be equal to 0.

pps_cabac_init_present_flag equal to 1 specifies that sh_cabac_init_flag is present in slice headers referring to the PPS. pps_cabac_init_present_flag equal to 0 specifies that sh_cabac_init_flag is not present in slice headers referring to the PPS.

pps_num_ref_idx_default_active_minus1[i] plus 1, when i is equal to 0, specifies the inferred value of the variable NumRefIdxActive[0] for P or B slices with sh_num_ref_idx_active_override_flag equal to 0, and, when i is equal to 1, specifies the inferred value of NumRefIdxActive[1] for B slices with sh_num_ref_idx_active_override_flag equal to 0. The value of pps_num_ref_idx_default_active_minus1[i] shall be in the range of 0 to 14, inclusive.

pps_rpl1_idx_present_flag equal to 0 specifies that rpl_sps_flag[1] and rpl_idx[1] are not present in the PH syntax structures or the slice headers for pictures referring to the PPS. pps_rpl1_idx_present_flag equal to 1 specifies that rpl_sps_flag[1] and rpl_idx[1] could be present in the PH syntax structures or the slice headers for pictures referring to the PPS.

pps_weighted_pred_flag equal to 0 specifies that weighted prediction is not applied to P slices referring to the PPS. pps_weighted_pred_flag equal to 1 specifies that weighted prediction is applied to P slices referring to the PPS. When sps_weighted_pred_flag is equal to 0, the value of pps_weighted_pred_flag shall be equal to 0.

pps_weighted_bipred_flag equal to 0 specifies that explicit weighted prediction is not applied to B slices referring to the PPS. pps_weighted_bipred_flag equal to 1 specifies that explicit weighted prediction is applied to B slices referring to the PPS. When sps_weighted_bipred_flag is equal to 0, the value of pps_weighted_bipred_flag shall be equal to 0.

pps_ref_wraparound_enabled_flag equal to 1 specifies that the horizontal wrap-around motion compensation is enabled for pictures referring to the PPS. pps_ref_wraparound_enabled_flag equal to 0 specifies that the horizontal wrap-around motion compensation is disabled for pictures referring to the PPS.

When sps_ref_wraparound_enabled_flag is equal to 0 or the value of CtbSizeY / MinCbSizeY + 1 is greater than pps_pic_width_in_luma_samples / MinCbSizeY − 1, the value of pps_ref_wraparound_enabled_flag shall be equal to 0.

pps_pic_width_minus_wraparound_offset specifies the difference between the picture width and the offset used for computing the horizontal wrap-around position in units of MinCbSizeY luma samples. The value of pps_pic_width_minus_wraparound_offset shall be less than or equal to (pps_pic_width_in_luma_samples / MinCbSizeY) − (CtbSizeY / MinCbSizeY) − 2.

The variable PpsRefWraparoundOffset is set equal to pps_pic_width_in_luma_samples / MinCbSizeY − pps_pic_width_minus_wraparound_offset.

pps_init_qp_minus26 plus 26 specifies the initial value of $SliceQp_Y$ for each slice referring to the PPS. The initial value of $SliceQp_Y$ is modified at the picture level when a non-zero value of ph_qp_delta is decoded or at the slice level when a non-zero value of sh_qp_delta is decoded. The value of pps_init_qp_minus26 shall be in the range of −(26 + QpBdOffset) to +37, inclusive.

pps_cu_qp_delta_enabled_flag equal to 1 specifies that either or both of the ph_cu_qp_delta_subdiv_intra_slice and ph_cu_qp_delta_subdiv_inter_slice syntax elements are present in PH syntax structures referring to the PPS, and the cu_qp_delta_abs and cu_qp_delta_sign_flag syntax elements could be present in the transform unit syntax and the palette coding syntax. pps_cu_qp_delta_enabled_flag equal to 0 specifies that the ph_cu_qp_delta_subdiv_intra_slice and ph_cu_qp_delta_subdiv_inter_slice syntax elements are not present in PH syntax structures referring to the PPS, and the cu_qp_delta_abs and cu_qp_delta_sign_flag syntax elements are not present in the transform unit syntax or the palette coding syntax.

pps_chroma_tool_offsets_present_flag equal to 1 specifies that chroma tool offsets related syntax elements are present in the PPS RBSP syntax structure and the chroma deblocking t_c and β offset syntax elements could be present in the PH syntax structures or the SHs of pictures referring to the PPS. pps_chroma_tool_offsets_present_flag equal to 0 specifies that chroma tool offsets related syntax elements are not present in the PPS RBSP syntax structure and the chroma deblocking t_c and β offset syntax elements are not present in the PH syntax structures or the SHs of pictures referring to the PPS. When sps_chroma_format_idc is equal to 0, the value of pps_chroma_tool_offsets_present_flag shall be equal to 0.

pps_cb_qp_offset and **pps_cr_qp_offset** specify the offsets to the luma quantization parameter Qp'_Y used for deriving Qp'_{Cb} and Qp'_{Cr}, respectively. The values of pps_cb_qp_offset and pps_cr_qp_offset shall be in the range of −12 to +12, inclusive. When sps_chroma_format_idc is equal to 0, pps_cb_qp_offset and pps_cr_qp_offset are not used in the decoding process and decoders shall ignore their value. When not present, the values of pps_cb_qp_offset and pps_cr_qp_offset are inferred to be equal to 0.

pps_joint_cbcr_qp_offset_present_flag equal to 1 specifies that pps_joint_cbcr_qp_offset_value and pps_joint_cbcr_qp_offset_list[i] are present in the PPS RBSP syntax structure. pps_joint_cbcr_qp_offset_present_flag equal to 0 specifies that pps_joint_cbcr_qp_offset_value and pps_joint_cbcr_qp_offset_list[i] are not present in the PPS RBSP syntax structure. When sps_chroma_format_idc is equal to 0 or sps_joint_cbcr_enabled_flag is equal to 0, the value of pps_joint_cbcr_qp_offset_present_flag shall be equal to 0. When not present, the value of pps_joint_cbcr_qp_offset_present_flag is inferred to be equal to 0.

pps_joint_cbcr_qp_offset_value specifies the offset to the luma quantization parameter Qp'_Y used for deriving Qp'_{CbCr}. The value of pps_joint_cbcr_qp_offset_value shall be in the range of −12 to +12, inclusive. When sps_chroma_format_idc is equal to 0 or sps_joint_cbcr_enabled_flag is equal to 0, pps_joint_cbcr_qp_offset_value is not used in the decoding process and decoders shall ignore its value. When pps_joint_cbcr_qp_offset_present_flag is equal to 0, pps_joint_cbcr_qp_offset_value is not present and is inferred to be equal to 0.

pps_slice_chroma_qp_offsets_present_flag equal to 1 specifies that the sh_cb_qp_offset and sh_cr_qp_offset syntax elements are present in the associated slice headers. pps_slice_chroma_qp_offsets_present_flag equal to 0 specifies that the sh_cb_qp_offset and sh_cr_qp_offset syntax elements are not present in the associated slice headers. When not present, the value of pps_slice_chroma_qp_offsets_present_flag is inferred to be equal to 0.

pps_cu_chroma_qp_offset_list_enabled_flag equal to 1 specifies that the ph_cu_chroma_qp_offset_subdiv_intra_slice and ph_cu_chroma_qp_offset_subdiv_inter_slice syntax elements are present in PH syntax structures referring to the PPS and cu_chroma_qp_offset_flag could be present in the transform unit syntax and the palette coding syntax. pps_cu_chroma_qp_offset_list_enabled_flag equal to 0 specifies that the ph_cu_chroma_qp_offset_subdiv_intra_slice and ph_cu_chroma_qp_offset_subdiv_inter_slice syntax elements are not present in PH syntax structures referring to the PPS and the cu_chroma_qp_offset_flag is not present in the transform unit syntax and the palette coding syntax. When not present, the value of pps_cu_chroma_qp_offset_list_enabled_flag is inferred to be equal to 0.

pps_chroma_qp_offset_list_len_minus1 plus 1 specifies the number of pps_cb_qp_offset_list[i], pps_cr_qp_offset_list[i], and pps_joint_cbcr_qp_offset_list[i], syntax elements that are present in the PPS RBSP syntax structure. The value of pps_chroma_qp_offset_list_len_minus1 shall be in the range of 0 to 5, inclusive.

pps_cb_qp_offset_list[i], **pps_cr_qp_offset_list**[i], and **pps_joint_cbcr_qp_offset_list**[i], specify offsets used in the derivation of Qp'_{Cb}, Qp'_{Cr}, and Qp'_{CbCr}, respectively. The values of pps_cb_qp_offset_list[i], pps_cr_qp_offset_list[i], and pps_joint_cbcr_qp_offset_list[i] shall be in the range of −12 to +12, inclusive. When pps_joint_cbcr_qp_offset_present_flag is equal to 0, pps_joint_cbcr_qp_offset_list[i] is not present and it is inferred to be equal to 0.

pps_deblocking_filter_control_present_flag equal to 1 specifies the presence of deblocking filter control syntax elements in the PPS. pps_deblocking_filter_control_present_flag equal to 0 specifies the absence of deblocking filter control syntax elements in the PPS and that the deblocking filter is applied for all slices referring to the PPS, using 0-valued deblocking β and t_C offsets.

pps_deblocking_filter_override_enabled_flag equal to 1 specifies that the deblocking behaviour for pictures referring to the PPS could be overridden in the picture level or slice level. pps_deblocking_filter_override_enabled_flag equal to 0 specifies that the deblocking behaviour for pictures referring to the PPS is not overridden in the picture level or slice level. When not present, the value of pps_deblocking_filter_override_enabled_flag is inferred to be equal to 0.

pps_deblocking_filter_disabled_flag equal to 1 specifies that the deblocking filter is disabled for pictures referring to the PPS unless overridden for a picture or slice by information present the PH or SH, respectively. pps_deblocking_filter_disabled_flag equal to 0 specifies that the deblocking filter is enabled for pictures referring to the PPS unless overridden for a picture or slice by information present the PH or SH, respectively. When not present, the value of pps_deblocking_filter_disabled_flag is inferred to be equal to 0.

NOTE 6 – When pps_deblocking_filter_disabled_flag equal is equal to 1 for a slice, the deblocking filter is disabled for the slice when one of the following two conditions is true: 1) ph_deblocking_filter_disabled_flag and sh_deblocking_filter_disabled_flag are not present and inferred to be equal to 1 and 2) ph_deblocking_filter_disabled_flag or sh_deblocking_filter_disabled_flag is present and equal to 1, and the deblocking filter is enabled for the slice when one of the following two conditions is true: 1) ph_deblocking_filter_disabled_flag and sh_deblocking_filter_disabled_flag are not present and inferred to be equal to 0 and 2) ph_deblocking_filter_disabled_flag or sh_deblocking_filter_disabled_flag is present and equal to 0.

NOTE 7 – When pps_deblocking_filter_disabled_flag is equal to 0 for a slice, the deblocking filter is enabled for the slice when one of the following two conditions is true: 1) ph_deblocking_filter_disabled_flag and sh_deblocking_filter_disabled_flag are not present and 2) ph_deblocking_filter_disabled_flag or sh_deblocking_filter_disabled_flag is present and equal to 0, and the deblocking filter is disabled for the slice when ph_deblocking_filter_disabled_flag or sh_deblocking_filter_disabled_flag is present and equal to 1.

pps_dbf_info_in_ph_flag equal to 1 specifies that deblocking filter information is present in the PH syntax structure and not present in slice headers referring to the PPS that do not contain a PH syntax structure. pps_dbf_info_in_ph_flag equal to 0 specifies that deblocking filter information is not present in the PH syntax structure and could be present in slice headers referring to the PPS. When not present, the value of pps_dbf_info_in_ph_flag is inferred to be equal to 0.

pps_luma_beta_offset_div2 and **pps_luma_tc_offset_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the luma component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of pps_luma_beta_offset_div2 and pps_luma_tc_offset_div2 shall both be in the range of −12 to 12, inclusive. When not present, the values of pps_luma_beta_offset_div2 and pps_luma_tc_offset_div2 are both inferred to be equal to 0.

pps_cb_beta_offset_div2 and **pps_cb_tc_offset_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of pps_cb_beta_offset_div2 and pps_cb_tc_offset_div2 shall both be in the range of −12 to 12, inclusive. When not present, the values of pps_cb_beta_offset_div2 and pps_cb_tc_offset_div2 are inferred to be equal to pps_luma_beta_offset_div2 and pps_luma_tc_offset_div2, respectively.

pps_cr_beta_offset_div2 and **pps_cr_tc_offset_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of pps_cr_beta_offset_div2 and pps_cr_tc_offset_div2 shall both be in the range of −12 to 12, inclusive. When not present, the values of pps_cr_beta_offset_div2 and pps_cr_tc_offset_div2 are inferred to be equal to pps_luma_beta_offset_div2 and pps_luma_tc_offset_div2, respectively.

pps_rpl_info_in_ph_flag equal to 1 specifies that RPL information is present in the PH syntax structure and not present in slice headers referring to the PPS that do not contain a PH syntax structure. pps_rpl_info_in_ph_flag equal to 0 specifies that RPL information is not present in the PH syntax structure and could be present in slice headers referring to the PPS. When not present, the value of pps_rpl_info_in_ph_flag is inferred to be equal to 0.

pps_sao_info_in_ph_flag equal to 1 specifies that SAO filter information could be present in the PH syntax structure and is not present in slice headers referring to the PPS that do not contain a PH syntax structure. pps_sao_info_in_ph_flag equal to 0 specifies that SAO filter information is not present in the PH syntax structure and could be present in slice headers referring to the PPS. When not present, the value of pps_sao_info_in_ph_flag is inferred to be equal to 0.

pps_alf_info_in_ph_flag equal to 1 specifies that ALF information could be present in the PH syntax structure and is not present in slice headers referring to the PPS that do not contain a PH syntax structure. pps_alf_info_in_ph_flag equal to 0 specifies that ALF information is not present in the PH syntax structure and could be present in slice headers referring to the PPS. When not present, the value of pps_alf_info_in_ph_flag is inferred to be equal to 0.

pps_wp_info_in_ph_flag equal to 1 specifies that weighted prediction information could be present in the PH syntax structure and is not present in slice headers referring to the PPS that do not contain a PH syntax structure. pps_wp_info_in_ph_flag equal to 0 specifies that weighted prediction information is not present in the PH syntax structure and could be present in slice headers referring to the PPS. When not present, the value of pps_wp_info_in_ph_flag is inferred to be equal to 0.

pps_qp_delta_info_in_ph_flag equal to 1 specifies that QP delta information is present in the PH syntax structure and not present in slice headers referring to the PPS that do not contain a PH syntax structure. pps_qp_delta_info_in_ph_flag equal to 0 specifies that QP delta information is not present in the PH syntax structure and is present in slice headers referring to the PPS. When not present, the value of pps_qp_delta_info_in_ph_flag is inferred to be equal to 0.

pps_picture_header_extension_present_flag equal to 0 specifies that no PH extension syntax elements are present in PH syntax structures referring to the PPS. pps_picture_header_extension_present_flag equal to 1 specifies that PH extension syntax elements are present in PH syntax structures referring to the PPS. pps_picture_header_extension_present_flag shall be equal to 0 in bitstreams conforming to this version of this Specification. However, some use of pps_picture_header_extension_present_flag equal to 1 could be specified in some future version of this Specification, and decoders conforming to this version of this Specification shall also allow the value of pps_picture_header_extension_present_flag equal to 1 to appear in the syntax.

pps_slice_header_extension_present_flag equal to 0 specifies that no slice header extension syntax elements are present in the slice headers for coded pictures referring to the PPS. pps_slice_header_extension_present_flag equal to 1 specifies that slice header extension syntax elements are present in the slice headers for coded pictures referring to the PPS. pps_slice_header_extension_present_flag shall be equal to 0 in bitstreams conforming to this version of this Specification. However, some use of pps_slice_header_extension_present_flag equal to 1 could be specified in some future version of this Specification, and decoders conforming to this version of this Specification shall also allow the value of pps_slice_header_extension_present_flag equal to 1 to appear in the syntax.

pps_extension_flag equal to 0 specifies that no pps_extension_data_flag syntax elements are present in the PPS RBSP syntax structure. pps_extension_flag equal to 1 specifies that pps_extension_data_flag syntax elements might be present in the PPS RBSP syntax structure. pps_extension_flag shall be equal to 0 in bitstreams conforming to this version of this Specification. However, some use of pps_extension_flag equal to 1 could be specified in some future version of this Specification, and decoders conforming to this version of this Specification shall also allow the value of pps_extension_flag equal to 1 to appear in the syntax.

pps_extension_data_flag could have any value. Its presence and value do not affect the decoding process specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore the pps_extension_data_flag syntax elements.

7.4.3.6 Adaptation parameter set semantics

Each APS RBSP shall be available to the decoding process, by inclusion in at least one AU with TemporalId less than or equal to the TemporalId of the coded slice NAL unit that refers it or provided through external means, prior to it being referenced by any of the following:

– a coded slice NAL unit in a PU with a PH NAL unit having a ph_alf_aps_id_luma[i] or ph_alf_aps_id_chroma or ph_alf_cc_cb_aps_id or ph_alf_cc_cr_aps_id syntax element that is present and equal to the aps_adaptation_parameter_set_id of an APS RBSP with aps_params_type equal to ALF_APS,

– a coded slice NAL unit in a PU having a PH NAL unit with a ph_lmcs_aps_id syntax element that is present and equal to the aps_adaptation_parameter_set_id of an APS RBSP with aps_params_type equal to LMCS_APS,

– a coded slice NAL unit in a PU having a PH NAL unit with a ph_scaling_list_aps_id syntax element that is present and equal to the aps_adaptation_parameter_set_id of an APS RBSP with aps_params_type equal to SCALING_APS,

– a coded slice NAL unit having a sh_alf_aps_id_luma[i] or sh_alf_aps_id_chroma or sh_alf_cc_cb_aps_id or sh_alf_cc_cr_aps_id syntax element that is present and equal to the aps_adaptation_parameter_set_id of an APS RBSP with aps_params_type equal to ALF_APS,

– a coded slice NAL unit having sh_picture_header_in_slice_header_flag equal to 1 with a ph_alf_aps_id_luma[i] or ph_alf_aps_id_chroma or ph_alf_cc_cb_aps_id or ph_alf_cc_cr_aps_id syntax element that is present and equal to the aps_adaptation_parameter_set_id of an APS RBSP with aps_params_type equal to ALF_APS,

– a coded slice NAL unit having sh_picture_header_in_slice_header_flag equal to 1 with a ph_lmcs_aps_id syntax element that is present and equal to the aps_adaptation_parameter_set_id of an APS RBSP with aps_params_type equal to LMCS_APS,

– a coded slice NAL unit having sh_picture_header_in_slice_header_flag equal to 1 with a ph_scaling_list_aps_id syntax element that is present and equal to the aps_adaptation_parameter_set_id of an APS RBSP with aps_params_type equal to SCALING_APS.

Such a coded slice NAL unit references the previous APS RBSP in decoding order (relative to the position of the coded slice NAL unit in decoding order) with the corresponding values of aps_adaptation_parameter_set_id and aps_params_type.

All APS NAL units with a particular value of nal_unit_type, a particular value of aps_adaptation_parameter_set_id, and a particular value of aps_params_type within a PU shall have the same content.

NOTE 1 – The content of an APS RBSP in a suffix APS NAL unit and the content of a prefix APS NAL unit with the same values of aps_adaptation_parameter_set_id (and aps_params_type) in the same PU can be different. When a suffix APS NAL unit is present in a PU, its APS RBSP cannot be referenced by the decoding process of that PU, since the suffix APS NAL unit cannot precede the PH NAL unit or any coded slice NAL units of that PU (see clause 7.4.2.4.4). However, the APS RBSP in a suffix APS NAL unit can be referenced by the decoding process of subsequent PUs in the bitstream (if any).

aps_params_type specifies the type of APS parameters carried in the APS as specified in Table 6. The value of aps_params_type shall be in the range of 0 to 2, inclusive, in bitstreams conforming to this version of this Specification. Other values of aps_params_type are reserved for future use by ITU-T | ISO/IEC. Decoders conforming to this version of this Specification shall ignore APS NAL units with reserved values of aps_params_type.

Table 6 – APS parameters type codes and types of APS parameters

aps_params_type	Name of aps_params_type	Type of APS parameters
0	ALF_APS	ALF parameters
1	LMCS_APS	LMCS parameters
2	SCALING_APS	Scaling list parameters

All APS NAL units with a particular value of aps_params_type, regardless of the nuh_layer_id values and whether they are prefix or suffix APS NAL units, share the same value space for aps_adaptation_parameter_set_id. APS NAL units with different values of aps_params_type use separate values spaces for aps_adaptation_parameter_set_id.

aps_adaptation_parameter_set_id provides an identifier for the APS for reference by other syntax elements.

When aps_params_type is equal to ALF_APS or SCALING_APS, the value of aps_adaptation_parameter_set_id shall be in the range of 0 to 7, inclusive.

When aps_params_type is equal to LMCS_APS, the value of aps_adaptation_parameter_set_id shall be in the range of 0 to 3, inclusive.

Let apsLayerId be the value of the nuh_layer_id of a particular APS NAL unit, and vclLayerId be the value of the nuh_layer_id of a particular VCL NAL unit. The particular VCL NAL unit shall not refer to the particular APS NAL unit unless apsLayerId is less than or equal to vclLayerId and all OLSs specified by the VPS that contain the layer with nuh_layer_id equal to vclLayerId also contain the layer with nuh_layer_id equal to apsLayerId.

NOTE 2 – In a CVS that contains only one layer, the nuh_layer_id of referenced APSs is equal to the nuh_layer_id of the VCL NAL units.

NOTE 3 – An APS NAL unit (with a particular value of nal_unit_type, a particular value of aps_adaptation_parameter_set_id, and a particular value of aps_params_type) could be shared across pictures, and different slices within a picture can refer to different ALF APSs.

NOTE 4 – A suffix APS NAL unit associated with a particular VCL NAL unit (a VCL NAL unit that precedes the suffix APS NAL unit in decoding order and is the last VCL NAL unit of the PU containing that VCL NAL unit) is not for use in the decoding process of that particular VCL NAL unit or any other VCL NAL unit of the PU containing that particular VCL NAL unit, but rather is for use in the decoding process of VCL NAL units of PUs that follow the suffix APS NAL unit in decoding order (if any).

aps_chroma_present_flag equal to 1 specifies that the APS NAL unit could include chroma related syntax elements. aps_chroma_present_flag equal to 0 specifies that the APS NAL unit does not include chroma related syntax elements.

aps_extension_flag equal to 0 specifies that no aps_extension_data_flag syntax elements are present in the APS RBSP syntax structure. aps_extension_flag equal to 1 specifies that aps_extension_data_flag syntax elements might be present in the APS RBSP syntax structure. aps_extension_data_flag shall be equal to 0 in bitstreams conforming to this version of this Specification. However, some use of aps_extension_data_flag equal to 1 could be specified in some future version of this Specification, and decoders conforming to this version of this Specification shall also allow the value of aps_extension_data_flag equal to 1 to appear in the syntax.

aps_extension_data_flag could have any value. Its presence and value do not affect the decoding process specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore the aps_extension_data_flag syntax elements.

7.4.3.7 Picture header RBSP semantics

The PH RBSP contains a PH syntax structure, i.e., picture_header_structure( ).

7.4.3.8 Picture header structure semantics

The PH syntax structure contains information that is common for all slices of the current picture.

ph_gdr_or_irap_pic_flag equal to 1 specifies that the current picture is a GDR or IRAP picture. ph_gdr_or_irap_pic_flag equal to 0 specifies that the current picture is not a GDR picture and might or might not be an IRAP picture.

ph_non_ref_pic_flag equal to 1 specifies that the current picture is never used as a reference picture. ph_non_ref_pic_flag equal to 0 specifies that the current picture might or might not be used as a reference picture.

ph_gdr_pic_flag equal to 1 specifies that the current picture is a GDR picture. ph_gdr_pic_flag equal to 0 specifies that the current picture is not a GDR picture. When not present, the value of ph_gdr_pic_flag is inferred to be equal to 0. When sps_gdr_enabled_flag is equal to 0, the value of ph_gdr_pic_flag shall be equal to 0.

NOTE 1 – When ph_gdr_or_irap_pic_flag is equal to 1 and ph_gdr_pic_flag is equal to 0, the current picture is an IRAP picture.

ph_inter_slice_allowed_flag equal to 0 specifies that all coded slices of the picture have sh_slice_type equal to 2. ph_inter_slice_allowed_flag equal to 1 specifies that there might or might not be one or more coded slices in the picture that have sh_slice_type equal to 0 or 1.

When ph_gdr_or_irap_pic_flag is equal to 1 and ph_gdr_pic_flag is equal to 0 (i.e., the picture is an IRAP picture), and vps_independent_layer_flag[GeneralLayerIdx[nuh_layer_id]] is equal to 1, the value of ph_inter_slice_allowed_flag shall be equal to 0.

ph_intra_slice_allowed_flag equal to 0 specifies that all coded slices of the picture have sh_slice_type equal to 0 or 1. ph_intra_slice_allowed_flag equal to 1 specifies that there might or might not be one or more coded slices in the picture that have sh_slice_type equal to 2.When not present, the value of ph_intra_slice_allowed_flag is inferred to be equal to 1.

NOTE 2 – For bitstreams that are suppposed to work for subpicture based bitstream merging without the need of changing PH NAL units, the encoder is expected to set the values of both ph_inter_slice_allowed_flag and ph_intra_slice_allowed_flag equal to 1.

ph_pic_parameter_set_id specifies the value of pps_pic_parameter_set_id for the PPS in use. The value of ph_pic_parameter_set_id shall be in the range of 0 to 63, inclusive.

It is a requirement of bitstream conformance that the value of TemporalId of the PH shall be greater than or equal to the value of TemporalId of the PPS that has pps_pic_parameter_set_id equal to ph_pic_parameter_set_id.

ph_pic_order_cnt_lsb specifies the picture order count modulo MaxPicOrderCntLsb for the current picture. The length of the ph_pic_order_cnt_lsb syntax element is sps_log2_max_pic_order_cnt_lsb_minus4 + 4 bits. The value of the ph_pic_order_cnt_lsb shall be in the range of 0 to MaxPicOrderCntLsb − 1, inclusive.

ph_recovery_poc_cnt specifies the recovery point of decoded pictures in output order.

When the current picture is a GDR picture, the variable recoveryPointPocVal is derived as follows:

$$\text{recoveryPointPocVal} = \text{PicOrderCntVal} + \text{ph\_recovery\_poc\_cnt} \quad (76)$$

If the current picture is a GDR picture and ph_recovery_poc_cnt is equal to 0, the current picture itself is also referred to as the recovery point picture. Otherwise, if the current picture is a GDR picture, and there is a picture picA that follows the current GDR picture in decoding order in the CLVS that has PicOrderCntVal equal to recoveryPointPocVal, the picture picA is referred to as the recovery point picture, otherwise, the first picture in output order that has PicOrderCntVal greater than recoveryPointPocVal in the CLVS is referred to as the recovery point picture. The recovery point picture shall not precede the current GDR picture in decoding order. The pictures that are associated with the current GDR picture and have PicOrderCntVal less than recoveryPointPocVal are referred to as the recovering pictures of the GDR picture. The value of ph_recovery_poc_cnt shall be in the range of 0 to MaxPicOrderCntLsb − 1, inclusive.

NOTE 3 – When sps_gdr_enabled_flag is equal to 1 and PicOrderCntVal of the current picture is greater than or equal to recoveryPointPocVal of the associated GDR picture, the current and subsequent decoded pictures in output order are exact match to the corresponding pictures produced by starting the decoding process from the previous IRAP picture, when present, preceding the associated GDR picture in decoding order.

ph_extra_bit[i] could have any value. Decoders conforming to this version of this Specification shall ignore the presence and value of ph_extra_bit[i]. Its value does not affect the decoding process specified in this version of this Specification.

ph_poc_msb_cycle_present_flag equal to 1 specifies that the syntax element ph_poc_msb_cycle_val is present in the PH syntax structure. ph_poc_msb_cycle_present_flag equal to 0 specifies that the syntax element ph_poc_msb_cycle_val is not present in the PH syntax structure. When vps_independent_layer_flag[GeneralLayerIdx[nuh_layer_id]] is equal to 0 and there is an ILRP entry in RefPicList[0] or RefPicList[1] of a slice of the current picture, the value of ph_poc_msb_cycle_present_flag shall be equal to 0.

ph_poc_msb_cycle_val specifies the value of the POC MSB cycle of the current picture. The length of the syntax element ph_poc_msb_cycle_val is sps_poc_msb_cycle_len_minus1 + 1 bits.

When present, **ph_alf_enabled_flag** equal to 1 specifies that the adaptive loop filter is enabled for the current picture, and ph_alf_enabled_flag equal to 0 specifies that the adaptive loop filter is disabled for the current picture. When not present, the value of ph_alf_enabled_flag is inferred to be equal to 0.

ph_num_alf_aps_ids_luma specifies the number of ALF APSs that the slices in the current picture refers to.

ph_alf_aps_id_luma[i] specifies the aps_adaptation_parameter_set_id of the i-th ALF APS that the luma component of the slices in the current picture refers to.

When ph_alf_aps_id_luma[i] is present, the following applies:

– The value of alf_luma_filter_signal_flag of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to ph_alf_aps_id_luma[i] shall be equal to 1.

– The TemporalId of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to ph_alf_aps_id_luma[i] shall be less than or equal to the TemporalId of the current picture.

– When sps_chroma_format_idc is equal to 0, the value of aps_chroma_present_flag of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to ph_alf_aps_id_luma[i] shall be equal to 0.

– When sps_ccalf_enabled_flag is equal to 0, the values of alf_cc_cb_filter_signal_flag and alf_cc_cr_filter_signal_flag of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to ph_alf_aps_id_luma[i] shall be equal to 0.

When present, **ph_alf_cb_enabled_flag** equal to 1 specifies that the adaptive loop filter is enabled for the Cb colour component of the current picture, and ph_alf_cb_enabled_flag equal to 0 specifies that the adaptive loop filter is disabled for the Cb colour component of the current picture. When ph_alf_cb_enabled_flag is not present, it is inferred to be equal to 0.

When present, **ph_alf_cr_enabled_flag** equal to 1 specifies that the adaptive loop filter is enabled for the Cr colour component of the current picture, and ph_alf_cr_enabled_flag equal to 0 specifies that the adaptive loop filter is disabled for the Cr colour component of the current picture. When ph_alf_cr_enabled_flag is not present, it is inferred to be equal to 0.

ph_alf_aps_id_chroma specifies the aps_adaptation_parameter_set_id of the ALF APS that the chroma component of the slices in the current picture refers to.

When ph_alf_aps_id_chroma is present, the following applies:

– The value of alf_chroma_filter_signal_flag of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to ph_alf_aps_id_chroma shall be equal to 1.

– The TemporalId of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to ph_alf_aps_id_chroma shall be less than or equal to the TemporalId of the current picture.

– When sps_ccalf_enabled_flag is equal to 0, the values of alf_cc_cb_filter_signal_flag and alf_cc_cr_filter_signal_flag of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to ph_alf_aps_id_chroma shall be equal to 0.

When present, **ph_alf_cc_cb_enabled_flag** equal to 1 specifies that the cross-component adaptive loop filter for the Cb colour component is enabled for the current picture, and ph_alf_cc_cb_enabled_flag equal to 0 specifies that the cross-component adaptive loop filter for the Cb colour component is disabled for the current picture. When not present, the value of ph_alf_cc_cb_enabled_flag is inferred to be equal to 0.

ph_alf_cc_cb_aps_id specifies the aps_adaptation_parameter_set_id of the ALF APS that the Cb colour component of the slices in the current picture refers to.

When ph_alf_cc_cb_aps_id is present, the following applies:

– The value of alf_cc_cb_filter_signal_flag of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to ph_alf_cc_cb_aps_id shall be equal to 1.

– The TemporalId of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to ph_alf_cc_cb_aps_id shall be less than or equal to the TemporalId of the current picture.

When present, **ph_alf_cc_cr_enabled_flag** equal to 1 specifies that the cross-compoent adaptive loop filter for the Cr colour component is enabled for the current picture, and ph_alf_cc_cr_enabled_flag equal to 0 specifies that the cross-component adaptive loop filter for the Cr colour component is disabled for the current picture. When not present, the value of ph_alf_cc_cr_enabled_flag is inferred to be equal to 0.

ph_alf_cc_cr_aps_id specifies the aps_adaptation_parameter_set_id of the ALF APS that the Cr colour component of the slices in the current picture refers to.

When ph_alf_cc_cr_aps_id is present, the following applies:

– The value of alf_cc_cr_filter_signal_flag of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to ph_alf_cc_cr_aps_id shall be equal to 1.

– The TemporalId of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to ph_alf_cc_cr_aps_id shall be less than or equal to the TemporalId of the current picture.

ph_lmcs_enabled_flag equal to 1 specifies that LMCS is enabled for the current picture. ph_lmcs_enabled_flag equal to 0 specifies that LMCS is disabled for the current picture. When not present, the value of ph_lmcs_enabled_flag is inferred to be equal to 0.

ph_lmcs_aps_id specifies the aps_adaptation_parameter_set_id of the LMCS APS that the slices in the current picture refers to.

When ph_lmcs_aps_id is present, the following applies:

– The TemporalId of the APS NAL unit having aps_params_type equal to LMCS_APS and aps_adaptation_parameter_set_id equal to ph_lmcs_aps_id shall be less than or equal to the TemporalId of the picture associated with PH.
– When sps_chroma_format_idc is equal to 0, the value of aps_chroma_present_flag of the APS NAL unit having aps_params_type equal to LMCS_APS and aps_adaptation_parameter_set_id equal to ph_lmcs_aps_id shall be equal to 0.
– The value of lmcs_delta_cw_prec_minus1 of the APS NAL unit having aps_params_type equal to LMCS_APS and aps_adaptation_parameter_set_id equal to ph_lmcs_aps_id shall be in the range of 0 to BitDepth − 2, inclusive.

ph_chroma_residual_scale_flag equal to 1 specifies that chroma residual scaling is enabled and could be used for the current picture. ph_chroma_residual_scale_flag equal to 0 specifies that chroma residual scaling is disabled and not used for the current picture. When ph_chroma_residual_scale_flag is not present, it is inferred to be equal to 0.

NOTE 4 – When the current picture is a GDR picture or a recovering picture of a GDR picture, and the current picture contains a non-CTU-aligned boundary between a "refreshed area" (i.e., an area that has an exact match of decoded sample values when starting the decoding process from the GDR picture compared to starting the decoding process from the previous IRAP picture in decoding order, when present) and a "dirty area" (i.e., an area that might not have an exact match of decoded sample values when starting the decoding process from the GDR picture compared to starting the decoding process from the previous IRAP picture in decoding order, when present), chroma residual scaling of LMCS would have to be disabled in the current picture to avoid the "dirty area" to affect decoded sample values of the "refreshed area".

ph_explicit_scaling_list_enabled_flag equal to 1 specifies that the explicit scaling list is enabled for the current picture. ph_explicit_scaling_list_enabled_flag equal to 0 specifies that the explicit scaling list is disabled for the picture. When not present, the value of ph_explicit_scaling_list_enabled_flag is inferred to be equal to 0.

ph_scaling_list_aps_id specifies the aps_adaptation_parameter_set_id of the scaling list APS.

When ph_scaling_list_aps_id is present, the following applies:

– The TemporalId of the APS NAL unit having aps_params_type equal to SCALING_APS and aps_adaptation_parameter_set_id equal to ph_scaling_list_aps_id shall be less than or equal to the TemporalId of the picture associated with PH.
– The value of aps_chroma_present_flag of the APS NAL unit having aps_params_type equal to SCALING_APS and aps_adaptation_parameter_set_id equal to ph_scaling_list_aps_id shall be equal to sps_chroma_format_idc = = 0 ? 0 : 1.

ph_virtual_boundaries_present_flag equal to 1 specifies that information of virtual boundaries is signalled in the PH syntax structure. ph_virtual_boundaries_present_flag equal to 0 specifies that information of virtual boundaries is not signalled in the PH syntax structure. When there is one or more than one virtual boundary signalled in the PH syntax structure, the in-loop filtering operations are disabled across the virtual boundaries in the picture. The in-loop filtering operations include the deblocking filter, sample adaptive offset filter, and adaptive loop filter operations. When not present, the value of ph_virtual_boundaries_present_flag is inferred to be equal to 0.

The variable VirtualBoundariesPresentFlag is derived as follows:

```
VirtualBoundariesPresentFlag = 0
if( sps_virtual_boundaries_enabled_flag )
    VirtualBoundariesPresentFlag = sps_virtual_boundaries_present_flag | |
            ph_virtual_boundaries_present_flag                                  (77)
```

ph_num_ver_virtual_boundaries specifies the number of ph_virtual_boundary_pos_x_minus1[i] syntax elements that are present in the PH syntax structure. The value of ph_num_ver_virtual_boundaries shall be in the range of 0 to (pps_pic_width_in_luma_samples <= 8 ? 0 : 3), inclusive.When ph_num_ver_virtual_boundaries is not present, it is inferred to be equal to 0.

The variable NumVerVirtualBoundaries is derived as follows:

```
NumVerVirtualBoundaries = 0
if( sps_virtual_boundaries_enabled_flag )
```

NumVerVirtualBoundaries = sps_virtual_boundaries_present_flag ?
    sps_num_ver_virtual_boundaries : ph_num_ver_virtual_boundaries (78)

ph_virtual_boundary_pos_x_minus1[i] plus 1 specifies the location of the i-th vertical virtual boundary in units of luma samples divided by 8. The value of ph_virtual_boundary_pos_x_minus1[i] shall be in the range of 0 to Ceil(pps_pic_width_in_luma_samples ÷ 8) − 2, inclusive.

The list VirtualBoundaryPosX[i] for i ranging from 0 to NumVerVirtualBoundaries − 1, inclusive, in units of luma samples, specifying the locations of the vertical virtual boundaries, is derived as follows:

```
for( i = 0; i < NumVerVirtualBoundaries; i++)
    VirtualBoundaryPosX[ i ] = ( sps_virtual_boundaries_present_flag ?
            ( sps_virtual_boundary_pos_x_minus1[ i ] + 1 ) :                    (79)
            ( ph_virtual_boundary_pos_x_minus1[ i ] + 1 ) ) * 8
```

The distance between any two vertical virtual boundaries shall be greater than or equal to CtbSizeY luma samples.

ph_num_hor_virtual_boundaries specifies the number of ph_virtual_boundary_pos_y_minus1[i] syntax elements that are present in the PH syntax structure. The value of ph_num_hor_virtual_boundaries shall be in the range of 0 to (pps_pic_height_in_luma_samples <= 8 ? 0: 3), inclusive. When ph_num_hor_virtual_boundaries is not present, it is inferred to be equal to 0.

The parameter NumHorVirtualBoundaries is derived as follows:

```
NumHorVirtualBoundaries = 0
if( sps_virtual_boundaries_enabled_flag )
    NumHorVirtualBoundaries = sps_virtual_boundaries_present_flag ?
            sps_num_hor_virtual_boundaries : ph_num_hor_virtual_boundaries      (80)
```

When sps_virtual_boundaries_enabled_flag is equal to 1 and ph_virtual_boundaries_present_flag is equal to 1, the sum of ph_num_ver_virtual_boundaries and ph_num_hor_virtual_boundaries shall be greater than 0.

ph_virtual_boundary_pos_y_minus1[i] plus 1 specifies the location of the i-th horizontal virtual boundary in units of luma samples divided by 8. The value of ph_virtual_boundary_pos_y_minus1[i] shall be in the range of 0 to Ceil(pps_pic_height_in_luma_samples ÷ 8) − 2, inclusive.

The list VirtualBoundaryPosY[i] for i ranging from 0 to NumHorVirtualBoundaries − 1, inclusive, in units of luma samples, specifying the locations of the horizontal virtual boundaries, is derived as follows:

```
for( i = 0; i < NumHorVirtualBoundaries; i++)
    VirtualBoundaryPosY[ i ] = ( sps_virtual_boundaries_present_flag ?
            ( sps_virtual_boundary_pos_y_minus1[ i ] + 1 ) :                    (81)
            ( ph_virtual_boundary_pos_y_minus1[ i ] + 1 ) ) * 8
```

The distance between any two horizontal virtual boundaries shall be greater than or equal to CtbSizeY luma samples.

ph_pic_output_flag affects the decoded picture output and removal processes as specified in Annex C. When ph_pic_output_flag is not present, it is inferred to be equal to 1.

It is a requirement of bitstream conformance that the bitstream shall contain at least one picture with pic_output_flag equal to 1 that is in an output layer.

NOTE 5 – There is no picture in the bitsteam that has ph_non_ref_pic_flag equal to 1 and ph_pic_output_flag equal to 0.

ph_partition_constraints_override_flag equal to 1 specifies that partition constraint parameters are present in the PH syntax structure. ph_partition_constraints_override_flag equal to 0 specifies that partition constraint parameters are not present in the PH syntax structure. When not present, the value of ph_partition_constraints_override_flag is inferred to be equal to 0.

ph_log2_diff_min_qt_min_cb_intra_slice_luma specifies the difference between the base 2 logarithm of the minimum size in luma samples of a luma leaf block resulting from quadtree splitting of a CTU and the base 2 logarithm of the minimum coding block size in luma samples for luma CUs in the slices with sh_slice_type equal to 2 (I) in the current picture. The value of ph_log2_diff_min_qt_min_cb_intra_slice_luma shall be in the range of 0 to Min(6, CtbLog2SizeY) − MinCbLog2SizeY, inclusive. When not present, the value of ph_log2_diff_min_qt_min_cb_intra_slice_luma is inferred to be equal to sps_log2_diff_min_qt_min_cb_intra_slice_luma.

The value of MinQtLog2SizeIntraY is updated as follows:

```
MinQtLog2SizeIntraY = ph_log2_diff_min_qt_min_cb_intra_slice_luma + MinCbLog2SizeY      (82)
```

ph_max_mtt_hierarchy_depth_intra_slice_luma specifies the maximum hierarchy depth for coding units resulting from multi-type tree splitting of a quadtree leaf in slices with sh_slice_type equal to 2 (I) in the current picture. The value of ph_max_mtt_hierarchy_depth_intra_slice_luma shall be in the range of 0 to 2*(CtbLog2SizeY − MinCbLog2SizeY), inclusive. When not present, the value of ph_max_mtt_hierarchy_depth_intra_slice_luma is inferred to be equal to sps_max_mtt_hierarchy_depth_intra_slice_luma.

ph_log2_diff_max_bt_min_qt_intra_slice_luma specifies the difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a luma coding block that can be split using a binary split and the base 2 logarithm of the minimum size (width or height) in luma samples of a luma leaf block resulting from quadtree splitting of a CTU in slices with sh_slice_type equal to 2 (I) in the current picture. The value of ph_log2_diff_max_bt_min_qt_intra_slice_luma shall be in the range of 0 to (sps_qtbtt_dual_tree_intra_flag ? Min(6, CtbLog2SizeY) : CtbLog2SizeY) − MinQtLog2SizeIntraY, inclusive. When not present, the value of ph_log2_diff_max_bt_min_qt_intra_slice_luma is inferred to be equal to sps_log2_diff_max_bt_min_qt_intra_slice_luma.

ph_log2_diff_max_tt_min_qt_intra_slice_luma specifies the difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a luma coding block that can be split using a ternary split and the base 2 logarithm of the minimum size (width or height) in luma samples of a luma leaf block resulting from quadtree splitting of a CTU in slices with sh_slice_type equal to 2 (I) in the current picture. The value of ph_log2_diff_max_tt_min_qt_intra_slice_luma shall be in the range of 0 to Min(6, CtbLog2SizeY) − MinQtLog2SizeIntraY, inclusive. When not present, the value of ph_log2_diff_max_tt_min_qt_intra_slice_luma is inferred to be equal to sps_log2_diff_max_tt_min_qt_intra_slice_luma.

ph_log2_diff_min_qt_min_cb_intra_slice_chroma specifies the difference between the base 2 logarithm of the minimum size in luma samples of a chroma leaf block resulting from quadtree splitting of a chroma CTU with treeType equal to DUAL_TREE_CHROMA and the base 2 logarithm of the minimum coding block size in luma samples for chroma CUs with treeType equal to DUAL_TREE_CHROMA in slices with sh_slice_type equal to 2 (I) in the current picture. The value of ph_log2_diff_min_qt_min_cb_intra_slice_chroma shall be in the range of 0 to Min(6, CtbLog2SizeY) − MinCbLog2SizeY, inclusive. When not present, the value of ph_log2_diff_min_qt_min_cb_intra_slice_chroma is inferred to be equal to sps_log2_diff_min_qt_min_cb_intra_slice_chroma.

The value of MinQtLog2SizeIntraC is updated as follows:

$$\text{MinQtLog2SizeIntraC} = \text{ph\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_chroma} + \text{MinCbLog2SizeY} \qquad (83)$$

ph_max_mtt_hierarchy_depth_intra_slice_chroma specifies the maximum hierarchy depth for chroma coding units resulting from multi-type tree splitting of a chroma quadtree leaf with treeType equal to DUAL_TREE_CHROMA in slices with sh_slice_type equal to 2 (I) in the current picture. The value of ph_max_mtt_hierarchy_depth_intra_slice_chroma shall be in the range of 0 to 2*(CtbLog2SizeY − MinCbLog2SizeY), inclusive. When not present, the value of ph_max_mtt_hierarchy_depth_intra_slice_chroma is inferred to be equal to sps_max_mtt_hierarchy_depth_intra_slice_chroma.

ph_log2_diff_max_bt_min_qt_intra_slice_chroma specifies the difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a chroma coding block that can be split using a binary split and the base 2 logarithm of the minimum size (width or height) in luma samples of a chroma leaf block resulting from quadtree splitting of a chroma CTU with treeType equal to DUAL_TREE_CHROMA in slices with sh_slice_type equal to 2 (I) in the current picture. The value of ph_log2_diff_max_bt_min_qt_intra_slice_chroma shall be in the range of 0 to Min(6, CtbLog2SizeY) − MinQtLog2SizeIntraC, inclusive. When not present, the value of ph_log2_diff_max_bt_min_qt_intra_slice_chroma is inferred to be equal to sps_log2_diff_max_bt_min_qt_intra_slice_chroma.

ph_log2_diff_max_tt_min_qt_intra_slice_chroma specifies the difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a chroma coding block that can be split using a ternary split and the base 2 logarithm of the minimum size (width or height) in luma samples of a chroma leaf block resulting from quadtree splitting of a chroma CTU with treeType equal to DUAL_TREE_CHROMA in slices with sh_slice_type equal to 2 (I) in the current picture. The value of ph_log2_diff_max_tt_min_qt_intra_slice_chroma shall be in the range of 0 to Min(6, CtbLog2SizeY) − MinQtLog2SizeIntraC, inclusive. When not present, the value of ph_log2_diff_max_tt_min_qt_intra_slice_chroma is inferred to be equal to sps_log2_diff_max_tt_min_qt_intra_slice_chroma.

ph_cu_qp_delta_subdiv_intra_slice specifies the maximum cbSubdiv value of coding units in intra slice that convey cu_qp_delta_abs and cu_qp_delta_sign_flag. The value of ph_cu_qp_delta_subdiv_intra_slice shall be in the range of 0 to 2 * (CtbLog2SizeY − MinQtLog2SizeIntraY + ph_max_mtt_hierarchy_depth_intra_slice_luma), inclusive.

When not present, the value of ph_cu_qp_delta_subdiv_intra_slice is inferred to be equal to 0.

ph_cu_chroma_qp_offset_subdiv_intra_slice specifies the maximum cbSubdiv value of coding units in intra slice that convey cu_chroma_qp_offset_flag. The value of ph_cu_chroma_qp_offset_subdiv_intra_slice shall be in the range of 0 to 2 * (CtbLog2SizeY − MinQtLog2SizeIntraY + ph_max_mtt_hierarchy_depth_intra_slice_luma), inclusive.

When not present, the value of ph_cu_chroma_qp_offset_subdiv_intra_slice is inferred to be equal to 0.

ph_log2_diff_min_qt_min_cb_inter_slice specifies the difference between the base 2 logarithm of the minimum size in luma samples of a luma leaf block resulting from quadtree splitting of a CTU and the base 2 logarithm of the minimum luma coding block size in luma samples for luma CUs in the slices with sh_slice_type equal to 0 (B) or 1 (P) in the current picture. The value of ph_log2_diff_min_qt_min_cb_inter_slice shall be in the range of 0 to Min(6, CtbLog2SizeY) − MinCbLog2SizeY, inclusive. When not present, the value of ph_log2_diff_min_qt_min_cb_inter_slice is inferred to be equal to sps_log2_diff_min_qt_min_cb_inter_slice.

The value of MinQtLog2SizeInterY is updated as follows:

$$\text{MinQtLog2SizeInterY} = \text{ph\_log2\_diff\_min\_qt\_min\_cb\_inter\_slice} + \text{MinCbLog2SizeY} \qquad (84)$$

ph_max_mtt_hierarchy_depth_inter_slice specifies the maximum hierarchy depth for coding units resulting from multi-type tree splitting of a quadtree leaf in slices with sh_slice_type equal to 0 (B) or 1 (P) in the current picture. The value of ph_max_mtt_hierarchy_depth_inter_slice shall be in the range of 0 to 2*(CtbLog2SizeY − MinCbLog2SizeY), inclusive. When not present, the value of ph_max_mtt_hierarchy_depth_inter_slice is inferred to be equal to sps_max_mtt_hierarchy_depth_inter_slice.

ph_log2_diff_max_bt_min_qt_inter_slice specifies the difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a luma coding block that can be split using a binary split and the base 2 logarithm of the minimum size (width or height) in luma samples of a luma leaf block resulting from quadtree splitting of a CTU in the slices with sh_slice_type equal to 0 (B) or 1 (P) in the current picture. The value of ph_log2_diff_max_bt_min_qt_inter_slice shall be in the range of 0 to CtbLog2SizeY − MinQtLog2SizeInterY, inclusive. When not present, the value of ph_log2_diff_max_bt_min_qt_inter_slice is inferred to be equal to sps_log2_diff_max_bt_min_qt_inter_slice.

ph_log2_diff_max_tt_min_qt_inter_slice specifies the difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a luma coding block that can be split using a ternary split and the base 2 logarithm of the minimum size (width or height) in luma samples of a luma leaf block resulting from quadtree splitting of a CTU in slices with sh_slice_type equal to 0 (B) or 1 (P) in the current picture. The value of ph_log2_diff_max_tt_min_qt_inter_slice shall be in the range of 0 to Min(6, CtbLog2SizeY) − MinQtLog2SizeInterY, inclusive. When not present, the value of ph_log2_diff_max_tt_min_qt_inter_slice is inferred to be equal to sps_log2_diff_max_tt_min_qt_inter_slice.

ph_cu_qp_delta_subdiv_inter_slice specifies the maximum cbSubdiv value of coding units that in inter slice convey cu_qp_delta_abs and cu_qp_delta_sign_flag. The value of ph_cu_qp_delta_subdiv_inter_slice shall be in the range of 0 to 2 * (CtbLog2SizeY − MinQtLog2SizeInterY + ph_max_mtt_hierarchy_depth_inter_slice), inclusive.

When not present, the value of ph_cu_qp_delta_subdiv_inter_slice is inferred to be equal to 0.

ph_cu_chroma_qp_offset_subdiv_inter_slice specifies the maximum cbSubdiv value of coding units in inter slice that convey cu_chroma_qp_offset_flag. The value of ph_cu_chroma_qp_offset_subdiv_inter_slice shall be in the range of 0 to 2 * (CtbLog2SizeY − MinQtLog2SizeInterY + ph_max_mtt_hierarchy_depth_inter_slice), inclusive.

When not present, the value of ph_cu_chroma_qp_offset_subdiv_inter_slice is inferred to be equal to 0.

ph_temporal_mvp_enabled_flag equal to 1 specifies that temporal motion vector predictor is enabled for the current picture. ph_temporal_mvp_enabled_flag equal to 0 specifies that temporal motion vector predictor is disabled for the current picture. When not present, the value of ph_temporal_mvp_enabled_flag is inferred to be equal to 0.

NOTE 6 – Due to the other existing constraints, the value of ph_temporal_mvp_enabled_flag could only be equal to 0 in a conforming bitstream when one or more of the following conditions are true: 1) no reference picture in the DPB has the same spatial resolution and the same scaling window offsets as the current picture, and 2) no reference picture in the DPB exists in the active entries of the RPLs of all slices in the current picture. Note that there are other, complicated conditions under which ph_temporal_mvp_enabled_flag could only be equal to 0 that are not listed.

The maximum number of subblock-based merging MVP candidates, MaxNumSubblockMergeCand, is derived as follows:

```
if( sps_affine_enabled_flag )
    MaxNumSubblockMergeCand = 5 − sps_five_minus_max_num_subblock_merge_cand          (85)
else
    MaxNumSubblockMergeCand = sps_sbtmvp_enabled_flag  &&  ph_temporal_mvp_enabled_flag
```

The value of MaxNumSubblockMergeCand shall be in the range of 0 to 5, inclusive.

ph_collocated_from_l0_flag equal to 1 specifies that the collocated picture used for temporal motion vector prediction is derived from RPL 0. ph_collocated_from_l0_flag equal to 0 specifies that the collocated picture used for temporal motion vector prediction is derived from RPL 1. When ph_temporal_mvp_enabled_flag and pps_rpl_info_in_ph_flag are both equal to 1 and num_ref_entries[1][RplsIdx[1]] is equal to 0, the value of ph_collocated_from_l0_flag is inferred to be equal to 1.

ph_collocated_ref_idx specifies the reference index of the collocated picture used for temporal motion vector prediction.

When ph_collocated_from_l0_flag is equal to 1, ph_collocated_ref_idx refers to an entry in RPL 0, and the value of ph_collocated_ref_idx shall be in the range of 0 to num_ref_entries[0][RplsIdx[0]] − 1, inclusive.

When ph_collocated_from_l0_flag is equal to 0, ph_collocated_ref_idx refers to an entry in RPL 1, and the value of ph_collocated_ref_idx shall be in the range of 0 to num_ref_entries[1][RplsIdx[1]] − 1, inclusive.

When not present, the value of ph_collocated_ref_idx is inferred to be equal to 0.

ph_mmvd_fullpel_only_flag equal to 1 specifies that the merge mode with motion vector difference uses only integer sample precision for the current picture. ph_mmvd_fullpel_only_flag equal to 0 specifies that the merge mode with motion vector difference could use either fractional or integer sample precision for the current picture. When not present, the value of ph_mmvd_fullpel_only_flag is inferred to be 0.

ph_mvd_l1_zero_flag equal to 1 specifies that the mvd_coding(x0, y0, 1, cpIdx) syntax structure is not parsed and MvdL1[x0][y0][compIdx] and MvdCpL1[x0][y0][cpIdx][compIdx] are set equal to 0 for compIdx = 0..1 and cpIdx = 0..2. ph_mvd_l1_zero_flag equal to 0 specifies that the mvd_coding(x0, y0, 1, cpIdx) syntax structure is parsed. When not present, the value of ph_mvd_l1_zero_flag is inferred to be 1.

ph_bdof_disabled_flag equal to 1 specifies that the bi-directional optical flow inter prediction based inter bi-prediction is disabled for the current picture. ph_bdof_disabled_flag equal to 0 specifies that the bi-directional optical flow inter prediction based inter bi-prediction is enabled for the current picture.

When not present, the value of ph_bdof_disabled_flag is inferred as follows:

– If sps_bdof_control_present_in_ph_flag is equal to 0, the value of ph_bdof_disabled_flag is inferred to be equal to 1 − sps_bdof_enabled_flag.

– Otherwise (sps_bdof_control_present_in_ph_flag is equal to 1), the value of ph_bdof_disabled_flag is inferred to be equal to 1.

ph_dmvr_disabled_flag equal to 1 specifies that the decoder motion vector refinement based inter bi-prediction is disabled for the current picture. ph_dmvr_disabled_flag equal to 0 specifies that the decoder motion vector refinement based inter bi-prediction is enabled for the current picture.

When not present, the value of ph_dmvr_disabled_flag is inferred as follows:

– If sps_dmvr_control_present_in_ph_flag is equal to 0, the value of ph_dmvr_disabled_flag is inferred to be equal to 1 − sps_dmvr_enabled_flag.

– Otherwise (sps_dmvr_control_present_in_ph_flag is equal to 1), the value of ph_dmvr_disabled_flag is inferred to be equal to 1.

ph_prof_disabled_flag equal to 1 specifies that prediction refinement with optical flow is disabled for the current picture. ph_prof_disabled_flag equal to 0 specifies that prediction refinement with optical flow is enabled for the current picture.

When ph_prof_disabled_flag is not present, it is inferred as follows:

– If sps_affine_prof_enabled_flag is equal to 1, the value of ph_prof_disabled_flag is inferred to be equal to 0.

– Otherwise (sps_affine_prof_enabled_flag is equal to 0), the value of ph_prof_disabled_flag is inferred to be equal to 1.

ph_qp_delta specifies the initial value of Qp_Y to be used for the coding blocks in the picture until modified by the value of CuQpDeltaVal in the coding unit layer.

When pps_qp_delta_info_in_ph_flag is equal to 1, the initial value of the Qp_Y quantization parameter for all slices of the picture, SliceQp_Y, is derived as follows:

$$\text{SliceQp}_\text{Y} = 26 + \text{pps\_init\_qp\_minus26} + \text{ph\_qp\_delta} \quad (86)$$

The value of SliceQp_Y shall be in the range of −QpBdOffset to +63, inclusive.

ph_joint_cbcr_sign_flag specifies whether, in transform units with tu_joint_cbcr_residual_flag[x0][y0] equal to 1, the collocated residual samples of both chroma components have inverted signs. When tu_joint_cbcr_residual_flag[x0][y0] equal to 1 for a transform unit, ph_joint_cbcr_sign_flag equal to 0 specifies that the sign of each residual sample of the Cr (or Cb) component is identical to the sign of the collocated Cb (or Cr) residual sample and ph_joint_cbcr_sign_flag equal to 1 specifies that the sign of each residual sample of the Cr (or Cb) component is given by the inverted sign of the collocated Cb (or Cr) residual sample.

When present, **ph_sao_luma_enabled_flag** equal to 1 specifies that SAO is enabled for the luma component of the current picture, and ph_sao_luma_enabled_flag equal to 0 specifies that SAO is disabled for the luma component of the current picture. When ph_sao_luma_enabled_flag is not present, it is inferred to be equal to 0.

When present, **ph_sao_chroma_enabled_flag** equal to 1 specifies that SAO is enabled for the chroma component of the current picture, and ph_sao_chroma_enabled_flag equal to 0 specifies that SAO is disabled for the chroma component of the current picture. When ph_sao_chroma_enabled_flag is not present, it is inferred to be equal to 0.

ph_deblocking_params_present_flag equal to 1 specifies that the deblocking parameters could be present in the PH syntax structure. ph_deblocking_params_present_flag equal to 0 specifies that the deblocking parameters are not present in the PH syntax structure. When not present, the value of ph_deblocking_params_present_flag is inferred to be equal to 0.

When present, **ph_deblocking_filter_disabled_flag** equal to 1 specifies that the deblocking filter is disabled for the current picture, and ph_deblocking_filter_disabled_flag equal to 0 specifies that the deblocking filter is enabled for the current picture.

NOTE 7 – When ph_deblocking_filter_disabled_flag is equal to 1, the deblocking filter is disabled for the slices of the current picture for which sh_deblocking_filter_disabled_flag is not present in the SHs and inferred to be equal to 1 or is present in the SHs and equal to 1, and the deblocking filter is enabled for the slices of the current picture for which sh_deblocking_filter_disabled_flag is not present in the SHs and inferred to be equal to 0 or is present in the SHs and equal to 0.

NOTE 8 – When ph_deblocking_filter_disabled_flag is equal to 0, the deblocking filter is enabled for the slices of the current picture for which sh_deblocking_filter_disabled_flag is not present in the SHs and inferred to be equal to 0 or is present in the SHs and equal to 0, and the deblocking filter is not applied for the slices of the current picture for which sh_deblocking_filter_disabled_flag is not present in the SHs and inferred to be equal to 1 or is present in the SHs and equal to 1.

When ph_deblocking_filter_disabled_flag is not present, it is inferred as follows:

– If pps_deblocking_filter_disabled_flag and ph_deblocking_params_present_flag are both equal to 1, the value of ph_deblocking_filter_disabled_flag is inferred to be equal to 0.
– Otherwise (pps_deblocking_filter_disabled_flag or ph_deblocking_params_present_flag is equal to 0), the value of ph_deblocking_filter_disabled_flag is inferred to be equal to pps_deblocking_filter_disabled_flag.

ph_luma_beta_offset_div2 and **ph_luma_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the luma component for the slices in the current picture. The values of ph_luma_beta_offset_div2 and ph_luma_tc_offset_div2 shall both be in the range of −12 to 12, inclusive. When not present, the values of ph_luma_beta_offset_div2 and ph_luma_tc_offset_div2 are inferred to be equal to pps_luma_beta_offset_div2 and pps_luma_tc_offset_div2, respectively.

ph_cb_beta_offset_div2 and **ph_cb_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for the slices in the current picture. The values of ph_cb_beta_offset_div2 and ph_cb_tc_offset_div2 shall both be in the range of −12 to 12, inclusive.

When not present, the values of ph_cb_beta_offset_div2 and ph_cb_tc_offset_div2 are inferred as follows:

– If pps_chroma_tool_offsets_present_flag is equal to 1, the values of ph_cb_beta_offset_div2 and ph_cb_tc_offset_div2 are inferred to be equal to pps_cb_beta_offset_div2 and pps_cb_tc_offset_div2, respectively.
– Otherwise (pps_chroma_tool_offsets_present_flag is equal to 0), the values of ph_cb_beta_offset_div2 and ph_cb_tc_offset_div2 are inferred to be equal to ph_luma_beta_offset_div2 and ph_luma_tc_offset_div2, respectively.

ph_cr_beta_offset_div2 and **ph_cr_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for the slices in the current picture. The values of ph_cr_beta_offset_div2 and ph_cr_tc_offset_div2 shall both be in the range of −12 to 12, inclusive.

When not present, the values of ph_cr_beta_offset_div2 and ph_cr_tc_offset_div2 are inferred as follows:

– If pps_chroma_tool_offsets_present_flag is equal to 1, the values of ph_cr_beta_offset_div2 and ph_cr_tc_offset_div2 are inferred to be equal to pps_cr_beta_offset_div2 and pps_cr_tc_offset_div2, respectively.
– Otherwise (pps_chroma_tool_offsets_present_flag is equal to 0), the values of ph_cr_beta_offset_div2 and ph_cr_tc_offset_div2 are inferred to be equal to ph_luma_beta_offset_div2 and ph_luma_tc_offset_div2, respectively.

ph_extension_length specifies the length of the PH extension data in bytes, not including the bits used for signalling ph_extension_length itself. When not present, the value of ph_extension_length is inferred to be equal to 0. Although ph_extension_length is not present in bitstreams conforming to this version of this Specification, some use of ph_extension_length could be specified in some future version of this Specification, and decoders conforming to this version of this Specification shall also allow ph_extension_length to be present and in the range of 0 to 256, inclusive.

ph_extension_data_byte could have any value. Its presence and value do not affect the decoding process specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore the value of ph_extension_data_byte.

7.4.3.9 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. An SEI RBSP contains one or more SEI messages.

7.4.3.10 AU delimiter RBSP semantics

The AU delimiter is used to indicate the start of an AU, whether the AU is an IRAP or GDR AU, and the type of slices present in the coded pictures in the AU containing the AU delimiter NAL unit. When the bitstream contains only one layer, there is no normative decoding process associated with the AU delimiter.

aud_irap_or_gdr_flag equal to 1 specifies that the AU containing the AU delimiter is an IRAP or GDR AU. aud_irap_or_gdr_flag equal to 0 specifies that the AU containing the AU delimiter is not an IRAP or GDR AU.

NOTE – Due to the conformance requirement of extracted sub-bitstreams specified in clause C.6, an AUD NAL unit is present in an AU that contains only IRAP or GDR pictures for the layers of an OLS that contains more than one layer.

aud_pic_type indicates that the sh_slice_type values for all slices of the coded pictures in the AU containing the AU delimiter NAL unit are members of the set listed in Table 7 for the given value of aud_pic_type. The value of aud_pic_type shall be equal to 0, 1 or 2 in bitstreams conforming to this version of this Specification. Other values of aud_pic_type are reserved for future use by ITU-T | ISO/IEC. Decoders conforming to this version of this Specification shall ignore reserved values of aud_pic_type.

Table 7 – Interpretation of aud_pic_type

aud_pic_type	sh_slice_type values that could be present in the AU
0	I
1	P, I
2	B, P, I

7.4.3.11 End of sequence RBSP semantics

When present in a bitstream, an EOS NAL unit is considered belonging to or being in the layer that has nuh_layer_id equal to the nuh_layer_id of the EOS NAL unit.

When present, the EOS RBSP specifies that the next subsequent PU that belongs to the same layer as the EOS NAL unit in the bitstream in decoding order (if any) is an IRAP or GDR PU. The syntax content of the SODB and RBSP for the EOS RBSP are empty.

When an AU auA contains an EOS NAL unit in a layer layerA, for each layer layerB that is present in the CVS and has layerA as a reference layer, the first picture in layerB in decoding order in an AU following auA in decoding order shall be a CLVSS picture.

7.4.3.12 End of bitstream RBSP semantics

The EOB RBSP indicates that no additional NAL units are present in the bitstream that are subsequent to the EOB RBSP in decoding order. The syntax content of the SODB and RBSP for the EOB RBSP are empty.

7.4.3.13 Filler data RBSP semantics

The filler data RBSP contains bytes whose value shall be equal to 0xFF. No normative decoding process is specified for a filler data RBSP.

fd_ff_byte is a byte equal to 0xFF.

7.4.3.14 Slice layer RBSP semantics

The slice layer RBSP consists of a slice header and slice data.

7.4.3.15 RBSP slice trailing bits semantics

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

Let the variable NumBytesInPicVclNalUnits be the sum of the values of NumBytesInNalUnit for all VCL NAL units of a coded picture.

Let the variable BinCountsInPicNalUnits be the number of times that the parsing process function DecodeBin( ), specified in clause 9.3.4.3.1, is invoked to decode the contents of all VCL NAL units of a coded picture.

Let the variable RawMinCuBits be derived as follows:

$$\begin{aligned}\text{RawMinCuBits} = {} & \text{MinCbSizeY} * \text{MinCbSizeY} * \\ & (\ \text{BitDepth} + 2 * \text{BitDepth} / (\ \text{SubWidthC} * \text{SubHeightC}\ )\ )\end{aligned} \qquad (87)$$

Let the variable vclByteScaleFactor be derived to be equal to (32 + 4 * general_tier_flag) ÷ 3.

Let the variable NumBytesInSubpicVclNalUnits be the sum of the values NumBytesInNalUnit for all VCL NAL units of a subpicture with subpicture index subpicIdxA.

Let the variable BinCountsInSubpicNalUnits be the number of times that the parsing process function DecodeBin( ), specified in clause 9.3.4.3.1, is invoked to decode the contents of all VCL NAL units of a subpicture with subpicture index subpicIdxA.

The variable subpicSizeInMinCbsY for the subpicture with subpicture index subpicIdxA is derived to be equal to ((sps_subpic_width_minus1[subpicIdxA] + 1) * CtbSizeY / MinCbSizeY * (sps_subpic_height_minus1[subpicIdxA] + 1) * CtbSizeY / MinCbSizeY).

7.4.3.16 RBSP trailing bits semantics

rbsp_stop_one_bit shall be equal to 1.

rbsp_alignment_zero_bit shall be equal to 0.

7.4.3.17 Byte alignment semantics

byte_alignment_bit_equal_to_one shall be equal to 1.

byte_alignment_bit_equal_to_zero shall be equal to 0.

7.4.3.18 Adaptive loop filter data semantics

alf_luma_filter_signal_flag equal to 1 specifies that a luma filter set is signalled. alf_luma_filter_signal_flag equal to 0 specifies that a luma filter set is not signalled.

alf_chroma_filter_signal_flag equal to 1 specifies that a chroma filter is signalled. alf_chroma_filter_signal_flag equal to 0 specifies that a chroma filter is not signalled. When not present, the value of alf_chroma_filter_signal_flag is inferred to be equal to 0.

The variable NumAlfFilters specifying the number of different adaptive loop filters is set equal to 25.

alf_cc_cb_filter_signal_flag equal to 1 specifies that cross-component filters for the Cb colour component are signalled. alf_cc_cb_filter_signal_flag equal to 0 specifies that cross-component filters for Cb colour component are not signalled. When not present, the value of alf_cc_cb_filter_signal_flag is inferred to be equal to 0.

alf_cc_cr_filter_signal_flag equal to 1 specifies that cross-component filters for the Cr colour component are signalled. alf_cc_cr_filter_signal_flag equal to 0 specifies that cross-component filters for the Cr colour component are not signalled. When not present, the value of alf_cc_cr_filter_signal_flag is inferred to be equal to 0.

At least one of the values of alf_luma_filter_signal_flag, alf_chroma_filter_signal_flag, alf_cc_cb_filter_signal_flag, and alf_cc_cr_filter_signal_flag shall be equal to 1.

alf_luma_clip_flag equal to 0 specifies that linear adaptive loop filtering is applied to the luma component. alf_luma_clip_flag equal to 1 specifies that non-linear adaptive loop filtering could be applied to the luma component.

alf_luma_num_filters_signalled_minus1 plus 1 specifies the number of adpative loop filter classes for which luma coefficients can be signalled. The value of alf_luma_num_filters_signalled_minus1 shall be in the range of 0 to NumAlfFilters − 1, inclusive.

alf_luma_coeff_delta_idx[filtIdx] specifies the indices of the signalled adaptive loop filter luma coefficient deltas for the filter class indicated by filtIdx ranging from 0 to NumAlfFilters − 1. When alf_luma_coeff_delta_idx[filtIdx] is not present, it is inferred to be equal to 0. The length of alf_luma_coeff_delta_idx[filtIdx] is Ceil(Log2(alf_luma_num_filters_signalled_minus1 + 1)) bits. The value of alf_luma_coeff_delta_idx[filtIdx] shall be in the range of 0 to alf_luma_num_filters_signalled_minus1, inclusive.

alf_luma_coeff_abs[sfIdx][j] specifies the absolute value of the j-th coefficient of the signalled luma filter indicated by sfIdx. When alf_luma_coeff_abs[sfIdx][j] is not present, it is inferred to be equal 0. The value of alf_luma_coeff_abs[sfIdx][j] shall be in the range of 0 to 128, inclusive.

alf_luma_coeff_sign[sfIdx][j] specifies the sign of the j-th luma coefficient of the filter indicated by sfIdx as follows:

– If alf_luma_coeff_sign[sfIdx][j] is equal to 0, the corresponding luma filter coefficient has a positive value.

– Otherwise (alf_luma_coeff_sign[sfIdx][j] is equal to 1), the corresponding luma filter coefficient has a negative value.

When alf_luma_coeff_sign[sfIdx][j] is not present, it is inferred to be equal to 0.

The variable filtCoeff[sfIdx][j] with sfIdx = 0..alf_luma_num_filters_signalled_minus1, j = 0..11 is initialized as follows:

```
filtCoeff[ sfIdx ][ j ] = alf_luma_coeff_abs[ sfIdx ][ j ] *
                          ( 1 − 2 * alf_luma_coeff_sign[ sfIdx ][ j ] )          (88)
```

The luma filter coefficients AlfCoeff_L[aps_adaptation_parameter_set_id] with elements AlfCoeff_L[aps_adaptation_parameter_set_id][filtIdx][j], with filtIdx = 0..NumAlfFilters − 1 and j = 0..11 are derived as follows:

```
AlfCoeffL[ aps_adaptation_parameter_set_id ][ filtIdx ][ j ] = filtCoeff[ alf_luma_coeff_delta_idx[ filtIdx ] ][ j ]

          (89)
```

The fixed filter coefficients AlfFixFiltCoeff[i][j] with i = 0..63, j = 0..11 and the class to filter mapping AlfClassToFiltMap[m][n] with m = 0..15 and n = 0..24 are derived as follows:

```
AlfFixFiltCoeff =                                                                  (90)
    {
      { 0,   0,   2,  -3,   1,  -4,   1,   7,  -1,   1,  -1,   5}
      { 0,   0,   0,   0,   0,  -1,   0,   1,   0,   0,  -1,   2}
      { 0,   0,   0,   0,   0,   0,   0,   1,   0,   0,   0,   0}
      { 0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  -1,   1}
      { 2,   2,  -7,  -3,   0,  -5,  13,  22,  12,  -3,  -3,  17}
      {-1,   0,   6,  -8,   1,  -5,   1,  23,   0,   2,  -5,  10}
      { 0,   0,  -1,  -1,   0,  -1,   2,   1,   0,   0,  -1,   4}
      { 0,   0,   3, -11,   1,   0,  -1,  35,   5,   2,  -9,   9}
      { 0,   0,   8,  -8,  -2,  -7,   4,   4,   2,   1,  -1,  25}
      { 0,   0,   1,  -1,   0,  -3,   1,   3,  -1,   1,  -1,   3}
      { 0,   0,   3,  -3,   0,  -6,   5,  -1,   2,   1,  -4,  21}
      {-7,   1,   5,   4,  -3,   5,  11,  13,  12,  -8,  11,  12}
      {-5,  -3,   6,  -2,  -3,   8,  14,  15,   2,  -7,  11,  16}
      { 2,  -1,  -6,  -5,  -2,  -2,  20,  14,  -4,   0,  -3,  25}
      { 3,   1,  -8,  -4,   0,  -8,  22,   5,  -3,   2, -10,  29}
      { 2,   1,  -7,  -1,   2, -11,  23,  -5,   0,   2, -10,  29}
      {-6,  -3,   8,   9,  -4,   8,   9,   7,  14,  -2,   8,   9}
      { 2,   1,  -4,  -7,   0,  -8,  17,  22,   1,  -1,  -4,  23}
      { 3,   0,  -5,  -7,   0,  -7,  15,  18,  -5,   0,  -5,  27}
      { 2,   0,   0,  -7,   1, -10,  13,  13,  -4,   2,  -7,  24}
      { 3,   3, -13,   4,  -2,  -5,   9,  21,  25,  -2,  -3,  12}
      {-5,  -2,   7,  -3,  -7,   9,   8,   9,  16,  -2,  15,  12}
      { 0,  -1,   0,  -7,  -5,   4,  11,  11,   8,  -6,  12,  21}
      { 3,  -2,  -3,  -8,  -4,  -1,  16,  15,  -2,  -3,   3,  26}
      { 2,   1,  -5,  -4,  -1,  -8,  16,   4,  -2,   1,  -7,  33}
      { 2,   1,  -4,  -2,   1, -10,  17,  -2,   0,   2, -11,  33}
      { 1,  -2,   7, -15, -16,  10,   8,   8,  20,  11,  14,  11}
      { 2,   2,   3, -13, -13,   4,   8,  12,   2,  -3,  16,  24}
      { 1,   4,   0,  -7,  -8,  -4,   9,   9,  -2,  -2,   8,  29}
      { 1,   1,   2,  -4,  -1,  -6,   6,   3,  -1,  -1,  -3,  30}
      {-7,   3,   2,  10,  -2,   3,   7,  11,  19,  -7,   8,  10}
      { 0,  -2,  -5,  -3,  -2,   4,  20,  15,  -1,  -3,  -1,  22}
      { 3,  -1,  -8,  -4,  -1,  -4,  22,   8,  -4,   2,  -8,  28}
      { 0,   3, -14,   3,   0,   1,  19,  17,   8,  -3,  -7,  20}
      { 0,   2,  -1,  -8,   3,  -6,   5,  21,   1,   1,  -9,  13}
      {-4,  -2,   8,  20,  -2,   2,   3,   5,  21,   4,   6,   1}
      { 2,  -2,  -3,  -9,  -4,   2,  14,  16,   3,  -6,   8,  24}
      { 2,   1,   5, -16,  -7,   2,   3,  11,  15,  -3,  11,  22}
      { 1,   2,   3, -11,  -2,  -5,   4,   8,   9,  -3,  -2,  26}
      { 0,  -1,  10,  -9,  -1,  -8,   2,   3,   4,   0,   0,  29}
      { 1,   2,   0,  -5,   1,  -9,   9,   3,   0,   1,  -7,  20}
```

```
  {-2,   8,  -6,  -4,   3,  -9,  -8,  45,  14,   2, -13,   7}
  { 1,  -1,  16, -19,  -8,  -4,  -3,   2,  19,   0,   4,  30}
  { 1,   1,  -3,   0,   2, -11,  15,  -5,   1,   2,  -9,  24}
  { 0,   1,  -2,   0,   1,  -4,   4,   0,   0,   1,  -4,   7}
  { 0,   1,   2,  -5,   1,  -6,   4,  10,  -2,   1,  -4,  10}
  { 3,   0,  -3,  -6,  -2,  -6,  14,   8,  -1,  -1,  -3,  31}
  { 0,   1,   0,  -2,   1,  -6,   5,   1,   0,   1,  -5,  13}
  { 3,   1,   9, -19, -21,   9,   7,   6,  13,   5,  15,  21}
  { 2,   4,   3, -12, -13,   1,   7,   8,   3,   0,  12,  26}
  { 3,   1,  -8,  -2,   0,  -6,  18,   2,  -2,   3, -10,  23}
  { 1,   1,  -4,  -1,   1,  -5,   8,   1,  -1,   2,  -5,  10}
  { 0,   1,  -1,   0,   0,  -2,   2,   0,   0,   1,  -2,   3}
  { 1,   1,  -2,  -7,   1,  -7,  14,  18,   0,   0,  -7,  21}
  { 0,   1,   0,  -2,   0,  -7,   8,   1,  -2,   0,  -3,  24}
  { 0,   1,   1,  -2,   2, -10,  10,   0,  -2,   1,  -7,  23}
  { 0,   2,   2, -11,   2,  -4,  -3,  39,   7,   1, -10,   9}
  { 1,   0,  13, -16,  -5,  -6,  -1,   8,   6,   0,   6,  29}
  { 1,   3,   1,  -6,  -4,  -7,   9,   6,  -3,  -2,   3,  33}
  { 4,   0, -17,  -1,  -1,   5,  26,   8,  -2,   3, -15,  30}
  { 0,   1,  -2,   0,   2,  -8,  12,  -6,   1,   1,  -6,  16}
  { 0,   0,   0,  -1,   1,  -4,   4,   0,   0,   0,  -3,  11}
  { 0,   1,   2,  -8,   2,  -6,   5,  15,   0,   2,  -7,   9}
  { 1,  -1,  12, -15,  -7,  -2,   3,   6,   6,  -1,   7,  30}
},
```

AlfClassToFiltMap = (91)

```
{
  {  8,  2,  2,  2,  3,  4, 53,  9,  9, 52,  4,  4,  5,  9,  2,  8, 10,  9,  1,  3, 39, 39, 10,  9, 52 }
  { 11, 12, 13, 14, 15, 30, 11, 17, 18, 19, 16, 20, 20,  4, 53, 21, 22, 23, 14, 25, 26, 26, 27, 28, 10 }
  { 16, 12, 31, 32, 14, 16, 30, 33, 53, 34, 35, 16, 20,  4,  7, 16, 21, 36, 18, 19, 21, 26, 37, 38, 39 }
  { 35, 11, 13, 14, 43, 35, 16,  4, 34, 62, 35, 35, 30, 56,  7, 35, 21, 38, 24, 40, 16, 21, 48, 57, 39 }
  { 11, 31, 32, 43, 44, 16,  4, 17, 34, 45, 30, 20, 20,  7,  5, 21, 22, 46, 40, 47, 26, 48, 63, 58, 10 }
  { 12, 13, 50, 51, 52, 11, 17, 53, 45,  9, 30,  4, 53, 19,  0, 22, 23, 25, 43, 44, 37, 27, 28, 10, 55 }
  { 30, 33, 62, 51, 44, 20, 41, 56, 34, 45, 20, 41, 41, 56,  5, 30, 56, 38, 40, 47, 11, 37, 42, 57,  8 }
  { 35, 11, 23, 32, 14, 35, 20,  4, 17, 18, 21, 20, 20, 20,  4, 16, 21, 36, 46, 25, 41, 26, 48, 49, 58 }
  { 12, 31, 59, 59,  3, 33, 33, 59, 59, 52,  4, 33, 17, 59, 55, 22, 36, 59, 59, 60, 22, 36, 59, 25, 55 }
  { 31, 25, 15, 60, 60, 22, 17, 19, 55, 55, 20, 20, 53, 19, 55, 22, 46, 25, 43, 60, 37, 28, 10, 55, 52 }
  { 12, 31, 32, 50, 51, 11, 33, 53, 19, 45, 16,  4,  4, 53,  5, 22, 36, 18, 25, 43, 26, 27, 27, 28, 10 }
  {  5,  2, 44, 52,  3,  4, 53, 45,  9,  3,  4, 56,  5,  0,  2,  5, 10, 47, 52,  3, 63, 39, 10,  9, 52 }
  { 12, 34, 44, 44,  3, 56, 56, 62, 45,  9, 56, 56,  7,  5,  0, 22, 38, 40, 47, 52, 48, 57, 39, 10,  9 }
  { 35, 11, 23, 14, 51, 35, 20, 41, 56, 62, 16, 20, 41, 56,  7, 16, 21, 38, 24, 40, 26, 26, 42, 57, 39 }
  { 33, 34, 51, 51, 52, 41, 41, 34, 62,  0, 41, 41, 56,  7,  5, 56, 38, 38, 40, 44, 37, 42, 57, 39, 10 }
  { 16, 31, 32, 15, 60, 30,  4, 17, 19, 25, 22, 20,  4, 53, 19, 21, 22, 46, 25, 55, 26, 48, 63, 58, 55 }
},
```

It is a requirement of bitstream conformance that the values of AlfCoeff_L[aps_adaptation_parameter_set_id][filtIdx][j] with filtIdx = 0..NumAlfFilters − 1, j = 0..11 shall be in the range of -2^7 to $2^7 - 1$, inclusive.

alf_luma_clip_idx[sfIdx][j] specifies the clipping index of the clipping value to use before multiplying by the j-th coefficient of the signalled luma filter indicated by sfIdx. When alf_luma_clip_idx[sfIdx][j] is not present, it is inferred to be equal to 0.

The luma filter clipping values AlfClip_L[aps_adaptation_parameter_set_id] with elements AlfClip_L[aps_adaptation_ parameter_set_id][filtIdx][j], with filtIdx = 0..NumAlfFilters − 1 and j = 0..11 are derived as specified in Table 8 depending on BitDepth and clipIdx set equal to alf_luma_clip_idx[alf_luma_coeff_delta_idx[filtIdx]][j].

alf_chroma_clip_flag equal to 0 specifies that linear adaptive loop filtering is applied to chroma components; alf_chroma_clip_flag equal to 1 specifies that non-linear adaptive loop filtering is applied to chroma components. When not present, alf_chroma_clip_flag is inferred to be equal to 0.

alf_chroma_num_alt_filters_minus1 plus 1 specifies the number of alternative filters for chroma components. The value of alf_chroma_num_alt_filters_minus1 shall be in the range of 0 to 7, inclusive.

alf_chroma_coeff_abs[altIdx][j] specifies the absolute value of the j-th chroma filter coefficient for the alternative chroma filter with index altIdx. When alf_chroma_coeff_abs[altIdx][j] is not present, it is inferred to be equal 0. The value of alf_chroma_coeff_abs[sfIdx][j] shall be in the range of 0 to 128, inclusive.

alf_chroma_coeff_sign[altIdx][j] specifies the sign of the j-th chroma filter coefficient for the alternative chroma filter with index altIdx as follows:

– If alf_chroma_coeff_sign[altIdx][j] is equal to 0, the corresponding chroma filter coefficient has a positive value.

– Otherwise (alf_chroma_coeff_sign[altIdx][j] is equal to 1), the corresponding chroma filter coefficient has a negative value.

When alf_chroma_coeff_sign[altIdx][j] is not present, it is inferred to be equal to 0.

The chroma filter coefficients $AlfCoeff_C$[aps_adaptation_parameter_set_id][altIdx] with elements $AlfCoeff_C$[aps_adaptation_parameter_set_id][altIdx][j], with altIdx = 0..alf_chroma_num_alt_filters_minus1, j = 0..5 are derived as follows:

$$AlfCoeff_C[\ aps\_adaptation\_parameter\_set\_id\ ][\ altIdx\ ][\ j\ ] = alf\_chroma\_coeff\_abs[\ altIdx\ ][\ j\ ] * (1 - 2 * alf\_chroma\_coeff\_sign[\ altIdx\ ][\ j\ ]) \tag{92}$$

It is a requirement of bitstream conformance that the values of $AlfCoeff_C$[aps_adaptation_parameter_set_id][altIdx][j] with altIdx = 0..alf_chroma_num_alt_filters_minus1, j = 0..5 shall be in the range of -2^7 to $2^7 - 1$, inclusive.

alf_chroma_clip_idx[altIdx][j] specifies the clipping index of the clipping value to use before multiplying by the j-th coefficient of the alternative chroma filter with index altIdx. When alf_chroma_clip_idx[altIdx][j] is not present, it is inferred to be equal to 0.

The chroma filter clipping values $AlfClip_C$[aps_adaptation_parameter_set_id][altIdx] with elements $AlfClip_C$[aps_adaptation_parameter_set_id][altIdx][j], with altIdx = 0..alf_chroma_num_alt_filters_minus1, j = 0..5 are derived as specified in Table 8 depending on BitDepth and clipIdx set equal to alf_chroma_clip_idx[altIdx][j].

Table 8 – Specification AlfClip depending on BitDepth and clipIdx

BitDepth	**clipIdx**			
	0	**1**	**2**	**3**
8	2^{8}	2^{5}	2^{3}	2^{1}
9	2^{9}	2^{6}	2^{4}	2^{2}
10	2^{10}	2^{7}	2^{5}	2^{3}
11	2^{11}	2^{8}	2^{6}	2^{4}
12	2^{12}	2^{9}	2^{7}	2^{5}
13	2^{13}	2^{10}	2^{8}	2^{6}
14	2^{14}	2^{11}	2^{9}	2^{7}
15	2^{15}	2^{12}	2^{10}	2^{8}
16	2^{16}	2^{13}	2^{11}	2^{9}

alf_cc_cb_filters_signalled_minus1 plus 1 specifies the number of cross-component filters for the Cb colour component signalled in the current ALF APS. The value of alf_cc_cb_filters_signalled_minus1 shall be in the range of 0 to 3, inclusive.

alf_cc_cb_mapped_coeff_abs[k][j] specifies the absolute value of the j-th mapped coefficient of the signalled k-th cross-component filter for the Cb colour component. When alf_cc_cb_mapped_coeff_abs[k][j] is not present, it is inferred to be equal to 0.

alf_cc_cb_coeff_sign[k][j] specifies the sign of the j-th coefficient of the signalled k-th cross-component filter for the Cb colour component as follows:

– If alf_cc_cb_coeff_sign[k][j] is equal to 0, the corresponding cross-component filter coefficient has a positive value.

– Otherwise (alf_cc_cb_sign[k][j] is equal to 1), the corresponding cross-component filter coefficient has a negative value.

When alf_cc_cb_coeff_sign[k][j] is not present, it is inferred to be equal to 0.

The signalled k-th cross-component filter coefficients for the Cb colour component $CcAlfApsCoeff_{Cb}$[aps_adaptation_parameter_set_id][k][j], with j = 0..6 are derived as follows:

– If alf_cc_cb_mapped_coeff_abs[k][j] is equal to 0, $CcAlfApsCoeff_{Cb}$[aps_adaptation_parameter_set_id][k][j] is set equal to 0.

– Otherwise, $CcAlfApsCoeff_{Cb}$[aps_adaptation_parameter_set_id][k][j] is set equal to $(1 - 2 * alf\_cc\_cb\_coeff\_sign[\ k\ ][\ j\ ]) * 2^{alf\_cc\_cb\_mapped\_coeff\_abs[\ k\ ][\ j\ ] - 1}$.

alf_cc_cr_filters_signalled_minus1 plus 1 specifies the number of cross-component filters for the Cr colour component signalled in the current ALF APS. The value of alf_cc_cr_filters_signalled_minus1 shall be in the range of 0 to 3, inclusive.

alf_cc_cr_mapped coeff_abs[k][j] specifies the absolute value of the j-th mapped coefficient of the signalled k-th cross-component filter for the Cr colour component. When alf_cc_cr_mapped coeff_abs[k][j] is not present, it is inferred to be equal to 0.

alf_cc_cr_coeff_sign[k][j] specifies the sign of the j-th coefficient of the signalled k-th cross-component filter for the Cr colour component as follows:

– If alf_cc_cr_coeff_sign[k][j] is equal to 0, the corresponding cross-component filter coefficient has a positive value.

– Otherwise (alf_cc_cr_sign[k][j] is equal to 1), the corresponding cross-component filter coefficient has a negative value.

When alf_cc_cr_coeff_sign[k][j] is not present, it is inferred to be equal to 0.

The signalled k-th cross-component filter coefficients for the Cr colour component $\text{CcAlfApsCoeff}_{\text{Cr}}$[aps_adaptation_parameter_set_id][k][j], with j = 0..6 are derived as follows:

– If alf_cc_cr_mapped_coeff_abs[k][j] is equal to 0, $\text{CcAlfApsCoeff}_{\text{Cr}}$[aps_adaptation_parameter_set_id][k][j] is set equal to 0.

– Otherwise, $\text{CcAlfApsCoeff}_{\text{Cr}}$[aps_adaptation_parameter_set_id][k][j] is set equal to (1 − 2 * alf_cc_cr_coeff_sign[k][j]) * $2^{\text{alf\_cc\_cr\_mapped\_coeff\_abs[k][j]} - 1}$.

7.4.3.19 Luma mapping with chroma scaling data semantics

lmcs_min_bin_idx specifies the minimum bin index used in the luma mapping with chroma scaling construction process. The value of lmcs_min_bin_idx shall be in the range of 0 to 15, inclusive.

lmcs_delta_max_bin_idx specifies the delta value between 15 and the maximum bin index LmcsMaxBinIdx used in the luma mapping with chroma scaling construction process. The value of lmcs_delta_max_bin_idx shall be in the range of 0 to 15, inclusive. The value of LmcsMaxBinIdx is set equal to 15 − lmcs_delta_max_bin_idx. The value of LmcsMaxBinIdx shall be greater than or equal to lmcs_min_bin_idx.

lmcs_delta_cw_prec_minus1 plus 1 specifies the number of bits used for the representation of the syntax lmcs_delta_abs_cw[i]. The value of lmcs_delta_cw_prec_minus1 shall be in the range of 0 to 14, inclusive.

lmcs_delta_abs_cw[i] specifies the absolute delta codeword value for the ith bin.

lmcs_delta_sign_cw_flag[i] specifies the sign of the variable lmcsDeltaCW[i] as follows:

– If lmcs_delta_sign_cw_flag[i] is equal to 0, lmcsDeltaCW[i] is a positive value.

– Otherwise (lmcs_delta_sign_cw_flag[i] is not equal to 0), lmcsDeltaCW[i] is a negative value.

When lmcs_delta_sign_cw_flag[i] is not present, it is inferred to be equal to 0.

The variable OrgCW is derived as follows:

OrgCW = (1 << BitDepth) / 16 (93)

The variable lmcsDeltaCW[i], with i = lmcs_min_bin_idx..LmcsMaxBinIdx, is derived as follows:

lmcsDeltaCW[i] = (1 − 2 * lmcs_delta_sign_cw_flag[i]) * lmcs_delta_abs_cw[i] (94)

The variable lmcsCW[i] is derived as follows:

– For i = 0.. lmcs_min_bin_idx − 1, lmcsCW[i] is set equal 0.

– For i = lmcs_min_bin_idx..LmcsMaxBinIdx, the following applies:

lmcsCW[i] = OrgCW + lmcsDeltaCW[i] (95)

The value of lmcsCW[i] shall be in the range of OrgCW >> 3 to (OrgCW << 3) − 1, inclusive.

– For i = LmcsMaxBinIdx + 1..15, lmcsCW[i] is set equal 0.

It is a requirement of bitstream conformance that the following condition is true:

$$\sum_{i=0}^{15} \text{lmcsCW[i]} \;<=\; (1 \;<<\; \text{BitDepth}) - 1 \qquad (96)$$

The variable InputPivot[i], with i = 0..15, is derived as follows:

InputPivot[i] = i * OrgCW (97)

The variable LmcsPivot[i] with i = 0..16, the variables ScaleCoeff[i] and InvScaleCoeff[i] with i = 0..15, are derived as follows:

```
LmcsPivot[ 0 ] = 0
for( i = 0; i  <=  15; i++ ) {
    LmcsPivot[ i + 1 ] = LmcsPivot[ i ] + lmcsCW[ i ]
    ScaleCoeff[ i ] = ( lmcsCW[ i ] * (1  <<  11 ) + ( 1  <<  ( Log2( OrgCW ) − 1 ) ) )  >>  ( Log2( OrgCW ) )
    if( lmcsCW[ i ]  = =  0 )                                                                      (98)
        InvScaleCoeff[ i ] = 0
    else
        InvScaleCoeff[ i ] = OrgCW * ( 1  <<  11 ) / lmcsCW[ i ]
}
```

It is a requirement of bitstream conformance that, for i = lmcs_min_bin_idx..LmcsMaxBinIdx, when the value of LmcsPivot[i] is not a multiple of 1 << (BitDepth − 5), the value of (LmcsPivot[i] >> (BitDepth − 5)) shall not be equal to the value of (LmcsPivot[i + 1] >> (BitDepth − 5)).

lmcs_delta_abs_crs specifies the absolute codeword value of the variable lmcsDeltaCrs. When not present, lmcs_delta_abs_crs is inferred to be equal to 0.

lmcs_delta_sign_crs_flag specifies the sign of the variable lmcsDeltaCrs. When not present, lmcs_delta_sign_crs_flag is inferred to be equal to 0.

The variable lmcsDeltaCrs is derived as follows:

lmcsDeltaCrs = (1 − 2 * lmcs_delta_sign_crs_flag) * lmcs_delta_abs_crs (99)

It is a requirement of bitstream conformance that, when lmcsCW[i] is not equal to 0, (lmcsCW[i] + lmcsDeltaCrs) shall be in the range of (OrgCW >> 3) to ((OrgCW << 3) − 1), inclusive.

The variable ChromaScaleCoeff[i], with i = 0..15, is derived as follows:

```
if( lmcsCW[ i ]  = =  0 )
      ChromaScaleCoeff[ i ] = ( 1  <<  11 )
else                                                                                               (100)
      ChromaScaleCoeff[ i ] = OrgCW * ( 1  <<  11 ) / ( lmcsCW[ i ] + lmcsDeltaCrs )
```

7.4.3.20 Scaling list data semantics

scaling_list_copy_mode_flag[id] equal to 1 specifies that the values of the scaling list are the same as the values of a reference scaling list. The reference scaling list is specified by scaling_list_pred_id_delta[id]. scaling_list_copy_mode_flag[id] equal to 0 specifies that scaling_list_pred_mode_flag[id] is present. When not present, the value of scaling_list_copy_mode_flag[id] is inferred to be equal to 1.

scaling_list_pred_mode_flag[id] equal to 1 specifies that the values of the scaling list can be predicted from a reference scaling list. The reference scaling list is specified by scaling_list_pred_id_delta[id]. scaling_list_pred_mode_flag[id] equal to 0 specifies that the values of the scaling list are explicitly signalled. When not present, the value of scaling_list_pred_mode_flag[id] is inferred to be equal to 0.

scaling_list_pred_id_delta[id] specifies the reference scaling list used to derive the predicted scaling matrix scalingMatrixPred. When not present, the value of scaling_list_pred_id_delta[id] is inferred to be equal to 0. The value of scaling_list_pred_id_delta[id] shall be in the range of 0 to maxIdDelta with maxIdDelta derived depending on id as follows:

maxIdDelta = (id < 2) ? id : ((id < 8) ? (id − 2) : (id − 8)) (101)

The variables refId and matrixSize are derived as follows:

refId = id − scaling_list_pred_id_delta[id] (102)

matrixSize = (id < 2) ? 2 : ((id < 8) ? 4 : 8) (103)

The (matrixSize)x(matrixSize) array scalingMatrixPred[x][y] with x = 0..matrixSize − 1, y = 0..matrixSize − 1 and the variable scalingMatrixDcPred are derived as follows:

– When both scaling_list_copy_mode_flag[id] and scaling_list_pred_mode_flag[id] are equal to 0, all elements of scalingMatrixPred are set equal to 8, and the value of scalingMatrixDcPred is set equal to 8.

– Otherwise, if scaling_list_pred_id_delta[id] is equal to 0, all elements of scalingMatrixPred are set equal to 16, and scalingMatrixDcPred is set equal to 16.

– Otherwise (either scaling_list_copy_mode_flag[id] or scaling_list_pred_mode_flag[id] is equal to 1 and scaling_list_pred_id_delta[id] is greater than 0), scalingMatrixPred is set equal to ScalingMatrixRec[refId], and the following applies for scalingMatrixDcPred:

  – If refId is greater than 13, scalingMatrixDcPred is set equal to ScalingMatrixDcRec[refId − 14].

  – Otherwise (refId is less than or equal to 13), scalingMatrixDcPred is set equal to scalingMatrixPred[0][0].

scaling_list_dc_coef[id − 14] is used to derive the value of the variable ScalingMatrixDcRec[id − 14] when id is greater than 13 as follows:

ScalingMatrixDcRec[id − 14] = (scalingMatrixDcPred + scaling_list_dc_coef[id − 14]) & 255 (104)

When not present, the value of scaling_list_dc_coef[id − 14] is inferred to be equal to 0. The value of scaling_list_dc_coef[id – 14] shall be in the range of −128 to 127, inclusive. The value of ScalingMatrixDcRec[id − 14] shall be greater than 0.

scaling_list_delta_coef[id][i] specifies the difference between the current matrix coefficient ScalingList[id][i] and the previous matrix coefficient ScalingList[id][i − 1], when scaling_list_copy_mode_flag[id] is equal to 0. The value of scaling_list_delta_coef[id][i] shall be in the range of −128 to 127, inclusive. When scaling_list_copy_mode_flag[id] is equal to 1, all elements of ScalingList[id] are set equal to 0.

The (matrixSize)x(matrixSize) array ScalingMatrixRec[id] is derived as follows:

```
ScalingMatrixRec[ id ][ x ][ y ] = ( scalingMatrixPred[ x ][ y ] + ScalingList[ id ][ k ] ) & 255        (105)
          with k = 0..( matrixSize * matrixSize − 1 ),
               x = DiagScanOrder[ Log2( matrixSize ) ][ Log2( matrixSize ) ][ k ][ 0 ], and
               y = DiagScanOrder[ Log2( matrixSize ) ][ Log2( matrixSize ) ][ k ][ 1 ]
```

The value of ScalingMatrixRec[id][x][y] shall be greater than 0.

7.4.3.21 VUI payload semantics

vui_reserved_payload_extension_data shall not be present in bitstreams conforming to this version of this Specification. However, decoders conforming to this version of this Specification shall ignore the presence and value of vui_reserved_payload_extension_data. When present, the length, in bits, of vui_reserved_payload_extension_data is equal to 8 * payloadSize − nEarlierBits − nPayloadZeroBits − 1, where nEarlierBits is the number of bits in the vui_payload( ) syntax structure that precede the vui_reserved_payload_extension_data syntax element, and nPayloadZeroBits is the number of vui_payload_bit_equal_to_zero syntax elements at the end of the vui_payload( ) syntax structure.

If more_data_in_payload( ) is TRUE after the parsing of the vui_parameters( ) syntax structure and nPayloadZeroBits is not equal to 7, PayloadBits is set equal to 8 * payloadSize − nPayloadZeroBits − 1; otherwise, PayloadBits is equal to 8 * payloadSize.

vui_payload_bit_equal_to_one shall be equal to 1.

vui_payload_bit_equal_to_zero shall be equal to 0.

7.4.3.22 Sequence parameter set range extension semantics

sps_extended_precision_flag equal to 1 specifies that an extended dynamic range is used for transform coefficients in the scaling and transformation processes and for binarization of the abs_remaining[ ] and dec_abs_level[ ] syntax elements. sps_extended_precision_flag equal to 0 specifies that the extended dynamic range is not used in the scaling and transformation processes and is not used for binarization of the abs_remaining[ ] and dec_abs_level[ ] syntax elements. When not present, the value of sps_extended_precision_flag is inferred to be equal to 0.

The variable Log2TransformRange is derived as follows:

Log2TransformRange = sps_extended_precision_flag ? Max(15, Min(20, BitDepth + 6)) : 15 (106)

sps_ts_residual_coding_rice_present_in_sh_flag equal to 1 specifies that sh_ts_residual_coding_rice_idx_minus1 may be present in slice_header( ) syntax structures referring to the SPS. sps_ts_residual_coding_rice_present_in_sh_flag equal to 0 specifies that sh_ts_residual_coding_rice_idx_minus1 is not present in slice_header( ) syntax structures referring to the SPS. When not present, the value of sps_ts_residual_coding_rice_present_in_sh_flag is inferred to be equal to 0.

sps_rrc_rice_extension_flag equal to 1 specifies that an alternative Rice parameter derivation for the binarization of abs_remaining[ ] and dec_abs_level[ ] is used. sps_rrc_rice_extension_flag equal to 0 specifies that the alternative Rice parameter derivation for the binarization of abs_remaining[ ] and dec_abs_level[ ] is not used. When not present, the value of sps_rrc_rice_extension_flag is inferred to be equal to 0.

sps_persistent_rice_adaptation_enabled_flag equal to 1 specifies that Rice parameter derivation for the binarization of abs_remainder[ ] and dec_abs_level[ ] is initialized at the start of each TU using statistics accumulated from previous TUs. sps_persistent_rice_adaptation_enabled_flag equal to 0 specifies that no previous TU state is used in Rice parameter derivation. When not present, the value of sps_persistent_rice_adaptation_enabled_flag is inferred to be equal to 0.

sps_reverse_last_sig_coeff_enabled_flag equal to 1 specifies that sh_reverse_last_sig_coeff_flag is present in slice_header( ) syntax structures referring to the SPS. sps_reverse_last_sig_coeff_enabled_flag equal to 0 specifies that sh_reverse_last_sig_coeff_flag is not present in slice_header( ) syntax structures referring to the SPS. When not present, the value of sps_reverse_last_sig_coeff_enabled_flag is inferred to be equal to 0.

7.4.4 Profile, tier, and level semantics

7.4.4.1 General profile, tier, and level semantics

A profile_tier_level( ) syntax structure provides level information and, optionally, profile, tier, sub-profile, and general constraints information to which the OlsInScope conforms.

When the profile_tier_level( ) syntax structure is included in a VPS, the OlsInScope is one or more OLSs specified by the VPS. When the profile_tier_level( ) syntax structure is included in an SPS, the OlsInScope is the OLS that includes only the layer that is the lowest layer among the layers that refer to the SPS, and this lowest layer is an independent layer.

general_profile_idc indicates a profile to which OlsInScope conforms as specified in Annex A. Bitstreams shall not contain values of general_profile_idc other than those specified in Annex A. Other values of general_profile_idc are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore OLSs associated with a reserved value of general_profile_idc.

general_tier_flag specifies the tier context for the interpretation of general_level_idc as specified in Annex A.

general_level_idc indicates a level to which OlsInScope conforms as specified in Annex A. Bitstreams shall not contain values of general_level_idc other than those specified in Annex A. Other values of general_level_idc are reserved for future use by ITU-T | ISO/IEC.

NOTE 1 – A greater value of general_level_idc indicates a higher level. The maximum level signalled in the DCI NAL unit for OlsInScope could be higher but not be lower than the level signalled in the SPS for a CLVS contained within OlsInScope.

NOTE 2 – When OlsInScope conforms to multiple profiles, general_profile_idc is expected to indicate the profile that provides the preferred decoded result or the preferred bitstream identification, as determined by the encoder (in a manner not specified in this Specification).

NOTE 3 – When the CVSs of OlsInScope conform to different profiles, multiple profile_tier_level( ) syntax structures could be included in the DCI NAL unit such that for each CVS of the OlsInScope there is at least one set of indicated profile, tier, and level for a decoder that is capable of decoding the CVS.

ptl_frame_only_constraint_flag equal to 1 specifies that sps_field_seq_flag for all pictures in OlsInScope shall be equal to 0. ptl_frame_only_constraint_flag equal to 0 does not impose such a constraint.

NOTE 4 – Decoders could ignore the value of ptl_frame_only_constraint_flag, as there are no decoding process requirements associated with it.

ptl_multilayer_enabled_flag equal to 1 specifies that the CVSs of OlsInScope might contain more than one layer. ptl_multilayer_enabled_flag equal to 0 specifies that all slices in OlsInScope shall have the same value of nuh_layer_id, i.e., there is only one layer in the CVSs of OlsInScope.

ptl_sublayer_level_present_flag[i] equal to 1 specifies that level information is present in the profile_tier_level( ) syntax structure for the sublayer representation with TemporalId equal to i. ptl_sublayer_level_present_flag[i] equal to 0 specifies that level information is not present in the profile_tier_level( ) syntax structure for the sublayer representation with TemporalId equal to i.

ptl_reserved_zero_bit shall be equal to 0. The value 1 for ptl_reserved_zero_bit is reserved for future use by ITU-T | ISO/IEC. Decoders conforming to this version of this Specification shall ignore the value of ptl_reserved_zero_bit.

The semantics of the syntax element **sublayer_level_idc**[i] is, apart from the specification of the inference of not present values, the same as the syntax element general_level_idc, but apply to the sublayer representation with TemporalId equal to i.

When not present, the value of sublayer_level_idc[i] is inferred as follows:

– The value of sublayer_level_idc[MaxNumSubLayersMinus1] is inferred to be equal to general_level_idc of the same profile_tier_level( ) structure,
– For i from MaxNumSubLayersMinus1 − 1 to 0 (in decreasing order of values of i), inclusive, sublayer_level_idc[i] is inferred to be equal to sublayer_level_idc[i + 1].

ptl_num_sub_profiles specifies the number of the general_sub_profile_idc[i] syntax elements.

general_sub_profile_idc[i] specifies the i-th interoperability indicator registered as specified by Rec. ITU-T T.35, the contents of which are not specified in this Specification.

7.4.4.2 General constraints information semantics

gci_present_flag equal to 1 specifies that additional syntax elements are present in the general_constraints_info( ) syntax structure before the gci_alignment_zero_bit syntax elements (when present). gci_present_flag equal to 0 specifies that no additional syntax elements are present in the general_constraints_info( ) syntax structure before the gci_alignment_zero_bit syntax elements (when present).

When gci_present_flag is equal to 0, the general_constraints_info( ) syntax structure does not impose any constraints.

gci_intra_only_constraint_flag equal to 1 specifies that sh_slice_type for all slices in OlsInScope shall be equal to 2. gci_intra_only_constraint_flag equal to 0 does not impose such a constraint.

gci_all_layers_independent_constraint_flag equal to 1 specifies that vps_all_independent_layers_flag for all pictures in OlsInScope shall be equal to 1. gci_all_layers_independent_constraint_flag equal to 0 does not impose such a constraint.

gci_one_au_only_constraint_flag equal to 1 specifies that there is only one AU in OlsInScope. gci_one_au_only_constraint_flag equal to 0 does not impose such a constraint.

gci_sixteen_minus_max_bitdepth_constraint_idc greater than 0 specifies that sps_bitdepth_minus8 plus 8 for all pictures in OlsInScope shall be in the range of 0 to 16 − gci_sixteen_minus_max_bitdepth_constraint_idc, inclusive. gci_sixteen_minus_max_bitdepth_constraint_idc equal to 0 does not impose such a constraint. The value of gci_sixteen_minus_max_bitdepth_constraint_idc shall be in the range of 0 to 8, inclusive.

gci_three_minus_max_chroma_format_constraint_idc greater than 0 specifies that sps_chroma_format_idc for all pictures in OlsInScope shall be in the range of 0 to 3 − gci_three_minus_max_chroma_format_constraint_idc, inclusive. gci_three_minus_max_chroma_format_constraint_idc equal to 0 does not impose such a constraint.

gci_no_mixed_nalu_types_in_pic_constraint_flag equal to 1 specifies that pps_mixed_nalu_types_in_pic_flag for all pictures in OlsInScope shall be equal to 0. gci_no_mixed_nalu_types_in_pic_constraint_flag equal to 0 does not impose such a constraint.

gci_no_trail_constraint_flag equal to 1 specifies that there shall be no NAL unit with nuh_unit_type equal to TRAIL_NUT present in OlsInScope. gci_no_trail_constraint_flag equal to 0 does not impose such a constraint.

gci_no_stsa_constraint_flag equal to 1 specifies that there shall be no NAL unit with nuh_unit_type equal to STSA_NUT present in OlsInScope. gci_no_stsa_constraint_flag equal to 0 does not impose such a constraint.

gci_no_rasl_constraint_flag equal to 1 specifies that there shall be no NAL unit with nuh_unit_type equal to RASL_NUT present in OlsInScope. gci_no_rasl_constraint_flag equal to 0 does not impose such a constraint.

gci_no_radl_constraint_flag equal to 1 specifies that there shall be no NAL unit with nuh_unit_type equal to RADL_NUT present in OlsInScope. gci_no_radl_constraint_flag equal to 0 does not impose such a constraint.

gci_no_idr_constraint_flag equal to 1 specifies that there shall be no NAL unit with nuh_unit_type equal to IDR_W_RADL or IDR_N_LP present in OlsInScope. gci_no_idr_constraint_flag equal to 0 does not impose such a constraint.

gci_no_cra_constraint_flag equal to 1 specifies that there shall be no NAL unit with nuh_unit_type equal to CRA_NUT present in OlsInScope. gci_no_cra_constraint_flag equal to 0 does not impose such a constraint.

gci_no_gdr_constraint_flag equal to 1 specifies that sps_gdr_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_gdr_constraint_flag equal to 0 does not impose such a constraint.

gci_no_aps_constraint_flag equal to 1 specifies that there shall be no NAL unit with nuh_unit_type equal to PREFIX_APS_NUT or SUFFIX_APS_NUT present in OlsInScope, sps_ccalf_enabled_flag, sps_lmcs_enabled_flag, sps_explicit_scaling_list_enabled_flag, ph_num_alf_aps_ids_luma, ph_alf_cb_enabled_flag, and ph_alf_cr_enabled_flag for all pictures in OlsInScope shall all be equal to 0, and sh_num_alf_aps_ids_luma, sh_alf_cb_enabled_flag, sh_alf_cr_enabled_flag for all slices in OlsInScope shall be equal to 0. gci_no_aps_constraint_flag equal to 0 does not impose such a constraint.

NOTE – When no APS is referenced, it is still possible to set sps_alf_enabled_flag equal to 1 and use ALF. Therefore, when gci_no_aps_constraint_flag is equal to 1, sps_alf_enabled_flag is not required to be equal to 0.

gci_no_idr_rpl_constraint_flag equal to 1 specifies that sps_idr_rpl_present_flag for all pictures in OlsInScope shall be equal to 0. gci_no_idr_rpl_constraint_flag equal to 0 does not impose such a constraint.

gci_one_tile_per_pic_constraint_flag equal to 1 specifies that each picture in OlsInScope shall contain only one tile, i.e., the value of NumTilesInPic for each picture shall be equal to 1. gci_one_tile_per_pic_constraint_flag equal to 0 does not impose such a constraint.

gci_pic_header_in_slice_header_constraint_flag equal to 1 specifies that each picture in OlsInScope shall contain only one slice and the value of sh_picture_header_in_slice_header_flag in each slice in OlsInScope shall be equal to 1. gci_pic_header_in_slice_header_constraint_flag equal to 0 does not impose such a constraint.

gci_one_slice_per_pic_constraint_flag equal to 1 specifies that each picture in OlsInScope shall contain only one slice, i.e., if pps_rect_slice_flag is equal to 1, the value of pps_num_slices_in_pic_minus1 for each picture in OlsInScope shall be equal to 0, otherwise, the value of sh_num_tiles_in_slice_minus1 present in each slice header in OlsInScope shall be equal to NumTilesInPic − 1. gci_one_slice_per_pic_constraint_flag equal to 0 does not impose such a constraint.

gci_no_rectangular_slice_constraint_flag equal to 1 specifies that pps_rect_slice_flag for all pictures in OlsInScope shall be equal to 0. gci_no_rectangular_slice_constraint_flag equal to 0 does not impose such a constraint.

gci_one_slice_per_subpic_constraint_flag equal to 1 specifies that the value of pps_single_slice_per_subpic_flag for all pictures in OlsInScope shall be equal to 1, gci_one_slice_per_subpic_constraint_flag equal to 0 does not impose such a constraint.

gci_no_subpic_info_constraint_flag equal to 1 specifies that sps_subpic_info_present_flag for all pictures in OlsInScope shall be equal to 0. gci_no_subpic_info_constraint_flag equal to 0 does not impose such a constraint.

gci_three_minus_max_log2_ctu_size_constraint_idc greater than 0 specifies that sps_log2_ctu_size_minus5 for all pictures in OlsInScope shall be in the range of 0 to 3 − gci_three_minus_max_log2_ctu_size_constraint_idc, inclusive. gci_three_minus_max_log2_ctu_size_constraint_idc equal to 0 does not impose such a constraint.

gci_no_partition_constraints_override_constraint_flag equal to 1 specifies that sps_partition_constraints_override_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_partition_constraints_override_constraint_flag equal to 0 does not impose such a constraint.

gci_no_mtt_constraint_flag equal to 1 specifies that sps_max_mtt_hierarchy_depth_intra_slice_luma, sps_max_mtt_hierarchy_depth_inter_slice, and sps_max_mtt_hierarchy_depth_intra_slice_chroma for all pictures in OlsInScope shall be equal to 0. gci_no_mtt_constraint_flag equal to 0 does not impose such a constraint.

gci_no_qtbtt_dual_tree_intra_constraint_flag equal to 1 specifies that sps_qtbtt_dual_tree_intra_flag for all pictures in OlsInScope shall be equal to 0. gci_no_qtbtt_dual_tree_intra_constraint_flag equal to 0 does not impose such a constraint.

gci_no_palette_constraint_flag equal to 1 specifies that sps_palette_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_palette_constraint_flag equal to 0 does not impose such a constraint.

gci_no_ibc_constraint_flag equal to 1 specifies that sps_ibc_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_ibc_constraint_flag equal to 0 does not impose such a constraint.

gci_no_isp_constraint_flag equal to 1 specifies that sps_isp_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_isp_constraint_flag equal to 0 does not impose such a constraint.

gci_no_mrl_constraint_flag equal to 1 specifies that sps_mrl_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_mrl_constraint_flag equal to 0 does not impose such a constraint.

gci_no_mip_constraint_flag equal to 1 specifies that sps_mip_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_mip_constraint_flag equal to 0 does not impose such a constraint.

gci_no_cclm_constraint_flag equal to 1 specifies that sps_cclm_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_cclm_constraint_flag equal to 0 does not impose such a constraint.

gci_no_ref_pic_resampling_constraint_flag equal to 1 specifies that sps_ref_pic_resampling_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_ref_pic_resampling_constraint_flag equal to 0 does not impose such a constraint.

gci_no_res_change_in_clvs_constraint_flag equal to 1 specifies that sps_res_change_in_clvs_allowed_flag for all pictures in OlsInScope shall be equal to 0. gci_no_res_change_in_clvs_constraint_flag equal to 0 does not impose such a constraint.

gci_no_weighted_prediction_constraint_flag equal to 1 specifies that sps_weighted_pred_flag and sps_weighted_bipred_flag for all pictures in OlsInScope shall both be equal to 0. gci_no_weighted_prediction_constraint_flag equal to 0 does not impose such a constraint.

gci_no_ref_wraparound_constraint_flag equal to 1 specifies that sps_ref_wraparound_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_ref_wraparound_constraint_flag equal to 0 does not impose such a constraint.

gci_no_temporal_mvp_constraint_flag equal to 1 specifies that sps_temporal_mvp_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_temporal_mvp_constraint_flag equal to 0 does not impose such a constraint.

gci_no_sbtmvp_constraint_flag equal to 1 specifies that sps_sbtmvp_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_sbtmvp_constraint_flag equal to 0 does not impose such a constraint.

gci_no_amvr_constraint_flag equal to 1 specifies that sps_amvr_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_amvr_constraint_flag equal to 0 does not impose such a constraint.

gci_no_bdof_constraint_flag equal to 1 specifies that sps_bdof_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_bdof_constraint_flag equal to 0 does not impose such a constraint.

gci_no_smvd_constraint_flag equal to 1 specifies that sps_smvd_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_smvd_constraint_flag equal to 0 does not impose such a constraint.

gci_no_dmvr_constraint_flag equal to 1 specifies that sps_dmvr_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_dmvr_constraint_flag equal to 0 does not impose such a constraint.

gci_no_mmvd_constraint_flag equal to 1 specifies that sps_mmvd_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_mmvd_constraint_flag equal to 0 does not impose such a constraint.

gci_no_affine_motion_constraint_flag equal to 1 specifies that sps_affine_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_affine_motion_constraint_flag equal to 0 does not impose such a constraint.

gci_no_prof_constraint_flag equal to 1 specifies that sps_affine_prof_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_prof_constraint_flag equal to 0 does not impose such a constraint.

gci_no_bcw_constraint_flag equal to 1 specifies that sps_bcw_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_bcw_constraint_flag equal to 0 does not impose such a constraint.

gci_no_ciip_constraint_flag equal to 1 specifies that sps_ciip_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_ciip_constraint_flag equal to 0 does not impose such a constraint.

gci_no_gpm_constraint_flag equal to 1 specifies that sps_gpm_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_gpm_constraint_flag equal to 0 does not impose such a constraint.

gci_no_luma_transform_size_64_constraint_flag equal to 1 specifies that sps_max_luma_transform_size_64_flag for all pictures in OlsInScope shall be equal to 0. gci_no_luma_transform_size_64_constraint_flag equal to 0 does not impose such a constraint.

gci_no_transform_skip_constraint_flag equal to 1 specifies that sps_transform_skip_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_transform_skip_constraint_flag equal to 0 does not impose such a constraint.

gci_no_bdpcm_constraint_flag equal to 1 specifies that sps_bdpcm_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_bdpcm_constraint_flag equal to 0 does not impose such a constraint.

gci_no_mts_constraint_flag equal to 1 specifies that sps_mts_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_mts_constraint_flag equal to 0 does not impose such a constraint.

gci_no_lfnst_constraint_flag equal to 1 specifies that sps_lfnst_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_lfnst_constraint_flag equal to 0 does not impose such a constraint.

gci_no_joint_cbcr_constraint_flag equal to 1 specifies that sps_joint_cbcr_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_joint_cbcr_constraint_flag equal to 0 does not impose such a constraint.

gci_no_sbt_constraint_flag equal to 1 specifies that sps_sbt_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_sbt_constraint_flag equal to 0 does not impose such a constraint.

gci_no_act_constraint_flag equal to 1 specifies that sps_act_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_act_constraint_flag equal to 0 does not impose such a constraint.

gci_no_explicit_scaling_list_constraint_flag equal to 1 specifies that sps_explicit_scaling_list_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_explicit_scaling_list_constraint_flag equal to 0 does not impose such a constraint.

gci_no_dep_quant_constraint_flag equal to 1 specifies that sps_dep_quant_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_dep_quant_constraint_flag equal to 0 does not impose such a constraint.

gci_no_sign_data_hiding_constraint_flag equal to 1 specifies that sps_sign_data_hiding_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_sign_data_hiding_constraint_flag equal to 0 does not impose such a constraint.

gci_no_cu_qp_delta_constraint_flag equal to 1 specifies that pps_cu_qp_delta_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_cu_qp_delta_constraint_flag equal to 0 does not impose such a constraint.

gci_no_chroma_qp_offset_constraint_flag equal to 1 specifies that pps_cu_chroma_qp_offset_list_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_chroma_qp_offset_constraint_flag equal to 0 does not impose such a constraint.

gci_no_sao_constraint_flag equal to 1 specifies that sps_sao_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_sao_constraint_flag equal to 0 does not impose such a constraint.

gci_no_alf_constraint_flag equal to 1 specifies that sps_alf_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_alf_constraint_flag equal to 0 does not impose such a constraint.

gci_no_ccalf_constraint_flag equal to 1 specifies that sps_ccalf_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_ccalf_constraint_flag equal to 0 does not impose such a constraint.

gci_no_lmcs_constraint_flag equal to 1 specifies that sps_lmcs_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_lmcs_constraint_flag equal to 0 does not impose such a constraint.

gci_no_ladf_constraint_flag equal to 1 specifies that sps_ladf_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_ladf_constraint_flag equal to 0 does not impose such a constraint.

gci_no_virtual_boundaries_constraint_flag equal to 1 specifies that sps_virtual_boundaries_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_virtual_boundaries_constraint_flag equal to 0 does not impose such a constraint.

gci_num_additional_bits specifies the number of the additional GCI bits in the general constraints information syntax structure other than gci_alignment_zero_bit syntax elements (when present). The value of gci_num_additional_bits shall be equal to 0 or 6 in bitstreams conforming to this version of this document. Values greater than 6 for gci_num_additional_bits are reserved for future use by ITU-T | ISO/IEC. Although the value of gci_num_additional_bits is required to be equal to 0 or 6 in this version of this document, decoders conforming to this version of this document shall also allow values of gci_num_additional_bits greater than 6 to appear in the syntax and shall ignore the values of the gci_reserved_bit[i] syntax elements when present.

gci_all_rap_pictures_constraint_flag equal to 1 specifies that all pictures in OlsInScope are GDR pictures with ph_recovery_poc_cnt equal to 0 or IRAP pictures. gci_all_rap_pictures_constraint_flag equal to 0 does not impose such a constraint. When gci_all_rap_pictures_constraint_flag is not present, its value is inferred to be equal to 0.

gci_no_extended_precision_processing_constraint_flag equal to 1 specifies that sps_extended_precision_flag for all pictures in OlsInScope shall be equal to 0. gci_no_extended_precision_processing_constraint_flag equal to 0 does not impose such a constraint. When gci_no_extended_precision_processing_constraint_flag is not present, its value is inferred to be equal to 0.

gci_no_ts_residual_coding_rice_constraint_flag equal to 1 specifies that sps_ts_residual_coding_rice_present_in_sh_flag for all pictures in OlsInScope shall be equal to 0. gci_no_ts_residual_coding_rice_constraint_flag equal to 0 does not impose such a constraint. When gci_no_ts_residual_coding_rice_constraint_flag is not present, its value is inferred to be equal to 0.

gci_no_rrc_rice_extension_constraint_flag equal to 1 specifies that sps_rrc_rice_extension_flag for all pictures in OlsInScope shall be equal to 0. gci_no_rrc_rice_extension_constraint_flag equal to 0 does not impose such a constraint. When gci_no_rrc_rice_extension_constraint_flag is not present, its value is inferred to be equal to 0.

gci_no_persistent_rice_adaptation_constraint_flag equal to 1 specifies that sps_persistent_rice_adaptation_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_persistent_rice_adaptation_constraint_flag equal to 0 does not impose such a constraint. When gci_no_persistent_rice_adaptation_constraint_flag is not present, its value is inferred to be equal to 0.

gci_no_reverse_last_sig_coeff_constraint_flag equal to 1 specifies that sps_reverse_last_sig_coeff_enabled_flag for all pictures in OlsInScope shall be equal to 0. gci_no_reverse_last_sig_coeff_constraint_flag equal to 0 does not impose such a constraint. When gci_no_reverse_last_sig_coeff_constraint_flag is not present, its value is inferred to be equal to 0.

gci_reserved_bit[i] could have any value. Its presence and value do not affect the decoding process specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore the values of the gci_reserved_bit[i] syntax elements.

gci_alignment_zero_bit shall be equal to 0.

7.4.5 DPB parameters semantics

The dpb_parameters( ) syntax structure provides information of DPB size, maximum picture reorder number, and maximum latency for one or more OLSs.

When a dpb_parameters( ) syntax structure is included in a VPS, the OLSs to which the dpb_parameters( ) syntax structure applies are specified by the VPS. When a dpb_parameters( ) syntax structure is included in an SPS, it applies to the OLS that includes only the layer that is the lowest layer among the layers that refer to the SPS, and this lowest layer is an independent layer.

dpb_max_dec_pic_buffering_minus1[i] plus 1 specifies the maximum required size of the DPB in units of picture storage buffers when Htid is equal to i. The value of dpb_max_dec_pic_buffering_minus1[i] shall be in the range of 0 to MaxDpbSize − 1, inclusive, where MaxDpbSize is as specified in clause A.4.2. When i is greater than 0, dpb_max_dec_pic_buffering_minus1[i] shall be greater than or equal to dpb_max_dec_pic_buffering_minus1[i − 1]. When dpb_max_dec_pic_buffering_minus1[i] is not present for i in the range of 0 to MaxSubLayersMinus1 − 1, inclusive, due to subLayerInfoFlag being equal to 0, it is inferred to be equal to dpb_max_dec_pic_buffering_minus1[MaxSubLayersMinus1].

dpb_max_num_reorder_pics[i] specifies the maximum allowed number of pictures of the OLS that can precede any picture in the OLS in decoding order and follow that picture in output order when Htid is equal to i. The value of dpb_max_num_reorder_pics[i] shall be in the range of 0 to dpb_max_dec_pic_buffering_minus1[i], inclusive. When i is greater than 0, dpb_max_num_reorder_pics[i] shall be greater than or equal to dpb_max_num_reorder_pics[i − 1]. When dpb_max_num_reorder_pics[i] is not present for i in the range of 0 to MaxSubLayersMinus1 − 1, inclusive, due to subLayerInfoFlag being equal to 0, it is inferred to be equal to dpb_max_num_reorder_pics[MaxSubLayersMinus1].

dpb_max_latency_increase_plus1[i] not equal to 0 is used to compute the value of MaxLatencyPictures[i], which specifies the maximum number of pictures in the OLS that can precede any picture in the OLS in output order and follow that picture in decoding order when Htid is equal to i.

When dpb_max_latency_increase_plus1[i] is not equal to 0, the value of MaxLatencyPictures[i] is specified as follows:

$$\text{MaxLatencyPictures[i]} = \text{dpb\_max\_num\_reorder\_pics[i]} + \text{dpb\_max\_latency\_increase\_plus1[i]} - 1 \qquad (107)$$

When dpb_max_latency_increase_plus1[i] is equal to 0, no corresponding limit is expressed.

The value of dpb_max_latency_increase_plus1[i] shall be in the range of 0 to $2^{32} - 2$, inclusive. When dpb_max_latency_increase_plus1[i] is not present for i in the range of 0 to MaxSubLayersMinus1 − 1, inclusive, due to subLayerInfoFlag being equal to 0, it is inferred to be equal to dpb_max_latency_increase_plus1[MaxSubLayersMinus1].

7.4.6 Timing and HRD parameters semantics

7.4.6.1 General timing and HRD parameters semantics

The general_timing_hrd_parameters( ) syntax structure provides some of the sequence-level HRD parameters used in the HRD operations.

It is a requirement of bitstream conformance that the content of the general_timing_hrd_parameters( ) syntax structure present in any VPSs or SPSs in the bitstream shall be identical.

When included in a VPS, the general_timing_hrd_parameters( ) syntax structure applies to all OLSs specified by the VPS. When included in an SPS, the general_timing_hrd_parameters( ) syntax structure applies to the OLS that includes only the layer that is the lowest layer among the layers that refer to the SPS, and this lowest layer is an independent layer.

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, in units of seconds, is equal to the quotient of num_units_in_tick divided by time_scale. For example, when the picture rate of a video signal is 25 Hz, time_scale and num_units_in_tick could be equal to 27 000 000 and 1 080 000, respectively, and consequently a clock tick would be equal to 0.04 seconds.

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. The value of time_scale shall be greater than 0.

general_nal_hrd_params_present_flag equal to 1 specifies that NAL HRD parameters (pertaining to Type II bitstream conformance point) are present in the general_timing_hrd_parameters( ) syntax structure. general_nal_hrd_params_present_flag equal to 0 specifies that NAL HRD parameters are not present in the general_timing_hrd_parameters( ) syntax structure.

NOTE 1 – When general_nal_hrd_params_present_flag is equal to 0, the conformance of the bitstream might not be verified without provision of the NAL HRD parameters and all BP SEI messages, and, when general_vcl_hrd_params_present_flag is also equal to 0, all PT and DUI SEI messages, by some means not specified in this Specification.

The variable NalHrdBpPresentFlag is derived as follows:

– If one or more of the following conditions are true, the value of NalHrdBpPresentFlag is set equal to 1:
  – general_nal_hrd_params_present_flag is present in the bitstream and is equal to 1.
  – The need for presence of BPs for NAL HRD operation to be present in the bitstream in BP SEI messages is determined by the application, by some means not specified in this Specification.
– Otherwise, the value of NalHrdBpPresentFlag is set equal to 0.

general_vcl_hrd_params_present_flag equal to 1 specifies that VCL HRD parameters (pertaining to Type I bitstream conformance point) are present in the general_timing_hrd_parameters( ) syntax structure. general_vcl_hrd_params_present_flag equal to 0 specifies that VCL HRD parameters are not present in the general_timing_hrd_parameters( ) syntax structure.

NOTE 2 – When general_vcl_hrd_params_present_flag is equal to 0, the conformance of the bitstream might not be verified without provision of the VCL HRD parameters and all BP SEI messages, and when general_nal_hrd_params_present_flag is also equal to 0, all PT and DUI SEI messages, by some means not specified in this Specification.

The variable VclHrdBpPresentFlag is derived as follows:

– If one or more of the following conditions are true, the value of VclHrdBpPresentFlag is set equal to 1:
  – general_vcl_hrd_params_present_flag is present in the bitstream and is equal to 1.
  – The need for presence of BPs for VCL HRD operation to be present in the bitstream in BP SEI messages is determined by the application, by some means not specified in this Specification.
– Otherwise, the value of VclHrdBpPresentFlag is set equal to 0.

The variable CpbDpbDelaysPresentFlag is derived as follows:

– If one or more of the following conditions are true, the value of CpbDpbDelaysPresentFlag is set equal to 1:
  – general_nal_hrd_params_present_flag is present in the bitstream and is equal to 1.
  – general_vcl_hrd_params_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 PT SEI messages is determined by the application, by some means not specified in this Specification.
– Otherwise, the value of CpbDpbDelaysPresentFlag is set equal to 0.

general_same_pic_timing_in_all_ols_flag equal to 1 specifies that the non-scalable-nested PT SEI message in each AU applies to the AU for any OLS in the bitstream and no scalable-nested PT SEI messages are present. general_same_pic_timing_in_all_ols_flag equal to 0 specifies that the non-scalable-nested PT SEI message in each AU might or might not apply to the AU for any OLS in the bitstream and scalable-nested PT SEI messages might be present.

general_du_hrd_params_present_flag equal to 1 specifies that DU level HRD parameters are present and the HRD could operate at the AU level or DU level. general_du_hrd_params_present_flag equal to 0 specifies that DU level HRD parameters are not present and the HRD operates at the AU level. When general_du_hrd_params_present_flag is not present, its value is inferred to be equal to 0.

tick_divisor_minus2 is used to specify the clock sub-tick. A clock sub-tick is the minimum interval of time that can be represented in the coded data when general_du_hrd_params_present_flag is equal to 1.

bit_rate_scale (together with bit_rate_value_minus1[i][j]) specifies the maximum input bit rate of the j-th CPB when Htid is equal to i.

cpb_size_scale (together with cpb_size_value_minus1[i][j]) specifies the CPB size of the j-th CPB when Htid is equal to i and when the CPB operates at the AU level.

cpb_size_du_scale (together with cpb_size_du_value_minus1[i][j]) specifies the CPB size of the j-th CPB when Htid is equal to i and when the CPB operates at the DU level.

hrd_cpb_cnt_minus1 plus 1 specifies the number of alternative CPB delivery schedules. The value of hrd_cpb_cnt_minus1 shall be in the range of 0 to 31, inclusive.

7.4.6.2 OLS timing and HRD parameters semantics

When an ols_timing_hrd_parameters( ) syntax structure is included in a VPS, the OLSs to which the ols_timing_hrd_parameters( ) syntax structure applies are specified by the VPS. When an ols_timing_hrd_parameters( ) syntax structure is included in an SPS, the ols_timing_hrd_parameters( ) syntax structure applies to the OLS that includes only the layer that is the lowest layer among the layers that refer to the SPS, and this lowest layer is an independent layer.

fixed_pic_rate_general_flag[i] equal to 1 indicates that, when Htid is equal to i, the temporal distance between the HRD output times of consecutive pictures in output order is constrained as specified in this clause using the variable DpbOutputElementalInterval[n]. fixed_pic_rate_general_flag[i] equal to 0 indicates that this constraint might not apply.

When fixed_pic_rate_general_flag[i] is not present, it is inferred to be equal to 0.

fixed_pic_rate_within_cvs_flag[i] equal to 1 indicates that, when Htid is equal to i, the temporal distance between the HRD output times of consecutive pictures in output order is constrained as specified in this clause using the variable DpbOutputElementalInterval[n]. fixed_pic_rate_within_cvs_flag[i] equal to 0 indicates that this constraint might not apply.

When fixed_pic_rate_general_flag[i] is equal to 1, the value of fixed_pic_rate_within_cvs_flag[i] is inferred to be equal to 1.

It is a requirement of bitstream conformance that when general_nal_hrd_params_present_flag and general_vcl_hrd_params_present_flag are both equal to 0, there shall be at least one value of fixed_pic_rate_within_cvs_flag[i] equal to 1 for i in the range of 0 to MaxSubLayersVal − 1, inclusive.

When present, **elemental_duration_in_tc_minus1**[i] plus 1 specifies, when Htid is equal to i, the temporal distance, in clock ticks, between the elemental units that specify the HRD output times of consecutive pictures in output order. The value of elemental_duration_in_tc_minus1[i] shall be in the range of 0 to 2047, inclusive.

When Htid is equal to i and fixed_pic_rate_within_cvs_flag[i] is equal to 1 for a CVS containing picture n, and picture n is a picture that is output and is not the last picture in the bitstream (in output order) that is output, the value of the variable DpbOutputElementalInterval[n] is specified by:

$$\text{DpbOutputElementalInterval[n]} = \text{DpbOutputInterval[n]} \div (\text{ pt\_display\_elemental\_periods\_minus1} + 1) \quad (108)$$

where DpbOutputInterval[n] is specified in Equation 1605.

When Htid is equal to i and fixed_pic_rate_within_cvs_flag[i] is equal to 1 for a CVS containing picture n, and picture n is a picture that is output and is not the last picture in the bitstream (in output order) that is output, the value computed for DpbOutputElementalInterval[n] shall be equal to ClockTick * (elemental_duration_in_tc_minus1[i] + 1), wherein ClockTick is as specified in Equation 1590 (using the value of ClockTick for the CVS containing picture n) when one of the following conditions is true for the following picture in output order nextPicInOutputOrder that is specified for use in Equation 1605:

– picture nextPicInOutputOrder is in the same CVS as picture n.

– picture nextPicInOutputOrder is in a different CVS and fixed_pic_rate_general_flag[i] is equal to 1 in the CVS containing picture nextPicInOutputOrder, the value of ClockTick is the same for both CVSs, and the value of elemental_duration_in_tc_minus1[i] is the same for both CVSs.

low_delay_hrd_flag[i] specifies the HRD operational mode, when Htid is equal to i, as specified in Annex C. When not present, the value of low_delay_hrd_flag[i] is inferred to be equal to 0.

NOTE 3 – When low_delay_hrd_flag[i] is equal to 1, "big pictures" that violate the nominal CPB removal times due to the number of bits used by an AU are permitted. It is expected, but not required, that such "big pictures" occur only occasionally.

7.4.6.3 Sublayer HRD parameters semantics

When the sublayer_hrd_parameters( ) syntax structure is included in the i-th ols_timing_hrd_parameters( ) syntax structure in a VPS, the value of the variable maxSubLayersMinus1 is set equal to vps_hrd_max_tid[i], and the value of the variable timingHrdParamsPresentFlag is set equal to vps_timing_hrd_params_present_flag. When the sublayer_hrd_parameters( ) syntax structure is included in the ols_timing_hrd_parameters( ) syntax structure in an SPS, the value of maxSubLayersMinus1 is set equal to sps_max_sublayers_minus1, and the value of timingHrdParamsPresentFlag is set equal to sps_timing_hrd_params_present_flag.

bit_rate_value_minus1[i][j] (together with bit_rate_scale) specifies the maximum input bit rate for the j-th CPB with Htid equal to i when the CPB operates at the AU level. bit_rate_value_minus1[i][j] shall be in the range of 0 to $2^{32} - 2$, inclusive. For any j greater than 0 and any particular value of i, bit_rate_value_minus1[i][j] shall be greater than bit_rate_value_minus1[i][j − 1].

When DecodingUnitHrdFlag is equal to 0, the following applies:

– The bit rate in bits per second is given by:

$$\text{BitRate[i][j]} = (\text{ bit\_rate\_value\_minus1[i][j]} + 1\text{ }) * 2^{(\text{ }6 + \text{bit\_rate\_scale })} \quad (109)$$

– When the bit_rate_value_minus1[i][j] syntax element is not present, it is inferred as follows:

 – If timingHrdParamsPresentFlag is equal to 1, bit_rate_value_minus1[i][j] is inferred to be equal to bit_rate_value_minus1[maxSubLayersMinus1][j].

 – Otherwise (timingHrdParamsPresentFlag is equal to 0), the value of BitRate[i][j] is inferred to be equal to BrVclFactor * MaxBR for VCL HRD parameters and to be equal to BrNalFactor * MaxBR for NAL HRD parameters, where MaxBR, BrVclFactor and BrNalFactor are specified in Annex A.

cpb_size_value_minus1[i][j] is used together with cpb_size_scale to specify the j-th CPB size with Htid equal to i when the CPB operates at the AU level. cpb_size_value_minus1[i][j] shall be in the range of 0 to $2^{32} - 2$, inclusive. For any j greater than 0 and any particular value of i, cpb_size_value_minus1[i][j] shall be less than or equal to cpb_size_value_minus1[i][j − 1].

When DecodingUnitHrdFlag is equal to 0, the following apples:

– The CPB size in bits is given by:

$$\text{CpbSize[i][j]} = (\text{ cpb\_size\_value\_minus1[i][j]} + 1\text{ }) * 2^{(\text{ }4 + \text{cpb\_size\_scale })} \quad (110)$$

– When the cpb_size_value_minus1[i][j] syntax element is not present, it is inferred as follows:

 – If timingHrdParamsPresentFlag is equal to 1, cpb_size_value_minus1[i][j] is inferred to be equal to cpb_size_value_minus1[maxSubLayersMinus1][j].

 – Otherwise (timingHrdParamsPresentFlag is equal to 0), the value of CpbSize[i][j] is inferred to be equal to BrVclFactor * MaxCPB for VCL HRD parameters and to be equal to BrNalFactor * MaxCPB for NAL HRD parameters, where MaxCPB, BrVclFactor and BrNalFactor are specified in Annex A.

cpb_size_du_value_minus1[i][j] is used together with cpb_size_du_scale to specify the i-th CPB size with Htid equal to i when the CPB operates at the DU level. cpb_size_du_value_minus1[i][j] shall be in the range of 0 to $2^{32} - 2$, inclusive. For any j greater than 0 and any particular value of i, cpb_size_du_value_minus1[i][j] shall be less than or equal to cpb_size_du_value_minus1[i][j − 1].

When DecodingUnitHrdFlag is equal to 1, the following applies:

– The CPB size in bits is given by:

$$\text{CpbSize[i][j]} = (\text{ cpb\_size\_du\_value\_minus1[i][j]} + 1\text{ }) * 2^{(\text{ }4 + \text{cpb\_size\_du\_scale })} \quad (111)$$

– When the cpb_size_du_value_minus1[i][j] syntax element is not present, it is inferred as follows:

 – If timingHrdParamsPresentFlag is equal to 1, cpb_size_du_value_minus1[i][j] is inferred to be equal to cpb_size_du_value_minus1[maxSubLayersMinus1][j].

 – Otherwise (timingHrdParamsPresentFlag is equal to 0), the value of CpbSize[i][j] is inferred to be equal to CpbVclFactor * MaxCPB for VCL HRD parameters and to be equal to CpbNalFactor * MaxCPB for NAL HRD parameters, where MaxCPB, CpbVclFactor and CpbNalFactor are specified in Annex A.

bit_rate_du_value_minus1[i][j] (together with bit_rate_scale) specifies the maximum input bit rate for the j-th CPB with Htid equal to i when the CPB operates at the DU level. bit_rate_du_value_minus1[i][j] shall be in the range of 0 to $2^{32} - 2$, inclusive. For any j greater than 0 and any particular value of i, bit_rate_du_value_minus1[i][j] shall be greater than bit_rate_du_value_minus1[i][j − 1].

When DecodingUnitHrdFlag is equal to 1, the folowing applies:

– The bit rate in bits per second is given by:

$$\text{BitRate[i][j]} = (\text{ bit\_rate\_du\_value\_minus1[i][j]} + 1\text{ }) * 2^{(\text{ }6 + \text{bit\_rate\_scale })} \quad (112)$$

– When the bit_rate_du_value_minus1[i][j] syntax element is not present, it is inferred as follows:

  – If timingHrdParamsPresentFlag is equal to 1, bit_rate_du_value_minus1[i][j] is inferred to be equal to bit_rate_du_value_minus1[maxSubLayersMinus1][j].

  – Otherwise (timingHrdParamsPresentFlag is equal to 0), the value of BitRate[i][j] is inferred to be equal to BrVclFactor * MaxBR for VCL HRD parameters and to be equal to BrNalFactor * MaxBR for NAL HRD parameters, where MaxBR, BrVclFactor and BrNalFactor are specified in Annex A.

cbr_flag[i][j] equal to 0 specifies that to decode this bitstream by the HRD using the j-th CPB specification, the hypothetical stream scheduler (HSS) operates in an intermittent bit rate mode. cbr_flag[i][j] equal to 1 specifies that the HSS operates in a constant bit rate (CBR) mode.

When not present, the value of cbr_flag[i][j] it is inferred as follows:

– When the cbr_flag[i][j] syntax element is not present, it is inferred as follows:

  – If timingHrdParamsPresentFlag is equal to 1, cbr_flag[i][j] is inferred to be equal to cbr_flag[maxSubLayersMinus1][j].

  – Otherwise (timingHrdParamsPresentFlag is equal to 0), the value of cbr_flag[i][j] is inferred to be equal to 0.

7.4.7 Supplemental enhancement information message semantics

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

payload_type_byte is a byte of the payload type of an SEI message.

payload_size_byte is a byte of the payload size of an SEI message.

7.4.8 Slice header semantics

The variable CuQpDeltaVal, specifying the difference between a luma quantization parameter for the coding unit containing cu_qp_delta_abs and its prediction, is set equal to 0. The variables $\text{CuQpOffset}_{\text{Cb}}$, $\text{CuQpOffset}_{\text{Cr}}$, and $\text{CuQpOffset}_{\text{CbCr}}$, specifying values to be used when determining the respective values of the Qp'_{Cb}, Qp'_{Cr}, and Qp'_{CbCr} quantization parameters for the coding unit containing cu_chroma_qp_offset_flag, are all set equal to 0.

sh_picture_header_in_slice_header_flag equal to 1 specifies that the PH syntax structure is present in the slice header. sh_picture_header_in_slice_header_flag equal to 0 specifies that the PH syntax structure is not present in the slice header.

It is a requirement of bitstream conformance that the value of sh_picture_header_in_slice_header_flag shall be the same in all coded slices in a CLVS.

When sh_picture_header_in_slice_header_flag is equal to 1 for a coded slice, it is a requirement of bitstream conformance that no NAL unit with nal_unit_type equal to PH_NUT shall be present in the CLVS.

When sh_picture_header_in_slice_header_flag is equal to 0, all coded slices in the current picture shall have sh_picture_header_in_slice_header_flag equal to 0, and the current PU shall have a PH NAL unit.

When any of the following conditions is true, the value of sh_picture_header_in_slice_header_flag shall be equal to 0:

– The value of sps_subpic_info_present_flag is equal to 1.

– The value of pps_rect_slice_flag is equal to 0.

– The value of pps_rpl_info_in_ph_flag, pps_dbf_info_in_ph_flag, pps_sao_info_in_ph_flag, pps_alf_info_in_ph_flag, pps_wp_info_in_ph_flag, or pps_qp_delta_info_in_ph_flag is equal to 1.

sh_subpic_id specifies the subpicture ID of the subpicture that contains the slice. If sh_subpic_id is present, the value of the variable CurrSubpicIdx is derived to be such that SubpicIdVal[CurrSubpicIdx] is equal to sh_subpic_id. Otherwise (sh_subpic_id is not present), CurrSubpicIdx is derived to be equal to 0. The length of sh_subpic_id is sps_subpic_id_len_minus1 + 1 bits.

sh_slice_address specifies the slice address of the slice. When not present, the value of sh_slice_address is inferred to be equal to 0.

If pps_rect_slice_flag is equal to 0, the following applies:

– The slice address is the raster scan tile index of the first tile in the slice.

– The length of sh_slice_address is Ceil(Log2 (NumTilesInPic)) bits.

– The value of sh_slice_address shall be in the range of 0 to NumTilesInPic − 1, inclusive.

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

– The slice address is the subpicture-level slice index of the current slice, i.e., SubpicLevelSliceIdx[j], where j is the picture-level slice index of the current slice.

– The length of sh_slice_address is Ceil(Log2(NumSlicesInSubpic[CurrSubpicIdx])) bits.

– The value of sh_slice_address shall be in the range of 0 to NumSlicesInSubpic[CurrSubpicIdx] − 1, inclusive.

It is a requirement of bitstream conformance that the following constraints apply:

– If pps_rect_slice_flag is equal to 0 or sps_subpic_info_present_flag is equal to 0, the value of sh_slice_address shall not be equal to the value of sh_slice_address of any other coded slice NAL unit of the same coded picture.

– Otherwise, the pair of sh_subpic_id and sh_slice_address values shall not be equal to the pair of sh_subpic_id and sh_slice_address values of any other coded slice NAL unit of the same coded picture.

– The shapes of the slices of a picture shall be such that each CTU, when decoded, shall have its entire left boundary and entire top boundary consisting of a picture boundary or consisting of boundaries of previously decoded CTU(s).

sh_extra_bit[i] could have any value. Decoders conforming to this version of this Specification shall ignore the presence and value of sh_extra_bit[i]. Its value does not affect the decoding process specified in this version of this Specification.

sh_num_tiles_in_slice_minus1 plus 1, when present, specifies the number of tiles in the slice. The value of sh_num_tiles_in_slice_minus1 shall be in the range of 0 to NumTilesInPic − 1, inclusive. When not present, the value of sh_num_tiles_in_slice_minus1 shall be inferred to be equal to 0.

The variable NumCtusInCurrSlice, which specifies the number of CTUs in the current slice, and the list CtbAddrInCurrSlice[i], for i ranging from 0 to NumCtusInCurrSlice − 1, inclusive, specifying the picture raster scan address of the i-th CTB within the slice, are derived as follows:

```
if( pps_rect_slice_flag ) {
    picLevelSliceIdx = sh_slice_address
    for( j = 0; j < CurrSubpicIdx; j++ )
        picLevelSliceIdx  +=  NumSlicesInSubpic[ j ]
    NumCtusInCurrSlice = NumCtusInSlice[ picLevelSliceIdx ]
    for( i = 0; i < NumCtusInCurrSlice; i++ )
        CtbAddrInCurrSlice[ i ] = CtbAddrInSlice[ picLevelSliceIdx ][ i ]                    (113)
} else {
    NumCtusInCurrSlice = 0
    for( tileIdx = sh_slice_address; tileIdx  <=  sh_slice_address + sh_num_tiles_in_slice_minus1; tileIdx++ )
{
        tileX = tileIdx % NumTileColumns
        tileY = tileIdx / NumTileColumns
        for( ctbY = TileRowBdVal[ tileY ]; ctbY < TileRowBdVal[ tileY + 1 ]; ctbY++ ) {
            for( ctbX = TileColBdVal[ tileX ]; ctbX < TileColBdVal[ tileX + 1 ]; ctbX++ ) {
                CtbAddrInCurrSlice[ NumCtusInCurrSlice ] = ctbY * PicWidthInCtbsY + ctbX
                NumCtusInCurrSlice++
            }
        }
    }
}
```

The variables SubpicLeftBoundaryPos, SubpicTopBoundaryPos, SubpicRightBoundaryPos, and SubpicBotBoundaryPos are derived as follows:

```
if( sps_subpic_treated_as_pic_flag[ CurrSubpicIdx ] ) {
    SubpicLeftBoundaryPos = sps_subpic_ctu_top_left_x[ CurrSubpicIdx ] * CtbSizeY
    SubpicRightBoundaryPos = Min( pps_pic_width_in_luma_samples − 1,
```

(sps_subpic_ctu_top_left_x[CurrSubpicIdx] +
sps_subpic_width_minus1[CurrSubpicIdx] + 1) * CtbSizeY − 1)
SubpicTopBoundaryPos = sps_subpic_ctu_top_left_y[CurrSubpicIdx] *CtbSizeY (114)
SubpicBotBoundaryPos = Min(pps_pic_height_in_luma_samples − 1,
(sps_subpic_ctu_top_left_y[CurrSubpicIdx] +
sps_subpic_height_minus1[CurrSubpicIdx] + 1) * CtbSizeY − 1)
}

sh_slice_type specifies the coding type of the slice according to Table 9.

Table 9 – Name association to sh_slice_type

sh_slice_type	Name of sh_slice_type
0	B (B slice)
1	P (P slice)
2	I (I slice)

When not present, the value of sh_slice_type is inferred to be equal to 2.

When ph_intra_slice_allowed_flag is equal to 0, the value of sh_slice_type shall be equal to 0 or 1.

When both of the following conditions are true, the value of sh_slice_type shall be equal to 2:

– The value of nal_unit_type is in the range of IDR_W_RADL to CRA_NUT, inclusive.

– The value of vps_independent_layer_flag[GeneralLayerIdx[nuh_layer_id]] is equal to 1 or the current picture is the first picture in the current AU.

When sps_subpic_treated_as_pic_flag[CurrSubpicIdx] is equal to 0, pps_mixed_nalu_types_in_pic_flag is equal to 1 (i.e., there are at least two subpictures in the current picture having different NAL unit types), the value of sh_slice_type shall be equal to 2.

NOTE 1 – This constraint is technically equivalent to the following: "When pps_mixed_nalu_types_in_pic_flag for a picture is equal to 1 (i.e., there are at least two subpictures in a picture having different NAL unit types), the value of sps_subpic_treated_as_pic_flag[ ] shall be equal to 1 for all the subpictures that are in the picture and contain at least one P or B slice."

sh_no_output_of_prior_pics_flag affects the output of previously-decoded pictures in the DPB after the decoding of a picture in a CVSS AU that is not the first AU in the bitstream as specified in Annex C.

It is a requirement of bitstream conformance that the value of sh_no_output_of_prior_pics_flag shall be the same for all slices in an AU that have sh_no_output_of_prior_pics_flag present in the SHs.

When all slices in an AU have sh_no_output_of_prior_pics_flag present in the SHs, the sh_no_output_of_prior_pics_flag in the SHs is also referred to as the sh_no_output_of_prior_pics_flag of the AU.

The variables MinQtLog2SizeY, MinQtLog2SizeC, MinQtSizeY, MinQtSizeC, MaxBtSizeY, MaxBtSizeC, MinBtSizeY, MaxTtSizeY, MaxTtSizeC, MinTtSizeY, MaxMttDepthY and MaxMttDepthC are derived as follows:

– If sh_slice_type equal to 2 (I), the following applies:

MinQtLog2SizeY = MinCbLog2SizeY + ph_log2_diff_min_qt_min_cb_intra_slice_luma (115)

MinQtLog2SizeC = MinCbLog2SizeY + ph_log2_diff_min_qt_min_cb_intra_slice_chroma (116)

MaxBtSizeY = 1 << (MinQtLog2SizeY + ph_log2_diff_max_bt_min_qt_intra_slice_luma) (117)

MaxBtSizeC = 1 << (MinQtLog2SizeC + ph_log2_diff_max_bt_min_qt_intra_slice_chroma) (118)

MaxTtSizeY = 1 << (MinQtLog2SizeY + ph_log2_diff_max_tt_min_qt_intra_slice_luma) (119)

MaxTtSizeC = 1 << (MinQtLog2SizeC + ph_log2_diff_max_tt_min_qt_intra_slice_chroma) (120)

MaxMttDepthY = ph_max_mtt_hierarchy_depth_intra_slice_luma (121)

MaxMttDepthC = ph_max_mtt_hierarchy_depth_intra_slice_chroma (122)

CuQpDeltaSubdiv = ph_cu_qp_delta_subdiv_intra_slice (123)

CuChromaQpOffsetSubdiv = ph_cu_chroma_qp_offset_subdiv_intra_slice (124)

– Otherwise (sh_slice_type equal to 0 (B) or 1 (P)), the following applies:

MinQtLog2SizeY = MinCbLog2SizeY + ph_log2_diff_min_qt_min_cb_inter_slice (125)

MinQtLog2SizeC = MinCbLog2SizeY + ph_log2_diff_min_qt_min_cb_inter_slice (126)

MaxBtSizeY = 1 << (MinQtLog2SizeY + ph_log2_diff_max_bt_min_qt_inter_slice) (127)

MaxBtSizeC = 1 << (MinQtLog2SizeC + ph_log2_diff_max_bt_min_qt_inter_slice) (128)

MaxTtSizeY = 1 << (MinQtLog2SizeY + ph_log2_diff_max_tt_min_qt_inter_slice) (129)

MaxTtSizeC = 1 << (MinQtLog2SizeC + ph_log2_diff_max_tt_min_qt_inter_slice) (130)

MaxMttDepthY = ph_max_mtt_hierarchy_depth_inter_slice (131)

MaxMttDepthC = ph_max_mtt_hierarchy_depth_inter_slice (132)

CuQpDeltaSubdiv = ph_cu_qp_delta_subdiv_inter_slice (133)

CuChromaQpOffsetSubdiv = ph_cu_chroma_qp_offset_subdiv_inter_slice (134)

– The following applies:

MinQtSizeY = 1 << MinQtLog2SizeY (135)

MinQtSizeC = 1 << MinQtLog2SizeC (136)

MinBtSizeY = 1 << MinCbLog2SizeY (137)

MinTtSizeY = 1 << MinCbLog2SizeY (138)

sh_alf_enabled_flag equal to 1 specifies that ALF is enabled for the Y, Cb, or Cr colour component of the current slice. sh_alf_enabled_flag equal to 0 specifies that ALF is disabled for all colour components in the current slice. When not present, the value of sh_alf_enabled_flag is inferred to be equal to ph_alf_enabled_flag.

sh_num_alf_aps_ids_luma specifies the number of ALF APSs that the slice refers to. When sh_alf_enabled_flag is equal to 1 and sh_num_alf_aps_ids_luma is not present, the value of sh_num_alf_aps_ids_luma is inferred to be equal to the value of ph_num_alf_aps_ids_luma.

sh_alf_aps_id_luma[i] specifies the aps_adaptation_parameter_set_id of the i-th ALF APS that the luma component of the slice refers to. When sh_alf_enabled_flag is equal to 1 and sh_alf_aps_id_luma[i] is not present, the value of sh_alf_aps_id_luma[i] is inferred to be equal to the value of ph_alf_aps_id_luma[i].

When sh_alf_aps_id_luma[i] is present, the following applies:

– The TemporalId of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to sh_alf_aps_id_luma[i] shall be less than or equal to the TemporalId of the coded slice NAL unit.

– The value of alf_luma_filter_signal_flag of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to sh_alf_aps_id_luma[i] shall be equal to 1.

– When sps_chroma_format_idc is equal to 0, the value of aps_chroma_present_flag of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to sh_alf_aps_id_luma[i] shall be equal to 0.

– When sps_ccalf_enabled_flag is equal to 0, the values of alf_cc_cb_filter_signal_flag and alf_cc_cr_filter_signal_flag of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to sh_alf_aps_id_luma[i] shall be equal to 0.

sh_alf_cb_enabled_flag equal to 1 specifies that ALF is enabled for the Cb colour component of the current slice. sh_alf_cb_enabled_flag equal to 0 specifies that ALF is disabled for the Cb colour component of the current slice. When sh_alf_cb_enabled_flag is not present, it is inferred to be equal to ph_alf_cb_enabled_flag.

sh_alf_cr_enabled_flag equal to 1 specifies that ALF is enabled for the Cr colour component of the current slice. sh_alf_cr_enabled_flag equal to 0 specifies that ALF is disabled for the Cr colour component of the current slice. When sh_alf_cr_enabled_flag is not present, it is inferred to be equal to ph_alf_cr_enabled_flag.

sh_alf_aps_id_chroma specifies the aps_adaptation_parameter_set_id of the ALF APS that the chroma component of the slice refers to. When sh_alf_enabled_flag is equal to 1 and sh_alf_aps_id_chroma is not present, the value of sh_alf_aps_id_chroma is inferred to be equal to the value of ph_alf_aps_id_chroma.

When sh_alf_aps_id_chroma is present, the following applies:

– The TemporalId of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to sh_alf_aps_id_chroma shall be less than or equal to the TemporalId of the coded slice NAL unit.

– The value of alf_chroma_filter_signal_flag of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to sh_alf_aps_id_chroma shall be equal to 1.

– When sps_ccalf_enabled_flag is equal to 0, the values of alf_cc_cb_filter_signal_flag and alf_cc_cr_filter_signal_flag of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to sh_alf_aps_id_chroma shall be equal to 0.

sh_alf_cc_cb_enabled_flag equal to 1 specifies that CCALF is enabled for the Cb colour component. sh_alf_cc_cb_enabled_flag equal to 0 specifies that CCALF is disabled for the Cb colour component. When sh_alf_cc_cb_enabled_flag is not present, it is inferred to be equal to ph_alf_cc_cb_enabled_flag.

sh_alf_cc_cb_aps_id specifies the aps_adaptation_parameter_set_id that the Cb colour component of the slice refers to.

The TemporalId of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to sh_alf_cc_cb_aps_id shall be less than or equal to the TemporalId of the coded slice NAL unit. When sh_alf_cc_cb_enabled_flag is equal to 1 and sh_alf_cc_cb_aps_id is not present, the value of sh_alf_cc_cb_aps_id is inferred to be equal to the value of ph_alf_cc_cb_aps_id.

The value of alf_cc_cb_filter_signal_flag of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to sh_alf_cc_cb_aps_id shall be equal to 1.

sh_alf_cc_cr_enabled_flag equal to 1 specifies that CCALF is enabled for the Cr colour component of the current slice. sh_alf_cc_cr_enabled_flag equal to 0 specifies that CCALF is disabled for the Cr colour component. When sh_alf_cc_cr_enabled_flag is not present, it is inferred to be equal to ph_alf_cc_cr_enabled_flag.

sh_alf_cc_cr_aps_id specifies the aps_adaptation_parameter_set_id that the Cr colour component of the slice refers to. The TemporalId of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to sh_alf_cc_cr_aps_id shall be less than or equal to the TemporalId of the coded slice NAL unit. When sh_alf_cc_cr_enabled_flag is equal to 1 and sh_alf_cc_cr_aps_id is not present, the value of sh_alf_cc_cr_aps_id is inferred to be equal to the value of ph_alf_cc_cr_aps_id.

The value of alf_cc_cr_filter_signal_flag of the APS NAL unit having aps_params_type equal to ALF_APS and aps_adaptation_parameter_set_id equal to sh_alf_cc_cr_aps_id shall be equal to 1.

sh_lmcs_used_flag equal to 1 specifies that luma mapping is used for the current slice and chroma scaling could be used for the current slice (depending on the value of ph_chroma_residual_scale_flag). sh_lmcs_used_flag equal to 0 specifies that luma mapping with chroma scaling is not used for the current slice. When sh_lmcs_used_flag is not present, it is inferred to be equal to sh_picture_header_in_slice_header_flag ? ph_lmcs_enabled_flag : 0.

sh_explicit_scaling_list_used_flag equal to 1 specifies that the explicit scaling list is used in the scaling process for transform coefficients when decoding the current slice. sh_explicit_scaling_list_used_flag equal to 0 specifies that the explicit scaling list is not used in the scaling process for transform coefficients when decoding the current slice. When not present, the value of sh_explicit_scaling_list_used_flag is inferred to be equal to sh_picture_header_in_slice_header_flag ? ph_explicit_scaling_list_enabled_flag : 0.

sh_num_ref_idx_active_override_flag equal to 1 specifies that the syntax element sh_num_ref_idx_active_minus1[0] is present for P and B slices when num_ref_entries[0][RplsIdx[0]] is greater than 1 and the syntax element sh_num_ref_idx_active_minus1[1] is present for B slices when num_ref_entries[1][RplsIdx[1]] is greater than 1. sh_num_ref_idx_active_override_flag equal to 0 specifies that the syntax elements sh_num_ref_idx_active_minus1[0] and sh_num_ref_idx_active_minus1[1] are not present. When not present, the value of sh_num_ref_idx_active_override_flag is inferred to be equal to 1.

sh_num_ref_idx_active_minus1[i] is used for the derivation of the variable NumRefIdxActive[i] as specified by Equation 139. The value of sh_num_ref_idx_active_minus1[i] shall be in the range of 0 to 14, inclusive.

For i equal to 0 or 1, when the current slice is a B slice, sh_num_ref_idx_active_override_flag is equal to 1, and sh_num_ref_idx_active_minus1[i] is not present, sh_num_ref_idx_active_minus1[i] is inferred to be equal to 0.

When the current slice is a P slice, sh_num_ref_idx_active_override_flag is equal to 1, and sh_num_ref_idx_active_minus1[0] is not present, sh_num_ref_idx_active_minus1[0] is inferred to be equal to 0.

The variable NumRefIdxActive[i] is derived as follows:

```
for( i = 0; i < 2; i++ ) {
    if( sh_slice_type = = B || ( sh_slice_type = = P && i = = 0 ) ) {
        if( sh_num_ref_idx_active_override_flag )
            NumRefIdxActive[ i ] = sh_num_ref_idx_active_minus1[ i ] + 1                    (139)
        else {
            if( num_ref_entries[ i ][ RplsIdx[ i ] ] >= pps_num_ref_idx_default_active_minus1[ i ] + 1 )
                NumRefIdxActive[ i ] = pps_num_ref_idx_default_active_minus1[ i ] + 1
            else
                NumRefIdxActive[ i ] = num_ref_entries[ i ][ RplsIdx[ i ] ]
        }
    } else /* sh_slice_type = = I || ( sh_slice_type = = P && i = = 1 ) */
        NumRefIdxActive[ i ] = 0
}
```

The value of NumRefIdxActive[i] − 1 specifies the maximum reference index for RPL i that may be used to decode the slice. When the value of NumRefIdxActive[i] is equal to 0, no reference index for RPL i is used to decode the slice.

When the current slice is a P slice, the value of NumRefIdxActive[0] shall be greater than 0.

When the current slice is a B slice, both NumRefIdxActive[0] and NumRefIdxActive[1] shall be greater than 0.

sh_cabac_init_flag specifies the method for determining the initialization table used in the initialization process for context variables. When sh_cabac_init_flag is not present, it is inferred to be equal to 0.

sh_collocated_from_l0_flag equal to 1 specifies that the collocated picture used for temporal motion vector prediction is derived from RPL 0. sh_collocated_from_l0_flag equal to 0 specifies that the collocated picture used for temporal motion vector prediction is derived from RPL 1.

When sh_slice_type is equal to B or P, ph_temporal_mvp_enabled_flag is equal to 1, and sh_collocated_from_l0_flag is not present, the following applies:

– If sh_slice_type is equal to B, sh_collocated_from_l0_flag is inferred to be equal to ph_collocated_from_l0_flag.

– Otherwise (sh_slice_type is equal to P), the value of sh_collocated_from_l0_flag is inferred to be equal to 1.

sh_collocated_ref_idx specifies the reference index of the collocated picture used for temporal motion vector prediction.

When sh_slice_type is equal to P or when sh_slice_type is equal to B and sh_collocated_from_l0_flag is equal to 1, sh_collocated_ref_idx refers to an entry in RPL 0, and the value of sh_collocated_ref_idx shall be in the range of 0 to NumRefIdxActive[0] − 1, inclusive.

When sh_slice_type is equal to B and sh_collocated_from_l0_flag is equal to 0, sh_collocated_ref_idx refers to an entry in RPL 1, and the value of sh_collocated_ref_idx shall be in the range of 0 to NumRefIdxActive[1] − 1, inclusive.

When sh_collocated_ref_idx is not present, the following applies:

– If pps_rpl_info_in_ph_flag is equal to 1, the value of sh_collocated_ref_idx is inferred to be equal to ph_collocated_ref_idx.

– Otherwise (pps_rpl_info_in_ph_flag is equal to 0), the value of sh_collocated_ref_idx is inferred to be equal to 0.

Let colPicList be set equal to sh_collocated_from_l0_flag ? 0 : 1. It is a requirement of bitstream conformance that the picture referred to by sh_collocated_ref_idx shall be the same for all non-I slices of a coded picture, the value of RprConstraintsActiveFlag[colPicList][sh_collocated_ref_idx] shall be equal to 0, and the value of sps_log2_ctu_size_minus5 for the picture referred to by sh_collocated_ref_idx shall be equal to the value of sps_log2_ctu_size_minus5 for the current picture.

NOTE 2 – The collocated picture has the same spatial resolution, the same scaling window offsets, the same number of subpictures, and the same CTU size as the current picture.

sh_qp_delta specifies the initial value of Qp_Y to be used for the coding blocks in the slice until modified by the value of CuQpDeltaVal in the coding unit layer.

When pps_qp_delta_info_in_ph_flag is equal to 0, the initial value of the Qp_Y quantization parameter for the slice, $SliceQp_Y$, is derived as follows:

$$SliceQp_Y = 26 + pps\_init\_qp\_minus26 + sh\_qp\_delta \quad (140)$$

The value of $SliceQp_Y$ shall be in the range of −QpBdOffset to +63, inclusive.

When either of the following conditions is true, the value of NumRefIdxActive[0] shall be less than or equal to the value of NumWeightsL0:

– The value of pps_wp_info_in_ph_flag is equal to 1, pps_weighted_pred_flag is equal to 1, and sh_slice_type is equal to P.

– The value of pps_wp_info_in_ph_flag is equal to 1, pps_weighted_bipred_flag is equal to 1, and sh_slice_type is equal to B.

When pps_wp_info_in_ph_flag is equal to 1, pps_weighted_bipred_flag is equal to 1, and sh_slice_type is equal to B, the value of NumRefIdxActive[1] shall be less than or equal to the value of NumWeightsL1.

When either of the following conditions is true, for each value of i in the range of 0 to NumRefIdxActive[0] − 1, inclusive, the values of luma_weight_l0_flag[i] and chroma_weight_l0_flag[i] are both inferred to be equal to 0:

– The value of pps_wp_info_in_ph_flag is equal to 1, pps_weighted_pred_flag is equal to 0, and sh_slice_type is equal to P.

– The value of pps_wp_info_in_ph_flag is equal to 1, pps_weighted_bipred_flag is equal to 0, and sh_slice_type is equal to B.

sh_cb_qp_offset specifies a difference to be added to the value of pps_cb_qp_offset when determining the value of the Qp'_{Cb} quantization parameter. The value of sh_cb_qp_offset shall be in the range of −12 to +12, inclusive. When sh_cb_qp_offset is not present, it is inferred to be equal to 0. The value of pps_cb_qp_offset + sh_cb_qp_offset shall be in the range of −12 to +12, inclusive.

sh_cr_qp_offset specifies a difference to be added to the value of pps_cr_qp_offset when determining the value of the Qp'_{Cr} quantization parameter. The value of sh_cr_qp_offset shall be in the range of −12 to +12, inclusive. When sh_cr_qp_offset is not present, it is inferred to be equal to 0. The value of pps_cr_qp_offset + sh_cr_qp_offset shall be in the range of −12 to +12, inclusive.

sh_joint_cbcr_qp_offset specifies a difference to be added to the value of pps_joint_cbcr_qp_offset_value when determining the value of the Qp'_{CbCr}. The value of sh_joint_cbcr_qp_offset shall be in the range of −12 to +12, inclusive. When sh_joint_cbcr_qp_offset is not present, it is inferred to be equal to 0. The value of pps_joint_cbcr_qp_offset_value + sh_joint_cbcr_qp_offset shall be in the range of −12 to +12, inclusive.

sh_cu_chroma_qp_offset_enabled_flag equal to 1 specifies that the cu_chroma_qp_offset_flag could be present in the transform unit and palette coding syntax of the current slice. sh_cu_chroma_qp_offset_enabled_flag equal to 0 specifies that the cu_chroma_qp_offset_flag is not present in the transform unit or palette coding syntax of the current slice. When not present, the value of sh_cu_chroma_qp_offset_enabled_flag is inferred to be equal to 0.

sh_sao_luma_used_flag equal to 1 specifies that SAO is used for the luma component in the current slice. sh_sao_luma_used_flag equal to 0 specifies that SAO is not used for the luma component in the current slice. When sh_sao_luma_used_flag is not present, it is inferred to be equal to ph_sao_luma_enabled_flag.

sh_sao_chroma_used_flag equal to 1 specifies that SAO is used for the chroma component in the current slice. sh_sao_chroma_used_flag equal to 0 specifies that SAO is not used for the chroma component in the current slice. When sh_sao_chroma_used_flag is not present, it is inferred to be equal to ph_sao_chroma_enabled_flag.

sh_deblocking_params_present_flag equal to 1 specifies that the deblocking parameters could be present in the slice header. sh_deblocking_params_present_flag equal to 0 specifies that the deblocking parameters are not present in the slice header. When not present, the value of sh_deblocking_params_present_flag is inferred to be equal to 0.

sh_deblocking_filter_disabled_flag equal to 1 specifies that the deblocking filter is disabled for the current slice. sh_deblocking_filter_disabled_flag equal to 0 specifies that the deblocking filter is enabled for the current slice.

When sh_deblocking_filter_disabled_flag is not present, it is inferred as follows:

– If pps_deblocking_filter_disabled_flag and sh_deblocking_params_present_flag are both equal to 1, the value of sh_deblocking_filter_disabled_flag is inferred to be equal to 0.

– Otherwise (pps_deblocking_filter_disabled_flag or sh_deblocking_params_present_flag is equal to 0), the value of sh_deblocking_filter_disabled_flag is inferred to be equal to ph_deblocking_filter_disabled_flag.

sh_luma_beta_offset_div2 and **sh_luma_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the luma component for the current slice. The values of sh_luma_beta_offset_div2 and sh_luma_tc_offset_div2 shall both be in the range of −12 to 12, inclusive. When not present, the values of sh_luma_beta_offset_div2 and sh_luma_tc_offset_div2 are inferred to be equal to ph_luma_beta_offset_div2 and ph_luma_tc_offset_div2, respectively.

sh_cb_beta_offset_div2 and **sh_cb_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for the current slice. The values of sh_cb_beta_offset_div2 and sh_cb_tc_offset_div2 shall both be in the range of −12 to 12, inclusive.

When not present, the values of sh_cb_beta_offset_div2 and sh_cb_tc_offset_div2 are inferred as follows:

– If pps_chroma_tool_offsets_present_flag is equal to 1, the values of sh_cb_beta_offset_div2 and sh_cb_tc_offset_div2 are inferred to be equal to ph_cb_beta_offset_div2 and ph_cb_tc_offset_div2, respectively.

– Otherwise (pps_chroma_tool_offsets_present_flag is equal to 0), the values of sh_cb_beta_offset_div2 and sh_cb_tc_offset_div2 are inferred to be equal to sh_luma_beta_offset_div2 and sh_luma_tc_offset_div2, respectively.

sh_cr_beta_offset_div2 and **sh_cr_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for the current slice. The values of sh_cr_beta_offset_div2 and sh_cr_tc_offset_div2 shall both be in the range of −12 to 12, inclusive.

When not present, the values of sh_cr_beta_offset_div2 and sh_cr_tc_offset_div2 are inferred as follows:

– If pps_chroma_tool_offsets_present_flag is equal to 1, the values of sh_cr_beta_offset_div2 and sh_cr_tc_offset_div2 are inferred to be equal to ph_cr_beta_offset_div2 and ph_cr_tc_offset_div2, respectively.

– Otherwise (pps_chroma_tool_offsets_present_flag is equal to 0), the values of sh_cr_beta_offset_div2 and sh_cr_tc_offset_div2 are inferred to be equal to sh_luma_beta_offset_div2 and sh_luma_tc_offset_div2, respectively.

sh_dep_quant_used_flag equal to 0 specifies that dependent quantization is not used for the current slice. sh_dep_quant_used_flag equal to 1 specifies that dependent quantization is used for the current slice. When sh_dep_quant_used_flag is not present, it is inferred to be equal to 0.

sh_sign_data_hiding_used_flag equal to 0 specifies that sign bit hiding is not used for the current slice. sh_sign_data_hiding_used_flag equal to 1 specifies that sign bit hiding is used for the current slice. When sh_sign_data_hiding_used_flag is not present, it is inferred to be equal to 0.

sh_ts_residual_coding_disabled_flag equal to 1 specifies that the residual_coding( ) syntax structure is used to parse the residual samples of a transform skip block for the current slice. sh_ts_residual_coding_disabled_flag equal to 0 specifies that the residual_ts_coding( ) syntax structure is used to parse the residual samples of a transform skip block for the current slice. When sh_ts_residual_coding_disabled_flag is not present, it is inferred to be equal to 0.

sh_ts_residual_coding_rice_idx_minus1 plus 1 specifies the Rice parameter used for the residual_ts_coding( ) syntax structure in the current slice. When not present, the value of sh_ts_residual_coding_rice_idx_minus1 is inferred to be equal to 0.

sh_reverse_last_sig_coeff_flag equal to 1 specifies that the coordinates of the last significant coefficient are coded relative to ((Log2ZoTbWidth << 1) − 1, (Log2ZoTbHeight << 1) − 1) for each transform block of the current slice. sh_reverse_last_sig_coeff_flag equal to 0 specifies that the coordinates of the last significant coefficient are coded relative to (0, 0) for each transform block of the current slice. When not present, the value of sh_reverse_last_sig_coeff_flag is inferred to be equal to 0.

sh_slice_header_extension_length specifies the length of the slice header extension data in bytes, not including the bits used for signalling sh_slice_header_extension_length itself. When not present, the value of sh_slice_header_extension_length is inferred to be equal to 0. Although sh_slice_header_extension_length is not present in bitstreams conforming to this version of this Specification, some use of sh_slice_header_extension_length could be specified in some future version of this Specification, and decoders conforming to this version of this Specification shall also allow sh_slice_header_extension_length to be present and in the range of 0 to 256, inclusive.

sh_slice_header_extension_data_byte[i] could have any value. Its presence and value do not affect the decoding process specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore the values of the sh_slice_header_extension_data_byte[i] syntax elements. Its value does not affect the decoding process specified in this version of specification.

The variable NumEntryPoints, which specifies the number of entry points in the current slice, is derived as follows:

```
NumEntryPoints = 0
if( sps_entry_point_offsets_present_flag )
    for( i = 1; i < NumCtusInCurrSlice; i++ ) {
        ctbAddrX = CtbAddrInCurrSlice[ i ] % PicWidthInCtbsY
        ctbAddrY = CtbAddrInCurrSlice[ i ] / PicWidthInCtbsY                    (141)
        prevCtbAddrX = CtbAddrInCurrSlice[ i − 1 ] % PicWidthInCtbsY
        prevCtbAddrY = CtbAddrInCurrSlice[ i − 1 ] / PicWidthInCtbsY
        if( CtbToTileRowBd[ ctbAddrY ] != CtbToTileRowBd[ prevCtbAddrY ] | |
            CtbToTileColBd[ ctbAddrX ] != CtbToTileColBd[ prevCtbAddrX ] | |
           ( ctbAddrY != prevCtbAddrY && sps_entropy_coding_sync_enabled_flag ) )
           NumEntryPoints++
    }
```

sh_entry_offset_len_minus1 plus 1 specifies the length, in bits, of the sh_entry_point_offset_minus1[i] syntax elements. The value of sh_entry_offset_len_minus1 shall be in the range of 0 to 31, inclusive.

sh_entry_point_offset_minus1[i] plus 1 specifies the i-th entry point offset in bytes, and is represented by sh_entry_offset_len_minus1 plus 1 bits. The slice data that follow the slice header consists of NumEntryPoints + 1 subsets, with subset index values ranging from 0 to NumEntryPoints, inclusive. The first byte of the slice data is considered byte 0. When present, emulation prevention bytes that appear in the slice data portion of the coded slice NAL unit are counted as part of the slice data for purposes of subset identification. Subset 0 consists of bytes 0 to sh_entry_point_offset_minus1[0], inclusive, of the coded slice data, subset k, with k in the range of 1 to NumEntryPoints − 1, inclusive, consists of bytes firstByte[k] to lastByte[k], inclusive, of the coded slice data with firstByte[k] and lastByte[k] derived as follows:

$$\text{firstByte}[\,k\,] = \sum_{n=1}^{k} (\,\text{sh\_entry\_point\_offset\_minus1}[\,n-1\,] + 1\,) \tag{142}$$

$$\text{lastByte}[\,k\,] = \text{firstByte}[\,k\,] + \text{sh\_entry\_point\_offset\_minus1}[\,k\,] \tag{143}$$

The last subset (with subset index equal to NumEntryPoints) consists of the remaining bytes of the coded slice data.

When sps_entropy_coding_sync_enabled_flag is equal to 0 and the slice contains one or more complete tiles, each subset shall consist of all coded bits of all CTUs in the slice that are within the same tile, and the number of subsets (i.e., the value of NumEntryPoints + 1) shall be equal to the number of tiles in the slice.

When sps_entropy_coding_sync_enabled_flag is equal to 0 and the slice contains a subset of CTU rows from a single tile, the value of NumEntryPoints shall be equal to 0. The subset shall consist of all coded bits of all CTUs in the slice.

When sps_entropy_coding_sync_enabled_flag is equal to 1, each subset k with k in the range of 0 to NumEntryPoints, inclusive, shall consist of all coded bits of all CTUs in a CTU row within a tile, and the number of subsets (i.e., the value of NumEntryPoints + 1) shall be equal to the total number of tile-specific CTU rows in the slice.

7.4.9 Weighted prediction parameters 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.

delta_chroma_log2_weight_denom is the difference of the base 2 logarithm of the denominator for all chroma weighting factors. When delta_chroma_log2_weight_denom is not present, it is inferred to be equal to 0.

The variable ChromaLog2WeightDenom is derived to be equal to luma_log2_weight_denom + delta_chroma_log2_weight_denom and the value shall be in the range of 0 to 7, inclusive.

num_l0_weights specifies the number of weights signalled for entries in RPL 0 when pps_wp_info_in_ph_flag is equal to 1. The value of num_l0_weights shall be in the range of 1 to Min(15, num_ref_entries[0][RplsIdx[0]]), inclusive.

If pps_wp_info_in_ph_flag is equal to 1, the variable NumWeightsL0 is set equal to num_l0_weights. Otherwise (pps_wp_info_in_ph_flag is equal to 0), NumWeightsL0 is set equal to NumRefIdxActive[0].

luma_weight_l0_flag[i] equal to 1 specifies that weighting factors for the luma component of list 0 prediction using RefPicList[0][i] are present. luma_weight_l0_flag[i] equal to 0 specifies that these weighting factors are not present.

chroma_weight_l0_flag[i] equal to 1 specifies that weighting factors for the chroma prediction values of list 0 prediction using RefPicList[0][i] are present. chroma_weight_l0_flag[i] equal to 0 specifies that these weighting factors are not present. When chroma_weight_l0_flag[i] is not present, it is inferred to be equal to 0.

delta_luma_weight_l0[i] is the difference of the weighting factor applied to the luma prediction value for list 0 prediction using RefPicList[0][i].

The variable LumaWeightL0[i] is derived to be equal to (1 << luma_log2_weight_denom) + delta_luma_weight_l0[i]. When luma_weight_l0_flag[i] is equal to 1, the value of delta_luma_weight_l0[i] shall be in the range of −128 to 127, inclusive. When luma_weight_l0_flag[i] is equal to 0, LumaWeightL0[i] is inferred to be equal to $2^{\text{luma\_log2\_weight\_denom}}$.

luma_offset_l0[i] is the additive offset applied to the luma prediction value for list 0 prediction using RefPicList[0][i]. The value of luma_offset_l0[i] shall be in the range of −128 to 127, inclusive. When luma_weight_l0_flag[i] is equal to 0, luma_offset_l0[i] is inferred to be equal to 0.

delta_chroma_weight_l0[i][j] is the difference of the weighting factor applied to the chroma prediction values for list 0 prediction using RefPicList[0][i] with j equal to 0 for Cb and j equal to 1 for Cr.

The variable ChromaWeightL0[i][j] is derived to be equal to (1 << ChromaLog2WeightDenom) + delta_chroma_weight_l0[i][j]. When chroma_weight_l0_flag[i] is equal to 1, the value of delta_chroma_weight_l0[i][j] shall be in the range of −128 to 127, inclusive. When chroma_weight_l0_flag[i] is equal to 0**,** ChromaWeightL0[i][j] is inferred to be equal to $2^{\text{ChromaLog2WeightDenom}}$.

delta_chroma_offset_l0[i][j] is the difference of the additive offset applied to the chroma prediction values for list 0 prediction using RefPicList[0][i] with j equal to 0 for Cb and j equal to 1 for Cr.

The variable ChromaOffsetL0[i][j] is derived as follows:

```
ChromaOffsetL0[ i ][ j ] = Clip3( −128, 127,
    ( 128 + delta_chroma_offset_l0[ i ][ j ] −
    ( ( 128 * ChromaWeightL0[ i ][ j ] ) >> ChromaLog2WeightDenom ) ) )          (144)
```

The value of delta_chroma_offset_l0[i][j] shall be in the range of −4 * 128 to 4 * 127, inclusive. When chroma_weight_l0_flag[i] is equal to 0**,** ChromaOffsetL0[i][j] is inferred to be equal to 0.

num_l1_weights specifies the number of weights signalled for entries in RPL 1 when pps_weighted_bipred_flag and pps_wp_info_in_ph_flag are both equal to 1. The value of num_l1_weights shall be in the range of 1 to Min(15, num_ref_entries[1][RplsIdx[1]]), inclusive.

The variable NumWeightsL1 is derived as follows:

```
if( !pps_weighted_bipred_flag | |
        ( pps_wp_info_in_ph_flag && num_ref_entries[ 1 ][ RplsIdx[ 1 ] ] = = 0 ) )
    NumWeightsL1 = 0
else if( pps_wp_info_in_ph_flag )                                            (145)
    NumWeightsL1 = num_l1_weights
else
    NumWeightsL1 = NumRefIdxActive[ 1 ]
```

luma_weight_l1_flag[i] equal to 1 specifies that weighting factors for the luma component of list 1 prediction using RefPicList[1][i] are present. luma_weight_l1_flag[i] equal to 0 specifies that these weighting factors are not present. When not present, the value of luma_weight_l1_flag[i] is inferred to be equal to 0.

chroma_weight_l1_flag[i], **delta_luma_weight_l1**[i], **luma_offset_l1**[i], **delta_chroma_weight_l1**[i][j], and **delta_chroma_offset_l1**[i][j] have the same semantics as chroma_weight_l0_flag[i], delta_luma_weight_l0[i], luma_offset_l0[i], delta_chroma_weight_l0[i][j] and delta_chroma_offset_l0[i][j], respectively, with l0, L0, list 0 and List0 replaced by l1, L1, list 1 and List1, respectively.

The variable sumWeightL0Flags is derived to be equal to the sum of luma_weight_l0_flag[i] + 2 * chroma_weight_l0_flag[i], for i = 0..NumRefIdxActive[0] − 1.

When sh_slice_type is equal to B, the variable sumWeightL1Flags is derived to be equal to the sum of luma_weight_l1_flag[i] + 2 * chroma_weight_l1_flag[i], for i = 0..NumRefIdxActive[1] − 1.

It is a requirement of bitstream conformance that, when sh_slice_type is equal to P, sumWeightL0Flags shall be less than or equal to 24 and when sh_slice_type is equal to B, the sum of sumWeightL0Flags and sumWeightL1Flags shall be less than or equal to 24.

7.4.10 Reference picture lists semantics

The ref_pic_lists( ) syntax structure could be present in the PH syntax structure or the slice header.

rpl_sps_flag[i] equal to 1 specifies that RPL i in ref_pic_lists( ) is derived based on one of the ref_pic_list_struct(listIdx, rplsIdx) syntax structures with listIdx equal to i in the SPS. rpl_sps_flag[i] equal to 0 specifies that RPL i of the picture is derived based on the ref_pic_list_struct(listIdx, rplsIdx) syntax structure with listIdx equal to i that is directly included in ref_pic_lists( ).

When rpl_sps_flag[i] is not present, it is inferred as follows:

– If sps_num_ref_pic_lists[i] is equal to 0, the value of rpl_sps_flag[i] is inferred to be equal to 0.
– Otherwise (sps_num_ref_pic_lists[i] is greater than 0), when pps_rpl1_idx_present_flag is equal to 0 and i is equal to 1, the value of rpl_sps_flag[1] is inferred to be equal to rpl_sps_flag[0].

rpl_idx[i] specifies the index, into the list of the ref_pic_list_struct(listIdx, rplsIdx) syntax structures with listIdx equal to i included in the SPS, of the ref_pic_list_struct(listIdx, rplsIdx) syntax structure with listIdx equal to i that is used for derivation of RPL i of the current picture or slice. The syntax element rpl_idx[i] is represented by Ceil(Log2(sps_num_ref_pic_lists[i])) bits.

When rpl_sps_flag[i] is equal to 1 and sps_num_ref_pic_lists[i] is equal to 1, the value of rpl_idx[i] is inferred to be equal to 0. When rpl_sps_flag[1] is equal to 1, pps_rpl1_idx_present_flag is equal to 0, and sps_num_ref_pic_lists[1] is greater than 1, the value of rpl_idx[1] is inferred to be equal to rpl_idx[0].

The value of rpl_idx[i] shall be in the range of 0 to sps_num_ref_pic_lists[i] − 1, inclusive.

The variable RplsIdx[i] is derived as follows:

```
RplsIdx[ i ] = rpl_sps_flag[ i ] ? rpl_idx[ i ] : sps_num_ref_pic_lists[ i ]          (146)
```

When pps_rpl_info_in_ph_flag is equal to 1 and ph_inter_slice_allowed_flag is equal to 1, the value of num_ref_entries[0][RplsIdx[0]] shall be greater than 0.

poc_lsb_lt[i][j] specifies the value of the picture order count modulo MaxPicOrderCntLsb of the j-th LTRP entry in the i-th RPL in the ref_pic_lists( ) syntax structure. The length of the poc_lsb_lt[i][j] syntax element is sps_log2_max_pic_order_cnt_lsb_minus4 + 4 bits.

The variable PocLsbLt[i][j] is derived as follows:

```
PocLsbLt[ i ][ j ] = ltrp_in_header_flag[ i ][ RplsIdx[ i ] ] ?                        (147)
              poc_lsb_lt[ i ][ j ] : rpls_poc_lsb_lt[ i ][ RplsIdx[ i ] ][ j ]
```

delta_poc_msb_cycle_present_flag[i][j] equal to 1 specifies that delta_poc_msb_cycle_lt[i][j] is present. delta_poc_msb_cycle_present_flag[i][j] equal to 0 specifies that delta_poc_msb_cycle_lt[i][j] is not present.

Let prevTid0Pic be the previous picture in decoding order that has nuh_layer_id the same as the slice or picture header referring to the ref_pic_lists( ) syntax structure, has TemporalId and ph_non_ref_pic_flag both equal to 0, and is not a RASL or RADL picture. Let setOfPrevPocVals be a set consisting of the following:

– the PicOrderCntVal of prevTid0Pic,
– the PicOrderCntVal of each picture that is referred to by entries in RefPicList[0] or RefPicList[1] of prevTid0Pic and has nuh_layer_id the same as the current picture,
– the PicOrderCntVal of each picture that follows prevTid0Pic in decoding order, has nuh_layer_id the same as the current picture, and precedes the current picture in decoding order.

When there is more than one value in setOfPrevPocVals for which the value modulo MaxPicOrderCntLsb is equal to PocLsbLt[i][j], the value of delta_poc_msb_cycle_present_flag[i][j] shall be equal to 1.

delta_poc_msb_cycle_lt[i][j] specifies the value of the variable FullPocLt[i][j] as follows:

```
if( j = = 0 )
    deltaPocMsbCycleLt[ i ][ j ] = delta_poc_msb_cycle_lt[ i ][ j ]
else                                                                                    (148)
    deltaPocMsbCycleLt[ i ][ j ] = delta_poc_msb_cycle_lt[ i ][ j ] + deltaPocMsbCycleLt[ i ][ j − 1 ]
FullPocLt[ i ][ j ] = PicOrderCntVal – deltaPocMsbCycleLt[ i ][ j ] * MaxPicOrderCntLsb −
        ( PicOrderCntVal & ( MaxPicOrderCntLsb − 1 ) ) + PocLsbLt[ i ][ j ]
```

The value of delta_poc_msb_cycle_lt[i][j] shall be in the range of 0 to $2^{(32 - \text{sps\_log2\_max\_pic\_order\_cnt\_lsb\_minus4} - 4)}$, inclusive. When not present, the value of delta_poc_msb_cycle_lt[i][j] is inferred to be equal to 0.

7.4.11 Reference picture list structure semantics

The ref_pic_list_struct(listIdx, rplsIdx) syntax structure could be present in an SPS, in a PH syntax structure, or in a slice header. Depending on whether the syntax structure is included in an SPS, a PH syntax structure, or a slice header, the following applies:

– If present in a PH syntax structure or slice header, the ref_pic_list_struct(listIdx, rplsIdx) syntax structure specifies RPL listIdx of the current picture (i.e., the coded picture containing the PH syntax structure or slice header).

– Otherwise (present in an SPS), the ref_pic_list_struct(listIdx, rplsIdx) syntax structure specifies a candidate for RPL listIdx, and the term "the current picture" in the semantics specified in the remainder of this clause refers to each picture that 1) has a PH syntax structure or one or more slices containing rpl_idx[listIdx] equal to an index into the list of the ref_pic_list_struct(listIdx, rplsIdx) syntax structures included in the SPS, and 2) is in a CLVS that refers to the SPS.

num_ref_entries[listIdx][rplsIdx] specifies the number of entries in the ref_pic_list_struct(listIdx, rplsIdx) syntax structure. The value of num_ref_entries[listIdx][rplsIdx] shall be in the range of 0 to MaxDpbSize + 13, inclusive, where MaxDpbSize is as specified in clause A.4.2.

ltrp_in_header_flag[listIdx][rplsIdx] equal to 0 specifies that the POC LSBs of the LTRP entries indicated in the ref_pic_list_struct(listIdx, rplsIdx) syntax structure are present in the same syntax structure. ltrp_in_header_flag[listIdx][rplsIdx] equal to 1 specifies that the POC LSBs of the LTRP entries indicated in the ref_pic_list_struct(listIdx, rplsIdx) syntax structure are not present in the same syntax structure. When sps_long_term_ref_pics_flag is equal to 1 and rplsIdx is equal to sps_num_ref_pic_lists[listIdx], the value of ltrp_in_header_flag[listIdx][rplsIdx] is inferred to be equal to 1.

inter_layer_ref_pic_flag[listIdx][rplsIdx][i] equal to 1 specifies that the i-th entry in the ref_pic_list_struct(listIdx, rplsIdx) syntax structure is an ILRP entry. inter_layer_ref_pic_flag[listIdx][rplsIdx][i] equal to 0 specifies that the i-th entry in the ref_pic_list_struct(listIdx, rplsIdx) syntax structure is not an ILRP entry. When not present, the value of inter_layer_ref_pic_flag[listIdx][rplsIdx][i] is inferred to be equal to 0.

st_ref_pic_flag[listIdx][rplsIdx][i] equal to 1 specifies that the i-th entry in the ref_pic_list_struct(listIdx, rplsIdx) syntax structure is an STRP entry. st_ref_pic_flag[listIdx][rplsIdx][i] equal to 0 specifies that the i-th entry in the ref_pic_list_struct(listIdx, rplsIdx) syntax structure is an LTRP entry. When inter_layer_ref_pic_flag[listIdx][rplsIdx][i] is equal to 0 and st_ref_pic_flag[listIdx][rplsIdx][i] is not present, the value of st_ref_pic_flag[listIdx][rplsIdx][i] is inferred to be equal to 1.

The variable NumLtrpEntries[listIdx][rplsIdx] is derived as follows:

```
for( i = 0, NumLtrpEntries[ listIdx ][ rplsIdx ] = 0; i < num_ref_entries[ listIdx ][ rplsIdx ]; i++ )
    if( !inter_layer_ref_pic_flag[ listIdx ][ rplsIdx ][ i ]  &&  !st_ref_pic_flag[ listIdx ][ rplsIdx ][ i ] )    (149)
        NumLtrpEntries[ listIdx ][ rplsIdx ]++
```

abs_delta_poc_st[listIdx][rplsIdx][i] specifies the value of the variable AbsDeltaPocSt[listIdx][rplsIdx][i] as follows:

```
if( ( sps_weighted_pred_flag  | |  sps_weighted_bipred_flag )  &&  i  !=  0 )
    AbsDeltaPocSt[ listIdx ][ rplsIdx ][ i ] = abs_delta_poc_st[ listIdx ][ rplsIdx ][ i ]    (150)
else
    AbsDeltaPocSt[ listIdx ][ rplsIdx ][ i ] =  abs_delta_poc_st[ listIdx ][ rplsIdx ][ i ] + 1
```

The value of abs_delta_poc_st[listIdx][rplsIdx][i] shall be in the range of 0 to $2^{15} - 1$, inclusive.

strp_entry_sign_flag[listIdx][rplsIdx][i] equal to 0 specifies that DeltaPocValSt[listIdx][rplsIdx] is greater than or equal to 0. strp_entry_sign_flag[listIdx][rplsIdx][i] equal to 1 specifies that DeltaPocValSt[listIdx][rplsIdx] is less than 0. When not present, the value of strp_entry_sign_flag[listIdx][rplsIdx][i] is inferred to be equal to 0.

The list DeltaPocValSt[listIdx][rplsIdx] is derived as follows:

```
for( i = 0; i < num_ref_entries[ listIdx ][ rplsIdx ]; i++ )
    if( !inter_layer_ref_pic_flag[ listIdx ][ rplsIdx ][ i ]  &&  st_ref_pic_flag[ listIdx ][ rplsIdx ][ i ] )    (151)
        DeltaPocValSt[ listIdx ][ rplsIdx ][ i ] = ( 1 − 2 * strp_entry_sign_flag[ listIdx ][ rplsIdx ][ i ] ) *
                AbsDeltaPocSt[ listIdx ][ rplsIdx ][ i ]
```

rpls_poc_lsb_lt[listIdx][rplsIdx][i] specifies the value of the picture order count modulo MaxPicOrderCntLsb of the picture referred to by the i-th entry in the ref_pic_list_struct(listIdx, rplsIdx) syntax structure. The length of the rpls_poc_lsb_lt[listIdx][rplsIdx][i] syntax element is sps_log2_max_pic_order_cnt_lsb_minus4 + 4 bits.

ilrp_idx[listIdx][rplsIdx][i] specifies the index, to the list of the direct reference layers, of the ILRP entry of the i-th entry in the ref_pic_list_struct(listIdx, rplsIdx) syntax structure. The value of ilrp_idx[listIdx][rplsIdx][i] shall be in the range of 0 to NumDirectRefLayers[GeneralLayerIdx[nuh_layer_id]] − 1, inclusive.

7.4.12 Slice data semantics

7.4.12.1 General slice data semantics

end_of_slice_one_bit shall be equal to 1.

end_of_tile_one_bit shall be equal to 1.

end_of_subset_one_bit shall be equal to 1.

7.4.12.2 Coding tree unit semantics

The CTU is the root node of the coding tree structure.

The array IsAvailable[cIdx][x][y] specifying whether the sample at (x, y) is available for use in the derivation process for neighbouring block availability as specified in clause 6.4.4 is initialized as follows for cIdx = 0..2, x = xCtb..xCtb + CtbSizeY − 1, and y = yCtb..yCtb + CtbSizeY − 1:

IsAvailable[cIdx][x][y] = FALSE (152)

alf_ctb_flag[cIdx][xCtb >> CtbLog2SizeY][yCtb >> CtbLog2SizeY] equal to 1 specifies that the adaptive loop filter is applied to the coding tree block of the colour component indicated by cIdx of the coding tree unit at luma location (xCtb, yCtb). alf_ctb_flag[cIdx][xCtb >> CtbLog2SizeY][yCtb >> CtbLog2SizeY] equal to 0 specifies that the adaptive loop filter is not applied to the coding tree block of the colour component indicated by cIdx of the coding tree unit at luma location (xCtb, yCtb).

When alf_ctb_flag[cIdx][xCtb >> CtbLog2SizeY][yCtb >> CtbLog2SizeY] is not present, it is inferred to be equal to 0.

alf_use_aps_flag equal to 0 specifies that one of the fixed filter sets is applied to the luma CTB. alf_use_aps_flag equal to 1 specifies that a filter set from an APS is applied to the luma CTB. When alf_use_aps_flag is not present, it is inferred to be equal to 0.

alf_luma_prev_filter_idx specifies the previous filter that is applied to the luma CTB. The value of alf_luma_prev_filter_idx shall be in a range of 0 to sh_num_alf_aps_ids_luma − 1, inclusive. When alf_luma_prev_filter_idx is not present, it is inferred to be equal to 0.

The variable AlfCtbFiltSetIdxY[xCtb >> CtbLog2SizeY][yCtb >> CtbLog2SizeY] specifying the filter set index for the luma CTB at location (xCtb, yCtb) is derived as follows:

– If alf_use_aps_flag is equal to 0, AlfCtbFiltSetIdxY[xCtb >> CtbLog2SizeY][yCtb >> CtbLog2SizeY] is set equal to alf_luma_fixed_filter_idx.

– Otherwise, AlfCtbFiltSetIdxY[xCtb >> CtbLog2SizeY][yCtb >> CtbLog2SizeY] is set equal to 16 + alf_luma_prev_filter_idx.

alf_luma_fixed_filter_idx specifies the fixed filter that is applied to the luma CTB. The value of alf_luma_fixed_filter_idx shall be in a range of 0 to 15, inclusive.

alf_ctb_filter_alt_idx[chromaIdx][xCtb >> CtbLog2SizeY][yCtb >> CtbLog2SizeY] specifies the index of the alternative chroma filter applied to the coding tree block of the chroma component, with chromaIdx equal to 0 for Cb and chromaIdx equal 1 for Cr, of the coding tree unit at luma location (xCtb, yCtb). When alf_ctb_filter_alt_idx[chromaIdx][xCtb >> CtbLog2SizeY][yCtb >> CtbLog2SizeY] is not present, it is inferred to be equal to zero.

alf_ctb_cc_cb_idc[xCtb >> CtbLog2SizeY][yCtb >> CtbLog2SizeY] equal to 0 specifies that the cross-component filter is not applied to the coding tree block of the Cb colour component at luma location (xCtb, yCtb). When alf_ctb_cc_cb_idc[xCtb >> CtbLog2SizeY][yCtb >> CtbLog2SizeY] is not equal to 0, alf_ctb_cc_cb_idc[xCtb >> CtbLog2SizeY][yCtb >> CtbLog2SizeY] − 1 specifies the filter set index of the cross-component filter applied to the coding tree block of the Cb colour component at luma location (xCtb, yCtb).

When alf_ctb_cc_cb_idc[xCtb >> CtbLog2SizeY][yCtb >> CtbLog2SizeY] is not present, it is inferred to be equal to 0.

alf_ctb_cc_cr_idc[xCtb >> CtbLog2SizeY][yCtb >> CtbLog2SizeY] equal to 0 specifies that the cross-component filter is not applied to the coding tree block of the Cr colour component at luma location (xCtb, yCtb). When alf_ctb_cc_cr_idc[xCtb >> CtbLog2SizeY][yCtb >> CtbLog2SizeY] is not equal to 0, alf_ctb_cc_cr_idc[xCtb >> CtbLog2SizeY][yCtb >> CtbLog2SizeY] − 1 specifies the filter set index of the cross-component filter applied to the coding tree block of the Cr colour component at luma location (xCtb, yCtb).

When alf_ctb_cc_cr_idc[xCtb >> CtbLog2SizeY][yCtb >> CtbLog2SizeY] is not present, it is inferred to be equal to 0.

7.4.12.3 Sample adaptive offset semantics

sao_merge_left_flag equal to 1 specifies that the syntax elements sao_type_idx_luma, sao_type_idx_chroma, sao_band_position, sao_eo_class_luma, sao_eo_class_chroma, sao_offset_abs and sao_offset_sign_flag are derived from the corresponding syntax elements of the left CTB. sao_merge_left_flag equal to 0 specifies that these syntax elements are not derived from the corresponding syntax elements of the left CTB. When sao_merge_left_flag is not present, it is inferred to be equal to 0.

sao_merge_up_flag equal to 1 specifies that the syntax elements sao_type_idx_luma, sao_type_idx_chroma, sao_band_position, sao_eo_class_luma, sao_eo_class_chroma, sao_offset_abs and sao_offset_sign_flag are derived from the corresponding syntax elements of the above CTB. sao_merge_up_flag equal to 0 specifies that these syntax elements are not derived from the corresponding syntax elements of the above CTB. When sao_merge_up_flag is not present, it is inferred to be equal to 0.

sao_type_idx_luma specifies the offset type for the luma component. The array SaoTypeIdx[cIdx][rx][ry] specifies the offset type as specified in Table 10 for the CTB at the location (rx, ry) for the colour component cIdx. The value of SaoTypeIdx[0][rx][ry] is derived as follows:

– If sao_type_idx_luma is present, SaoTypeIdx[0][rx][ry] is set equal to sao_type_idx_luma.

– Otherwise (sao_type_idx_luma is not present), SaoTypeIdx[0][rx][ry] is derived as follows:

  – If sao_merge_left_flag is equal to 1, SaoTypeIdx[0][rx][ry] is set equal to SaoTypeIdx[0][rx − 1][ry].

  – Otherwise, if sao_merge_up_flag is equal to 1, SaoTypeIdx[0][rx][ry] is set equal to SaoTypeIdx[0][rx][ry − 1].

  – Otherwise, SaoTypeIdx[0][rx][ry] is set equal to 0.

sao_type_idx_chroma specifies the offset type for the chroma components. The values of SaoTypeIdx[cIdx][rx][ry] are derived as follows for cIdx equal to 1..2:

– If sao_type_idx_chroma is present, SaoTypeIdx[cIdx][rx][ry] is set equal to sao_type_idx_chroma.

– Otherwise (sao_type_idx_chroma is not present), SaoTypeIdx[cIdx][rx][ry] is derived as follows:

  – If sao_merge_left_flag is equal to 1, SaoTypeIdx[cIdx][rx][ry] is set equal to SaoTypeIdx[cIdx][rx − 1][ry].

  – Otherwise, if sao_merge_up_flag is equal to 1, SaoTypeIdx[cIdx][rx][ry] is set equal to SaoTypeIdx[cIdx][rx][ry − 1].

  – Otherwise, SaoTypeIdx[cIdx][rx][ry] is set equal to 0.

Table 10 – Specification of the SAO type

SaoTypeIdx[cIdx][rx][ry]	SAO type (informative)
0	Not applied
1	Band offset
2	Edge offset

sao_offset_abs[cIdx][rx][ry][i] specifies the offset value of i-th category for the CTB at the location (rx, ry) for the colour component cIdx.

When sao_offset_abs[cIdx][rx][ry][i] is not present, it is inferred as follows:

– If sao_merge_left_flag is equal to 1, sao_offset_abs[cIdx][rx][ry][i] is inferred to be equal to sao_offset_abs[cIdx][rx − 1][ry][i].

– Otherwise, if sao_merge_up_flag is equal to 1, sao_offset_abs[cIdx][rx][ry][i] is inferred to be equal to sao_offset_abs[cIdx][rx][ry − 1][i].

– Otherwise, sao_offset_abs[cIdx][rx][ry][i] is inferred to be equal to 0.

sao_offset_sign_flag[cIdx][rx][ry][i] specifies the sign of the offset value of i-th category for the CTB at the location (rx, ry) for the colour component cIdx.

When sao_offset_sign_flag[cIdx][rx][ry][i] is not present, it is inferred as follows:

– If sao_merge_left_flag is equal to 1, sao_offset_sign_flag[cIdx][rx][ry][i] is inferred to be equal to sao_offset_sign_flag[cIdx][rx − 1][ry][i].

– Otherwise, if sao_merge_up_flag is equal to 1, sao_offset_sign_flag[cIdx][rx][ry][i] is inferred to be equal to sao_offset_sign_flag[cIdx][rx][ry − 1][i].

– Otherwise, if SaoTypeIdx[cIdx][rx][ry] is equal to 2, the following applies:

  – If i is equal to 0 or 1, sao_offset_sign_flag[cIdx][rx][ry][i] is inferred to be equal 0.

  – Otherwise (i is equal to 2 or 3), sao_offset_sign_flag[cIdx][rx][ry][i] is inferred to be equal 1.

– Otherwise, sao_offset_sign_flag[cIdx][rx][ry][i] is inferred to be equal 0.

The list SaoOffsetVal[cIdx][rx][ry][i] for i ranging from 0 to 4, inclusive, is derived as follows:

```
SaoOffsetVal[ cIdx ][ rx ][ ry ][ 0 ] = 0
for( i = 0; i < 4; i++ )
SaoOffsetVal[ cIdx ][ rx ][ ry ][ i + 1 ] = ( 1 − 2 * sao_offset_sign_flag[ cIdx ][ rx ][ ry ][ i ] ) *         (153)
        ( sao_offset_abs[ cIdx ][ rx ][ ry ][ i ] << ( BitDepth − Min( 10, BitDepth ) ) )
```

sao_band_position[cIdx][rx][ry] specifies the displacement of the band offset of the sample range when SaoTypeIdx[cIdx][rx][ry] is equal to 1.

When sao_band_position[cIdx][rx][ry] is not present, it is inferred as follows:

– If sao_merge_left_flag is equal to 1, sao_band_position[cIdx][rx][ry] is inferred to be equal to sao_band_position[cIdx][rx − 1][ry].

– Otherwise, if sao_merge_up_flag is equal to 1, sao_band_position[cIdx][rx][ry] is inferred to be equal to sao_band_position[cIdx][rx][ry − 1].

– Otherwise, sao_band_position[cIdx][rx][ry] is inferred to be equal to 0.

sao_eo_class_luma specifies the edge offset class for the luma component. The array SaoEoClass[cIdx][rx][ry] specifies the offset type as specified in Table 11 for the CTB at the location (rx, ry) for the colour component cIdx. The value of SaoEoClass[0][rx][ry] is derived as follows:

– If sao_eo_class_luma is present, SaoEoClass[0][rx][ry] is set equal to sao_eo_class_luma.

– Otherwise (sao_eo_class_luma is not present), SaoEoClass[0][rx][ry] is derived as follows:

  – If sao_merge_left_flag is equal to 1, SaoEoClass[0][rx][ry] is set equal to SaoEoClass[0][rx − 1][ry].

  – Otherwise, if sao_merge_up_flag is equal to 1, SaoEoClass[0][rx][ry] is set equal to SaoEoClass[0][rx][ry − 1].

  – Otherwise, SaoEoClass[0][rx][ry] is set equal to 0.

sao_eo_class_chroma specifies the edge offset class for the chroma components. The values of SaoEoClass[cIdx][rx][ry] are derived as follows for cIdx equal to 1..2:

– If sao_eo_class_chroma is present, SaoEoClass[cIdx][rx][ry] is set equal to sao_eo_class_chroma.

– Otherwise (sao_eo_class_chroma is not present), SaoEoClass[cIdx][rx][ry] is derived as follows:

  – If sao_merge_left_flag is equal to 1, SaoEoClass[cIdx][rx][ry] is set equal to SaoEoClass[cIdx][rx − 1][ry].

  – Otherwise, if sao_merge_up_flag is equal to 1, SaoEoClass[cIdx][rx][ry] is set equal to SaoEoClass[cIdx][rx][ry − 1].

  – Otherwise, SaoEoClass[cIdx][rx][ry] is set equal to 0.

Table 11 – Specification of the SAO edge offset class

SaoEoClass[cIdx][rx][ry]	SAO edge offset class (informative)
0	1D 0-degree edge offset
1	1D 90-degree edge offset
2	1D 135-degree edge offset
3	1D 45-degree edge offset

7.4.12.4 Coding tree semantics

The variables allowSplitQt, allowSplitBtVer, allowSplitBtHor, allowSplitTtVer, and allowSplitTtHor are derived as follows:

– The allowed quad split process as specified in clause 6.4.1 is invoked with the coding block size cbSize set equal to cbWidth, the current multi-type tree depth mttDepth, treeTypeCurr and modeTypeCurr as inputs, and the output is assigned to allowSplitQt.

– The variables minQtSize, maxBtSize, maxTtSize and maxMttDepth are derived as follows:

  – If treeTypeCurr is equal to DUAL_TREE_CHROMA, minQtSize, maxBtSize, maxTtSize and maxMttDepth are set equal to MinQtSizeC, MaxBtSizeC, MaxTtSizeC and MaxMttDepthC + depthOffset, respectively.

  – Otherwise, minQtSize, maxBtSize, maxTtSize and maxMttDepth are set equal to MinQtSizeY, MaxBtSizeY, MaxTtSizeY and MaxMttDepthY + depthOffset, respectively.

– The allowed binary split process as specified in clause 6.4.2 is invoked with the binary split mode SPLIT_BT_VER, the coding block width cbWidth, the coding block height cbHeight, the location (x0, y0), the current multi-type tree depth mttDepth, the maximum multi-type tree depth with offset maxMttDepth, the maximum binary tree size maxBtSize, the minimum quadtree size minQtSize, the current partition index partIdx, treeTypeCurr and modeTypeCurr as inputs, and the output is assigned to allowSplitBtVer.

– The allowed binary split process as specified in clause 6.4.2 is invoked with the binary split mode SPLIT_BT_HOR, the coding block height cbHeight, the coding block width cbWidth, the location (x0, y0), the current multi-type tree depth mttDepth, the maximum multi-type tree depth with offset maxMttDepth, the maximum binary tree size maxBtSize, the minimum quadtree size minQtSize, the current partition index partIdx, treeTypeCurr and modeTypeCurr as inputs, and the output is assigned to allowSplitBtHor.

– The allowed ternary split process as specified in clause 6.4.3 is invoked with the ternary split mode SPLIT_TT_VER, the coding block width cbWidth, the coding block height cbHeight, the location (x0, y0), the current multi-type tree depth mttDepth, the maximum multi-type tree depth with offset maxMttDepth, the maximum ternary tree size maxTtSize, treeTypeCurr and modeTypeCurr as inputs, and the output is assigned to allowSplitTtVer.

– The allowed ternary split process as specified in clause 6.4.3 is invoked with the ternary split mode SPLIT_TT_HOR, the coding block height cbHeight, the coding block width cbWidth, the location (x0, y0), the current multi-type tree depth mttDepth, the maximum multi-type tree depth with offset maxMttDepth, the maximum ternary tree size maxTtSize, treeTypeCurr and modeTypeCurr as inputs, and the output is assigned to allowSplitTtHor.

split_cu_flag equal to 0 specifies that a coding unit is not split. split_cu_flag equal to 1 specifies that a coding unit is split into four coding units using a quad split as indicated by the syntax element split_qt_flag, or into two coding units using a binary split or into three coding units using a ternary split as indicated by the syntax element mtt_split_cu_binary_flag. The binary or ternary split can be either vertical or horizontal as indicated by the syntax element mtt_split_cu_vertical_flag.

When split_cu_flag is not present, the value of split_cu_flag is inferred as follows:

– If one or more of the following conditions are true, the value of split_cu_flag is inferred to be equal to 1:

  – x0 + cbWidth is greater than pps_pic_width_in_luma_samples.

  – y0 + cbHeight is greater than pps_pic_height_in_luma_samples.

– Otherwise, the value of split_cu_flag is inferred to be equal to 0.

split_qt_flag specifies whether a coding unit is split into coding units with half horizontal and vertical size.

When split_qt_flag is not present, the following applies:

– If all of the following conditions are true, split_qt_flag is inferred to be equal to 1:

  – split_cu_flag is equal to 1.

  – allowSplitQt, allowSplitBtHor, allowSplitBtVer, allowSplitTtHor and allowSplitTtVer are equal to FALSE.

– Otherwise, if allowSplitQt is equal to TRUE, the value of split_qt_flag is inferred to be equal to 1.

– Otherwise, the value of split_qt_flag is inferred to be equal to 0.

mtt_split_cu_vertical_flag equal to 0 specifies that a coding unit is split horizontally. mtt_split_cu_vertical_flag equal to 1 specifies that a coding unit is split vertically

When mtt_split_cu_vertical_flag is not present, it is inferred as follows:

– If allowSplitBtHor is equal to TRUE or allowSplitTtHor is equal to TRUE, the value of mtt_split_cu_vertical_flag is inferred to be equal to 0.

– Otherwise, the value of mtt_split_cu_vertical_flag is inferred to be equal to 1.

mtt_split_cu_binary_flag equal to 0 specifies that a coding unit is split into three coding units using a ternary split. mtt_split_cu_binary_flag equal to 1 specifies that a coding unit is split into two coding units using a binary split.

When mtt_split_cu_binary_flag is not present, it is inferred as follows:

– If allowSplitBtVer is equal to FALSE and allowSplitBtHor is equal to FALSE, the value of mtt_split_cu_binary_flag is inferred to be equal to 0.

– Otherwise, if allowSplitTtVer is equal to FALSE and allowSplitTtHor is equal to FALSE, the value of mtt_split_cu_binary_flag is inferred to be equal to 1.

– Otherwise, if allowSplitBtHor is equal to TRUE and allowSplitTtVer is equal to TRUE, the value of mtt_split_cu_binary_flag is inferred to be equal to 1 − mtt_split_cu_vertical_flag.

– Otherwise (allowSplitBtVer is equal to TRUE and allowSplitTtHor is equal to TRUE), the value of mtt_split_cu_binary_flag is inferred to be equal to mtt_split_cu_vertical_flag.

The variable MttSplitMode[x][y][mttDepth] is derived from the value of mtt_split_cu_vertical_flag and from the value of mtt_split_cu_binary_flag as defined in Table 12 for x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1.

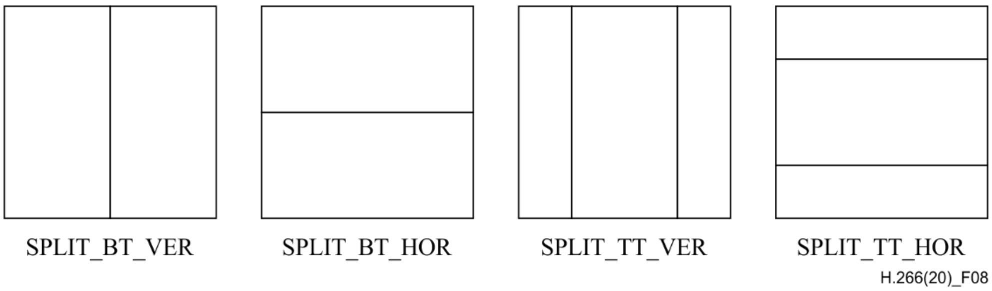

Figure 8 – Multi-type tree splitting modes indicated by MttSplitMode (informative)

MttSplitMode[x0][y0][mttDepth] represents horizontal and vertical binary and ternary splittings of a coding unit within the multi-type tree as illustrated in Figure 8. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

Table 12 – Specification of MttSplitMode[x][y][mttDepth] for x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1

MttSplitMode[x0][y0][mttDepth]	mtt_split_cu_vertical_flag	mtt_split_cu_binary_flag
SPLIT_TT_HOR	0	0
SPLIT_BT_HOR	0	1
SPLIT_TT_VER	1	0
SPLIT_BT_VER	1	1

The variable ModeTypeCondition is derived as follows:

– If one or more of the following conditions are true, ModeTypeCondition is set equal to 0:
  – sh_slice_type is equal to I and sps_qtbtt_dual_tree_intra_flag is equal to 1;
  – modeTypeCurr is not equal to MODE_TYPE_ALL;
  – sps_chroma_format_idc is equal to 0;
  – sps_chroma_format_idc is equal to 3;
– Otherwise, if one or more of the following conditions is true, ModeTypeCondition is set equal to 1:
  – cbWidth * cbHeight is equal to 64 and split_qt_flag is equal to 1;
  – cbWidth * cbHeight is equal to 64 and split_qt_flag is equal to 0 and MttSplitMode[x0][y0][mttDepth] is equal to SPLIT_TT_HOR or SPLIT_TT_VER;
  – cbWidth * cbHeight is equal to 32 and MttSplitMode[x0][y0][mttDepth] is equal to SPLIT_BT_HOR or SPLIT_BT_VER;
– Otherwise, if one of the following conditions is true, ModeTypeCondition is set equal to 1 + (sh_slice_type != I ? 1 : 0):
  – cbWidth * cbHeight is equal to 64 and MttSplitMode[x0][y0][mttDepth] is equal to SPLIT_BT_HOR or SPLIT_BT_VER and sps_chroma_format_idc is equal to 1;
  – cbWidth * cbHeight is equal to 128 and MttSplitMode[x0][y0][mttDepth] is equal to SPLIT_TT_HOR or SPLIT_TT_VER and sps_chroma_format_idc is equal to 1;
  – cbWidth is equal to 8 and MttSplitMode[x0][y0][mttDepth] is equal to SPLIT_BT_VER;
  – cbWidth is equal to 16 and split_qt_flag is equal to 0 and MttSplitMode[x0][y0][mttDepth] is equal to SPLIT_TT_VER;
– Otherwise, ModeTypeCondition is set equal to 0.

non_inter_flag equal to 0 specifies that coding units inside the current coding tree node can only use inter prediction coding modes. non_inter_flag equal to 1 specifies that coding units inside the current coding tree node cannot use inter prediction coding modes.

7.4.12.5 Coding unit semantics

The following assignments are made for x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1:

CbPosX[chType][x][y] = x0 (154)

CbPosY[chType][x][y] = y0 (155)

CbWidth[chType][x][y] = cbWidth (156)

CbHeight[chType][x][y] = cbHeight (157)

CqtDepth[chType][x][y] = cqtDepth (158)

The variable MvdLX[x0][y0][compIdx], with X = 0..1 and compIdx = 0..1, is set equal to 0.

The variable MvdCpLX[x0][y0][cpIdx][compIdx], with X = 0..1, cpIdx = 0..2 and compIdx = 0..1, is set equal to 0.

The variable CclmEnabled is derived by invoking the cross-component chroma intra prediction mode checking process specified in clause 8.4.4 with the luma location (xCb, yCb) set equal to (x0, y0) as input.

cu_skip_flag[x0][y0] equal to 1 specifies that for the current coding unit, when decoding a P or B slice, no more syntax elements except one or more of the following are parsed after cu_skip_flag[x0][y0]: the IBC mode flag pred_mode_ibc_flag [x0][y0], and the merge_data( ) syntax structure; when decoding an I slice, no more syntax elements except merge_idx[x0][y0] are parsed after cu_skip_flag[x0][y0]. cu_skip_flag[x0][y0] equal to 0 specifies that the coding unit is not skipped. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When cu_skip_flag[x0][y0] is not present, it is inferred to be equal to 0.

When treeType is not equal to DUAL_TREE_CHROMA, the variable CuSkipFlag[x][y] is set equal to cu_skip_flag[x0][y0] for x = x0..x0 + cbWidth − 1, y = y0..y0 + cbHeight − 1.

pred_mode_flag equal to 0 specifies that the current coding unit is coded in inter prediction mode. pred_mode_flag equal to 1 specifies that the current coding unit is coded in intra prediction mode.

When pred_mode_flag is not present, it is inferred as follows:

– If cbWidth is equal to 4 and cbHeight is equal to 4, pred_mode_flag is inferred to be equal to 1.

– Otherwise, if modeType is equal to MODE_TYPE_INTRA, pred_mode_flag is inferred to be equal to 1.

– Otherwise, if modeType is equal to MODE_TYPE_INTER, pred_mode_flag is inferred to be equal to 0.

– Otherwise, pred_mode_flag is inferred to be equal to 1 when decoding an I slice, and equal to 0 when decoding a P or B slice, respectively.

The variable CuPredMode[chType][x][y] is derived as follows for c = (treeType = = SINGLE_TREE ? (0..1) : (chType..chType)), x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1:

– If pred_mode_flag is equal to 0, CuPredMode[c][x][y] is set equal to MODE_INTER.

– Otherwise (pred_mode_flag is equal to 1), CuPredMode[c][x][y] is set equal to MODE_INTRA.

pred_mode_ibc_flag equal to 1 specifies that the current coding unit is coded in IBC prediction mode. pred_mode_ibc_flag equal to 0 specifies that the current coding unit is not coded in IBC prediction mode.

When pred_mode_ibc_flag is not present, it is inferred as follows:

– If cu_skip_flag[x0][y0] is equal to 1, and cbWidth is equal to 4, and cbHeight is equal to 4, pred_mode_ibc_flag is inferred to be equal 1.

– Otherwise, if cu_skip_flag[x0][y0] is equal to 1 and modeType is equal to MODE_TYPE_INTRA, pred_mode_ibc_flag is inferred to be equal 1.

– Otherwise, if either cbWidth or cbHeight are equal to 128, pred_mode_ibc_flag is inferred to be equal to 0.

– Otherwise, if modeType is equal to MODE_TYPE_INTER, pred_mode_ibc_flag is inferred to be equal to 0.

– Otherwise, if treeType is equal to DUAL_TREE_CHROMA, pred_mode_ibc_flag is inferred to be equal to 0.

– Otherwise, pred_mode_ibc_flag is inferred to be equal to the value of sps_ibc_enabled_flag when decoding an I slice, and 0 when decoding a P or B slice, respectively.

When pred_mode_ibc_flag is equal to 1, the variable CuPredMode[c][x][y] is set to be equal to MODE_IBC for c = (treeType = = SINGLE_TREE ? (0..1) : (chType..chType)), x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1.

pred_mode_plt_flag specifies the use of palette mode in the current coding unit. pred_mode_plt_flag equal to 1 indicates that palette mode is applied in the current coding unit. pred_mode_plt_flag equal to 0 indicates that palette mode is not applied in the current coding unit. When pred_mode_plt_flag is not present, it is inferred to be equal to 0.

When pred_mode_plt_flag is equal to 1, the variable CuPredMode[c][x][y] is set to be equal to MODE_PLT for c = (treeType = = SINGLE_TREE ? (0..1) : (chType..chType)), x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1.

cu_act_enabled_flag equal to 1 specifies that the decoded residuals of the current coding unit are applied using a colour space conversion. cu_act_enabled_flag equal to 0 specifies that the decoded residuals of the current coding unit are applied without a colour space conversion. When cu_act_enabled_flag is not present, it is inferred to be equal to 0.

intra_bdpcm_luma_flag equal to 1 specifies that BDPCM is applied to the current luma coding block at the location (x0, y0), i.e., the transform is skipped, the luma intra prediction mode is specified by intra_bdpcm_luma_dir_flag. intra_bdpcm_luma_flag equal to 0 specifies that BDPCM is not applied to the current luma coding block at the location (x0, y0).

When intra_bdpcm_luma_flag is not present it is inferred to be equal to 0.

The variable BdpcmFlag[x][y][cIdx] is set equal to intra_bdpcm_luma_flag for x = x0..x0 + cbWidth − 1, y = y0..y0 + cbHeight − 1 and cIdx = 0.

intra_bdpcm_luma_dir_flag equal to 0 specifies that the BDPCM prediction direction is horizontal. intra_bdpcm_luma_dir_flag equal to 1 specifies that the BDPCM prediction direction is vertical.

The variable BdpcmDir[x][y][cIdx] is set equal to intra_bdpcm_luma_dir_flag for x = x0..x0 + cbWidth − 1, y = y0..y0 + cbHeight − 1 and cIdx = 0.

intra_mip_flag equal to 1 specifies that the intra prediction type for luma samples is matrix-based intra prediction. intra_mip_flag equal to 0 specifies that the intra prediction type for luma samples is not matrix-based intra prediction.

When intra_mip_flag is not present, it is inferred to be equal to 0.

When treeType is not equal to DUAL_TREE_CHROMA, the variable IntraMipFlag[x][y] is set equal to intra_mip_flag for x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1.

intra_mip_transposed_flag[x0][y0] specifies whether the input vector for matrix-based intra prediction mode for luma samples is transposed or not.

intra_mip_mode[x0][y0] specifies the matrix-based intra prediction mode for luma samples. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

intra_luma_ref_idx specifies the intra prediction reference line index.

When intra_luma_ref_idx is not present it is inferred to be equal to 0.

The variable IntraLumaRefLineIdx[x][y] is set equal to intra_luma_ref_idx for x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1.

intra_subpartitions_mode_flag equal to 1 specifies that the current intra coding unit is partitioned into NumIntraSubPartitions[x0][y0] transform block subpartitions. intra_subpartitions_mode_flag equal to 0 specifies that the current intra coding unit is not partitioned into transform block subpartitions.

When intra_subpartitions_mode_flag is not present, it is inferred to be equal to 0.

When treeType is not equal to DUAL_TREE_CHROMA, the variable IntraSubPartitionsModeFlag[x][y] is set equal to intra_subpartitions_mode_flag for x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1.

intra_subpartitions_split_flag specifies whether the intra subpartitions split type is horizontal or vertical.

The variable IntraSubPartitionsSplitType specifies the type of split used for the current luma coding block as illustrated in Table 13. IntraSubPartitionsSplitType is derived as follows:

– If intra_subpartitions_mode_flag is equal to 0, IntraSubPartitionsSplitType is set equal to 0.

– Otherwise, the IntraSubPartitionsSplitType is set equal to 1 + intra_subpartitions_split_flag.

Table 13 – Name association to IntraSubPartitionsSplitType

IntraSubPartitionsSplitType	Name of IntraSubPartitionsSplitType
0	ISP_NO_SPLIT
1	ISP_HOR_SPLIT
2	ISP_VER_SPLIT

The variable NumIntraSubPartitions specifies the number of transform block subpartitions into which an intra luma coding block is divided. NumIntraSubPartitions is derived as follows:

– If IntraSubPartitionsSplitType is equal to ISP_NO_SPLIT, NumIntraSubPartitions is set equal to 1.

– Otherwise, if one of the following conditions is true, NumIntraSubPartitions is set equal to 2:

  – cbWidth is equal to 4 and cbHeight is equal to 8,

  – cbWidth is equal to 8 and cbHeight is equal to 4.

– Otherwise, NumIntraSubPartitions is set equal to 4.

The syntax elements **intra_luma_mpm_flag**[x0][y0], **intra_luma_not_planar_flag**[x0][y0], **intra_luma_mpm_idx**[x0][y0] and **intra_luma_mpm_remainder**[x0][y0] specify the intra prediction mode for luma samples. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture. When intra_luma_mpm_flag[x0][y0] is equal to 1, the intra prediction mode is inferred from a neighbouring intra-predicted coding unit according to clause 8.4.2.

When intra_luma_mpm_flag[x0][y0] is not present, it is inferred to be equal to 1.

When intra_luma_not_planar_flag[x0][y0] is not present, it is inferred to be equal to 1.

intra_bdpcm_chroma_flag equal to 1 specifies that BDPCM is applied to the current chroma coding blocks at the location (x0, y0), i.e., the transform is skipped, the chroma intra prediction mode is specified by intra_bdpcm_chroma_dir_flag. intra_bdpcm_chroma_flag equal to 0 specifies that BDPCM is not applied to the current chroma coding blocks at the location (x0, y0).

When intra_bdpcm_chroma_flag is not present it is inferred to be equal to 0.

The variable BdpcmFlag[x][y][cIdx] is set equal to intra_bdpcm_chroma_flag for x = x0..x0 + cbWidth − 1, y = y0..y0 + cbHeight − 1 and cIdx = 1..2.

intra_bdpcm_chroma_dir_flag equal to 0 specifies that the BDPCM prediction direction is horizontal. intra_bdpcm_chroma_dir_flag equal to 1 specifies that the BDPCM prediction direction is vertical.

The variable BdpcmDir[x][y][cIdx] is set equal to intra_bdpcm_chroma_dir_flag for x = x0..x0 + cbWidth − 1, y = y0..y0 + cbHeight − 1 and cIdx = 1..2.

cclm_mode_flag equal to 1 specifies that one of the INTRA_LT_CCLM, INTRA_L_CCLM and INTRA_T_CCLM chroma intra prediction modes is applied. cclm_mode_flag equal to 0 specifies that none of the INTRA_LT_CCLM, INTRA_L_CCLM and INTRA_T_CCLM chroma intra prediction modes is applied.

When cclm_mode_flag is not present, it is inferred to be equal to 0.

cclm_mode_idx specifies which one of the INTRA_LT_CCLM, INTRA_L_CCLM and INTRA_T_CCLM chroma intra prediction modes is applied.

intra_chroma_pred_mode specifies the intra prediction mode for chroma samples. When intra_chroma_pred_mode is not present, it is inferred to be equal to 0.

general_merge_flag[x0][y0] specifies whether the inter prediction parameters for the current coding unit are inferred from a neighbouring inter-predicted partition. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When general_merge_flag[x0][y0] is not present, it is inferred as follows:

– If cu_skip_flag[x0][y0] is equal to 1, general_merge_flag[x0][y0] is inferred to be equal to 1.

– Otherwise, general_merge_flag[x0][y0] is inferred to be equal to 0.

mvp_l0_flag[x0][y0] specifies the motion vector predictor index of list 0 where x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When mvp_l0_flag[x0][y0] is not present, it is inferred to be equal to 0.

mvp_l1_flag[x0][y0] has the same semantics as mvp_l0_flag, with l0 and list 0 replaced by l1 and list 1, respectively.

inter_pred_idc[x0][y0] specifies whether list0, list1, or bi-prediction is used for the current coding unit according to Table 14. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

Table 14 – Name association to inter prediction mode

inter_pred_idc	**Name of inter_pred_idc**		
	(cbWidth + cbHeight) > 12	**(cbWidth + cbHeight) = = 12**	**(cbWidth + cbHeight) = = 8**
0	PRED_L0	PRED_L0	n.a.
1	PRED_L1	PRED_L1	n.a.
2	PRED_BI	n.a.	n.a.

When inter_pred_idc[x0][y0] is not present, it is inferred to be equal to PRED_L0.

sym_mvd_flag[x0][y0] equal to 1 specifies that the syntax elements ref_idx_l0[x0][y0] and ref_idx_l1[x0][y0], and the mvd_coding(x0, y0, refList, cpIdx) syntax structure for refList equal to 1 are not present. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When sym_mvd_flag[x0][y0] is not present, it is inferred to be equal to 0.

ref_idx_l0[x0][y0] specifies the list 0 reference picture index for the current coding unit. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When ref_idx_l0[x0][y0] is not present it is inferred as follows:

– If sym_mvd_flag[x0][y0] is equal to 1, ref_idx_l0[x0][y0] is inferred to be equal to RefIdxSymL0.

– Otherwise (sym_mvd_flag[x0][y0] is equal to 0), ref_idx_l0[x0][y0] is inferred to be equal to 0.

ref_idx_l1[x0][y0] has the same semantics as ref_idx_l0, with l0, L0 and list 0 replaced by l1, L1 and list 1, respectively.

inter_affine_flag[x0][y0] equal to 1 specifies that for the current coding unit, when decoding a P or B slice, affine model based motion compensation is used to generate the prediction samples of the current coding unit. inter_affine_flag[x0][y0] equal to 0 specifies that the coding unit is not predicted by affine model based motion compensation. When inter_affine_flag[x0][y0] is not present, it is inferred to be equal to 0.

The variable InterAffineFlag[x][y] is set equal to inter_affine_flag[x0][y0] for x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1.

cu_affine_type_flag[x0][y0] equal to 1 specifies that for the current coding unit, when decoding a P or B slice, 6-parameter affine model based motion compensation is used to generate the prediction samples of the current coding unit. cu_affine_type_flag[x0][y0] equal to 0 specifies that 4-parameter affine model based motion compensation is used to generate the prediction samples of the current coding unit. When cu_affine_type_flag[x0][y0] is not present, it is inferred to be equal to 0.

MotionModelIdc[x][y] represents motion model of a coding unit as illustrated in Table 15. The array indices x, y specify the luma sample location (x, y) relative to the top-left luma sample of the picture.

The variable MotionModelIdc[x][y] is derived as follows for x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1:

– If general_merge_flag[x0][y0] is equal to 1, the following applies:

MotionModelIdc[x][y] = merge_subblock_flag[x0][y0] (159)

– Otherwise (general_merge_flag[x0][y0] is equal to 0), the following applies:

$$\text{MotionModelIdc[x][y] = inter\_affine\_flag[x0][y0] + cu\_affine\_type\_flag[x0][y0]} \quad (160)$$

Table 15 – Interpretation of MotionModelIdc[x0][y0]

MotionModelIdc[x][y]	Motion model for motion compensation
0	Translational motion
1	4-parameter affine motion
2	6-parameter affine motion

amvr_flag[x0][y0] specifies the resolution of the motion vector differences for the current CU. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture. amvr_flag[x0][y0] equal to 0 specifies that the resolution of the motion vector differences is 1/4 of a luma sample. amvr_flag[x0][y0] equal to 1 specifies that the resolution of the motion vector differences is further specified by amvr_precision_idx[x0][y0].

When amvr_flag[x0][y0] is not present, it is inferred as follows:

– If CuPredMode[chType][x0][y0] is equal to MODE_IBC, amvr_flag[x0][y0] is inferred to be equal to 1.

– Otherwise (CuPredMode[chType][x0][y0] is not equal to MODE_IBC), amvr_flag[x0][y0] is inferred to be equal to 0.

amvr_precision_idx[x0][y0] specifies that the resolution of the motion vector difference with AmvrShift is defined in Table 16. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When amvr_precision_idx[x0][y0] is not present, it is inferred to be equal to 0.

The motion vector differences are modified as follows:

– If inter_affine_flag[x0][y0] is equal to 0, the variables MvdL0[x0][y0][0], MvdL0[x0][y0][1], MvdL1[x0][y0][0], MvdL1[x0][y0][1] are modified as follows:

$$\text{MvdL0[x0][y0][0] = MvdL0[x0][y0][0]} \ll \text{AmvrShift} \quad (161)$$

$$\text{MvdL0[x0][y0][1] = MvdL0[x0][y0][1]} \ll \text{AmvrShift} \quad (162)$$

$$\text{MvdL1[x0][y0][0] = MvdL1[x0][y0][0]} \ll \text{AmvrShift} \quad (163)$$

$$\text{MvdL1[x0][y0][1] = MvdL1[x0][y0][1]} \ll \text{AmvrShift} \quad (164)$$

– Otherwise (inter_affine_flag[x0][y0] is equal to 1), the variables MvdCpL0[x0][y0][0][0], MvdCpL0[x0][y0][0][1], MvdCpL0[x0][y0][1][0], MvdCpL0[x0][y0][1][1], MvdCpL0[x0][y0][2][0] and MvdCpL0[x0][y0][2][1] are modified as follows:

$$\text{MvdCpL0[x0][y0][0][0] = MvdCpL0[x0][y0][0][0]} \ll \text{AmvrShift} \quad (165)$$

$$\text{MvdCpL1[x0][y0] [0][1] = MvdCpL1[x0][y0][0][1]} \ll \text{AmvrShift} \quad (166)$$

$$\text{MvdCpL0[x0][y0][1][0] = MvdCpL0[x0][y0][1][0]} \ll \text{AmvrShift} \quad (167)$$

$$\text{MvdCpL1[x0][y0] [1][1] = MvdCpL1[x0][y0][1][1]} \ll \text{AmvrShift} \quad (168)$$

$$\text{MvdCpL0[x0][y0][2][0] = MvdCpL0[x0][y0][2][0]} \ll \text{AmvrShift} \quad (169)$$

$$\text{MvdCpL1[x0][y0] [2][1] = MvdCpL1[x0][y0][2][1]} \ll \text{AmvrShift} \quad (170)$$

Table 16 – Specification of AmvrShift

amvr_flag	amvr_precision_idx	AmvrShift		
		inter_affine_flag[x0][y0] = = 1	**CuPredMode[chType][x0][y0] = = MODE_IBC)**	**inter_affine_flag[x0][y0] = =0 && CuPredMode[chType][x0][y0] != MODE_IBC**
0	-	2 (1/4 luma sample)	-	2 (1/4 luma sample)
1	0	0 (1/16 luma sample)	4 (1 luma sample)	3 (1/2 luma sample)
1	1	4 (1 luma sample)	6 (4 luma samples)	4 (1 luma sample)
1	2	-	-	6 (4 luma samples)

bcw_idx[x0][y0] specifies the weight index of bi-prediction with CU weights. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When bcw_idx[x0][y0] is not present, it is inferred to be equal to 0.

cu_coded_flag equal to 1 specifies that the transform_tree( ) syntax structure is present for the current coding unit. cu_coded_flag equal to 0 specifies that the transform_tree( ) syntax structure is not present for the current coding unit.

When cu_coded_flag is not present, it is inferred as follows:

– If cu_skip_flag[x0][y0] is equal to 1 or pred_mode_plt_flag is equal to 1, cu_coded_flag is inferred to be equal to 0.

– Otherwise, cu_coded_flag is inferred to be equal to 1.

cu_sbt_flag equal to 1 specifies that for the current coding unit, subblock transform is used. cu_sbt_flag equal to 0 specifies that for the current coding unit, subblock transform is not used.

When cu_sbt_flag is not present, its value is inferred to be equal to 0.

NOTE – When subblock transform is used, a coding unit is split into two transform units; one transform unit has residual data, the other does not have residual data.

cu_sbt_quad_flag equal to 1 specifies that for the current coding unit, the subblock transform includes a transform unit of 1/4 size of the current coding unit. cu_sbt_quad_flag equal to 0 specifies that for the current coding unit the subblock transform includes a transform unit of 1/2 size of the current coding unit.

When cu_sbt_quad_flag is not present, its value is inferred to be equal to 0.

cu_sbt_horizontal_flag equal to 1 specifies that the current coding unit is split horizontally into 2 transform units. cu_sbt_horizontal_flag[x0][y0] equal to 0 specifies that the current coding unit is split vertically into 2 transform units.

When cu_sbt_horizontal_flag is not present, its value is derived as follows:

– If cu_sbt_quad_flag is equal to 1, cu_sbt_horizontal_flag is set to be equal to allowSbtHorQ.

– Otherwise (cu_sbt_quad_flag is equal to 0), cu_sbt_horizontal_flag is set to be equal to allowSbtHorH.

cu_sbt_pos_flag equal to 1 specifies that the tu_y_coded_flag, tu_cb_coded_flag and tu_cr_coded_flag of the first transform unit in the current coding unit are not present. cu_sbt_pos_flag equal to 0 specifies that the tu_y_coded_flag, tu_cb_coded_flag and tu_cr_coded_flag of the second transform unit in the current coding unit are not present.

The variable SbtNumFourthsTb0 is derived as follows:

$$\text{sbtMinNumFourths} = \text{cu\_sbt\_quad\_flag} \; ? \; 1 : 2 \quad (171)$$

$$\text{SbtNumFourthsTb0} = \text{cu\_sbt\_pos\_flag} \; ? \; (4 - \text{sbtMinNumFourths}) : \text{sbtMinNumFourths} \quad (172)$$

lfnst_idx specifies whether and which one of the two low frequency non-separable transform kernels in a selected transform set is used. lfnst_idx equal to 0 specifies that the low frequency non-separable transform is not used in the current coding unit.

When lfnst_idx is not present, it is inferred to be equal to 0.

The variable ApplyLfnstFlag[cIdx] is derived as follows:

– When treeType is equal to SINGLE_TREE or DUAL_TREE_LUMA, the following applies:

$$\text{ApplyLfnstFlag[0] = (lfnst\_idx > 0) ? 1 : 0} \quad (173)$$

– The following applies for cIdx = 1, 2:

$$\text{ApplyLfnstFlag[cIdx] = (lfnst\_idx > 0 \&\& treeType == DUAL\_TREE\_CHROMA) ? 1 : 0} \quad (174)$$

mts_idx specifies which transform kernels are applied along the horizontal and vertical direction of the associated luma transform blocks in the current coding unit.

When mts_idx is not present, it is inferred to be equal to 0.

When ResetIbcBuf is equal to 1, the following applies:

– For x = 0..IbcBufWidthY − 1 and y = 0..CtbSizeY − 1, the following assignments are made:

$$\text{IbcVirBuf[0][x][y] = −1} \quad (175)$$

– The variable ResetIbcBuf is set equal to 0.

When x0 % VSize is equal to 0 and y0 % VSize is equal to 0, the following assignments are made for x = x0..x0 + Max(cbWidth, VSize) − 1 and y = y0..y0 + Max(cbHeight, VSize) − 1:

$$\text{IbcVirBuf[0][(x + (IbcBufWidthY >> 1)) \% IbcBufWidthY][y \% CtbSizeY] = −1} \quad (176)$$

7.4.12.6 Palette coding semantics

In the following semantics, the array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture. The array indices xC, yC specify the location (xC, yC) of the sample relative to the top-left luma sample of the picture, when treeType is equal to SINGLE_TREE or DUAL_TREE_LUMA; and relative to the top-left chroma sample of the picture, when treeType is equal to DUAL_TREE_CHROMA. The array index startComp specifies the first colour component of the current palette table. startComp equal to 0 indicates the Y component; startComp equal to 1 indicates the Cb component; startComp equal to 2 indicates the Cr component. numComps specifies the number of colour components in the current palette table.

The predictor palette consists of palette entries from previous coding units that are used to predict the entries in the current palette.

PredictorPaletteSize[startComp] specifies the size of the predictor palette for the first colour component of the current palette table startComp. PredictorPaletteSize[startComp] is derived as specified in clause 8.4.5.3.

PalettePredictorEntryReuseFlags[i] equal to 1 specifies that the i-th entry in the predictor palette is reused in the current palette. PalettePredictorEntryReuseFlags[i] equal to 0 specifies that the i-th entry in the predictor palette is not an entry in the current palette. All elements of the array PalettePredictorEntryReuseFlags[i] are initialized to 0.

palette_predictor_run is used to determine the number of zeros that precede a non-zero entry in the array PalettePredictorEntryReuseFlags.

It is a requirement of bitstream conformance that the value of palette_predictor_run shall be in the range of 0 to (PredictorPaletteSize[startComp] − predictorEntryIdx), inclusive, where predictorEntryIdx corresponds to the current position in the array PalettePredictorEntryReuseFlags. The variable NumPredictedPaletteEntries specifies the number of entries in the current palette that are reused from the predictor palette. The value of NumPredictedPaletteEntries shall be in the range of 0 to maxNumPaletteEntries, inclusive.

num_signalled_palette_entries specifies the number of entries in the current palette that are explicitly signalled for the first colour component of the current palette table startComp.

When num_signalled_palette_entries is not present, it is inferred to be equal to 0.

The variable CurrentPaletteSize[startComp] specifies the size of the current palette for the first colour component of the current palette table startComp and is derived as follows:

$$\text{CurrentPaletteSize[startComp] = NumPredictedPaletteEntries + num\_signalled\_palette\_entries} \quad (177)$$

The value of CurrentPaletteSize[startComp] shall be in the range of 0 to maxNumPaletteEntries, inclusive.

new_palette_entries[cIdx][i] specifies the value for the i-th signalled palette entry for the colour component cIdx.

The variable LocalDualTreeFlag is derived as follows:

```
LocalDualTreeFlag = ( treeType  !=  SINGLE_TREE  &&
    ( sh_slice_type  !=  I  | |  ( sh_slice_type = = I  &&  sps_qtbtt_dual_tree_intra_flag = = 0 ) ) ) ? 1 : 0       (178)
```

The variable PredictorPaletteEntries[cIdx][i] specifies the i-th element in the predictor palette for the colour component cIdx.

The variable CurrentPaletteEntries[cIdx][i] specifies the i-th element in the current palette for the colour component cIdx and is derived as follows:

```
numPredictedPaletteEntries = 0
for( i = 0; i < PredictorPaletteSize[ startComp ]; i++ )
    if( PalettePredictorEntryReuseFlags[ i ] ) {
        for( cIdx = LocalDualTreeFlag ? 0 : startComp; cIdx < LocalDualTreeFlag ? 3 :
                ( startComp + numComps ); cIdx++ )
            CurrentPaletteEntries[ cIdx ][ numPredictedPaletteEntries ] = PredictorPaletteEntries[ cIdx ][ i ]
        numPredictedPaletteEntries++
    }
for( cIdx = startComp; cIdx < (startComp + numComps); cIdx++)                                  (179)
    for( i = 0; i < num_signalled_palette_entries; i++ )
        CurrentPaletteEntries[ cIdx ][ numPredictedPaletteEntries + i ] = new_palette_entries[ cIdx ][ i ]
```

palette_escape_val_present_flag equal to 1 specifies that the current coding unit contains at least one escape coded sample. palette_escape_val_present_flag equal to 0 specifies that there are no escape coded samples in the current coding unit. When not present, the value of palette_escape_val_present_flag is inferred to be equal to 1.

The variable MaxPaletteIndex specifies the maximum possible value for a palette index for the current coding unit. The value of MaxPaletteIndex is set equal to CurrentPaletteSize[startComp] − 1 + palette_escape_val_present_flag.

palette_idx_idc is an indication of an index to the palette table, CurrentPaletteEntries. The value of palette_idx_idc shall be in the range of 0 to MaxPaletteIndex, inclusive, for the first index in the block and in the range of 0 to (MaxPaletteIndex − 1), inclusive, for the remaining indices in the block.

When palette_idx_idc is not present, it is inferred to be equal to 0.

palette_transpose_flag equal to 1 specifies that vertical traverse scan is applied for scanning the indices for samples in the current coding unit. palette_transpose_flag equal to 0 specifies that horizontal traverse scan is applied for scanning the indices for samples in the current coding unit. When not present, the value of palette_transpose_flag is inferred to be equal to 0.

The array TraverseScanOrder specifies the scan order array for palette coding. If palette_transpose_flag is equal to 0, TraverseScanOrder is assigned the horizontal scan order HorTravScanOrder. Otherwise (palette_transpose_flag is equal to 1), TraverseScanOrder is assigned the vertical scan order VerTravScanOrder.

run_copy_flag equal to 1 specifies that the palette run type is the same as the run type at the previously scanned position and palette index is the same as the index at the previous scanned position if CopyAboveIndicesFlag[xC][yC] is equal to 0. Otherwise, run_copy_flag equal to 0 specifies that the palette run type is different from the run type at the previously scanned position.

copy_above_palette_indices_flag equal to 1 specifies that the palette index is equal to the palette index at the same location in the row above if horizontal traverse scan is used or the same location in the left column if vertical traverse scan is used. copy_above_palette_indices_flag equal to 0 specifies that an indication of the palette index of the sample is coded in the bitstream or inferred.

The variable CopyAboveIndicesFlag[xC][yC] equal to 1 specifies that the palette index is copied from the palette index in the row above (horizontal scan) or left column (vertical scan). CopyAboveIndicesFlag[xC][yC] equal to 0 specifies that the palette index is explicitly coded in the bitstream or inferred.

The variable PaletteIndexMap[xC][yC] specifies a palette index, which is an index to the array represented by CurrentPaletteEntries. The value of PaletteIndexMap[xC][yC] shall be in the range of 0 to MaxPaletteIndex, inclusive.

The variable adjustedRefPaletteIndex is derived as follows:

```
adjustedRefPaletteIndex = MaxPaletteIndex + 1
if( PaletteScanPos > 0 ) {
    xcPrev = x0 + TraverseScanOrder[ log2CbWidth ][ log2bHeight ][ PaletteScanPos − 1 ][ 0 ]
    ycPrev = y0 + TraverseScanOrder[ log2CbWidth ][ log2bHeight ][ PaletteScanPos − 1 ][ 1 ]
    if( CopyAboveIndicesFlag[ xcPrev ][ ycPrev ] = = 0 )
        adjustedRefPaletteIndex = PaletteIndexMap[ xcPrev ][ ycPrev ]                    (180)
    else {
        if( !palette_transpose_flag )
            adjustedRefPaletteIndex = PaletteIndexMap[ xC ][ yC − 1 ]
        else
            adjustedRefPaletteIndex = PaletteIndexMap[ xC − 1 ][ yC ]
    }
}
```

When CopyAboveIndicesFlag[xC][yC] is equal to 0, the variable CurrPaletteIndex is derived as follows:

```
if( CurrPaletteIndex >= adjustedRefPaletteIndex )
    CurrPaletteIndex++                                                                   (181)
```

palette_escape_val specifies the quantized escape coded sample value for a component.

The variable PaletteEscapeVal[cIdx][xC][yC] specifies the escape value of a sample for which PaletteIndexMap[xC][yC] is equal to MaxPaletteIndex and palette_escape_val_present_flag is equal to 1. The array index cIdx specifies the colour component.

It is a requirement of bitstream conformance that PaletteEscapeVal[cIdx][xC][yC] shall be in the range of 0 to (1 << BitDepth) − 1, inclusive.

7.4.12.7 Merge data semantics

merge_idx[x0][y0] specifies the merging candidate index of the merging candidate list where x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When merge_idx[x0][y0] is not present, it is inferred as follows:

– If mmvd_merge_flag[x0][y0] is equal to 1, merge_idx[x0][y0] is inferred to be equal to mmvd_cand_flag[x0][y0].

– Otherwise (mmvd_merge_flag[x0][y0] is equal to 0), merge_idx[x0][y0] is inferred to be equal to 0.

merge_subblock_flag[x0][y0] specifies whether the subblock-based inter prediction parameters for the current coding unit are inferred from neighbouring blocks. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture. When merge_subblock_flag[x0][y0] is not present, it is inferred to be equal to 0.

The variable MergeSubblockFlag[x][y] is set equal to merge_subblock_flag[x0][y0] for x = x0..x0 + cbWidth − 1, y = y0..y0 + cbHeight − 1.

merge_subblock_idx[x0][y0] specifies the merging candidate index of the subblock-based merging candidate list where x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When merge_subblock_idx[x0][y0] is not present, it is inferred to be equal to 0.

regular_merge_flag[x0][y0] equal to 1 specifies that regular merge mode or merge mode with motion vector difference is used to generate the inter prediction parameters of the current coding unit. regular_merge_flag[x0][y0] equal to 0 specifies that neither the regular merge mode nor the merge mode with motion vector difference is used to generate the inter prediction parameters of the current coding unit. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When regular_merge_flag[x0][y0] is not present, it is inferred to be equal to general_merge_flag[x0][y0] && !merge_subblock_flag[x0][y0].

mmvd_merge_flag[x0][y0] equal to 1 specifies that merge mode with motion vector difference is used to generate the inter prediction parameters of the current coding unit. mmvd_merge_flag[x0][y0] equal to 0 specifies that merge mode with motion vector difference is not used to generate the inter prediction parameters. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When mmvd_merge_flag[x0][y0] is not present, it is inferred to be equal to 0.

mmvd_cand_flag[x0][y0] specifies whether the first (0) or the second (1) candidate in the merging candidate list is used with the motion vector difference derived from mmvd_distance_idx[x0][y0] and mmvd_direction_idx[x0][y0]. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When mmvd_cand_flag[x0][y0] is not present, it is inferred to be equal to 0.

mmvd_distance_idx[x0][y0] specifies the index used to derive MmvdDistance[x0][y0] as specified in Table 17. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

Table 17 – Specification of MmvdDistance[x0][y0] based on mmvd_distance_idx[x0][y0]

mmvd_distance_idx[x0][y0]	**MmvdDistance[x0][y0]**	
	ph_mmvd_fullpel_only_flag = = 0	**ph_mmvd_fullpel_only_flag = = 1**
0	1	4
1	2	8
2	4	16
3	8	32
4	16	64
5	32	128
6	64	256
7	128	512

mmvd_direction_idx[x0][y0] specifies index used to derive MmvdSign[x0][y0] as specified in Table 18. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

Table 18 – Specification of MmvdSign[x0][y0] based on mmvd_direction_idx[x0][y0]

mmvd_direction_idx[x0][y0]	**MmvdSign[x0][y0][0]**	**MmvdSign[x0][y0][1]**
0	+1	0
1	−1	0
2	0	+1
3	0	−1

Both components of the merge plus MVD offset MmvdOffset[x0][y0] are derived as follows:

MmvdOffset[x0][y0][0] = (MmvdDistance[x0][y0] << 2) * MmvdSign[x0][y0][0] (182)

MmvdOffset[x0][y0][1] = (MmvdDistance[x0][y0] << 2) * MmvdSign[x0][y0][1] (183)

ciip_flag[x0][y0] specifies whether the combined inter-picture merge and intra-picture prediction is applied for the current coding unit. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When ciip_flag[x0][y0] is not present, it is inferred as follows:

– If all the following conditions are true, ciip_flag[x0][y0] is inferred to be equal to 1:
  – sps_ciip_enabled_flag is equal to 1.
  – general_merge_flag[x0][y0] is equal to 1.
  – merge_subblock_flag[x0][y0] is equal to 0.
  – regular_merge_flag[x0][y0] is equal to 0.
  – cu_skip_flag[x0][y0] is equal to 0.

  – cbWidth is less than 128.
  – cbHeight is less than 128.
  – cbWidth * cbHeight is greater than or equal to 64.
– Otherwise, ciip_flag[x0][y0] is inferred to be equal to 0.

When ciip_flag[x0][y0] is equal to 1, the variable IntraPredModeY[x][y] with x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1 is set to be equal to INTRA_PLANAR.

The variable MergeGpmFlag[x0][y0], which specifies whether geometric partitioning based motion compensation is used to generate the prediction samples of the current coding unit, when decoding a B slice, is derived as follows:

– If all the following conditions are true, MergeGpmFlag[x0][y0] is set equal to 1:
  – sps_gpm_enabled_flag is equal to 1.
  – sh_slice_type is equal to B.
  – general_merge_flag[x0][y0] is equal to 1.
  – cbWidth is greater than or equal to 8.
  – cbHeight is greater than or equal to 8.
  – cbWidth is less than 8 * cbHeight.
  – cbHeight is less than 8 * cbWidth.
  – regular_merge_flag[x0][y0] is equal to 0.
  – merge_subblock_flag[x0][y0] is equal to 0.
  – ciip_flag[x0][y0] is equal to 0.
– Otherwise, MergeGpmFlag[x0][y0] is set equal to 0.

merge_gpm_partition_idx[x0][y0] specifies the partitioning shape of the geometric partitioning merge mode. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When merge_gpm_partition_idx[x0][y0] is not present, it is inferred to be equal to 0.

merge_gpm_idx0[x0][y0] specifies the first merging candidate index of the geometric partitioning based motion compensation candidate list where x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When merge_gpm_idx0[x0][y0] is not present, it is inferred to be equal to 0.

merge_gpm_idx1[x0][y0] specifies the second merging candidate index of the geometric partitioning based motion compensation candidate list where x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When merge_gpm_idx1[x0][y0] is not present, it is inferred to be equal to 0.

7.4.12.8 Motion vector difference semantics

abs_mvd_greater0_flag[compIdx] specifies whether the absolute value of a motion vector component difference is greater than 0.

abs_mvd_greater1_flag[compIdx] specifies whether the absolute value of a motion vector component difference is greater than 1.

When abs_mvd_greater1_flag[compIdx] is not present, it is inferred to be equal to 0.

abs_mvd_minus2[compIdx] plus 2 specifies the absolute value of a motion vector component difference.

When abs_mvd_minus2[compIdx] is not present, it is inferred to be equal to −1.

mvd_sign_flag[compIdx] specifies the sign of a motion vector component difference as follows:

– If mvd_sign_flag[compIdx] is equal to 0, the corresponding motion vector component difference has a positive value.

– Otherwise (mvd_sign_flag[compIdx] is equal to 1), the corresponding motion vector component difference has a negative value.

When mvd_sign_flag[compIdx] is not present, it is inferred to be equal to 0.

The motion vector difference lMvd[compIdx] for compIdx = 0..1 is derived as follows:

$$\text{lMvd[compIdx]} = \text{abs\_mvd\_greater0\_flag[compIdx]} * (\text{abs\_mvd\_minus2[compIdx]} + 2) * (1 - 2 * \text{mvd\_sign\_flag[compIdx]}) \quad (184)$$

The value of lMvd[compIdx] shall be in the range of -2^{17} to $2^{17} - 1$, inclusive.

Depending in the value of MotionModelIdc[x0][y0], motion vector differences are derived as follows:

– If MotionModelIdc[x0][y0] is equal to 0, the variables MvdLX[x0][y0][compIdx], with X being 0 or 1, specify the difference between a list X vector component to be used and its prediction, the array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture, and the array index compIdx specifies horizontal or vertical component motion vector difference, are derived as follows:
  – If refList is equal to 0, MvdL0[x0][y0][compIdx] is set equal to lMvd[compIdx] for compIdx = 0..1.
  – Otherwise (refList is equal to 1), MvdL1[x0][y0][compIdx] is set equal to lMvd[compIdx] for compIdx = 0..1.
– Otherwise (MotionModelIdc[x0][y0] is not equal to 0), the variables MvdCpLX[x0][y0][cpIdx][compIdx], with X being 0 or 1, specify the difference between a list X vector component to be used and its prediction, the array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture, the array index cpIdx specifies the control point index, and the array index compIdx specifies horizontal or vertical component motion vector difference, are derived as follows:
  – If refList is equal to 0, MvdCpL0[x0][y0][cpIdx][compIdx] is set equal to lMvd[compIdx] for compIdx = 0..1.
  – Otherwise (refList is equal to 1), MvdCpL1[x0][y0][cpIdx][compIdx] is set equal to lMvd[compIdx] for compIdx = 0..1.

When sym_mvd_flag[x0][y0] is equal to 1, the value of MvdL0[x0][y0][compIdx] shall not be equal to -2^{17}.

7.4.12.9 Transform tree semantics

The transform tree is a recursive syntax structure that contains one or more transform unit syntax structures, i.e., transform_unit( ). The root of the transform tree is a coding unit syntax structure, i.e., coding_unit( ).

7.4.12.10 Transform unit semantics

The transform coefficient levels are represented by the arrays TransCoeffLevel[x0][y0][cIdx][xC][yC]. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered transform block relative to the top-left luma sample of the picture. The array index cIdx specifies an indicator for the colour component; it is equal to 0 for Y, 1 for Cb, and 2 for Cr. The array indices xC and yC specify the transform coefficient location (xC, yC) within the current transform block. When the value of TransCoeffLevel[x0][y0][cIdx][xC][yC] is not specified in clause 7.3.11.11, it is inferred to be equal to 0.

tu_cb_coded_flag[x0][y0] equal to 1 specifies that the Cb transform block contains one or more transform coefficient levels not equal to 0. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered transform block relative to the top-left luma sample of the picture.

When tu_cb_coded_flag[x0][y0] is not present, its value is inferred to be equal to 0.

tu_cr_coded_flag[x0][y0] equal to 1 specifies that the Cr transform block contains one or more transform coefficient levels not equal to 0. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered transform block relative to the top-left luma sample of the picture.

When tu_cr_coded_flag[x0][y0] is not present, its value is inferred to be equal to 0.

tu_y_coded_flag[x0][y0] equal to 1 specifies that the luma transform block contains one or more transform coefficient levels not equal to 0. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered transform block relative to the top-left luma sample of the picture.

When tu_y_coded_flag[x0][y0] is not present and treeType is not equal to DUAL_TREE_CHROMA, its value is inferred as follows:

– If cu_sbt_flag is equal to 1 and one of the following conditions is true, tu_y_coded_flag[x0][y0] is inferred to be equal to 0:

  – subTuIndex is equal to 0 and cu_sbt_pos_flag is equal to 1;

  – subTuIndex is equal to 1 and cu_sbt_pos_flag is equal to 0.

– Otherwise, tu_y_coded_flag[x0][y0] is inferred to be equal to 1.

tu_joint_cbcr_residual_flag[x0][y0] specifies whether the residual samples for both chroma components Cb and Cr are coded as a single transform block. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered transform block relative to the top-left luma sample of the picture.

tu_joint_cbcr_residual_flag[x0][y0] equal to 1 specifies that the transform unit syntax includes the transform coefficient levels for a single transform block from which the residual samples for both Cb and Cr are derived. tu_joint_cbcr_residual_flag[x0][y0] equal to 0 specifies that the transform coefficient levels of the chroma components are coded as indicated by the syntax elements tu_cb_coded_flag[x0][y0] and tu_cr_coded_flag[x0][y0].

When tu_joint_cbcr_residual_flag[x0][y0] is not present, it is inferred to be equal to 0.

Depending on tu_joint_cbcr_residual_flag[x0][y0], tu_cb_coded_flag[x0][y0], and tu_cr_coded_flag[x0][y0], the variable TuCResMode[x0][y0] is derived as follows:

– If tu_joint_cbcr_residual_flag[x0][y0] is equal to 0, the variable TuCResMode[x0][y0] is set equal to 0.

– Otherwise, if tu_cb_coded_flag[x0][y0] is equal to 1 and tu_cr_coded_flag[x0][y0] is equal to 0, the variable TuCResMode[x0][y0] is set equal to 1.

– Otherwise, if tu_cb_coded_flag[x0][y0] is equal to 1, the variable TuCResMode[x0][y0] is set equal to 2.

– Otherwise, the variable TuCResMode[x0][y0] is set equal to 3.

cu_qp_delta_abs specifies the absolute value of the difference CuQpDeltaVal between the quantization parameter of the current coding unit and its prediction.

cu_qp_delta_sign_flag specifies the sign of CuQpDeltaVal as follows:

– If cu_qp_delta_sign_flag is equal to 0, the corresponding CuQpDeltaVal has a positive value.

– Otherwise (cu_qp_delta_sign_flag is equal to 1), the corresponding CuQpDeltaVal has a negative value.

When cu_qp_delta_sign_flag is not present, it is inferred to be equal to 0.

When cu_qp_delta_abs is present, the variables IsCuQpDeltaCoded and CuQpDeltaVal are derived as follows:

$$\text{IsCuQpDeltaCoded} = 1 \quad (185)$$

$$\text{CuQpDeltaVal} = \text{cu\_qp\_delta\_abs} * (\ 1 - 2 * \text{cu\_qp\_delta\_sign\_flag}\ ) \quad (186)$$

The value of CuQpDeltaVal shall be in the range of −(32 + QpBdOffset / 2) to +(31 + QpBdOffset / 2), inclusive.

cu_chroma_qp_offset_flag when present and equal to 1, specifies that an entry in the pps_cb_qp_offset_list[ ] is used to determine the value of CuQpOffset_{Cb}, a corresponding entry in the pps_cr_qp_offset_list[ ] is used to determine the value of CuQpOffset_{Cr}, and a corresponding entry in the pps_joint_cbcr_qp_offset_list[ ] is used to determine the value of CuQpOffset_{CbCr}. cu_chroma_qp_offset_flag equal to 0 specifies that these lists are not used to determine the values of CuQpOffset_{Cb}, CuQpOffset_{Cr}, and CuQpOffset_{CbCr}.

cu_chroma_qp_offset_idx, when present, specifies the index into the pps_cb_qp_offset_list[ ], pps_cr_qp_offset_list[ ], and pps_joint_cbcr_qp_offset_list[ ] that is used to determine the value of CuQpOffset_{Cb}, CuQpOffset_{Cr}, and CuQpOffset_{CbCr}. When present, the value of cu_chroma_qp_offset_idx shall be in the range of 0 to pps_chroma_qp_offset_list_len_minus1, inclusive. When not present, the value of cu_chroma_qp_offset_idx is inferred to be equal to 0.

When cu_chroma_qp_offset_flag is present, the following applies:

– The variable IsCuChromaQpOffsetCoded is set equal to 1.

– The variables CuQpOffset_{Cb}, CuQpOffset_{Cr}, and CuQpOffset_{CbCr} are derived as follows:

  – If cu_chroma_qp_offset_flag is equal to 1, the following applies:

$$\text{CuQpOffset}_{Cb} = \text{pps\_cb\_qp\_offset\_list}[\ \text{cu\_chroma\_qp\_offset\_idx}\ ] \quad (187)$$

$$\text{CuQpOffset}_{Cr} = \text{pps\_cr\_qp\_offset\_list}[\ \text{cu\_chroma\_qp\_offset\_idx}\ ] \tag{188}$$

$$\text{CuQpOffset}_{CbCr} = \text{pps\_joint\_cbcr\_qp\_offset\_list}[\ \text{cu\_chroma\_qp\_offset\_idx}\ ] \tag{189}$$

– Otherwise (cu_chroma_qp_offset_flag is equal to 0), CuQpOffset_{Cb}, CuQpOffset_{Cr}, and CuQpOffset_{CbCr} are all set equal to 0.

transform_skip_flag[x0][y0][cIdx] specifies whether a transform is applied to the associated transform block or not. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered transform block relative to the top-left luma sample of the picture. The array index cIdx specifies an indicator for the colour component; it is equal to 0 for Y, 1 for Cb, and 2 for Cr. transform_skip_flag[x0][y0][cIdx] equal to 1 specifies that no transform is applied to the associated transform block. transform_skip_flag[x0][y0][cIdx] equal to 0 specifies that the decision whether transform is applied to the associated transform block or not depends on other syntax elements.

When transform_skip_flag[x0][y0][cIdx] is not present, it is inferred as follows:

– If BdpcmFlag[x0][y0][cIdx] is equal to 1, transform_skip_flag[x0][y0][cIdx] is inferred to be equal to 1.

– Otherwise (BdpcmFlag[x0][y0][cIdx] is equal to 0), transform_skip_flag[x0][y0][cIdx] is inferred to be equal to 0.

7.4.12.11 Residual coding semantics

The array AbsLevel[xC][yC] represents an array of absolute values of transform coefficient levels for the current transform block and the array AbsLevelPass1[xC][yC] represents an array of partially reconstructed absolute values of transform coefficient levels for the current transform block. The array indices xC and yC specify the transform coefficient location (xC, yC) within the current transform block. When the value of AbsLevel[xC][yC] is not specified in clause 7.3.11.11, it is inferred to be equal to 0. When the value of AbsLevelPass1[xC][yC] is not specified in clause 7.3.11.11, it is inferred to be equal to 0.

The variables CoeffMin and CoeffMax specifying the minimum and maximum transform coefficient values are derived as follows:

$$\text{CoeffMin} = -(\ 1\ <<\ (\ \text{sps\_extended\_precision\_flag}\ ?\ \text{Max}(\ 15, \text{Min}(\ 20, \text{BitDepth} + 6\ )\ ) : 15\ )\ ) \tag{190}$$

$$\text{CoeffMax} = (\ 1\ <<\ (\ \text{sps\_extended\_precision\_flag}\ ?\ \text{Max}(\ 15, \text{Min}(\ 20, \text{BitDepth} + 6\ )\ ) : 15\ )\ ) - 1 \tag{191}$$

The array QStateTransTable[ ][ ] is specified as follows:

$$\text{QStateTransTable}[\ ][\ ] = \{\ \{\ 0, 2\ \}, \{\ 2, 0\ \}, \{\ 1, 3\ \}, \{\ 3, 1\ \}\ \} \tag{192}$$

last_sig_coeff_x_prefix specifies the prefix of the column position of the last significant coefficient in scanning order within a transform block. The values of last_sig_coeff_x_prefix shall be in the range of 0 to (Log2ZoTbWidth << 1) − 1, inclusive.

When last_sig_coeff_x_prefix is not present, it is inferred to be 0.

last_sig_coeff_y_prefix specifies the prefix of the row position of the last significant coefficient in scanning order within a transform block. The values of last_sig_coeff_y_prefix shall be in the range of 0 to (Log2ZoTbHeight << 1) − 1, inclusive.

When last_sig_coeff_y_prefix is not present, it is inferred to be 0.

last_sig_coeff_x_suffix specifies the suffix of the column position of the last significant coefficient in scanning order within a transform block. The values of last_sig_coeff_x_suffix shall be in the range of 0 to (1 << ((last_sig_coeff_x_prefix >> 1) − 1)) − 1, inclusive.

The column position of the last significant coefficient in scanning order within a transform block LastSignificantCoeffX is derived as follows:

– If last_sig_coeff_x_suffix is not present, the following applies:

$$\text{LastSignificantCoeffX} = \text{last\_sig\_coeff\_x\_prefix} \tag{193}$$

– Otherwise (last_sig_coeff_x_suffix is present), the following applies:

$$\begin{aligned}\text{LastSignificantCoeffX} = (\ 1\ <<\ (\ (\text{last\_sig\_coeff\_x\_prefix}\ >>\ 1\ ) - 1\ )\ ) *\\ (\ 2 + (\text{last\_sig\_coeff\_x\_prefix}\ \&\ 1\ )\ ) + \text{last\_sig\_coeff\_x\_suffix}\end{aligned} \tag{194}$$

When sh_reverse_last_sig_coeff_flag is equal to 1, the value of LastSignificantCoeffX is modified as follows:

$$\text{LastSignificantCoeffX} = (1 \ll \text{Log2ZoTbWidth}) - 1 - \text{LastSignificantCoeffX} \tag{195}$$

last_sig_coeff_y_suffix specifies the suffix of the row position of the last significant coefficient in scanning order within a transform block. The values of last_sig_coeff_y_suffix shall be in the range of 0 to (1 << ((last_sig_coeff_y_prefix >> 1) − 1)) − 1, inclusive.

The row position of the last significant coefficient in scanning order within a transform block LastSignificantCoeffY is derived as follows:

– If last_sig_coeff_y_suffix is not present, the following applies:

$$\text{LastSignificantCoeffY} = \text{last\_sig\_coeff\_y\_prefix} \tag{196}$$

– Otherwise (last_sig_coeff_y_suffix is present), the following applies:

$$\begin{aligned}\text{LastSignificantCoeffY} = {} & (1 \ll ((\text{last\_sig\_coeff\_y\_prefix} \gg 1) - 1)) * \\ & (2 + (\text{last\_sig\_coeff\_y\_prefix} \,\&\, 1)) + \text{last\_sig\_coeff\_y\_suffix}\end{aligned} \tag{197}$$

When sh_reverse_last_sig_coeff_flag is equal to 1, the value of LastSignificantCoeffY is modified as follows:

$$\text{LastSignificantCoeffY} = (1 \ll \text{Log2ZoTbHeight}) - 1 - \text{LastSignificantCoeffY} \tag{198}$$

sb_coded_flag[xS][yS] specifies the following for the subblock at location (xS, yS) within the current transform block, where a subblock is an array of numSbCoeff transform coefficient levels:

– If transform_skip_flag[x0][y0][cIdx] is equal to 0 or sh_ts_residual_coding_disabled_flag is equal to 1, the following applies:
  – When sb_coded_flag[xS][yS] is not present, it is inferred as follows:
    – If one or more of the following conditions are true, sb_coded_flag[xS][yS] is inferred to be equal to 1:
      – (xS, yS) is equal to (0, 0).
      – (xS, yS) is equal to (LastSignificantCoeffX >> log2SbW, LastSignificantCoeffY >> log2SbH).
    – Otherwise, sb_coded_flag[xS][yS] is inferred to be equal to 0.
  – If sb_coded_flag[xS][yS] is equal to 0, all transform coefficient levels of the subblock at location (xS, yS) are inferred to be equal to 0.
  – Otherwise (sb_coded_flag[xS][yS] is equal to 1), the following applies:
    – If (xS, yS) is equal to (0, 0) and (LastSignificantCoeffX, LastSignificantCoeffY) is not equal to (0, 0), at least one of the sig_coeff_flag syntax elements is present for the subblock at location (xS, yS).
    – Otherwise, at least one of the transform coefficient levels of the subblock at location (xS, yS) has a non-zero value.
– Otherwise (transform_skip_flag[x0][y0][cIdx] is equal to 1 and sh_ts_residual_coding_disabled_flag is equal to 0), the following applies:
  – When sb_coded_flag[xS][yS] is not present, it is inferred to be equal to 1.
  – If sb_coded_flag[xS][yS] is equal to 0, all transform coefficient levels of the subblock at location (xS, yS) are inferred to be equal to 0.
  – Otherwise (sb_coded_flag[xS][yS] is equal to 1), at least one of the transform coefficient levels of the subblock at location (xS, yS) has a non-zero value.

sig_coeff_flag[xC][yC] specifies for the transform coefficient location (xC, yC) within the current transform block whether the corresponding transform coefficient level at the location (xC, yC) is non-zero as follows:

– If sig_coeff_flag[xC][yC] is equal to 0, the transform coefficient level at the location (xC, yC) is set equal to 0.
– Otherwise (sig_coeff_flag[xC][yC] is equal to 1), the transform coefficient level at the location (xC, yC) has a non-zero value.

When sig_coeff_flag[xC][yC] is not present, it is inferred as follows:

– If transform_skip_flag[x0][y0][cIdx] is equal to 0 or sh_ts_residual_coding_disabled_flag is equal to 1, the following applies:

  – If (xC, yC) is the last significant location (LastSignificantCoeffX, LastSignificantCoeffY) in scan order or all of the following conditions are true, sig_coeff_flag[xC][yC] is inferred to be equal to 1:

    – (xC & ((1 << log2SbW) − 1), yC & ((1 << log2SbH) − 1)) is equal to (0, 0).

    – inferSbDcSigCoeffFlag is equal to 1.

    – sb_coded_flag[xS][yS] is equal to 1.

  – Otherwise, sig_coeff_flag[xC][yC] is inferred to be equal to 0.

– Otherwise (transform_skip_flag[x0][y0][cIdx] is equal to 1 and sh_ts_residual_coding_disabled_flag is equal to 0), the following applies:

  – If all of the following conditions are true, sig_coeff_flag[xC][yC] is inferred to be equal to 1:

    – (xC & ((1 << log2SbW) − 1), yC & ((1 << log2SbH) − 1)) is equal to
    ((1 << log2SbW) − 1, (1 << log2SbH) − 1).

    – inferSbSigCoeffFlag is equal to 1.

    – sb_coded_flag[xS][yS] is equal to 1.

  – Otherwise, sig_coeff_flag[xC][yC] is inferred to be equal to 0.

abs_level_gtx_flag[n][j] specifies whether the absolute value of the transform coefficient level (at scanning position n) is greater than (j << 1) + 1. When abs_level_gtx_flag[n][j] is not present, it is inferred to be equal to 0.

par_level_flag[n] specifies the parity of the transform coefficient level at scanning position n. When par_level_flag[n] is not present, it is inferred to be equal to 0.

abs_remainder[n] is the remaining absolute value of a transform coefficient level that is coded with Golomb-Rice code at the scanning position n. When abs_remainder[n] is not present, it is inferred to be equal to 0.

It is a requirement of bitstream conformance that the value of abs_remainder[n] shall be constrained such that the corresponding value of TransCoeffLevel[x0][y0][cIdx][xC][yC] is in the range of CoeffMin to CoeffMax, inclusive.

dec_abs_level[n] is an intermediate value that is coded with Golomb-Rice code at the scanning position n. Given ZeroPos[n] that is derived in clause 9.3.3.2 during the parsing of dec_abs_level[n], the absolute value of a transform coefficient level at location (xC, yC) AbsLevel[xC][yC] is derived as follows:

– If dec_abs_level[n] is not present or equal to ZeroPos[n], AbsLevel[xC][yC] is set equal to 0.

– Otherwise, if dec_abs_level[n] is less than ZeroPos[n], AbsLevel[xC][yC] is set equal to dec_abs_level[n] + 1;

– Otherwise (dec_abs_level[n] is greater than ZeroPos[n]), AbsLevel[xC][yC] is set equal to dec_abs_level[n].

It is a requirement of bitstream conformance that the value of dec_abs_level[n] shall be constrained such that the corresponding value of TransCoeffLevel[x0][y0][cIdx][xC][yC] is in the range of CoeffMin to CoeffMax, inclusive.

coeff_sign_flag[n] specifies the sign of a transform coefficient level for the scanning position n as follows:

– If coeff_sign_flag[n] is equal to 0, the corresponding transform coefficient level has a positive value.

– Otherwise (coeff_sign_flag[n] is equal to 1), the corresponding transform coefficient level has a negative value.

When coeff_sign_flag[n] is not present, it is inferred to be equal to 0.

The value of CoeffSignLevel[xC][yC] specifies the sign of a transform coefficient level at the location (xC, yC) as follows:

– If CoeffSignLevel[xC][yC] is equal to 0, the corresponding transform coefficient level is equal to zero

– Otherwise, if CoeffSignLevel[xC][yC] is equal to 1, the corresponding transform coefficient level has a positive value.

– Otherwise (CoeffSignLevel[xC][yC] is equal to −1), the corresponding transform coefficient level has a negative value.

8 Decoding process

8.1 General decoding process

8.1.1 General

Input to this process is a bitstream BitstreamToDecode. Output of this process is a list of decoded pictures.

The decoding process is specified such that all decoders that conform to a specified profile and level will produce numerically identical cropped decoded output pictures when invoking the decoding process associated with that profile for a bitstream conforming to that profile and level. Any decoding process that produces identical cropped decoded output pictures to those produced by the process described herein (with the correct output order or output timing, as specified) conforms to the decoding process requirements of this Specification.

For each IRAP picture in the bitstream, the following applies:

- If the picture is the first picture of a layer in the bitstream in decoding order, an IDR picture, or the first picture of a layer that follows an EOS NAL unit of the layer in decoding order, the variable NoOutputBeforeRecoveryFlag for the picture is set equal to 1.
- Otherwise, when the picture is a CRA picture, the following applies:
  - If some external means not specified in this Specification is available to set the variable HandleCraAsClvsStartFlag for the picture to a value, HandleCraAsClvsStartFlag for the picture is set equal to the value provided by the external means and NoOutputBeforeRecoveryFlag is set equal to HandleCraAsClvsStartFlag.
  - Otherwise, HandleCraAsClvsStartFlag and NoOutputBeforeRecoveryFlag are both set equal to 0 for the picture.

For each GDR picture in the bitstream, the following applies:

- If the picture is the first picture of a layer in the bitstream in decoding order or the first picture of a layer that follows an EOS NAL unit of the layer in decoding order, the variable NoOutputBeforeRecoveryFlag for the picture is set equal to 1.
- Otherwise, if some external means not specified in this Specification is available to set the variable HandleGdrAsClvsStartFlag for the picture to a value, HandleGdrAsClvsStartFlag is set equal to the value provided by the external means and NoOutputBeforeRecoveryFlag is set equal to HandleGdrAsClvsStartFlag for the picture.
- Otherwise, HandleGdrAsClvsStartFlag and NoOutputBeforeRecoveryFlag for the picture are both set equal to 0.

The NoOutputBeforeRecoveryFlag of an IRAP or GDR picture is also referred to as the NoOutputBeforeRecoveryFlag of the PU containing the IRAP or GDR picture.

> NOTE 1 – The operations specified above in this clause, for both IRAP pictures and GDR pictures, enable identification of the CLVSS pictures and CLVSs of each layer and consequently enable identification of the CVSS AUs and the boundaries of the CVSs in the bitstream.

The variables TargetOlsIdx, which identifies the OLS index of the target OLS to be decoded, is derived as follows:

- The following applies in the first AU of the bitstream:
  - If some external means not specified in this Specification are available for setting TargetOlsIdx, TargetOlsIdx is set by the external means.
  - Otherwise, if opi_ols_idx is present in an OPI NAL unit in the first AU of the bitstream, the variable TargetOlsIdx is set equal to opi_ols_idx.
  - Otherwise, the variable TargetOlsIdx is set to be equal to the lowest OLS index that contains the largest number of layers among all OLSs specified by the VPS and the largest number of output layers among the OLSs with the largest number of layers.

    > NOTE 2 – When sps_video_parameter_set_id is equal to 0, the phrase "layers specified by the VPS" refers to the only present layer, and the value of TargetOlsIdx would be equal to 0.
- The following applies in the first AU of each CVS that is not the first AU of the bitstream:
  - If some external means not specified in this Specification are available for setting TargetOlsIdx, TargetOlsIdx is set by the external means.

– Otherwise, if opi_ols_idx is present in an OPI NAL unit in the first AU of the CVS, the variable TargetOlsIdx is set equal to opi_ols_idx.

The variable Htid, which identifies the highest temporal sublayer to be decoded, is derived as follows:

– The following applies in the first AU of the bitstream:

  – If some external means not specified in this Specification are available for setting Htid, Htid is set by the external means.

  – Otherwise, if opi_htid_plus1 is present in an OPI NAL unit in the first AU of the bitstream, Htid is set equal to ((opi_htid_plus1 > 0) ? opi_htid_plus1 − 1 : 0).

  – Otherwise, Htid is set equal to vps_ptl_max_tid[vps_ols_ptl_idx[TargetOlsIdx]].

    NOTE 3 – When sps_video_parameter_set_id is equal to 0, Htid would be set equal to sps_max_sublayers_minus1.

– The following applies in the first AU of each CVS that is not the first AU of the bitstream:

  – If some external means not specified in this Specification are available for setting Htid, Htid is set by the external means.

  – Otherwise, if opi_htid_plus1 is present in an OPI NAL unit in the first AU of the CVS, Htid is set equal to ((opi_htid_plus1 > 0) ? opi_htid_plus1 − 1 : 0).

Each CVS in the bitstream BitstreamToDecode does not contain any other layers than those included in the target OLS indicated by the TargetOlsIdx value applying to that CVS and does not include any NAL unit with TemporalId greater than Htid applying to that CVS.

The variable DuHrdPreferredFlag is either specified by external means, or when not specified by external means, set equal to 0.

The variable DecodingUnitHrdFlag is specified as follows:

– If the decoding process is invoked in a bitstream conformance test as specified in clause C.1, DecodingUnitHrdFlag is set as specified in clause C.1.

– Otherwise, DecodingUnitHrdFlag is set equal to (DuHrdPreferredFlag && general_du_hrd_params_present_flag), where general_du_hrd_params_present_flag is found in the general_timing_hrd_parameters( ) syntax structure that applies to the target OLS identified by TargetOlsIdx.

Clause 8.1.2 is repeatedly invoked for each coded picture in BitstreamToDecode in decoding order.

8.1.2 Decoding process for a coded picture

The decoding processes specified in this clause apply to each coded picture, referred to as the current picture, in BitstreamToDecode.

Depending on the value of sps_chroma_format_idc, the number of sample arrays of the current picture is as follows:

– If sps_chroma_format_idc is equal to 0, the current picture consists of 1 sample array S_L.

– Otherwise (sps_chroma_format_idc is not equal to 0), the current picture consists of 3 sample arrays S_L, S_{Cb}, S_{Cr}.

The decoding process for the current picture takes as inputs the syntax elements and upper-case variables from clause 7. When interpreting the semantics of each syntax element in each NAL unit, and in the remaining parts of clause 8, the term "the bitstream" (or part thereof, e.g., a CVS of the bitstream) refers to BitstreamToDecode (or part thereof).

The decoding process operates as follows for the current picture:

1. The decoding of NAL units is specified in clause 8.2.

2. The processes in clause 8.3 specify the following decoding processes using syntax elements in the slice header layer and above:

   – Variables and functions relating to picture order count are derived as specified in clause 8.3.1. This needs to be invoked only for the first slice of a picture.

   – At the beginning of the decoding process for each slice of a picture, the decoding process for RPLs construction specified in clause 8.3.2 is invoked for derivation of RPL 0 (RefPicList[0]) and RPL 1 (RefPicList[1]).

- The decoding process for reference picture marking in clause 8.3.3 is invoked, wherein reference pictures might be marked as "unused for reference" or "used for long-term reference". This needs to be invoked only for the first slice of a picture.
- When the current picture is an IDR picture with sps_idr_rpl_present_flag equal to 1 or pps_rpl_info_in_ph_flag equal to 1, a CRA picture with NoOutputBeforeRecoveryFlag equal to 1, or a GDR picture with NoOutputBeforeRecoveryFlag equal to 1, the decoding process for generating unavailable reference pictures specified in clause 8.3.4 is invoked, which needs to be invoked only for the first slice of a picture.
- At the beginning of the decoding process for each B slice, the decoding process for symmetric motion vector difference reference indices specified in clause 8.3.5 is invoked for derivation of the variables RefIdxSymL0 and RefIdxSymL1.
- At the beginning of the decoding process for each P or B slice, the decoding process for collocated picture and no backward prediction flag specified in clause 8.3.6 is invoked for derivation of the variables ColPic and NoBackwardPredFlag.

3. The processes in clauses 8.4, 8.5, 8.6, 8.7, and 8.8 specify decoding processes using syntax elements in all syntax structure layers. When any NAL units are present in a PU that follow the last VCL NAL unit of the picture in decoding order, their decoding is deferred until after all slices of the current picture have been decoded and the in-loop filter process of clause 8.8 has been applied. It is a requirement of bitstream conformance that the coded slices of the picture shall contain slice data for every CTU of the picture, such that the division of the picture into slices, and the division of the slices into CTUs each forms a partitioning of the picture.
4. After all slices of the current picture have been decoded, the in-loop filter process of clause 8.8 has been applied, and all remaining NAL units of the PU have been decoded, the current decoded picture is marked as "used for short-term reference", the picture referred to by each ILRP entry, when present, in RefPicList[0] or RefPicList[1] is marked as "used for short-term reference", and the variable PictureOutputFlag of the current picture is derived as follows:
   - If sps_video_parameter_set_id is greater than 0 and the current layer is not an output layer (i.e., nuh_layer_id is not equal to OutputLayerIdInOls[TargetOlsIdx][i] for any value of i in the range of 0 to NumOutputLayersInOls[TargetOlsIdx] − 1, inclusive), or one of the following conditions is true, PictureOutputFlag is set equal to 0:
     - The current picture is a RASL picture and NoOutputBeforeRecoveryFlag of the associated IRAP picture is equal to 1.
     - The current picture is a GDR picture with NoOutputBeforeRecoveryFlag equal to 1 or is a recovering picture of a GDR picture with NoOutputBeforeRecoveryFlag equal to 1.
   - Otherwise, PictureOutputFlag is set equal to ph_pic_output_flag.

     NOTE – In an implementation, the decoder could output a picture not belonging to an output layer. For example, when there is only one output layer while in an AU the picture of the output layer is not available, e.g., due to a loss or layer down-switching, the decoder could set PictureOutputFlag set equal to 1 for the picture that has the highest value of nuh_layer_id among all pictures of the AU available to the decoder and having ph_pic_output_flag equal to 1, and set PictureOutputFlag equal to 0 for all other pictures of the AU available to the decoder.

8.2 NAL unit decoding process

Inputs to this process are NAL units of the current picture and their associated non-VCL NAL units.

Outputs of this process are the parsed 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 parses the RBSP syntax structure.

8.3 Slice decoding process

8.3.1 Decoding process for picture order count

Output of this process is PicOrderCntVal, the picture order count of the current picture.

Each coded picture is associated with a picture order count variable, denoted as PicOrderCntVal.

Let the variable currLayerIdx be set equal to GeneralLayerIdx[nuh_layer_id].

PicOrderCntVal is derived as follows:

– If vps_independent_layer_flag[currLayerIdx] is equal to 0 and there is a picture picA in the current AU with nuh_layer_id equal to layerIdA such that GeneralLayerIdx[layerIdA] is in the list ReferenceLayerIdx[currLayerIdx], PicOrderCntVal is derived to be equal to the PicOrderCntVal of picA, and the value of ph_pic_order_cnt_lsb shall be the same in all VCL NAL units of the current AU.

– Otherwise, PicOrderCntVal of the current picture is derived as specified in the remainder of this clause.

When ph_poc_msb_cycle_val is not present and the current picture is not a CLVSS picture, the variables prevPicOrderCntLsb and prevPicOrderCntMsb are derived as follows:

– Let prevTid0Pic be the previous picture in decoding order that has nuh_layer_id equal to the nuh_layer_id of the current picture, has TemporalId and ph_non_ref_pic_flag both equal to 0, and is not a RASL or RADL picture.

NOTE 1 – In a sub-bitstream consisting of only intra pictures, extracted from a single-layer bitstream and used in intra-picture-only trick play, the prevTid0Pic is the previous intra picture in decoding order that has TemporalId equal to 0. To ensure correct POC derivation, encoders could choose either to include ph_poc_msb_cycle_val for each intra picture, or set the value of sps_log2_max_pic_order_cnt_lsb_minus4 to be large enough such that the POC difference between the current picture and the prevTid0Pic is less than MaxPicOrderCntLsb / 2. However, on the other hand, the bitstream conformance constraint specified by item 8 in clause C.4 disallows the value of POC difference between the current picture and the prevTid0Pic to be greater than or equal to MaxPicOrderCntLsb / 2. Consequently, encoders could either leave such a sub-bitstream to be a non-conforming bitstream and expect decoders capable of intra-only trick play decoding to be able to appropriately decode such sub-bitstreams, or set the value of sps_log2_max_pic_order_cnt_lsb_minus4 to be large enough such that the POC difference between the current picture and the prevTid0Pic is less than MaxPicOrderCntLsb / 2.

NOTE 2 – When vps_max_tid_il_ref_pics_plus1[i][j] is equal to 0 for any value of i among the layer indices and j equal to the layer index of the current layer, the prevTid0Pic in the extracted sub-bitstream for some of the OLSs would be the previous IRAP or GDR picture with ph_recovery_poc_cnt equal to 0 in the current layer in decoding order. To ensure such a sub-bitstream to be a conforming bitstream, which is required by the general sub-bitstream extraction process specified in clause C.6, encoders are expected to set the value of sps_log2_max_pic_order_cnt_lsb_minus4 to be large enough such that the POC difference between the current picture and the prevTid0Pic is less than MaxPicOrderCntLsb / 2. Due to that the bitstream conformance constraint specified by item 8 in clause C.4 disallows the value of POC difference between the current picture and the prevTid0Pic to be greater than or equal to MaxPicOrderCntLsb / 2, the other option that ensures correct POC derivation by including ph_poc_msb_cycle_val for each IRAP picture and each GDR picture with ph_recovery_poc_cnt equal to 0 is precluded.

– The variable prevPicOrderCntLsb is set equal to ph_pic_order_cnt_lsb of prevTid0Pic.

– The variable prevPicOrderCntMsb is set equal to PicOrderCntMsb of prevTid0Pic.

The variable PicOrderCntMsb of the current picture is derived as follows:

– If ph_poc_msb_cycle_val is present, PicOrderCntMsb is set equal to ph_poc_msb_cycle_val * MaxPicOrderCntLsb.

– Otherwise (ph_poc_msb_cycle_val is not present), if the current picture is a CLVSS picture, PicOrderCntMsb is set equal to 0.

– Otherwise, PicOrderCntMsb is derived as follows:

```
if( ( ph_pic_order_cnt_lsb < prevPicOrderCntLsb )  &&
        ( ( prevPicOrderCntLsb − ph_pic_order_cnt_lsb )  >=  ( MaxPicOrderCntLsb / 2 ) ) )
    PicOrderCntMsb = prevPicOrderCntMsb + MaxPicOrderCntLsb                          (199)
else if( ( ph_pic_order_cnt_lsb > prevPicOrderCntLsb )  &&
        ( ( ph_pic_order_cnt_lsb − prevPicOrderCntLsb ) > ( MaxPicOrderCntLsb / 2 ) ) )
    PicOrderCntMsb = prevPicOrderCntMsb − MaxPicOrderCntLsb
else
    PicOrderCntMsb = prevPicOrderCntMsb
```

PicOrderCntVal is derived as follows:

```
PicOrderCntVal = PicOrderCntMsb + ph_pic_order_cnt_lsb                               (200)
```

NOTE 3 – All CLVSS pictures for which ph_poc_msb_cycle_val is not present have PicOrderCntVal equal to ph_pic_order_cnt_lsb since for those pictures PicOrderCntMsb is set equal to 0.

The value of PicOrderCntVal shall be in the range of -2^{31} to $2^{31} - 1$, inclusive.

In one CVS, the PicOrderCntVal values for any two coded pictures with the same value of nuh_layer_id shall not be the same.

All pictures in any particular AU shall have the same value of PicOrderCntVal.

The function PicOrderCnt(picX) is specified as follows:

$$\text{PicOrderCnt(picX) = PicOrderCntVal of the picture picX} \tag{201}$$

The function DiffPicOrderCnt(picA, picB) is specified as follows:

$$\text{DiffPicOrderCnt(picA, picB) = PicOrderCnt(picA)} - \text{PicOrderCnt(picB)} \tag{202}$$

The bitstream shall not contain data that result in values of DiffPicOrderCnt(picA, picB) used in the decoding process that are not in the range of -2^{15} to $2^{15} - 1$, inclusive.

NOTE 4 – Let X be the current picture and Y and Z be two other pictures in the same CVS, 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.

8.3.2 Decoding process for reference picture lists construction

This process is invoked at the beginning of the decoding process for each slice of a picture.

Reference pictures are addressed through reference indices. A reference index is an index into an RPL. When decoding an I slice, no RPL is used in decoding of the slice data. When decoding a P slice, only RPL 0 (i.e., RefPicList[0]), is used in decoding of the slice data. When decoding a B slice, both RPL 0 and RPL 1 (i.e., RefPicList[1]) are used in decoding of the slice data.

At the beginning of the decoding process for each slice of a picture, the RPLs RefPicList[0] and RefPicList[1] are derived. The RPLs are used in marking of reference pictures as specified in clause 8.3.3 or in decoding of the slice data.

NOTE 1 – For an I slice of a picture, RefPicList[0] and RefPicList[1] could be derived for bitstream conformance checking purpose, but their derivation is not necessary for decoding of the current picture or pictures following the current picture in decoding order. For a P slice of a picture, RefPicList[1] could be derived for bitstream conformance checking purpose, but its derivation is not necessary for decoding of the current picture or pictures following the current picture in decoding order.

If sps_idr_rpl_present_flag is equal to 0, pps_rpl_info_in_ph_flag is equal to 0, and nal_unit_type is equal to IDR_W_RADL or IDR_N_LP, the RPLs RefPicList[0] and RefPicList[1] are both derived to be empty, i.e., to contain 0 entries, and the following applies for each i equal to 0 or 1:

– The value of RplsIdx[i] is inferred to be equal to sps_num_ref_pic_lists[i].

– The value of num_ref_entries[i][RplsIdx[i]] is inferred to be equal to 0.

– The value of NumRefIdxActive[i] is inferred to be equal to 0.

Otherwise, the RPLs RefPicList[0] and RefPicList[1], the reference picture scaling ratios RefPicScale[i][j][0] and RefPicScale[i][j][1], and the reference picture scaled flags RprConstraintsActiveFlag[0][j] and RprConstraintsActiveFlag[1][j] are derived as follows:

```
for( i = 0; i < 2; i++ ) {
    for( j = 0, k = 0, pocBase = PicOrderCntVal; j < num_ref_entries[ i ][ RplsIdx[ i ] ]; j++) {
        if( !inter_layer_ref_pic_flag[ i ][ RplsIdx[ i ] ][ j ] ) {
            if( st_ref_pic_flag[ i ][ RplsIdx[ i ] ][ j ] ) {
                RefPicPocList[ i ][ j ] = pocBase + DeltaPocValSt[ i ][ RplsIdx[ i ] ][ j ]
                if( there is a reference picture picA in the DPB with the same nuh_layer_id as the current picture
                        and PicOrderCntVal equal to RefPicPocList[ i ][ j ] )
                    RefPicList[ i ][ j ] = picA
                else
                    RefPicList[ i ][ j ] = "no reference picture"                                        (203)
                pocBase = RefPicPocList[ i ][ j ]
            } else {
                if( !delta_poc_msb_cycle_present_flag[ i ][ k ] ) {
                    if( there is a reference picA in the DPB with the same nuh_layer_id as the current picture and
                            PicOrderCntVal & ( MaxPicOrderCntLsb − 1 ) equal to PocLsbLt[ i ][ k ] )
                        RefPicList[ i ][ j ] = picA
                    else
                        RefPicList[ i ][ j ] = "no reference picture"
                    RefPicLtPocList[ i ][ j ] = PocLsbLt[ i ][ k ]
                } else {
                    if( there is a reference picA in the DPB with the same nuh_layer_id as the current picture and
                            PicOrderCntVal equal to FullPocLt[ i ][ k ] )
                        RefPicList[ i ][ j ] = picA
```

```
                    else
                        RefPicList[ i ][ j ] = "no reference picture"
                    RefPicLtPocList[ i ][ j ] = FullPocLt[ i ][ k ]
                }
                k++
            }
        } else {
            layerIdx = DirectRefLayerIdx[ GeneralLayerIdx[ nuh_layer_id ] ][ ilrp_idx[ i ][ RplsIdx[ i ] ][ j ] ]
            refPicLayerId = vps_layer_id[ layerIdx ]
            if( there is a reference picture picA in the DPB with nuh_layer_id equal to refPicLayerId and
                    the same PicOrderCntVal as the current picture )
                RefPicList[ i ][ j ] = picA
            else
                RefPicList[ i ][ j ] = "no reference picture"
        }
        fRefWidth is set equal to CurrPicScalWinWidthL of the reference picture RefPicList[ i ][ j ]
        fRefHeight is set equal to CurrPicScalWinHeightL of the reference picture RefPicList[ i ][ j ]

        refPicWidth, refPicHeight, refScalingWinLeftOffset, refScalingWinRightOffset,
    refScalingWinTopOffset,
                and refScalingWinBottomOffset, are set equal to the values of pps_pic_width_in_luma_samples,
                pps_pic_height_in_luma_samples, pps_scaling_win_left_offset, pps_scaling_win_right_offset,
                pps_scaling_win_top_offset, and pps_scaling_win_bottom_offset, respectively, of the reference
                picture RefPicList[ i ][ j ]
        fRefNumSubpics is set equal to sps_num_subpics_minus1 of the reference picture RefPicList[ i ][ j ]

        RefPicScale[ i ][ j ][ 0 ] = ( ( fRefWidth  <<  14 ) + ( CurrPicScalWinWidthL  >>  1 ) ) /
                CurrPicScalWinWidthL
        RefPicScale[ i ][ j ][ 1 ] = ( ( fRefHeight  <<  14 ) + ( CurrPicScalWinHeightL  >>  1 ) ) /
                CurrPicScalWinHeightL
        RprConstraintsActiveFlag[ i ][ j ] = ( pps_pic_width_in_luma_samples  !=  refPicWidth  | |
                pps_pic_height_in_luma_samples  !=  refPicHeight  | |
                pps_scaling_win_left_offset  !=  refScalingWinLeftOffset  | |
                pps_scaling_win_right_offset  !=  refScalingWinRightOffset  | |
                pps_scaling_win_top_offset  !=  refScalingWinTopOffset  | |
                pps_scaling_win_bottom_offset  !=  refScalingWinBottomOffset  | |
                sps_num_subpics_minus1  !=  fRefNumSubpics )
    }
}
```

For each i equal to 0 or 1, the first NumRefIdxActive[i] entries in RefPicList[i] are referred to as the active entries in RefPicList[i], and the other entries in RefPicList[i] are referred to as the inactive entries in RefPicList[i].

NOTE 2 – It is possible that a particular picture is referred to by both an entry in RefPicList[0] and an entry in RefPicList[1]. It is also possible that a particular picture is referred to by more than one entry in RefPicList[0] or by more than one entry in RefPicList[1].

NOTE 3 – The active entries in RefPicList[0] and the active entries in RefPicList[1] collectively refer to all reference pictures that could be used for inter prediction of the current picture and one or more pictures in the same layer that follow the current picture in decoding order. The inactive entries in RefPicList[0] and the inactive entries in RefPicList[1] collectively refer to all reference pictures that are *not* used for inter prediction of the current picture but could be used in inter prediction for one or more pictures in the same layer that follow the current picture in decoding order.

NOTE 4 – There could be one or more entries in RefPicList[0] or RefPicList[1] that are equal to "no reference picture" because the corresponding pictures are not present in the DPB. Each inactive entry in RefPicList[0] or RefPicList[1] that is equal to "no reference picture" is expected to be ignored. An unintentional picture loss is expected to be inferred for each active entry in RefPicList[0] or RefPicList[1] that is equal to "no reference picture".

It is a requirement of bitstream conformance that the following constraints apply:

NOTE 5 – When the bitstream conformance is checked for some of these constraints, the conformance check is expected to be performed after invoking the decoding process for generating unavailable reference pictures specified in clause 8.3.4.

– For each i equal to 0 or 1, num_ref_entries[i][RplsIdx[i]] shall not be less than NumRefIdxActive[i].

– The picture referred to by each active entry in RefPicList[0] or RefPicList[1] shall be present in the DPB and shall have TemporalId less than or equal to that of the current picture.

– The picture referred to by each entry in RefPicList[0] or RefPicList[1] shall not be the current picture and shall have ph_non_ref_pic_flag equal to 0.

– An STRP entry in RefPicList[0] or RefPicList[1] of a slice of a picture and an LTRP entry in RefPicList[0] or RefPicList[1] of the same slice or a different slice of the same picture shall not refer to the same picture.

– There shall be no LTRP entry in RefPicList[0] or RefPicList[1] for which the difference between the PicOrderCntVal of the current picture and the PicOrderCntVal of the picture referred to by the entry is greater than or equal to 2^{24}.

– Let setOfRefPics be the set of unique pictures referred to by all entries in RefPicList[0] that have the same nuh_layer_id as the current picture and all entries in RefPicList[1] that have the same nuh_layer_id as the current picture. The number of pictures in setOfRefPics shall be less than or equal to MaxDpbSize − 1, inclusive, where MaxDpbSize is as specified in clause A.4.2, and setOfRefPics shall be the same for all slices of a picture.

– When the current slice has nal_unit_type equal to STSA_NUT, there shall be no active entry in RefPicList[0] or RefPicList[1] that has TemporalId equal to that of the current picture and nuh_layer_id equal to that of the current picture.

– When the current picture is a picture that follows, in decoding order, an STSA picture that has TemporalId equal to that of the current picture and nuh_layer_id equal to that of the current picture, there shall be no picture that precedes the STSA picture in decoding order, has TemporalId equal to that of the current picture, and has nuh_layer_id equal to that of the current picture included as an active entry in RefPicList[0] or RefPicList[1].

– When the current subpicture, with TemporalId equal to a particular value tId, nuh_layer_id equal to a particular value layerId, and subpicture index equal to a particular value subpicIdx, is a subpicture that follows, in decoding order, an STSA subpicture with TemporalId equal to tId, nuh_layer_id equal to layerId, and subpicture index equal to subpicIdx, there shall be no picture with TemporalId equal to tId and nuh_layer_id equal to layerId that precedes the picture containing the STSA subpicture in decoding order included as an active entry in RefPicList[0] or RefPicList[1].

– When the current picture, with nuh_layer_id equal to a particular value layerId, is an IRAP picture, there shall be no picture referred to by an entry in RefPicList[0] or RefPicList[1] that precedes, in output order or decoding order, any preceding IRAP picture with nuh_layer_id equal to layerId in decoding order (when present).

– When the current subpicture, with nuh_layer_id equal to a particular value layerId and subpicture index equal to a particular value subpicIdx, is an IRAP subpicture, there shall be no picture referred to by an entry in RefPicList[0] or RefPicList[1] that precedes, in output order or decoding order, any preceding picture, in decoding order (when present), containing an IRAP subpicture with nuh_layer_id equal to layerId and subpicture index equal to subpicIdx.

– When the current picture is not a RASL picture associated with a CRA picture with NoOutputBeforeRecoveryFlag equal to 1, there shall be no picture referred to by an active entry in RefPicList[0] or RefPicList[1] that was generated by the decoding process for generating unavailable reference pictures for the CRA picture associated with the current picture.

– When the current subpicture is not a RASL subpicture associated with a CRA subpicture in a CRA picture with NoOutputBeforeRecoveryFlag equal to 1, there shall be no picture referred to by an active entry in RefPicList[0] or RefPicList[1] that was generated by the decoding process for generating unavailable reference pictures for the CRA picture containing the CRA subpicture associated with the current subpicture.

– When the current picture, with nuh_layer_id equal to a particular value layerId, is not any of the following, there shall be no picture referred to by an entry in RefPicList[0] or RefPicList[1] that was generated by the decoding process for generating unavailable reference pictures for the IRAP or GDR picture associated with the current picture:

  o An IDR picture with sps_idr_rpl_present_flag equal to 1 or pps_rpl_info_in_ph_flag equal to 1

  o A CRA picture with NoOutputBeforeRecoveryFlag equal to 1

  o When sps_field_seq_flag is equal to 1, a picture, associated with a CRA picture with NoOutputBeforeRecoveryFlag equal to 1, that precedes, in decoding order, all the leading pictures associated with the same CRA picture

  o A leading picture associated with a CRA picture with NoOutputBeforeRecoveryFlag equal to 1

  o A GDR picture with NoOutputBeforeRecoveryFlag equal to 1

- A recovering picture of a GDR picture with NoOutputBeforeRecoveryFlag equal to 1 and nuh_layer_id equal to layerId

– When the current subpicture, with nuh_layer_id equal to a particular value layerId and subpicture index equal to a particular value subpicIdx, is not any of the following, there shall be no picture referred to by an entry in RefPicList[0] or RefPicList[1] that was generated by the decoding process for generating unavailable reference pictures for the IRAP or GDR picture containing the IRAP or GDR subpicture associated with the current subpicture:
  - An IDR subpicture in an IDR picture with sps_idr_rpl_present_flag equal to 1 or pps_rpl_info_in_ph_flag equal to 1
  - A CRA subpicture in a CRA picture with NoOutputBeforeRecoveryFlag equal to 1
  - When sps_field_seq_flag is equal to 1, a subpicture, associated with a CRA subpicture in a CRA picture with NoOutputBeforeRecoveryFlag equal to 1, that precedes, in decoding order, all the leading pictures associated with the same CRA picture
  - A leading subpicture associated with a CRA subpicture in a CRA picture with NoOutputBeforeRecoveryFlag equal to 1
  - A GDR subpicture in a GDR picture with NoOutputBeforeRecoveryFlag equal to 1
  - A subpicture in a recovering picture of a GDR picture with NoOutputBeforeRecoveryFlag equal to 1 and nuh_layer_id equal to layerId

– When the current picture follows an IRAP picture having the same value of nuh_layer_id in both decoding order and output order, there shall be no picture referred to by an active entry in RefPicList[0] or RefPicList[1] that precedes that IRAP picture in output order or decoding order.

– When the current subpicture follows an IRAP subpicture having the same value of nuh_layer_id and the same value of subpicture index in both decoding and output order, there shall be no picture referred to by an active entry in RefPicList[0] or RefPicList[1] that precedes the picture containing that IRAP subpicture in output order or decoding order.

– When the current picture follows an IRAP picture having the same value of nuh_layer_id and all the leading pictures, if any, associated with that IRAP picture in both decoding order and output order, there shall be no picture referred to by an entry in RefPicList[0] or RefPicList[1] that precedes that IRAP picture in output order or decoding order.

– When the current subpicture follows an IRAP subpicture having the same value of nuh_layer_id and the same value of subpicture index and all the leading subpictures, if any, associated with that IRAP subpicture in both decoding and output order, there shall be no picture referred to by an entry in RefPicList[0] or RefPicList[1] that precedes the picture containing that IRAP subpicture in output order or decoding order.

– When the current picture is a RADL picture, there shall be no active entry in RefPicList[0] or RefPicList[1] that is any of the following:
  - A RASL picture with pps_mixed_nalu_types_in_pic_flag is equal to 0

    NOTE 6 – An active entry in RefPicList[0] or RefPicList[1] of the current picture that is a RADL picture could refer to a RASL picture with pps_mixed_nalu_types_in_pic_flag equal to 1 in the layer or a reference layer of the current layer. However, when decoding starts from the associated CRA picture, such a RADL picture could still be correctly decoded, because the RADL subpicture(s) in that referenced RASL picture would be correctly decoded, as the RADL picture would only refer to the RADL subpictures in the referenced RASL picture, as imposed by the next constraint that disallows RADL subpictures referring to a RASL subpicture.
  - A picture that precedes the associated IRAP picture of the RADL picture in decoding order

– When the current subpicture, with nuh_layer_id equal to a particular value layerId and subpicture index equal to a particular value subpicIdx, is a RADL subpicture, there shall be no active entry in RefPicList[0] or RefPicList[1] that is any of the following:
  - A picture containing a RASL subpicture with subpicture index equal to subpicIdx
  - A RASL picture for which the value of nuh_layer_id is not equal to layerId and the value of sps_num_subpics_minus1 is equal to 0
  - A picture that precedes the picture containing the associated IRAP subpicture of the RADL subpicture in decoding order

– The following constraints apply for the picture referred to by each ILRP entry, when present, in RefPicList[0] or RefPicList[1] of a slice of the current picture:
  - o The picture shall be in the same AU as the current picture.
  - o The picture shall be present in the DPB.
  - o The picture shall have nuh_layer_id refPicLayerId less than the nuh_layer_id of the current picture.
  - o Either of the following conditions shall apply:
    - o The picture is a GDR picture with ph_recovery_poc_cnt equal to 0 or an IRAP picture.
    - o The picture has TemporalId less than vps_max_tid_il_ref_pics_plus1[currLayerIdx][refLayerIdx], where currLayerIdx and refLayerIdx are equal to GeneralLayerIdx[nuh_layer_id] and GeneralLayerIdx[refpicLayerId], respectively.

– Each ILRP entry, when present, in RefPicList[0] or RefPicList[1] of a slice shall be an active entry.

– When sps_num_subpics_minus1 is greater than 0 and the current subpicture with subpicture index subpicIdx has sps_subpic_treated_as_pic_flag[subpicIdx] equal to 1, it is a requirement of bitstream conformance that both of the following conditions shall be true:
  - o Exactly one and not both of the following two conditions shall be true:
    - ▪ The picture referred to by each active entry in RefPicList[0] or RefPicList[1] and the current picture have the same value for each of the following:
      - • pps_pic_width_in_luma_samples
      - • pps_pic_height_in_luma_samples
      - • sps_num_subpics_minus1
      - • sps_log2_ctu_size_minus5
      - • sps_subpic_ctu_top_left_x[i], sps_subpic_ctu_top_left_y[i], sps_subpic_width_minus1[i], sps_subpic_height_minus1[i], and sps_subpic_treated_as_pic_flag[i], respectively, for each value of i in the range of 0 to sps_num_subpics_minus1, inclusive
      - • SubpicIdVal[subpicIdx]
    - ▪ The picture referred to by each active entry in RefPicList[0] or RefPicList[1] is an ILRP entry for which the value of sps_num_subpics_minus1 is equal to 0.
  - o The current picture and any picture that is in a different layer than the current layer and has the current picture in an active entry in RefPicList[0] or RefPicList[1] have the same value for each of the following:
    - ▪ pps_pic_width_in_luma_samples
    - ▪ pps_pic_height_in_luma_samples
    - ▪ sps_num_subpics_minus1
    - ▪ sps_log2_ctu_size_minus5
    - ▪ sps_subpic_ctu_top_left_x[i], sps_subpic_ctu_top_left_y[i], sps_subpic_width_minus1[i], sps_subpic_height_minus1[i], sps_subpic_treated_as_pic_flag[i], SubpicIdVal[i], respectively, for each value of i in the range of 0 to sps_num_subpics_minus1, inclusive

8.3.3 Decoding process for reference picture marking

This process is invoked once per picture, after decoding of a slice header and the decoding process for RPL construction for the slice as specified in clause 8.3.2, but prior to the decoding of the slice data. This process might result in one or more reference pictures in the DPB being marked as "unused for reference" or "used for long-term reference".

A decoded picture in the DPB can be marked as "unused for reference", "used for short-term reference" or "used for long-term reference", but only one among these three at any given moment during the operation of the decoding process. Assigning one of these markings to a picture implicitly removes another of these markings when applicable. When a 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).

STRPs and ILRPs are identified by their nuh_layer_id and PicOrderCntVal values. LTRPs are identified by their nuh_layer_id values and by the Log2(MaxPicOrderCntLsb) LSBs of their PicOrderCntVal values or their PicOrderCntVal values.

If the current picture is a CLVSS picture, all reference pictures currently in the DPB (if any) with the same nuh_layer_id as the current picture are marked as "unused for reference". Otherwise, the following applies:

- For each LTRP entry in RefPicList[0] or RefPicList[1], when the picture is marked as "used for short-term reference" and has the same nuh_layer_id as the current picture, the picture is marked as "used for long-term reference".
- Each reference picture with the same nuh_layer_id as the current picture in the DPB that is not referred to by any entry in RefPicList[0] or RefPicList[1] is marked as "unused for reference".

For each ILRP entry in RefPicList[0] or RefPicList[1], the picture is marked as "used for long-term reference".

8.3.4 Decoding process for generating unavailable reference pictures

8.3.4.1 General decoding process for generating unavailable reference pictures

This process is invoked once per coded picture when the current picture is an IDR picture with sps_idr_rpl_present_flag equal to 1 or pps_rpl_info_in_ph_flag equal to 1, a CRA picture with NoOutputBeforeRecoveryFlag equal to 1, or a GDR picture with NoOutputBeforeRecoveryFlag equal to 1.

When this process is invoked, the following applies:

- For each RefPicList[i][j], with i in the range of 0 to 1, inclusive, and j in the range of 0 to num_ref_entries[i][RplsIdx[i]] − 1, inclusive, that is equal to "no reference picture", a picture is generated as specified in clause 8.3.4.2 and the following applies:
  - The value of nuh_layer_id for the generated picture is set equal to nuh_layer_id of the current picture.
  - If st_ref_pic_flag[i][RplsIdx[i]][j] is equal to 1 and inter_layer_ref_pic_flag[i][RplsIdx[i]][j] is equal to 0, the value of PicOrderCntVal for the generated picture is set equal to RefPicPocList[i][j] and the generated picture is marked as "used for short-term reference".
  - Otherwise, when st_ref_pic_flag[i][RplsIdx[i]][j] is equal to 0 and inter_layer_ref_pic_flag[i][RplsIdx[i]][j] is equal to 0, the value of PicOrderCntVal for the generated picture is set equal to RefPicLtPocList[i][j], the value of ph_pic_order_cnt_lsb for the generated picture is inferred to be equal to (RefPicLtPocList[i][j] & (MaxPicOrderCntLsb – 1)), and the generated picture is marked as "used for long-term reference".
  - The value of PictureOutputFlag for the generated reference picture is set equal to 0.
  - RefPicList[i][j] is set to be the generated reference picture.
  - The value of TemporalId for the generated picture is set equal to TemporalId of the current picture.
  - The value of ph_non_ref_pic_flag for the generated picture is set equal to 0.
  - The value of ph_pic_parameter_set_id for the generated picture is set equal to ph_pic_parameter_set_id of the current picture.

8.3.4.2 Generation of one unavailable picture

When this process is invoked, an unavailable picture is generated as follows:

- The value of each element in the sample array S_L for the picture is set equal to 1 << (BitDepth − 1).
- When sps_chroma_format_idc is not equal to 0, the value of each element in the sample arrays S_{Cb} and S_{Cr} for the picture is set equal to 1 << (BitDepth − 1).
- The prediction mode CuPredMode[0][x][y] is set equal to MODE_INTRA for x ranging from 0 to pps_pic_width_in_luma_samples − 1, inclusive, and y ranging from 0 to pps_pic_height_in_luma_samples − 1, inclusive.

NOTE – The output of the recovery point picture of a GDR picture with NoOutputBeforeRecoveryFlag equal to 1 and the pictures following that recovery point picture in output order and decoding order is independent of the values set for the elements of S_L, S_{Cb}, S_{Cr} and CuPredMode[0][x][y].

8.3.5 Decoding process for symmetric motion vector difference reference indices

This process is invoked at the beginning of the decoding process for each B slice, after decoding of the slice header as well as the invocation of the decoding process for RPL construction for the slice as specified in clause 8.3.2, but prior to the parsing and decoding of any coding unit.

Outputs of this process are RefIdxSymL0 and RefIdxSymL1 specifying the list 0 and list 1 reference picture indices for symmetric motion vector differences, i.e., when sym_mvd_flag is equal to 1 for a coding unit.

The variable RefIdxSymLX, with X = 0..1, is derived as follows:

- The variable currPic specifies the current picture.
- RefIdxSymL0 is set equal to −1.
- For each index i with i = 0..NumRefIdxActive[0] − 1, the following applies:
  - When all of the following conditions are true, RefIdxSymL0 is set equal to i:
    - RefPicList[0][i] is a short-term-reference picture,
    - DiffPicOrderCnt(currPic, RefPicList[0][i]) > 0,
    - DiffPicOrderCnt(currPic, RefPicList[0][i]) <
      DiffPicOrderCnt(currPic, RefPicList[0][RefIdxSymL0]) or RefIdxSymL0 is equal to −1.
- RefIdxSymL1 is set equal to −1.
- For each index i with i = 0..NumRefIdxActive[1] − 1, the following applies:
  - When all of the following conditions are true, RefIdxSymL1 is set equal to i:
    - RefPicList[1][i] is a short-term-reference picture,
    - DiffPicOrderCnt(currPic, RefPicList[1][i]) < 0,
    - DiffPicOrderCnt(currPic, RefPicList[1][i]) >
      DiffPicOrderCnt(currPic, RefPicList[1][RefIdxSymL1]) or RefIdxSymL1 is equal to −1.
- When RefIdxSymL0 is equal to −1 or RefIdxSymL1 is equal to −1, the following applies:
  - RefIdxSymL0 is set equal to −1 and RefIdxSymL1 is set equal to −1.
  - For each index i with i = 0..NumRefIdxActive[0] − 1, the following applies:
    - When all of the following conditions are true, RefIdxSymL0 is set equal to i:
      - RefPicList[0][i] is a short-term-reference picture,
      - DiffPicOrderCnt(currPic, RefPicList[0][i]) < 0,
      - DiffPicOrderCnt(currPic, RefPicList[0][i]) >
        DiffPicOrderCnt(currPic, RefPicList[0][RefIdxSymL0]) or RefIdxSymL0 is equal to −1.
  - For each index i with i = 0..NumRefIdxActive[1] − 1, the following applies:
    - When all of the following conditions are true, RefIdxSymL1 is set equal to i:
      - RefPicList[1][i] is a short-term-reference picture,
      - DiffPicOrderCnt(currPic, RefPicList[1][i]) > 0,
      - DiffPicOrderCnt(currPic, RefPicList[1][i]) <
        DiffPicOrderCnt(currPic, RefPicList[1][RefIdxSymL1]) or RefIdxSymL1 is equal to −1.

8.3.6 Decoding process for collocated picture and no backward prediction

This process is invoked at the beginning of the decoding process for each P or B slice, after decoding of the slice header as well as the invocation of the decoding process for RPL construction for the slice as specified in clause 8.3.2, but prior to the decoding of any coding unit.

The variable currPic specifies the current picture.

When ph_temporal_mvp_enabled_flag is equal to 1, the variable ColPic is derived as follows:

– If sh_slice_type is equal to B and sh_collocated_from_l0_flag is equal to 0, ColPic is set equal to RefPicList1[sh_collocated_ref_idx].

– Otherwise (sh_slice_type is equal to B and sh_collocated_from_l0_flag is equal to 1, or sh_slice_type is equal to P), ColPic is set equal to RefPicList0[sh_collocated_ref_idx].

The variable NoBackwardPredFlag is derived as follows:

– If DiffPicOrderCnt(aPic, currPic) is less than or equal to 0 for each active picture aPic in RefPicList0 or RefPicList1 of the current slice, NoBackwardPredFlag is set equal to 1.

– Otherwise, NoBackwardPredFlag is set equal to 0.

8.4 Decoding process for coding units coded in intra prediction mode

8.4.1 General decoding process for coding units coded in intra prediction mode

Inputs to this process are:

– a luma location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples,

– a variable treeType specifying whether a single or a dual tree is used and if a dual tree is used, it specifies whether the current tree corresponds to the luma or chroma components.

Output of this process is a modified reconstructed picture before in-loop filtering.

The derivation process for quantization parameters as specified in clause 8.7.1 is invoked with the luma location (xCb, yCb), the width of the current coding block in luma samples cbWidth and the height of the current coding block in luma samples cbHeight, and the variable treeType as inputs.

When treeType is equal to SINGLE_TREE or treeType is equal to DUAL_TREE_LUMA, the decoding process for luma samples is specified as follows:

– If pred_mode_plt_flag is equal to 1, the general decoding process for palette blocks as specified in clause 8.4.5.3 is invoked with (xCbComp, yCbComp) set equal to the luma location (xCb, yCb), the variable treeType, the variable cIdx set equal to 0, the variable nCbW set equal to cbWidth, the variable nCbH set equal to cbHeight.

– Otherwise (pred_mode_plt_flag is equal to 0), the following applies:

  1. The luma intra prediction mode is derived as follows:

     – If IntraMipFlag[xCb][yCb] is equal to 1, IntraPredModeY[x][y] with x = xCb..xCb + cbWidth − 1 and y = yCb..yCb + cbHeight − 1 is set to be equal to intra_mip_mode[xCb][yCb].

     – Otherwise, the derivation process for the luma intra prediction mode as specified in clause 8.4.2 is invoked with the luma location (xCb, yCb), the width of the current coding block in luma samples cbWidth and the height of the current coding block in luma samples cbHeight as input.

  2. The variable predModeIntra is set equal to IntraPredModeY[xCb][yCb].

  3. When cu_act_enabled_flag[xCb][yCb] is equal to 1, the following applies:

     – The general decoding process for intra blocks as specified in clause 8.4.5.1 is invoked with the sample location (xTb0, yTb0) set equal to the luma location (xCb, yCb), the variable nCbW set equal to cbWidth, the variable nCbH set equal to cbHeight, the variable nTbW set equal to cbWidth, the variable nTbH set equal to cbHeight, predModeIntra, and the variable cIdx set equal to 0 and controlPara set equal to 1 as inputs, and the output is a residual sample array $resSamples_L$.

     – The general decoding process for intra blocks as specified in clause 8.4.5.1 is invoked with the sample location (xTb0, yTb0) set equal to the luma location (xCb, yCb), the variable nCbW set equal to cbWidth, the variable nCbH set equal to cbHeight, the variable nTbW set equal to cbWidth, the variable nTbH set equal to cbHeight, predModeIntra, and the variable cIdx set equal to 1 and controlPara set equal to 1 as inputs, and the output is a residual sample array $resSamples_{Cb}$.

– The general decoding process for intra blocks as specified in clause 8.4.5.1 is invoked with the sample location (xTb0, yTb0) set equal to the luma location (xCb, yCb), the variable nCbW set equal to cbWidth, the variable nCbH set equal to cbHeight, the variable nTbW set equal to cbWidth, the variable nTbH set equal to cbHeight, predModeIntra, and the variable cIdx set equal to 2 and controlPara set equal to 1 as inputs, and the output is a residual sample array $resSamples_{Cr}$.

– The residual modification process for residual blocks using colour space conversion as specified in clause 8.7.4.6 is invoked with the variable nTbW set equal to cbWidth, the variable nTbH set equal to cbHeight, the array r_Y set equal to $resSamples_L$, the array r_{Cb} set equal to $resSamples_{Cb}$, and the array r_{Cr} set equal to $resSamples_{Cr}$ as inputs, and the output are modified versions of the arrays $resSamples_L$, $resSamples_{Cb}$ and $resSamples_{Cr}$.

4. The general decoding process for intra blocks as specified in clause 8.4.5.1 is invoked with the sample location (xTb0, yTb0) set equal to the luma location (xCb, yCb), the variable nCbW set equal to cbWidth, the variable nCbH set equal to cbHeight, the variable nTbW set equal to cbWidth, the variable nTbH set equal to cbHeight, predModeIntra, the variable cIdx set equal to 0, controlPara set equal to (cu_act_enabled_flag[xCb][yCb] ? 2 : 3) and, when controlPara is equal to 2, the array of residual samples $resSamples_L$ as inputs, and the output is a modified reconstructed picture before in-loop filtering.

When treeType is equal to SINGLE_TREE or treeType is equal to DUAL_TREE_CHROMA, and when sps_chroma_format_idc is not equal to 0, the decoding process for chroma samples is specified as follows:

– If pred_mode_plt_flag is equal to 1, the following applies:

  – The general decoding process for palette blocks as specified in clause 8.4.5.3 is invoked with (xCbComp, yCbComp) set equal to the chroma location (xCb / SubWidthC, yCb / SubHeightC), the variable treeType, the variable cIdx set equal to 1, the variable nCbW set equal to (cbWidth / SubWidthC), the variable nCbH set equal to (cbHeight / SubHeightC).

  – The general decoding process for palette blocks as specified in clause 8.4.5.3 is invoked with (xCbComp, yCbComp) set equal to the chroma location (xCb / SubWidthC, yCb / SubHeightC), the variable treeType, the variable cIdx set equal to 2, the variable nCbW set equal to (cbWidth / SubWidthC), the variable nCbH set equal to (cbHeight / SubHeightC).

– Otherwise (pred_mode_plt_flag is equal to 0), the following applies:

  1. The derivation process for the chroma intra prediction mode as specified in clause 8.4.3 is invoked with the luma location (xCb, yCb), the width of the current coding block in luma samples cbWidth, the height of the current coding block in luma samples cbHeight, and the tree type treeType as inputs.

  2. The general decoding process for intra blocks as specified in clause 8.4.5.1 is invoked with the sample location (xTb0, yTb0) set equal to the chroma location (xCb / SubWidthC, yCb / SubHeightC), the variable nCbW set equal to (cbWidth / SubWidthC), the variable nCbH set equal to (cbHeight / SubHeightC), the variable nTbW set equal to (cbWidth / SubWidthC), the variable nTbH set equal to (cbHeight / SubHeightC), the variable predModeIntra set equal to IntraPredModeC[xCb][yCb], the variable cIdx set equal to 1, controlPara set equal to (cu_act_enabled_flag[xCb][yCb] ? 2 : 3) and, when controlPara is equal to 2, the array of residual samples $resSamples_{Cb}$ as inputs, and the output is a modified reconstructed picture before in-loop filtering.

  3. The general decoding process for intra blocks as specified in clause 8.4.5.1 is invoked with the sample location (xTb0, yTb0) set equal to the chroma location (xCb / SubWidthC, yCb / SubHeightC), the variable nCbW set equal to (cbWidth / SubWidthC), the variable nCbH set equal to (cbHeight / SubHeightC), the variable nTbW set equal to (cbWidth / SubWidthC), the variable nTbH set equal to (cbHeight / SubHeightC), the variable predModeIntra set equal to IntraPredModeC[xCb][yCb], the variable cIdx set equal to 2, and controlPara set equal to (cu_act_enabled_flag[xCb][yCb] ? 2 : 3) and, when controlPara is equal to 2, the array of residual samples $resSamples_{Cr}$ as inputs, and the output is a modified reconstructed picture before in-loop filtering.

8.4.2 Derivation process for luma intra prediction mode

Input to this process are:

– a luma location (xCb, yCb) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples.

In this process, the luma intra prediction mode IntraPredModeY[xCb][yCb] is derived.

Table 19 specifies the value for the intra prediction mode IntraPredModeY[xCb][yCb] and the associated names.

Table 19 – Specification of intra prediction mode and associated names

Intra prediction mode	Associated name
0	INTRA_PLANAR
1	INTRA_DC
2..66	INTRA_ANGULAR2..INTRA_ANGULAR66
81..83	INTRA_LT_CCLM, INTRA_L_CCLM, INTRA_T_CCLM

NOTE – The intra prediction modes INTRA_LT_CCLM, INTRA_L_CCLM and INTRA_T_CCLM are only applicable to chroma components.

IntraPredModeY[xCb][yCb] is derived as follows:

– If intra_luma_not_planar_flag[xCb][yCb] is equal to 0, IntraPredModeY[xCb][yCb] is set equal to INTRA_PLANAR.

– Otherwise, if BdpcmFlag[xCb][yCb][0] is equal to 1, IntraPredModeY[xCb][yCb] is set equal to BdpcmDir[xCb][yCb][0] ? INTRA_ANGULAR50 : INTRA_ANGULAR18.

– Otherwise (intra_luma_not_planar_flag[xCb][yCb] is equal to 1 and BdpcmFlag[xCb][yCb][0] is equal to 0), the following ordered steps apply:

1. The neighbouring locations (xNbA, yNbA) and (xNbB, yNbB) are set equal to (xCb − 1, yCb + cbHeight − 1) and (xCb + cbWidth − 1, yCb − 1), respectively.

2. For X being replaced by either A or B, the variables candIntraPredModeX are derived as follows:

   – The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring location (xNbY, yNbY) set equal to (xNbX, yNbX), checkPredModeY set equal to FALSE, and cIdx set equal to 0 as inputs, and the output is assigned to availableX.

   – The candidate intra prediction mode candIntraPredModeX is derived as follows:

     – If one or more of the following conditions are true, candIntraPredModeX is set equal to INTRA_PLANAR:

       – The variable availableX is equal to FALSE.

       – CuPredMode[0][xNbX][yNbX] is not equal to MODE_INTRA.

       – IntraMipFlag[xNbX][yNbX] is equal to 1.

       – X is equal to B and yCb − 1 is less than ((yCb >> CtbLog2SizeY) << CtbLog2SizeY).

     – Otherwise, candIntraPredModeX is set equal to IntraPredModeY[xNbX][yNbX].

3. The candModeList[x] with x = 0..4 is derived as follows:

   – If candIntraPredModeB is equal to candIntraPredModeA and candIntraPredModeA is greater than INTRA_DC, candModeList[x] with x = 0..4 is derived as follows:

$$\text{candModeList[0]} = \text{candIntraPredModeA} \tag{204}$$

$$\text{candModeList[1]} = 2 + ((\text{candIntraPredModeA} + 61) \,\%\, 64) \tag{205}$$

$$\text{candModeList[2]} = 2 + ((\text{candIntraPredModeA} - 1) \,\%\, 64) \tag{206}$$

$$\text{candModeList[3]} = 2 + ((\text{candIntraPredModeA} + 60) \,\%\, 64) \tag{207}$$

$$\text{candModeList[4]} = 2 + (\text{candIntraPredModeA} \,\%\, 64) \tag{208}$$

   – Otherwise, if candIntraPredModeB is not equal to candIntraPredModeA and candIntraPredModeA or candIntraPredModeB is greater than INTRA_DC, the following applies:

     – The variables minAB and maxAB are derived as follows:

$$\text{minAB} = \text{Min}(\text{candIntraPredModeA}, \text{candIntraPredModeB}) \tag{209}$$

$$\text{maxAB} = \text{Max}(\text{candIntraPredModeA}, \text{candIntraPredModeB}) \tag{210}$$

– If candIntraPredModeA and candIntraPredModeB are both greater than INTRA_DC, candModeList[x] with x = 0..4 is derived as follows:

candModeList[0] = candIntraPredModeA (211)

candModeList[1] = candIntraPredModeB (212)

– If maxAB − minAB is equal to 1, the following applies:

candModeList[2] = 2 + ((minAB + 61) % 64) (213)

candModeList[3] = 2 + ((maxAB − 1) % 64) (214)

candModeList[4] = 2 + ((minAB + 60) % 64) (215)

– Otherwise, if maxAB − minAB is greater than or equal to 62, the following applies:

candModeList[2] = 2 + ((minAB − 1) % 64) (216)

candModeList[3] = 2 + ((maxAB + 61) % 64) (217)

candModeList[4] = 2 + (minAB % 64) (218)

– Otherwise, if maxAB − minAB is equal to 2, the following applies:

candModeList[2] = 2 + ((minAB − 1) % 64) (219)

candModeList[3] = 2 + ((minAB + 61) % 64) (220)

candModeList[4] = 2 + ((maxAB − 1) % 64) (221)

– Otherwise, the following applies:

candModeList[2] = 2 + ((minAB + 61) % 64) (222)

candModeList[3] = 2 + ((minAB − 1) % 64) (223)

candModeList[4] = 2 + ((maxAB + 61) % 64) (224)

– Otherwise (candIntraPredModeA or candIntraPredModeB is greater than INTRA_DC), candModeList[x] with x = 0..4 is derived as follows:

candModeList[0] = maxAB (225)

candModeList[1] = 2 + ((maxAB + 61) % 64) (226)

candModeList[2] = 2 + ((maxAB − 1) % 64) (227)

candModeList[3] = 2 + ((maxAB + 60) % 64) (228)

candModeList[4] = 2 + (maxAB % 64) (229)

– Otherwise, the following applies:

candModeList[0] = INTRA_DC (230)

candModeList[1] = INTRA_ANGULAR50 (231)

candModeList[2] = INTRA_ANGULAR18 (232)

candModeList[3] = INTRA_ANGULAR46 (233)

candModeList[4] = INTRA_ANGULAR54 (234)

4. IntraPredModeY[xCb][yCb] is derived by applying the following procedure:

   – If intra_luma_mpm_flag[xCb][yCb] is equal to 1, the IntraPredModeY[xCb][yCb] is set equal to candModeList[intra_luma_mpm_idx[xCb][yCb]].

   – Otherwise, IntraPredModeY[xCb][yCb] is derived by applying the following ordered steps:

     1. When candModeList[i] is greater than candModeList[j] for i = 0..3 and for each i, j = (i + 1)..4, both values are swapped as follows:

        (candModeList[i], candModeList[j]) = Swap(candModeList[i], candModeList[j]) (235)

     2. IntraPredModeY[xCb][yCb] is derived by the following ordered steps:

        i. IntraPredModeY[xCb][yCb] is set equal to intra_luma_mpm_remainder[xCb][yCb].

        ii. The value of IntraPredModeY[xCb][yCb] is incremented by one.

        iii. For i equal to 0 to 4, inclusive, when IntraPredModeY[xCb][yCb] is greater than or equal to candModeList[i], the value of IntraPredModeY[xCb][yCb] is incremented by one.

The variable IntraPredModeY[x][y] with x = xCb..xCb + cbWidth − 1 and y = yCb..yCb + cbHeight − 1 is set to be equal to IntraPredModeY[xCb][yCb].

8.4.3 Derivation process for chroma intra prediction mode

Input to this process are:

– a luma location (xCb, yCb) specifying the top-left sample of the current chroma coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples,

– a variable treeType specifying whether a single or a dual tree is used.

In this process, the chroma intra prediction mode IntraPredModeC[xCb][yCb] and the MIP chroma direct mode flag MipChromaDirectFlag[xCb][yCb] are derived.

The MIP chroma direct mode flag MipChromaDirectFlag[xCb][yCb] is derived as follows:

– If all of the following conditions are true, MipChromaDirectFlag[xCb][yCb] is set equal to 1:

  – treeType is equal to SINGLE_TREE.

  – sps_chroma_format_idc is equal to 3.

  – intra_chroma_pred_mode is equal to 4 or cu_act_enabled_flag[xCb][yCb] is equal to 1.

  – IntraMipFlag[xCb][yCb] is equal to 1.

– Otherwise, MipChromaDirectFlag[xCb][yCb] is set equal to 0.

The chroma intra prediction mode IntraPredModeC[xCb][yCb] is derived as follows:

– If MipChromaDirectFlag[xCb][yCb] is equal to 1, the chroma intra prediction mode IntraPredModeC[xCb][yCb] is set equal to IntraPredModeY[xCb][yCb].

– Otherwise, the following applies:

  – The corresponding luma intra prediction mode lumaIntraPredMode is derived as follows:

    – If IntraMipFlag[xCb + cbWidth / 2][yCb + cbHeight / 2] is equal to 1, the following applies:

      – If treeType is equal to SINGLE_TREE and sps_chroma_format_idc is equal to 3, lumaIntraPredMode is set equal to −1.

      – Otherwise, lumaIntraPredMode is set equal to INTRA_PLANAR.

    – Otherwise, if CuPredMode[0][xCb + cbWidth / 2][yCb + cbHeight / 2] is equal to MODE_IBC or MODE_PLT, lumaIntraPredMode is set equal to INTRA_DC.

    – Otherwise, lumaIntraPredMode is set equal to IntraPredModeY[xCb + cbWidth / 2][yCb + cbHeight / 2].

– The chroma intra prediction mode IntraPredModeC[xCb][yCb] is derived as follows:

  – If cu_act_enabled_flag[xCb][yCb] is equal to 1, the chroma intra prediction mode IntraPredModeC[xCb][yCb] is set equal to lumaIntraPredMode.

  – Otherwise, if BdpcmFlag[xCb][yCb][1] is equal to 1, IntraPredModeC[xCb][yCb] is set equal to BdpcmDir[xCb][yCb][1] ? INTRA_ANGULAR50 : INTRA_ANGULAR18.

  – Otherwise (cu_act_enabled_flag[xCb][yCb] is equal to 0 and BdpcmFlag[xCb][yCb][1] is equal to 0), the chroma intra prediction mode IntraPredModeC[xCb][yCb] is derived using cclm_mode_flag, cclm_mode_idx, intra_chroma_pred_mode and lumaIntraPredMode as specified in Table 20.

Table 20 – Specification of IntraPredModeC[xCb][yCb] depending on cclm_mode_flag, cclm_mode_idx, intra_chroma_pred_mode and lumaIntraPredMode

cclm_mode_flag	**cclm_mode_idx**	**intra_chroma_pred_mode**	**lumaIntraPredMode**				
			0	**50**	**18**	**1**	**X (−1 <= X <= 66)**
0	–	0	66	0	0	0	0
0	–	1	50	66	50	50	50
0	–	2	18	18	66	18	18
0	–	3	1	1	1	66	1
0	–	4	0	50	18	1	X
1	0	–	81	81	81	81	81
1	1	–	82	82	82	82	82
1	2	–	83	83	83	83	83

– When sps_chroma_format_idc is equal to 2, the chroma intra prediction mode Y is derived using the chroma intra prediction mode X in Table 20 as specified in Table 21, and the chroma intra prediction mode X is set equal to the chroma intra prediction mode Y afterwards.

Table 21 – Specification of the 4:2:2 mapping process from chroma intra prediction mode X to mode Y when sps_chroma_format_idc is equal to 2

mode X	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
mode Y	0	1	61	62	63	64	65	66	2	3	5	6	8	10	12	13	14	16
mode X	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
mode Y	18	20	22	23	24	26	28	30	31	33	34	35	36	37	38	39	40	41
mode X	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53
mode Y	41	42	43	43	44	44	45	45	46	47	48	48	49	49	50	51	51	52
mode X	54	55	56	57	58	59	60	61	62	63	64	65	66					
mode Y	52	53	54	55	55	56	56	57	57	58	59	59	60					

8.4.4 Cross-component chroma intra prediction mode checking process

Input to this process is:

– a luma location (xCb, yCb) specifying the top-left sample of the current chroma coding block relative to the top-left luma sample of the current picture.

Output of this process is:

– a variable CclmEnabled specifying if a cross-component chroma intra prediction mode is enabled (TRUE) or not enabled (FALSE) for the current chroma coding block.

The variable CclmEnabled is derived as follows:

– If sps_cclm_enabled_flag is equal to 0, CclmEnabled is set equal to 0.

– Otherwise, if one or more of the following conditions are true, CclmEnabled is set equal to 1:

  – sps_qtbtt_dual_tree_intra_flag is equal to 0.

  – sh_slice_type is not equal to I.

  – CtbLog2SizeY is less than 6.

– Otherwise, the following applies:

  – The variables xCb64, yCb64, yCb32 are derived as follows:

$$xCb64 = (xCb >> 6) << 6 \quad (236)$$

$$yCb64 = (yCb >> 6) << 6 \quad (237)$$

$$yCb32 = (yCb >> 5) << 5 \quad (238)$$

  – The variable CclmEnabled is derived as follows:

    – If one or more of the following conditions are true, the variable CclmEnabled is set equal to 1:

      – CbWidth[1][xCb64][yCb64] is equal to 64 and CbHeight[1][xCb64][yCb64] is equal to 64.

      – CqtDepth[1][xCb64][yCb64] is equal to CtbLog2SizeY − 6, MttSplitMode[xCb64][yCb64][0] is equal to SPLIT_BT_HOR, CbWidth[1][xCb64][yCb32] is equal to 64 and CbHeight[1][xCb64][yCb32] is equal to 32.

      – CqtDepth[1][xCb64][yCb64] is greater than CtbLog2SizeY − 6.

      – CqtDepth[1][xCb64][yCb64] is equal to CtbLog2SizeY − 6, MttSplitMode[xCb64][yCb64][0] is equal to SPLIT_BT_HOR, and MttSplitMode[xCb64][yCb32][1] is equal to SPLIT_BT_VER.

    – Otherwise, the variable CclmEnabled is set equal to 0.

  – When CclmEnabled is equal to 1 and one of the following conditions is true, CclmEnabled is set equal to 0:

    – CbWidth[0][xCb64][yCb64] and CbHeight[0][xCb64][yCb64] are both equal to 64, and IntraSubPartitionsModeFlag[xCb64][yCb64] is equal to 1.

    – CbWidth[0][xCb64][yCb64] or CbHeight[0][xCb64][yCb64] is less than 64, and CqtDepth[0][xCb64][yCb64] is equal to CtbLog2SizeY − 6.

8.4.5 Decoding process for intra blocks

8.4.5.1 General decoding process for intra blocks

Inputs to this process are:

– a sample location (xTb0, yTb0) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture,

– a variable nCbW specifying the width of the current coding block,

– a variable nCbH specifying the height of the current coding block,

– a variable nTbW specifying the width of the current transform block,

– a variable nTbH specifying the height of the current transform block,

– a variable predModeIntra specifying the intra prediction mode,

– a variable cIdx specifying the colour component of the current block,

– a variable controlPara specifying the output of the process,

– when controlPara is equal to 2, an array of residual samples resSamplesRec specifying the reconstructed residual samples for the colour component of the current block.

Output of this process is a modified reconstructed picture before in-loop filtering when controlPara is not equal to 1 or a residual sample array when controlPara is equal to 1.

The maximum transform block width maxTbWidth and height maxTbHeight are derived as follows:

$$\text{maxTbWidth} = (\text{cIdx} == 0) \mathbin{?} \text{MaxTbSizeY} : \text{MaxTbSizeY} / \text{SubWidthC} \tag{239}$$

$$\text{maxTbHeight} = (\text{cIdx} == 0) \mathbin{?} \text{MaxTbSizeY} : \text{MaxTbSizeY} / \text{SubHeightC} \tag{240}$$

The luma sample location is derived as follows:

$$(\text{xTbY}, \text{yTbY}) = (\text{cIdx} == 0) \mathbin{?} (\text{xTb0}, \text{yTb0}) : (\text{xTb0} * \text{SubWidthC}, \text{yTb0} * \text{SubHeightC}) \tag{241}$$

Depending on maxTbWidth and maxTbHeight, the following applies:

– If IntraSubPartitionsSplitType is equal to ISP_NO_SPLIT and nTbW is greater than maxTbWidth or nTbH is greater than maxTbHeight, the following ordered steps apply.

  1. The variables verSplitFirst, newTbW, and newTbH are derived as follows:

$$\begin{aligned}\text{verSplitFirst} = {} & (\text{nTbW} * (\text{cIdx} == 0 \mathbin{?} 1 : \text{SubWidthC}) > \text{nTbH} * (\text{cIdx} == 0 \mathbin{?} 1 : \text{SubHeightC})) \\ & \&\& \; (\text{nTbW} > \text{maxTbWidth})\end{aligned} \tag{242}$$

$$\text{newTbW} = \text{verSplitFirst} \mathbin{?} (\text{nTbW} / 2) : \text{nTbW} \tag{243}$$

$$\text{newTbH} = !\text{verSplitFirst} \mathbin{?} (\text{nTbH} / 2) : \text{nTbH} \tag{244}$$

  2. The general decoding process for intra blocks as specified in this clause is invoked with the location (xTb0, yTb0), the coding block width nCbW and the height nCbH, the transform block width nTbW set equal to newTbW and the height nTbH set equal to newTbH, the intra prediction mode predModeIntra, the variable cIdx, the variable controlPara and, when controlPara is equal to 2, the array of residual samples resSamplesRec as inputs, and the output is a modified reconstructed picture before in-loop filtering.

  3. The following applies:

     – If verSplitFirst is equal to TRUE, the general decoding process for intra blocks as specified in this clause is invoked with the location (xTb0, yTb0) set equal to (xTb0 + newTbW, yTb0), the coding block width nCbW and the height nCbH, the transform block width nTbW set equal to newTbW and the height nTbH set equal to newTbH, the intra prediction mode predModeIntra, the variable cIdx, the variable controlPara and, when controlPara is equal to 2, the array of residual samples resSamplesRec as inputs, and the output is a modified reconstructed picture before in-loop filtering.

     – Otherwise (verSplitFirst is equal to FALSE), the general decoding process for intra blocks as specified in this clause is invoked with the location (xTb0, yTb0) set equal to (xTb0, yTb0 + newTbH), the coding block width nCbW and the height nCbH, the transform block width nTbW set equal to newTbW and the height nTbH set equal to newTbH, the intra prediction mode predModeIntra, the variable cIdx, the variable controlPara and, when controlPara is equal to 2, the array of residual samples resSamplesRec as inputs, and the output is a modified reconstructed picture before in-loop filtering.

– Otherwise, the following ordered steps apply:

  – The variables nW, nH, nPbW, pbFactor, xPartInc and yPartInc are derived as follows:

$$\begin{aligned}\text{nW} = {} & (\text{cIdx} == 0 \; \&\& \; \text{IntraSubPartitionsSplitType} == \text{ISP\_VER\_SPLIT}) \mathbin{?} \\ & \text{nTbW} / \text{NumIntraSubPartitions} : \text{nTbW}\end{aligned} \tag{245}$$

$$\begin{aligned}\text{nH} = {} & (\text{cIdx} == 0 \; \&\& \; \text{IntraSubPartitionsSplitType} == \text{ISP\_HOR\_SPLIT}) \mathbin{?} \\ & \text{nTbH} / \text{NumIntraSubPartitions} : \text{nTbH}\end{aligned} \tag{246}$$

$$\text{xPartInc} = (\text{cIdx} == 0 \; \&\& \; \text{IntraSubPartitionsSplitType} == \text{ISP\_VER\_SPLIT}) \mathbin{?} 1 : 0 \tag{247}$$

$$\text{yPartInc} = (\text{cIdx} == 0 \; \&\& \; \text{IntraSubPartitionsSplitType} == \text{ISP\_HOR\_SPLIT}) \mathbin{?} 1 : 0 \tag{248}$$

$$\text{nPbW} = \text{Max}(4, \text{nW}) \tag{249}$$

$$\text{pbFactor} = \text{nPbW} / \text{nW} \tag{250}$$

$$\text{numPartitions} = (\text{cIdx} == 0) ? \text{NumIntraSubPartitions} : 1 \quad (251)$$

– For i = 0..numPartitions − 1, the following applies:

1. The variables xPartIdx, yPartIdx, and xPartPbIdx are derived as follows:

$$\text{xPartIdx} = \text{i} * \text{xPartInc} \quad (252)$$

$$\text{yPartIdx} = \text{i} * \text{yPartInc} \quad (253)$$

$$\text{xPartPbIdx} = \text{xPartIdx} \% \text{pbFactor} \quad (254)$$

2. When controlPara is not equal to 1 and xPartPbIdx is equal to 0, the intra sample prediction process as specified in clause 8.4.5.2.1 is invoked with the location (xTbCmp, yTbCmp) set equal to (xTb0 + nW * xPartIdx, yTb0 + nH * yPartIdx), the intra prediction mode predModeIntra, the transform block width nTbW and height nTbH set equal to nPbW and nH, the coding block width nCbW and height nCbH set equal to nTbW and nTbH, and the variable cIdx as inputs, and the output is an (nPbW)x(nH) array predSamples.

3. The (nW)x(nH) array resSamples is derived as follows:
   – If controlPara is equal to 2, the (nW)x(nH) array resSamples is derived by setting resSamples[x][y] equal to resSamplesRec[x][y] with x = 0..nW − 1, y = 0..nH − 1.
   – Otherwise (controlPara is not equal to 2), the scaling and transformation process as specified in clause 8.7.2 is invoked with the luma location (xTbY, yTbY) set equal to (xTbY + nW * xPartIdx, yTbY + nH * yPartIdx), the variable cIdx, the variable predMode set equal to MODE_INTRA, nCbW, nCbH, the transform width nTbW and the transform height nTbH set equal to nW and nH as inputs, and the output is an (nW)x(nH) array resSamples.

4. When controlPara is not equal to 1, the picture reconstruction process for a colour component as specified in clause 8.7.5.1 is invoked with the current block location (xCurr, yCurr) set equal to (xTb0 + nW * xPartIdx, yTb0 + nH * yPartIdx), the current block width nCurrSw and the current block height nCurrSh set equal to nW and nH, respectively, the variable cIdx, the (nW)x(nH) array predSamples[x][y] with x = xPartPbIdx * nW..(xPartPbIdx +1) * nW − 1, y = 0..nH − 1, and the (nW)x(nH) array resSamples as inputs, and the output is a modified reconstructed picture before in-loop filtering.

8.4.5.2 Intra sample prediction

8.4.5.2.1 General

Inputs to this process are:

– a sample location (xTbCmp, yTbCmp) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture,
– a variable predModeIntra specifying the intra prediction mode,
– a variable nTbW specifying the transform block width,
– a variable nTbH specifying the transform block height,
– a variable nCbW specifying the coding block width,
– a variable nCbH specifying the coding block height,
– a variable cIdx specifying the colour component of the current block.

Outputs of this process are the predicted samples predSamples[x][y], with x = 0..nTbW − 1, y = 0..nTbH − 1.

The predicted samples predSamples[x][y] are derived as follows:

– If IntraMipFlag[xTbCmp][yTbCmp] is equal to 1 and cIdx is equal to 0, or if MipChromaDirectFlag[xTbCmp][yTbCmp] is equal to 1 and cIdx is not equal to 0, the matrix-based intra sample prediction process as specified in clause 8.4.5.2.2 is invoked with the location (xTbCmp, yTbCmp), the intra prediction mode predModeIntra, the transform block width nTbW and height nTbH, and the variable cIdx as inputs, and the output is predSamples.

– Otherwise, the general intra sample prediction process as specified in clause 8.4.5.2.6 is invoked with the location (xTbCmp, yTbCmp), the intra prediction mode predModeIntra, the transform block width nTbW and height nTbH, the coding block width nCbW and height nCbH, and the variable cIdx as inputs, and the output is predSamples.

8.4.5.2.2 Matrix-based intra sample prediction

Inputs to this process are:

– a sample location (xTbCmp, yTbCmp) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture,

– a variable predModeIntra specifying the intra prediction mode,

– a variable nTbW specifying the transform block width,

– a variable nTbH specifying the transform block height,

– a variable cIdx specifying the colour component of the current block.

Outputs of this process are the predicted samples predSamples[x][y], with x = 0..nTbW − 1, y = 0..nTbH − 1.

The variable mipSizeId is derived as follows:

– If both nTbW and nTbH are equal to 4, mipSizeId is set equal to 0.

– Otherwise, if either nTbW or nTbH is equal to 4, or both nTbW and nTbH are equal to 8, mipSizeId is set equal to 1.

– Otherwise, mipSizeId is set equal to 2.

Variables boundarySize and predSize are derived using mipSizeId as specified in Table 22.

Table 22 – Specification of boundary size boundarySize and prediction size predSize using mipSizeId

mipSizeId	boundarySize	predSize
0	2	4
1	4	4
2	4	8

The flag isTransposed is derived as follows:

isTransposed = intra_mip_transposed_flag[xTbCmp][yTbCmp] (255)

The variables inSize, variables refW and refH are derived as follows:

inSize = (2 * boundarySize) − ((mipSizeId = = 2) ? 1 : 0) (256)

refW = nTbW + 1 (257)

refH = nTbH + 1 (258)

For the generation of the reference samples refT[x] with x = 0..nTbW − 1 and refL[y] with y = 0..nTbH − 1, the following applies:

– The reference sample availability marking process as specified in clause 8.4.5.2.8 is invoked with the sample location (xTbCmp, yTbCmp), reference line index equal to 0, the reference sample width refW, the reference sample height refH, and the colour component index cIdx as inputs, and the reference samples refUnfilt[x][y] with x = −1, y = −1..refH − 1 and x = 0..refW − 1, y = −1 as output.

– When at least one sample refUnfilt[x][y] with x = −1, y = −1..refH − 1 and x = 0..refW − 1, y = −1 is marked as "not available for intra prediction", the reference sample substitution process as specified in clause 8.4.5.2.9 is invoked with reference line index 0, the reference sample width refW, the reference sample height refH, the reference samples refUnfilt[x][y] with x = −1, y = −1..refH − 1 and x = 0..refW − 1, y = −1, and the colour component index cIdx as inputs, and the modified reference samples refUnfilt[x][y] with x = −1, y = −1..refH − 1 and x = 0..refW − 1, y = −1 as output.

– The reference samples refT[x] with x = 0..nTbW − 1 and refL[y] with y = 0..nTbH − 1 are assigned as follows:

refT[x] = refUnfilt[x][−1] (259)

$$\text{refL}[\ y\ ] = \text{refUnfilt}[\ -1\ ][\ y\ ] \tag{260}$$

For the generation of the input samples p[x] with x = 0..inSize − 1, the following applies:

– The MIP boundary downsampling process as specified in clause 8.4.5.2.3 is invoked for the top reference samples with the block size nTbW, the reference samples refT[x] with x = 0..nTbW − 1, and the boundary size boundarySize as inputs, and reduced boundary samples redT[x] with x = 0..boundarySize − 1 as outputs.

– The MIP boundary downsampling process as specified in clause 8.4.5.2.3 is invoked for the left reference samples with the block size nTbH, the reference samples refL[y] with y = 0..nTbH − 1, and the boundary size boundarySize as inputs, and reduced boundary samples redL[x] with x = 0..boundarySize − 1 as outputs.

– The reduced top and left boundary samples redT and redL are assigned to the boundary sample array pTemp[x] with x = 0..2 * boundarySize − 1 as follows:

  – If isTransposed is equal to 1, pTemp[x] is set equal to redL[x] with x = 0..boundarySize − 1 and pTemp[x + boundarySize] is set equal to redT[x] with x = 0..boundarySize − 1.

  – Otherwise, pTemp[x] is set equal to redT[x] with x = 0..boundarySize − 1 and pTemp[x + boundarySize] is set equal to redL[x] with x = 0..boundarySize − 1.

– The input values p[x] with x = 0.. inSize − 1 are derived as follows:

  – If mipSizeId is equal to 2, the following applies:

$$p[\ x\ ] = \text{pTemp}[\ x + 1\ ] - \text{pTemp}[\ 0\ ] \tag{261}$$

  – Otherwise (mipSizeId is less than 2), the following applies:

$$\begin{aligned} &p[\ 0\ ] = (\ 1\ \ll\ (\ \text{BitDepth} - 1\ )\ ) - \text{pTemp}[\ 0\ ] \\ &p[\ x\ ] = \text{pTemp}[\ x\ ] - \text{pTemp}[\ 0\ ] \qquad \text{for } x = 1..\text{inSize} - 1 \end{aligned} \tag{262}$$

For the intra sample prediction process according to predModeIntra, the following ordered steps apply:

1. The matrix-based intra prediction samples predMip[x][y], with x = 0..predSize − 1, y = 0..predSize − 1 are derived as follows:

   – The variable modeId is set equal to predModeIntra.

   – The weight matrix mWeight[x][y] with x = 0..inSize − 1, y = 0..predSize * predSize − 1 is derived by invoking the MIP weight matrix derivation process as specified in clause 8.4.5.2.4 with mipSizeId and modeId as inputs.

   – The matrix-based intra prediction samples predMip[x][y], with x = 0..predSize − 1, y = 0..predSize − 1 are derived as follows:

$$\text{oW} = 32 - 32 * (\textstyle\sum_{i=0}^{\text{inSize}-1} p[\ i\ ]) \tag{263}$$

$$\begin{aligned} \text{predMip}[\ x\ ][\ y\ ] = (\ (\ (\ &\textstyle\sum_{i=0}^{\text{inSize}-1} \text{mWeight}[\ i\ ][\ y * \text{predSize} + x\ ] * p[\ i\ ]\ ) + \\ &\text{oW}\ ) \gg 6\ ) + \text{pTemp}[\ 0\ ] \end{aligned} \tag{264}$$

2. The matrix-based intra prediction samples predMip[x][y], with x = 0..predSize − 1, y = 0..predSize − 1 are clipped as follows:

$$\text{predMip}[\ x\ ][\ y\ ] = \text{Clip1}(\ \text{predMip}[\ x\ ][\ y\ ]\ ) \tag{265}$$

3. When isTransposed is equal to TRUE, the predSize x predSize array predMip[x][y] with x = 0..predSize − 1, y = 0..predSize − 1 is transposed as follows:

$$\text{predTemp}[\ y\ ][\ x\ ] = \text{predMip}[\ x\ ][\ y\ ] \tag{266}$$

$$\text{predMip} = \text{predTemp} \tag{267}$$

4. The predicted samples predSamples[x][y], with x = 0..nTbW − 1, y = 0..nTbH − 1 are derived as follows:

   – If nTbW is greater than predSize or nTbH is greater than predSize, the MIP prediction upsampling process as specified in clause 8.4.5.2.5 is invoked with the input block size predSize, matrix-based intra prediction samples predMip[x][y] with x = 0..predSize − 1, y = 0..predSize − 1, the transform block width nTbW, the transform block height nTbH, the top reference samples refT[x] with x = 0..nTbW − 1, and the left reference samples refL[y] with y = 0..nTbH − 1 as inputs, and the output is the predicted sample array predSamples.

– Otherwise, predSamples[x][y], with x = 0..nTbW − 1, y = 0..nTbH − 1 is set equal to predMip[x][y].

8.4.5.2.3 MIP boundary sample downsampling process

Inputs to this process are:

– a variable nTbS specifying the transform block size,

– reference samples refS[x] with x = 0..nTbS − 1,

– a variable boundarySize specifying the downsampled boundary size.

Outputs of this process are the reduced boundary samples redS[x] with x = 0..boundarySize − 1.

The reduced boundary samples redS[x] with x = 0..boundarySize − 1 are derived as follows:

– If boundarySize is less than nTbS, the following applies:

$$\text{bDwn} = \text{nTbS} / \text{boundarySize} \quad (268)$$

$$\text{redS}[\ x\ ] = (\ \textstyle\sum_{i=0}^{\text{bDwn}-1} \text{refS}[\ x * \text{bDwn} + i\ ] + (\ 1 \ll (\ \text{Log2}(\ \text{bDwn}\ ) - 1\ )\ )\ ) \gg \text{Log2}(\ \text{bDwn}\ ) \quad (269)$$

– Otherwise (boundarySize is equal to nTbS), redS[x] is set equal to refS[x].

8.4.5.2.4 MIP weight matrix derivation process

Inputs to this process are:

– a variable mipSizeId,

– a variable modeId.

Output of this process is the MIP weight matrix mWeight[x][y].

The MIP weight matrix mWeight[x][y] is derived depending on mipSizeId and modeId as follows:

– If mipSizeId is equal to 0 and modeId is equal to 0, the following applies:

mWeight[x][y] = (270)

```
{
{ 32,  30,  90,  28}, { 32,  32,  72,  28}, { 34,  77,  53,  30}, { 51, 124,  36,  37},
{ 31,  31,  95,  37}, { 33,  31,  70,  50}, { 52,  80,  25,  60}, { 78, 107,   1,  65},
{ 31,  29,  37,  95}, { 38,  34,  19, 101}, { 73,  85,   0,  81}, { 92,  99,   0,  65},
{ 34,  29,  14, 111}, { 48,  48,   7, 100}, { 80,  91,   0,  74}, { 89,  97,   0,  64}
},
```

– Otherwise, if mipSizeId is equal to 0 and modeId is equal to 1, the following applies:

mWeight[x][y] = (271)

```
{
{ 31,  23,  34,  29}, { 31,  43,  34,  31}, { 30,  95,  34,  32}, { 29, 100,  35,  33},
{ 31,  23,  34,  29}, { 31,  43,  34,  31}, { 30,  95,  34,  32}, { 29,  99,  35,  33},
{ 31,  24,  35,  29}, { 31,  44,  34,  31}, { 30,  95,  35,  32}, { 29,  99,  35,  33},
{ 31,  24,  35,  30}, { 31,  44,  35,  31}, { 30,  95,  35,  32}, { 29,  99,  35,  33}
},
```

– Otherwise, if mipSizeId is equal to 0 and modeId is equal to 2, the following applies:

mWeight[x][y] = (272)

```
{
{ 32,  32,  36,  58}, { 32,  29,  26,  66}, { 36,  37,  23,  61}, { 79,  84,   3,  37},
{ 32,  32,  30,  69}, { 33,  29,  24,  71}, { 44,  16,  21,  70}, { 96,  18,   0,  57},
{ 32,  31,  24,  74}, { 33,  30,  23,  71}, { 36,  24,  24,  71}, { 59,   9,  16,  68},
{ 32,  32,  23,  75}, { 33,  30,  24,  70}, { 32,  30,  25,  71}, { 36,  26,  25,  70}
},
```

– Otherwise, if mipSizeId is equal to 0 and modeId is equal to 3, the following applies:

mWeight[x][y] = (273)

```
{
{ 32,  33,  34,  32}, { 32,  30,  22,  38}, { 29,  46,  25,  38}, { 53, 123,  28,  22},
{ 32,  33,  30,  37}, { 32,  30,  21,  38}, { 32,  40,  24,  38}, { 64, 116,  26,  17},
{ 32,  32,  23,  49}, { 32,  30,  21,  39}, { 34,  39,  24,  37}, { 72, 109,  23,  16},
{ 33,  31,  17,  60}, { 32,  31,  21,  39}, { 35,  41,  24,  37}, { 72, 106,  22,  18}
},
```

– Otherwise, if mipSizeId is equal to 0 and modeId is equal to 4, the following applies:

mWeight[x][y] = (274)

```
{
{ 34,  25,  89,  20}, { 38,  32,  47,  24}, { 40,  86,  29,  27}, { 38,  98,  32,  29},
{ 34,  31,  94,  40}, { 44,  25,  83,  27}, { 54,  72,  43,  16}, { 47,  94,  33,  22},
{ 33,  31,  36,  94}, { 43,  23,  51,  76}, { 62,  55,  64,  25}, { 57,  89,  38,  15},
{ 32,  32,  28, 101}, { 38,  26,  33,  94}, { 55,  38,  68,  47}, { 59,  80,  52,  16}
},
```

– Otherwise, if mipSizeId is equal to 0 and modeId is equal to 5, the following applies:

mWeight[x][y] = (275)

```
{
{ 28,  30,  68,  29}, { 23,  48,  23,  48}, { 39,  98,  16,  42}, { 84,  86,  20,  17},
{ 25,  31,  52,  74}, { 38,  68,   5,  70}, { 95,  78,   7,  21}, {127,  54,  12,   0},
{ 30,  47,  14, 107}, { 79,  76,   0,  53}, {127,  59,   7,   1}, {127,  51,   9,   0},
{ 50,  71,   1,  96}, {109,  69,   7,  25}, {127,  56,   9,   0}, {123,  53,  13,   0}
},
```

– Otherwise, if mipSizeId is equal to 0 and modeId is equal to 6, the following applies:

mWeight[x][y] = (276)

```
{
{ 40,  20,  72,  18}, { 48,  29,  44,  18}, { 53,  81,  35,  18}, { 48,  96,  33,  22},
{ 45,  23,  79,  49}, { 61,  21,  56,  49}, { 72,  52,  32,  48}, { 65,  69,  20,  50},
{ 41,  27,  29,  96}, { 49,  22,  28,  94}, { 52,  22,  28,  93}, { 49,  27,  27,  92},
{ 37,  29,  26,  98}, { 39,  28,  28,  97}, { 38,  28,  30,  97}, { 38,  29,  30,  95}
},
```

– Otherwise, if mipSizeId is equal to 0 and modeId is equal to 7, the following applies:

mWeight[x][y] = (277)

```
{
{ 33,  27,  43,  27}, { 32,  29,  31,  31}, { 31,  73,  33,  31}, { 35, 104,  34,  28},
{ 32,  30,  63,  22}, { 33,  26,  33,  29}, { 33,  57,  33,  30}, { 37, 100,  35,  27},
{ 32,  31,  85,  25}, { 34,  25,  39,  25}, { 35,  39,  32,  28}, { 40,  91,  35,  25},
{ 32,  30,  77,  50}, { 34,  26,  54,  22}, { 37,  31,  34,  27}, { 45,  75,  34,  23}
},
```

– Otherwise, if mipSizeId is equal to 0 and modeId is equal to 8, the following applies:

mWeight[x][y] = (278)

```
{
{ 34,  25,  77,  19}, { 36,  34,  56,  24}, { 41,  83,  39,  30}, { 47,  96,  28,  35},
{ 34,  31,  70,  65}, { 38,  29,  53,  77}, { 43,  36,  37,  83}, { 48,  39,  28,  83},
{ 33,  31,  31,  98}, { 33,  31,  30,  99}, { 34,  30,  31,  98}, { 36,  29,  31,  96},
{ 32,  32,  30,  97}, { 32,  32,  31,  96}, { 31,  33,  33,  96}, { 32,  33,  34,  94}
},
```

– Otherwise, if mipSizeId is equal to 0 and modeId is equal to 9, the following applies:

mWeight[x][y] = (279)

```
{
{ 30,  30,  93,  19}, { 31,  59,  67,  34}, { 31,  79,  36,  59}, { 30,  67,  17,  79},
{ 30,  38,  68,  69}, { 29,  40,  43,  91}, { 26,  35,  32, 101}, { 23,  32,  30, 101},
{ 26,  34,  30, 101}, { 23,  33,  30, 102}, { 20,  32,  31, 102}, { 18,  33,  32, 102},
{ 23,  33,  31, 100}, { 20,  34,  32, 100}, { 18,  35,  33, 100}, { 18,  35,  33, 100}
},
```

– Otherwise, if mipSizeId is equal to 0 and modeId is equal to 10, the following applies:

mWeight[x][y] = (280)

```
{
{ 31,  54,  90,  26}, { 32,  60,  53,  61}, { 34,  49,  37,  84}, { 34,  39,  35,  89},
{ 35,  38,  41,  88}, { 35,  35,  32,  96}, { 35,  31,  33,  96}, { 35,  32,  35,  94},
{ 34,  34,  30,  97}, { 35,  32,  33,  95}, { 35,  32,  34,  94}, { 35,  34,  34,  93},
{ 34,  34,  34,  93}, { 35,  34,  34,  93}, { 35,  34,  34,  92}, { 36,  34,  35,  91}
},
```

– Otherwise, if mipSizeId is equal to 0 and modeId is equal to 11, the following applies:

mWeight[x][y] = (281)

```
{
{ 32,  29,  54,  24}, { 31,  32,  34,  29}, { 31,  43,  34,  29}, { 32,  67,  36,  28},
{ 31,  34,  69,  37}, { 31,  35,  46,  33}, { 30,  35,  39,  33}, { 30,  42,  39,  36},
{ 31,  35,  39,  88}, { 30,  38,  41,  84}, { 30,  39,  40,  81}, { 39,  46,  38,  78},
{ 31,  36,  34,  96}, { 34,  38,  37,  93}, { 55,  42,  38,  82}, { 89,  53,  38,  65}
},
```

– Otherwise, if mipSizeId is equal to 0 and modeId is equal to 12, the following applies:

mWeight[x][y] = (282)

```
{
{ 32,  33,  43,  29}, { 32,  30,  29,  33}, { 31,  47,  31,  33}, { 33, 100,  31,  31},
{ 32,  33,  74,  25}, { 32,  32,  34,  31}, { 32,  33,  30,  33}, { 32,  68,  30,  32},
{ 32,  31,  91,  40}, { 32,  32,  58,  26}, { 31,  31,  30,  32}, { 31,  42,  30,  33},
{ 32,  31,  49,  85}, { 32,  31,  83,  35}, { 31,  33,  48,  29}, { 31,  36,  32,  33}
},
```

– Otherwise, if mipSizeId is equal to 0 and modeId is equal to 13, the following applies:

mWeight[x][y] = (283)

```
{
{ 31,  29,  81,  35}, { 32,  28,  34,  50}, { 31,  75,  16,  43}, { 34, 103,  29,  32},
{ 32,  32,  53,  78}, { 31,  28,  36,  88}, { 30,  52,  18,  73}, { 52,  88,  17,  35},
{ 32,  32,  35,  94}, { 30,  31,  35,  95}, { 36,  29,  31,  92}, {100,  43,  16,  40},
{ 32,  32,  35,  93}, { 30,  32,  38,  93}, { 55,  18,  37,  83}, {127,   0,  30,  40}
},
```

– Otherwise, if mipSizeId is equal to 0 and modeId is equal to 14, the following applies:

mWeight[x][y] = (284)

```
{
{ 31,  22,  47,  30}, { 31,  48,  25,  34}, { 30,  95,  31,  32}, { 32, 103,  33,  32},
{ 30,  24,  57,  31}, { 30,  47,  26,  34}, { 31,  95,  31,  32}, { 43,  97,  35,  25},
{ 29,  26,  44,  63}, { 37,  38,  24,  47}, { 74,  63,  28,  20}, {110,  58,  34,   3},
{ 46,  22,   5, 108}, { 93,   5,   9,  77}, {127,   0,  17,  52}, {127,   0,  15,  50}
},
```

– Otherwise, if mipSizeId is equal to 0 and modeId is equal to 15, the following applies:

mWeight[x][y] = (285)

```
{
{ 32,  27,  68,  24}, { 35,  23,  35,  28}, { 35,  64,  29,  29}, { 37, 104,  33,  28},
{ 32,  32,  91,  40}, { 36,  23,  67,  36}, { 49,  23,  39,  28}, { 60,  67,  30,  20},
{ 32,  32,  36,  95}, { 35,  29,  38,  93}, { 50,  16,  30,  84}, { 72,  16,  15,  65},
{ 32,  32,  27, 100}, { 33,  32,  29, 100}, { 37,  29,  30,  98}, { 48,  21,  29,  90}
},
```

– Otherwise, if mipSizeId is equal to 1 and modeId is equal to 0, the following applies:

mWeight[x][y] = (286)

```
{
{ 30,  63,  46,  37,  25,  33,  33,  34}, { 30,  60,  66,  38,  32,  31,  32,  33},
{ 29,  45,  74,  42,  32,  32,  32,  33}, { 30,  39,  62,  58,  32,  33,  32,  33},
{ 30,  66,  55,  39,  32,  30,  30,  36}, { 29,  54,  69,  40,  33,  31,  31,  33},
{ 28,  48,  71,  43,  32,  33,  32,  33}, { 28,  41,  72,  46,  32,  34,  32,  33},
```

```
{ 30,  66,  56,  40,  32,  33,  28,  33}, { 29,  55,  69,  39,  33,  33,  30,  32},
{ 27,  46,  72,  43,  33,  33,  32,  33}, { 27,  42,  69,  48,  32,  34,  32,  33},
{ 30,  63,  55,  40,  32,  33,  35,  30}, { 29,  56,  66,  40,  33,  33,  33,  30},
{ 27,  47,  69,  44,  33,  33,  33,  32}, { 27,  42,  65,  50,  32,  34,  32,  33}
},
```

– Otherwise, if mipSizeId is equal to 1 and modeId is equal to 1, the following applies:

mWeight[x][y] = (287)

```
{
{ 32,  33,  30,  31,  74,  30,  31,  32}, { 33,  56,  28,  30,  41,  29,  32,  32},
{ 33,  77,  52,  26,  29,  34,  30,  32}, { 33,  37,  80,  41,  31,  34,  30,  32},
{ 32,  32,  33,  31,  59,  76,  28,  31}, { 33,  31,  31,  30,  78,  40,  28,  32},
{ 33,  47,  28,  29,  53,  27,  31,  31}, { 33,  61,  44,  28,  34,  32,  31,  31},
{ 32,  31,  34,  30,  26,  64,  76,  27}, { 32,  31,  34,  29,  45,  86,  36,  29},
{ 33,  27,  34,  29,  73,  55,  25,  32}, { 33,  33,  34,  30,  62,  33,  30,  31},
{ 32,  31,  34,  30,  30,  29,  58,  74}, { 32,  31,  35,  29,  27,  53,  77,  35},
{ 32,  30,  36,  29,  40,  80,  44,  31}, { 33,  28,  37,  30,  58,  60,  31,  33}
},
```

– Otherwise, if mipSizeId is equal to 1 and modeId is equal to 2, the following applies:

mWeight[x][y] = (288)

```
{
{ 32,  51,  27,  32,  27,  50,  29,  32}, { 32,  95,  42,  29,  29,  42,  30,  32},
{ 32,  27,  99,  34,  31,  41,  29,  32}, { 32,  34,  21, 104,  31,  42,  30,  32},
{ 32,  45,  30,  32,   9,  88,  40,  30}, { 32,  77,  38,  30,   9,  76,  38,  30},
{ 32,  38,  78,  33,  14,  67,  37,  30}, { 32,  30,  30,  87,  20,  59,  38,  31},
{ 33,  37,  32,  32,  27,  18, 106,  34}, { 34,  44,  34,  31,  25,  17, 108,  31},
{ 36,  39,  45,  31,  24,  15, 108,  30}, { 37,  31,  31,  54,  25,  14, 101,  32},
{ 36,  33,  32,  30,  29,  37,  13, 110}, { 39,  32,  32,  29,  27,  37,  15, 108},
{ 44,  33,  31,  27,  25,  37,  16, 106}, { 47,  30,  31,  32,  25,  34,  19, 102}
},
```

– Otherwise, if mipSizeId is equal to 1 and modeId is equal to 3, the following applies:

mWeight[x][y] = (289)

```
{
{ 32,  48,  35,  35,  47,  68,  31,  31}, { 32,  33,  59,  40,  27,  71,  33,  30},
{ 32,  29,  47,  65,  24,  62,  37,  30}, { 33,  33,  31,  81,  26,  50,  42,  32},
{ 32,  30,  40,  38,  30,  70,  55,  31}, { 32,  20,  46,  50,  26,  55,  64,  31},
{ 33,  30,  29,  66,  25,  41,  72,  33}, { 36,  34,  27,  69,  26,  31,  67,  39},
{ 33,  28,  36,  40,  30,  26,  85,  47}, { 36,  27,  33,  50,  31,  20,  79,  53},
{ 43,  30,  26,  57,  28,  17,  67,  62}, { 51,  27,  28,  55,  22,  23,  49,  70},
{ 38,  29,  32,  39,  28,  30,  22, 104}, { 51,  31,  28,  43,  24,  31,  17, 102},
{ 69,  23,  30,  40,  15,  38,  10,  95}, { 77,  13,  35,  38,   8,  43,   8,  90}
},
```

– Otherwise, if mipSizeId is equal to 1 and modeId is equal to 4, the following applies:

mWeight[x][y] = (290)

```
{
{ 32,  38,  32,  33, 101,  40,  29,  32}, { 32,  40,  37,  32, 100,  36,  30,  32},
{ 32,  37,  46,  35,  94,  33,  30,  31}, { 33,  34,  30,  62,  81,  35,  30,  31},
{ 32,  32,  33,  32,  22, 102,  39,  29}, { 32,  31,  33,  33,  26, 104,  34,  28},
{ 33,  33,  33,  33,  31, 103,  32,  28}, { 33,  32,  34,  36,  37,  94,  33,  28},
{ 32,  33,  32,  32,  34,  24,  99,  36}, { 32,  34,  33,  33,  33,  30,  98,  32},
{ 33,  33,  34,  33,  31,  37,  95,  29}, { 33,  33,  33,  36,  30,  46,  85,  31},
{ 32,  33,  32,  33,  30,  34,  23, 104}, { 32,  34,  33,  33,  31,  32,  30,  98},
{ 32,  33,  34,  34,  31,  29,  39,  91}, { 33,  33,  32,  37,  32,  30,  47,  82}
},
```

– Otherwise, if mipSizeId is equal to 1 and modeId is equal to 5, the following applies:

mWeight[x][y] = (291)

```
{
{ 32,  52,  48,  31,  38,  76,  26,  32}, { 33,  19,  62,  50,  25,  50,  51,  31},
{ 33,  30,  20,  74,  29,  29,  54,  51}, { 34,  35,  23,  56,  31,  25,  41,  76},
{ 33,  25,  38,  39,  28,  39,  83,  35}, { 35,  28,  25,  47,  31,  23,  57,  74},
{ 37,  35,  22,  38,  31,  27,  30, 101}, { 38,  32,  33,  29,  30,  31,  27, 103},
{ 34,  32,  27,  37,  32,  25,  41,  92}, { 38,  33,  28,  32,  30,  31,  18, 111},
{ 40,  32,  33,  27,  29,  33,  18, 111}, { 40,  32,  34,  27,  28,  33,  23, 105},
```

```
{ 35,  32,  30,  33,  31,  33,  20, 107}, { 38,  31,  33,  30,  29,  33,  21, 106},
{ 40,  32,  33,  29,  29,  34,  22, 105}, { 40,  32,  33,  30,  29,  34,  24, 101}
},
```

– Otherwise, if mipSizeId is equal to 1 and modeId is equal to 6, the following applies:

mWeight[x][y] = (292)

```
{
{ 32,  28,  31,  33,  92,  33,  30,  31}, { 33,  30,  28,  33,  71,  26,  32,  30},
{ 33,  60,  26,  33,  47,  28,  33,  30}, { 33,  63,  44,  36,  37,  31,  33,  30},
{ 33,  30,  31,  33,  43,  90,  33,  29}, { 33,  28,  29,  34,  71,  71,  26,  30},
{ 33,  30,  26,  33,  86,  45,  28,  30}, { 33,  38,  29,  32,  74,  32,  33,  29},
{ 33,  32,  30,  32,  29,  41,  95,  27}, { 34,  31,  29,  33,  26,  71,  73,  22},
{ 34,  31,  29,  33,  37,  88,  46,  25}, { 33,  32,  28,  34,  55,  75,  36,  28},
{ 34,  31,  30,  32,  33,  27,  43,  89}, { 35,  32,  28,  33,  33,  23,  77,  59},
{ 34,  33,  28,  33,  30,  35,  91,  37}, { 34,  34,  28,  34,  33,  53,  74,  31}
},
```

– Otherwise, if mipSizeId is equal to 1 and modeId is equal to 7, the following applies:

mWeight[x][y] = (293)

```
{
{ 33,  49,  26,  32,  26,  52,  28,  31}, { 33,  71,  72,  24,  30,  32,  34,  31},
{ 32,  23,  70,  68,  32,  32,  32,  32}, { 31,  33,  21, 106,  33,  32,  32,  33},
{ 34,  47,  32,  29,   5,  86,  44,  26}, { 34,  44,  89,  28,  28,  37,  33,  30},
{ 32,  27,  46,  89,  33,  31,  31,  32}, { 30,  33,  20, 107,  33,  33,  32,  33},
{ 35,  39,  42,  27,  26,  24,  92,  35}, { 34,  27,  87,  43,  30,  34,  38,  31},
{ 31,  31,  32, 100,  32,  33,  30,  32}, { 29,  32,  22, 106,  33,  33,  32,  33},
{ 35,  29,  47,  32,  32,  32,  17, 100}, { 34,  24,  69,  60,  34,  33,  28,  44},
{ 31,  33,  31,  99,  32,  33,  32,  31}, { 29,  33,  25, 103,  33,  33,  32,  35}
},
```

– Otherwise, if mipSizeId is equal to 2 and modeId is equal to 0, the following applies:

mWeight[x][y] = (294)

```
{
{ 42,  37,  33,  27,  44,  33,  35}, { 71,  39,  34,  24,  36,  35,  36},
{ 77,  46,  35,  33,  30,  34,  36}, { 64,  60,  35,  33,  31,  32,  36},
{ 49,  71,  38,  32,  32,  31,  36}, { 42,  66,  50,  33,  31,  32,  36},
{ 40,  52,  67,  33,  31,  32,  35}, { 38,  43,  75,  33,  32,  32,  35},
{ 56,  40,  33,  26,  43,  38,  36}, { 70,  49,  34,  30,  28,  38,  38},
{ 65,  57,  36,  34,  28,  33,  39}, { 59,  60,  39,  33,  30,  31,  38},
{ 55,  60,  43,  33,  30,  31,  38}, { 51,  61,  47,  33,  30,  32,  37},
{ 46,  62,  51,  34,  30,  32,  37}, { 42,  60,  55,  33,  31,  32,  37},
{ 60,  42,  34,  30,  37,  43,  38}, { 68,  52,  35,  35,  22,  37,  40},
{ 62,  58,  37,  34,  28,  31,  40}, { 58,  59,  41,  33,  30,  30,  39},
{ 56,  59,  44,  34,  30,  31,  38}, { 53,  60,  45,  33,  30,  31,  38},
{ 49,  65,  45,  33,  30,  31,  38}, { 45,  64,  47,  33,  31,  32,  38},
{ 59,  44,  35,  31,  34,  43,  41}, { 66,  53,  36,  35,  25,  31,  43},
{ 61,  58,  38,  34,  29,  30,  40}, { 59,  57,  41,  33,  30,  31,  39},
{ 57,  58,  43,  33,  30,  31,  39}, { 54,  61,  43,  33,  31,  31,  39},
{ 51,  64,  43,  33,  31,  31,  39}, { 48,  64,  45,  33,  32,  31,  39},
{ 57,  45,  35,  30,  35,  40,  44}, { 65,  54,  37,  33,  33,  24,  44},
{ 63,  56,  38,  34,  30,  29,  39}, { 61,  56,  41,  34,  30,  32,  39},
{ 58,  58,  42,  33,  31,  31,  39}, { 54,  62,  41,  33,  31,  31,  39},
{ 51,  65,  42,  33,  31,  31,  39}, { 48,  63,  43,  33,  32,  31,  39},
{ 55,  46,  35,  30,  36,  38,  47}, { 65,  53,  37,  32,  36,  26,  40},
{ 65,  54,  38,  33,  31,  30,  38}, { 63,  55,  39,  33,  30,  32,  38},
{ 59,  58,  40,  33,  31,  31,  39}, { 54,  64,  40,  33,  31,  30,  40},
{ 49,  66,  40,  32,  32,  30,  41}, { 48,  64,  42,  32,  32,  30,  41},
{ 54,  46,  35,  30,  34,  39,  49}, { 64,  52,  36,  32,  34,  34,  35},
{ 65,  53,  37,  33,  32,  32,  37}, { 63,  55,  38,  33,  31,  31,  39},
{ 59,  60,  38,  33,  31,  31,  40}, { 54,  64,  38,  33,  32,  30,  40},
{ 49,  66,  39,  33,  32,  29,  41}, { 47,  64,  42,  32,  33,  29,  42},
{ 51,  46,  35,  31,  33,  37,  54}, { 61,  51,  36,  32,  33,  38,  36},
{ 63,  53,  37,  32,  32,  34,  37}, { 62,  55,  37,  33,  32,  32,  39},
{ 58,  59,  37,  33,  32,  31,  40}, { 53,  63,  38,  33,  32,  31,  40},
{ 49,  64,  40,  33,  33,  30,  41}, { 46,  62,  42,  33,  33,  30,  42}
},
```

– Otherwise, if mipSizeId is equal to 2 and modeId is equal to 1, the following applies:

mWeight[x][y] = (295)

```
{
{ 39,  34,  33,  58,  44,  31,  32}, { 60,  38,  32,  40,  51,  30,  31},
{ 73,  49,  31,  39,  48,  32,  31}, { 60,  73,  30,  39,  46,  33,  32},
{ 43,  87,  35,  38,  45,  33,  32}, { 35,  78,  54,  36,  45,  33,  32},
{ 33,  47,  86,  35,  44,  33,  32}, { 31,  17, 114,  34,  44,  34,  33},
{ 43,  37,  32,  53,  70,  30,  31}, { 53,  50,  30,  42,  72,  31,  30},
{ 52,  66,  30,  39,  70,  32,  30}, { 46,  78,  35,  37,  68,  34,  30},
{ 43,  75,  48,  37,  66,  34,  30}, { 40,  62,  68,  35,  65,  35,  30},
{ 33,  37,  97,  33,  62,  37,  31}, { 26,  14, 122,  32,  59,  38,  33},
{ 40,  39,  33,  34,  87,  37,  30}, { 45,  54,  32,  34,  84,  41,  29},
{ 41,  70,  35,  33,  83,  40,  29}, { 37,  73,  44,  32,  82,  40,  30},
{ 37,  65,  60,  31,  81,  41,  29}, { 35,  48,  82,  30,  79,  43,  29},
{ 28,  27, 108,  28,  76,  45,  30}, { 19,  11, 127,  27,  70,  46,  32},
{ 38,  40,  34,  27,  73,  62,  28}, { 39,  54,  35,  30,  73,  62,  28},
{ 33,  65,  41,  29,  75,  59,  28}, { 30,  65,  53,  27,  76,  58,  29},
{ 29,  53,  72,  26,  77,  58,  29}, { 27,  35,  95,  24,  77,  60,  28},
{ 19,  19, 117,  23,  74,  61,  30}, {  9,  16, 127,  23,  68,  60,  34},
{ 35,  40,  35,  29,  44,  89,  30}, { 33,  51,  39,  29,  49,  86,  30},
{ 28,  57,  49,  28,  53,  83,  30}, { 24,  52,  65,  26,  56,  82,  30},
{ 22,  39,  86,  24,  58,  82,  30}, { 18,  22, 108,  23,  59,  82,  31},
{ 10,  13, 125,  22,  58,  80,  33}, {  0,  19, 127,  22,  56,  74,  40},
{ 33,  40,  36,  31,  28,  90,  45}, { 29,  46,  44,  29,  31,  92,  43},
{ 24,  45,  58,  28,  34,  91,  43}, { 19,  37,  78,  26,  37,  91,  43},
{ 15,  22,  99,  25,  38,  91,  42}, { 11,  11, 118,  24,  39,  90,  44},
{  2,  11, 127,  23,  41,  85,  48}, {  0,  17, 127,  23,  43,  75,  55},
{ 31,  37,  39,  30,  28,  54,  82}, { 27,  37,  52,  28,  30,  58,  79},
{ 22,  30,  70,  27,  32,  58,  79}, { 15,  19,  91,  26,  33,  58,  79},
{ 10,   8, 111,  25,  34,  58,  79}, {  5,   2, 125,  25,  35,  57,  80},
{  0,   9, 127,  25,  36,  53,  84}, {  0,  13, 127,  25,  39,  47,  88},
{ 28,  29,  46,  28,  39,   2, 123}, { 24,  24,  62,  27,  41,   1, 125},
{ 19,  14,  81,  25,  43,   0, 126}, { 13,   4, 101,  24,  44,   0, 127},
{  6,   0, 116,  23,  45,   0, 127}, {  0,   0, 126,  23,  45,   1, 127},
{  0,   4, 127,  25,  44,   2, 127}, {  0,   9, 127,  25,  44,   3, 127}
},
```

– Otherwise, if mipSizeId is equal to 2 and modeId is equal to 2, the following applies:

mWeight[x][y] = (296)

```
{
{ 30,  32,  32,  42,  34,  32,  32}, { 63,  26,  34,  16,  38,  32,  32},
{ 98,  26,  34,  25,  34,  33,  32}, { 75,  61,  30,  31,  32,  33,  32},
{ 36,  94,  32,  30,  33,  32,  32}, { 26,  76,  58,  30,  33,  32,  32},
{ 30,  39,  91,  31,  32,  33,  31}, { 32,  23, 105,  32,  32,  32,  32},
{ 34,  30,  33,  31,  52,  29,  32}, { 66,  24,  34,  11,  41,  33,  32},
{ 97,  28,  34,  24,  34,  33,  32}, { 71,  65,  30,  30,  32,  33,  32},
{ 34,  92,  35,  30,  33,  32,  32}, { 26,  70,  64,  29,  34,  32,  32},
{ 30,  37,  94,  30,  33,  32,  31}, { 32,  23, 105,  31,  33,  33,  31},
{ 37,  29,  33,   8,  79,  27,  32}, { 71,  22,  35,   5,  50,  32,  32},
{ 98,  29,  34,  23,  34,  34,  32}, { 66,  70,  30,  31,  31,  33,  32},
{ 31,  92,  38,  30,  33,  32,  32}, { 26,  66,  68,  29,  34,  32,  31},
{ 30,  34,  97,  30,  34,  33,  31}, { 31,  22, 106,  30,  34,  33,  31},
{ 40,  28,  34,   0,  76,  46,  28}, { 76,  21,  35,   0,  55,  35,  32},
{ 97,  32,  34,  21,  37,  33,  33}, { 61,  75,  29,  30,  32,  32,  32},
{ 29,  92,  40,  29,  33,  32,  32}, { 26,  62,  73,  29,  34,  32,  31},
{ 29,  32,  99,  30,  34,  33,  30}, { 31,  22, 107,  30,  34,  33,  31},
{ 42,  27,  34,   1,  48,  79,  25}, { 80,  20,  35,   0,  48,  47,  31},
{ 94,  36,  32,  17,  40,  33,  33}, { 55,  80,  29,  27,  35,  31,  32},
{ 27,  90,  43,  28,  34,  32,  31}, { 26,  58,  76,  29,  33,  33,  30},
{ 29,  30, 101,  29,  34,  34,  30}, { 31,  21, 108,  29,  35,  34,  30},
{ 44,  26,  34,   6,  30,  80,  40}, { 81,  21,  35,   0,  41,  52,  35},
{ 90,  41,  31,  14,  41,  35,  33}, { 51,  82,  29,  24,  37,  32,  32},
{ 27,  87,  47,  27,  35,  32,  31}, { 26,  54,  79,  29,  34,  33,  30},
{ 29,  29, 102,  28,  34,  33,  30}, { 31,  21, 108,  28,  35,  33,  31},
{ 47,  26,  34,   7,  34,  44,  75}, { 80,  24,  34,   0,  41,  41,  50},
{ 84,  45,  31,  12,  40,  36,  36}, { 49,  81,  31,  22,  37,  33,  32},
{ 28,  81,  51,  26,  35,  33,  31}, { 28,  51,  81,  28,  34,  33,  30},
{ 29,  30, 101,  28,  35,  33,  31}, { 31,  22, 107,  28,  35,  33,  32},
{ 48,  27,  34,  10,  40,  16,  97}, { 75,  27,  34,   3,  42,  26,  66},
{ 77,  47,  33,  12,  40,  32,  43}, { 49,  75,  36,  21,  37,  33,  35},
{ 32,  72,  55,  25,  36,  33,  32}, { 30,  49,  81,  27,  35,  33,  31},
{ 30,  32,  98,  28,  35,  32,  32}, { 31,  24, 104,  28,  35,  32,  33}
},
```

– Otherwise, if mipSizeId is equal to 2 and modeId is equal to 3, the following applies:

mWeight[x][y] = (297)

```
{
```

```
{ 36,  29,  33,  43,  47,  29,  31}, { 74,  20,  35,  19,  47,  34,  32},
{ 92,  35,  32,  29,  31,  40,  34}, { 53,  80,  26,  33,  28,  36,  37},
{ 24,  91,  41,  31,  31,  31,  38}, { 25,  57,  74,  31,  32,  30,  37},
{ 32,  28,  99,  32,  32,  29,  36}, { 34,  20, 105,  33,  32,  30,  35},
{ 50,  26,  34,  33,  74,  30,  31}, { 75,  28,  33,  23,  46,  47,  33},
{ 64,  58,  29,  30,  26,  46,  40}, { 31,  85,  37,  31,  27,  33,  44},
{ 22,  67,  64,  30,  31,  28,  42}, { 29,  35,  93,  31,  32,  27,  40},
{ 33,  20, 105,  32,  33,  27,  37}, { 34,  19, 106,  33,  32,  29,  36},
{ 51,  29,  33,  25,  72,  51,  30}, { 61,  42,  31,  30,  31,  60,  39},
{ 40,  70,  34,  32,  24,  41,  50}, { 22,  72,  54,  30,  31,  27,  50},
{ 25,  44,  83,  30,  33,  25,  44}, { 32,  23, 102,  32,  33,  26,  40},
{ 34,  18, 107,  32,  33,  28,  37}, { 34,  19, 105,  33,  32,  30,  35},
{ 45,  35,  32,  30,  39,  79,  33}, { 43,  53,  33,  35,  24,  53,  55},
{ 27,  67,  45,  32,  29,  27,  61}, { 22,  53,  72,  30,  33,  22,  52},
{ 28,  31,  95,  31,  33,  25,  43}, { 32,  20, 105,  32,  33,  27,  38},
{ 34,  18, 107,  32,  32,  29,  36}, { 34,  20, 105,  33,  31,  31,  35},
{ 38,  40,  32,  35,  23,  72,  54}, { 31,  55,  39,  34,  29,  32,  73},
{ 22,  57,  60,  31,  35,  18,  64}, { 25,  39,  86,  31,  35,  22,  49},
{ 30,  24, 101,  32,  33,  27,  40}, { 33,  19, 106,  32,  32,  30,  36},
{ 34,  18, 107,  33,  31,  31,  35}, { 34,  20, 104,  33,  31,  32,  34},
{ 33,  42,  35,  34,  28,  39,  82}, { 26,  51,  50,  33,  34,  18,  80},
{ 23,  46,  74,  31,  35,  20,  59}, { 27,  32,  93,  32,  34,  26,  44},
{ 31,  22, 103,  32,  32,  30,  37}, { 33,  19, 106,  33,  31,  31,  35},
{ 34,  19, 106,  33,  31,  32,  34}, { 35,  21, 103,  34,  31,  32,  34},
{ 29,  41,  41,  33,  34,  20,  92}, { 24,  44,  62,  34,  35,  18,  73},
{ 24,  37,  83,  34,  33,  25,  52}, { 28,  28,  97,  33,  32,  30,  40},
{ 32,  23, 103,  33,  31,  32,  36}, { 34,  20, 105,  34,  30,  33,  34},
{ 35,  20, 104,  34,  30,  33,  33}, { 35,  22, 102,  34,  30,  33,  34},
{ 27,  38,  51,  34,  34,  20,  86}, { 26,  37,  71,  35,  34,  24,  64},
{ 27,  33,  87,  35,  32,  30,  47}, { 30,  28,  96,  34,  31,  32,  39},
{ 32,  24, 100,  35,  30,  32,  36}, { 34,  23, 101,  34,  30,  33,  34},
{ 35,  23, 101,  34,  30,  32,  34}, { 34,  24,  99,  35,  30,  33,  34}
},
```

– Otherwise, if mipSizeId is equal to 2 and modeId is equal to 4, the following applies:

mWeight[x][y] = (298)

```
{
{ 39,  30,  31,  67,  33,  34,  31}, { 72,  21,  32,  43,  39,  33,  31},
{100,  23,  32,  35,  39,  34,  31}, { 75,  63,  24,  32,  38,  34,  32},
{ 32,  98,  26,  29,  37,  35,  32}, { 22,  77,  55,  29,  36,  35,  31},
{ 31,  37,  90,  31,  35,  35,  32}, { 35,  22, 100,  33,  33,  36,  33},
{ 47,  29,  32,  74,  54,  32,  31}, { 71,  24,  32,  60,  50,  36,  30},
{ 86,  31,  30,  46,  48,  37,  30}, { 65,  63,  25,  34,  46,  39,  30},
{ 33,  85,  32,  28,  43,  40,  30}, { 26,  64,  60,  27,  39,  41,  30},
{ 33,  33,  87,  29,  35,  41,  31}, { 37,  23,  93,  32,  33,  41,  32},
{ 41,  32,  32,  45,  84,  32,  32}, { 55,  31,  32,  50,  70,  40,  30},
{ 62,  37,  31,  45,  61,  45,  29}, { 53,  55,  31,  36,  55,  48,  29},
{ 38,  63,  40,  29,  48,  50,  28}, { 34,  49,  60,  27,  43,  51,  29},
{ 38,  30,  78,  28,  38,  50,  31}, { 40,  24,  83,  30,  36,  48,  33},
{ 35,  33,  33,  29,  75,  58,  29}, { 39,  35,  33,  34,  68,  59,  29},
{ 41,  39,  34,  36,  61,  62,  29}, { 41,  43,  37,  33,  54,  64,  28},
{ 41,  43,  45,  30,  48,  65,  29}, { 42,  36,  56,  27,  44,  63,  30},
{ 42,  30,  65,  27,  41,  60,  33}, { 42,  28,  68,  28,  37,  56,  36},
{ 33,  34,  33,  31,  42,  88,  30}, { 31,  36,  34,  31,  44,  84,  31},
{ 31,  37,  35,  32,  43,  83,  31}, { 35,  35,  39,  32,  40,  82,  31},
{ 40,  32,  44,  31,  38,  81,  31}, { 44,  30,  48,  30,  37,  78,  33},
{ 44,  30,  52,  28,  37,  72,  36}, { 43,  30,  55,  29,  35,  66,  40},
{ 32,  33,  33,  34,  25,  85,  48}, { 30,  34,  34,  33,  25,  88,  44},
{ 30,  34,  36,  34,  25,  90,  41}, { 33,  32,  38,  34,  25,  90,  40},
{ 38,  29,  41,  34,  26,  88,  40}, { 42,  29,  41,  33,  27,  85,  41},
{ 43,  30,  42,  31,  28,  80,  43}, { 42,  31,  45,  31,  30,  72,  47},
{ 32,  33,  33,  33,  26,  54,  79}, { 31,  32,  34,  35,  20,  68,  68},
{ 32,  32,  35,  36,  17,  76,  62}, { 34,  31,  36,  36,  17,  79,  59},
{ 37,  29,  37,  36,  18,  78,  58}, { 39,  29,  37,  35,  20,  77,  58},
{ 41,  30,  37,  34,  22,  74,  58}, { 40,  31,  40,  32,  26,  68,  59},
{ 33,  31,  34,  33,  29,  31,  98}, { 34,  30,  34,  35,  23,  45,  88},
{ 34,  31,  34,  36,  20,  54,  82}, { 35,  31,  34,  36,  18,  59,  78},
{ 36,  31,  34,  37,  19,  60,  76}, { 38,  30,  34,  36,  20,  61,  74},
{ 39,  31,  35,  35,  22,  60,  73}, { 39,  31,  37,  34,  24,  59,  71}
},
```

– Otherwise (mipSizeId is equal to 2 and modeId is equal to 5), the following applies:

mWeight[x][y] = (299)

```
{
{ 30,  33,  32,  55,  32,  32,  32}, { 47,  30,  31,  29,  36,  32,  32},
```

```
{ 81,  28,  32,  28,  34,  32,  32}, { 85,  46,  29,  32,  32,  33,  32},
{ 54,  82,  26,  32,  32,  33,  32}, { 30,  90,  38,  31,  32,  33,  32},
{ 30,  56,  73,  31,  33,  32,  32}, { 37,  21, 102,  32,  32,  32,  32},
{ 33,  32,  31,  68,  39,  31,  31}, { 38,  32,  31,  43,  34,  33,  31},
{ 63,  30,  31,  29,  34,  32,  32}, { 82,  37,  30,  29,  33,  32,  32},
{ 71,  63,  27,  31,  32,  33,  32}, { 44,  86,  30,  30,  33,  33,  32},
{ 33,  72,  55,  30,  32,  32,  31}, { 37,  37,  86,  31,  32,  33,  31},
{ 34,  33,  32,  60,  61,  29,  32}, { 36,  33,  31,  56,  38,  32,  31},
{ 51,  30,  31,  38,  33,  33,  32}, { 75,  31,  31,  30,  33,  33,  32},
{ 80,  47,  29,  30,  32,  33,  31}, { 60,  73,  27,  30,  33,  33,  31},
{ 41,  78,  41,  30,  33,  32,  31}, { 38,  53,  68,  30,  32,  33,  31},
{ 33,  33,  32,  43,  77,  35,  30}, { 35,  33,  31,  55,  54,  29,  32},
{ 43,  32,  31,  46,  39,  31,  32}, { 64,  30,  31,  35,  34,  33,  32},
{ 79,  37,  30,  31,  32,  33,  31}, { 73,  57,  28,  30,  32,  33,  31},
{ 54,  73,  33,  30,  32,  33,  31}, { 43,  64,  52,  30,  32,  33,  31},
{ 33,  33,  32,  34,  68,  58,  28}, { 34,  33,  31,  45,  70,  33,  31},
{ 38,  33,  31,  48,  52,  29,  32}, { 54,  31,  31,  40,  39,  31,  32},
{ 73,  32,  31,  34,  34,  33,  31}, { 77,  45,  29,  31,  32,  32,  32},
{ 65,  63,  30,  31,  31,  33,  31}, { 51,  66,  42,  30,  32,  33,  31},
{ 33,  32,  32,  34,  44,  81,  31}, { 34,  33,  31,  38,  66,  52,  28},
{ 36,  33,  30,  44,  62,  34,  31}, { 47,  31,  31,  43,  48,  30,  32},
{ 64,  31,  31,  38,  38,  32,  32}, { 75,  38,  30,  33,  34,  32,  32},
{ 71,  53,  30,  31,  32,  33,  32}, { 59,  61,  37,  30,  32,  33,  32},
{ 33,  32,  31,  35,  31,  71,  54}, { 34,  33,  31,  37,  49,  70,  33},
{ 36,  33,  31,  41,  60,  48,  30}, { 43,  32,  31,  43,  54,  35,  31},
{ 56,  31,  31,  40,  44,  32,  32}, { 68,  35,  30,  36,  37,  32,  32},
{ 70,  45,  30,  33,  34,  33,  32}, { 63,  55,  35,  31,  33,  33,  32},
{ 33,  32,  31,  33,  34,  36,  87}, { 34,  32,  31,  36,  38,  62,  52},
{ 36,  33,  31,  39,  50,  57,  36}, { 41,  33,  31,  41,  53,  43,  33},
{ 50,  33,  31,  41,  48,  36,  32}, { 59,  35,  31,  37,  41,  34,  32},
{ 65,  42,  31,  35,  36,  33,  32}, { 62,  49,  35,  33,  34,  34,  33}
}
```

8.4.5.2.5 MIP prediction upsampling process

Inputs to this process are:

- a variable predSize specifying the input block size,
- matrix-based intra prediction samples predMip[x][y], with x = 0..predSize − 1, y = 0..predSize − 1,
- a variable nTbW specifying the transform block width,
- a variable nTbH specifying the transform block height,
- top reference samples refT[x] with x = 0..nTbW − 1,
- left reference samples refL[y] with y = 0..nTbH − 1.

Outputs of this process are the predicted samples predSamples[x][y], with x = 0..nTbW − 1, y = 0..nTbH − 1.

The sparse predicted samples predSamples[m][n] are derived from predMip[x][y], with x = 0..predSize − 1, y = 0..predSize − 1 as follows:

$$\text{upHor} = \text{nTbW} \,/\, \text{predSize} \tag{300}$$

$$\text{upVer} = \text{nTbH} \,/\, \text{predSize} \tag{301}$$

$$\text{predSamples}[\ (\ x + 1\ ) * \text{upHor} - 1\ ][\ (\ y + 1\ ) * \text{upVer} - 1\ ] = \text{predMip}[\ x\ ][\ y\ ] \tag{302}$$

The top reference samples refT[x] are assigned to predSamples[x][−1] with x = 0..nTbW − 1.

The left reference samples refL[y] are assigned to predSamples[−1][y] with y = 0..nTbH − 1.

The predicted samples predSamples[x][y], with x = 0..nTbW − 1, y = 0..nTbH − 1 are derived by the following ordered steps:

1. When upHor is greater than 1, horizontal upsampling for all sparse positions (xHor, yHor) = (m * upHor − 1, n * upVer − 1) with m = 0..predSize − 1, n = 1..predSize is applied with dX = 1..upHor − 1 as follows:

$$\text{sum} = (\ \text{upHor} - \text{dX}\ ) * \text{predSamples}[\ \text{xHor}\ ][\ \text{yHor}\ ] + \text{dX} * \text{predSamples}[\ \text{xHor} + \text{upHor}\ ][\ \text{yHor}\ ] \tag{303}$$

$$\text{predSamples}[\ \text{xHor} + \text{dX}\ ][\ \text{yHor}\ ] = (\ \text{sum} + \text{upHor}\ /2\ )\ /\ \text{upHor} \tag{304}$$

2. When upVer is greater than 1, vertical upsampling for all sparse positions (xVer, yVer) = (m, n * upVer − 1) with m = 0..nTbW − 1, n = 0..predSize − 1 is applied with dY = 1..upVer − 1 as follows:

$$\text{sum} = (\text{upVer} - \text{dY}) * \text{predSamples}[\text{xVer}][\text{yVer}] + \text{dY} * \text{predSamples}[\text{xVer}][\text{yVer} + \text{upVer}] \quad (305)$$

$$\text{predSamples}[\text{xVer}][\text{yVer} + \text{dY}] = (\text{sum} + \text{upVer} / 2) / \text{upVer} \quad (306)$$

8.4.5.2.6 General intra sample prediction

Inputs to this process are:

– a sample location (xTbCmp, yTbCmp) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture,

– a variable predModeIntra specifying the intra prediction mode,

– a variable nTbW specifying the transform block width,

– a variable nTbH specifying the transform block height,

– a variable nCbW specifying the coding block width,

– a variable nCbH specifying the coding block height,

– a variable cIdx specifying the colour component of the current block.

Outputs of this process are the predicted samples predSamples[x][y], with x = 0..nTbW − 1, y = 0..nTbH − 1.

The variables refW and refH are derived as follows:

– If IntraSubPartitionsSplitType is equal to ISP_NO_SPLIT or cIdx is not equal to 0, the following applies:

$$\text{refW} = \text{nTbW} * 2 \quad (307)$$

$$\text{refH} = \text{nTbH} * 2 \quad (308)$$

– Otherwise (IntraSubPartitionsSplitType is not equal to ISP_NO_SPLIT and cIdx is equal to 0), the following applies:

$$\text{refW} = \text{nCbW} + \text{nTbW} \quad (309)$$

$$\text{refH} = \text{nCbH} + \text{nTbH} \quad (310)$$

The variable refIdx specifying the intra prediction reference line index is derived as follows:

$$\text{refIdx} = (\text{cIdx} == 0) \ ? \ \text{IntraLumaRefLineIdx}[\text{xTbCmp}][\text{yTbCmp}] : 0 \quad (311)$$

The wide angle intra prediction mode mapping process as specified in clause 8.4.5.2.7 is invoked with predModeIntra, nCbW, nCbH, nTbW, nTbH and cIdx as inputs, and the modified predModeIntra as output.

The variable refFilterFlag is derived as follows:

– If predModeIntra is equal to 0, −14, −12, −10, −6, 2, 34, 66, 72, 76, 78, or 80, refFilterFlag is set equal to 1.

– Otherwise, refFilterFlag is set equal to 0.

For the generation of the reference samples p[x][y] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx, the following ordered steps apply:

1. The reference sample availability marking process as specified in clause 8.4.5.2.8 is invoked with the sample location (xTbCmp, yTbCmp), the intra prediction reference line index refIdx, the reference sample width refW, the reference sample height refH, the colour component index cIdx as inputs, and the reference samples refUnfilt[x][y] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = − refIdx..refW − 1, y = −1 − refIdx as output.

2. When at least one sample refUnfilt[x][y] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx is marked as "not available for intra prediction", the reference sample substitution process as specified in clause 8.4.5.2.9 is invoked with the intra prediction reference line index refIdx, the reference sample width refW, the reference sample height refH, the reference samples refUnfilt[x][y] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx, and

the colour component index cIdx as inputs, and the modified reference samples refUnfilt[x][y] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx as output.

3. The reference sample filtering process as specified in clause 8.4.5.2.10 is invoked with the intra prediction reference line index refIdx, the transform block width nTbW and height nTbH, the reference sample width refW, the reference sample height refH, the reference filter flag refFilterFlag, the unfiltered samples refUnfilt[x][y] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx, and the colour component index cIdx as inputs, and the reference samples p[x][y] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx as output.

The intra sample prediction process according to predModeIntra applies as follows:

– If predModeIntra is equal to INTRA_PLANAR, the corresponding intra prediction mode process specified in clause 8.4.5.2.11 is invoked with the transform block width nTbW, and the transform block height nTbH, and the reference sample array p as inputs, and the output is the predicted sample array predSamples.

– Otherwise, if predModeIntra is equal to INTRA_DC, the corresponding intra prediction mode process specified in clause 8.4.5.2.12 is invoked with the transform block width nTbW, the transform block height nTbH, the intra prediction reference line index refIdx, and the reference sample array p as inputs, and the output is the predicted sample array predSamples.

– Otherwise, if predModeIntra is equal to INTRA_LT_CCLM, INTRA_L_CCLM or INTRA_T_CCLM, the corresponding intra prediction mode process specified in clause 8.4.5.2.14 is invoked with the intra prediction mode predModeIntra, the sample location (xTbC, yTbC) set equal to (xTbCmp, yTbCmp), the transform block width nTbW and height nTbH, the colour component index cIdx, and the reference sample array p as inputs, and the output is the predicted sample array predSamples.

– Otherwise, the corresponding intra prediction mode process specified in clause 8.4.5.2.13 is invoked with the intra prediction mode predModeIntra, the intra prediction reference line index refIdx, the transform block width nTbW, the transform block height nTbH, the reference sample width refW, the reference sample height refH, the coding block width nCbW and height nCbH, the reference filter flag refFilterFlag, the colour component index cIdx, and the reference sample array p as inputs, and the predicted sample array predSamples as outputs.

When all of the following conditions are true, the position-dependent prediction sample filtering process specified in clause 8.4.5.2.15 is invoked with the intra prediction mode predModeIntra, the transform block width nTbW, the transform block height nTbH, the predicted samples predSamples[x][y], with x = 0..nTbW − 1, y = 0..nTbH − 1, the reference sample width refW, the reference sample height refH, and the reference samples p[x][y], with x = −1, y = −1..refH − 1 and x = 0..refW − 1, y = −1 as inputs, and the output is the modified predicted sample array predSamples:

– nTbW is greater than or equal to 4 and nTbH is greater than or equal to 4;

– refIdx is equal to 0;

– BdpcmFlag[xTbCmp * (cIdx > 0 ? SubWidthC : 1)][yTbCmp * (cIdx > 0 ? SubHeightC : 1)][cIdx] is equal to 0;

– One of the following conditions is true:

  – predModeIntra is equal to INTRA_PLANAR;

  – predModeIntra is equal to INTRA_DC;

  – predModeIntra is less than or equal to INTRA_ANGULAR18;

  – predModeIntra is greater than or equal to INTRA_ANGULAR50 and less than INTRA_LT_CCLM.

8.4.5.2.7 Wide angle intra prediction mode mapping process

Inputs to this process are:

– a variable predModeIntra specifying the intra prediction mode,

– a variable nCbW specifying the coding block width,

– a variable nCbH specifying the coding block height,

– a variable nTbW specifying the transform block width,

– a variable nTbH specifying the transform block height,

– a variable cIdx specifying the colour component of the current block.

Output of this process is the modified intra prediction mode predModeIntra.

The variables nW and nH are derived as follows:

– If IntraSubPartitionsSplitType is equal to ISP_NO_SPLIT or cIdx is not equal to 0, the following applies:

$$nW = nTbW \quad (312)$$

$$nH = nTbH \quad (313)$$

– Otherwise (IntraSubPartitionsSplitType is not equal to ISP_NO_SPLIT and cIdx is equal to 0), the following applies:

$$nW = nCbW \quad (314)$$

$$nH = nCbH \quad (315)$$

The variable whRatio is set equal to Abs(Log2(nW) − Log2(nH)).

For non-square blocks (nW is not equal to nH), the intra prediction mode predModeIntra is modified as follows:

– If all of the following conditions are true, predModeIntra is set equal to (predModeIntra + 65):
  – nW is greater than nH;
  – predModeIntra is greater than or equal to 2;
  – predModeIntra is less than (whRatio > 1) ? (8 + 2 * whRatio) : 8.
– Otherwise, if all of the following conditions are true, predModeIntra is set equal to (predModeIntra − 67):
  – nH is greater than nW;
  – predModeIntra is less than or equal to 66;
  – predModeIntra is greater than (whRatio > 1) ? (60 − 2 * whRatio) : 60.

8.4.5.2.8 Reference sample availability marking process

Inputs to this process are:

– a sample location (xTbCmp, yTbCmp) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture,
– a variable refIdx specifying the intra prediction reference line index,
– a variable refW specifying the width of the reference area in units of samples,
– a variable refH specifying the height of the reference area in units of samples,
– a variable cIdx specifying the colour component of the current block.

Outputs of this process are the reference samples refUnfilt[x][y] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx for intra sample prediction.

The refW + refH + 1 + (2 * refIdx) neighbouring samples refUnfilt[x][y] that are constructed samples prior to the in-loop filter process, with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx, are derived as follows:

– The neighbouring location (xNbCmp, yNbCmp) is specified by:

$$(xNbCmp, yNbCmp) = (xTbCmp + x, yTbCmp + y) \quad (316)$$

– The current luma location (xTbY, yTbY) and the neighbouring luma location (xNbY, yNbY) are derived as follows:

$$\begin{aligned}(xTbY, yTbY) = (cIdx == 0) \ ? \ & (xTbCmp, yTbCmp) : \\ & (xTbCmp * SubWidthC, yTbCmp * SubHeightC)\end{aligned} \quad (317)$$

$$\begin{aligned}(xNbY, yNbY) = (cIdx == 0) \ ? \ & (xNbCmp, yNbCmp) : \\ & (xNbCmp * SubWidthC, yNbCmp * SubHeightC)\end{aligned} \quad (318)$$

– The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xTbY, yTbY), the neighbouring luma location (xNbY, yNbY), checkPredModeY set equal to FALSE, and cIdx as inputs, and the output is assigned to availableN.

– Each sample refUnfilt[x][y] is derived as follows:

  – If availableN is equal to FALSE, the sample refUnfilt[x][y] is marked as "not available for intra prediction".

  – Otherwise, the sample refUnfilt[x][y] is marked as "available for intra prediction" and the sample at the location (xNbCmp, yNbCmp) is assigned to refUnfilt[x][y].

8.4.5.2.9 Reference sample substitution process

Inputs to this process are:

– a variable refIdx specifying the intra prediction reference line index,

– a variable refW specifying the width of the reference area in units of samples,

– a variable refH specifying the height of the reference area in units of samples,

– reference samples refUnfilt[x][y] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx for intra sample prediction.

Outputs of this process are the modified reference samples refUnfilt[x][y] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx for intra sample prediction.

The values of the samples refUnfilt[x][y] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx are modified as follows:

– If all samples refUnfilt[x][y] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx are marked as "not available for intra prediction", all values of refUnfilt[x][y] are set equal to 1 << (BitDepth − 1).

– Otherwise (at least one but not all samples refUnfilt[x][y] are marked as "not available for intra prediction"), the following ordered steps apply:

  1. When refUnfilt[−1 − refIdx][refH − 1] is marked as "not available for intra prediction", search sequentially starting from x = −1 − refIdx, y = refH − 1 to x = −1 − refIdx, y = −1 − refIdx, then from x = −refIdx, y = −1 − refIdx to x = refW − 1, y = −1 − refIdx, for a sample refUnfilt[x][y] that is marked as "available for intra prediction". Once a sample refUnfilt[x][y] marked as "available for intra prediction" is found, the search is terminated and the value of refUnfilt[−1 − refIdx][refH − 1] is set equal to the value of refUnfilt[x][y].

  2. For x = −1 − refIdx, y = −(refH − 2)..1 + refIdx, when refUnfilt[x][−y] is marked as "not available for intra prediction", the value of refUnfilt[x][−y] is set equal to the value of refUnfilt[x][−y + 1].

  3. For x = −refIdx..refW − 1, y = −1 − refIdx, when refUnfilt[x][y] is marked as "not available for intra prediction", the value of refUnfilt[x][y] is set equal to the value of refUnfilt[x − 1][y].

All samples refUnfilt[x][y] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx are marked as "available for intra prediction".

8.4.5.2.10 Reference sample filtering process

Inputs to this process are:

– a variable refIdx specifying the intra prediction reference line index,

– a variable nTbW specifying the transform block width,

– a variable nTbH specifying the transform block height,

– a variable refW specifying the reference samples width,

– a variable refH specifying the reference samples height,

– a variable refFilterFlag specifying the value of reference filter flag,

– the (unfiltered) neighbouring samples refUnfilt[x][y], with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx,

– a variable cIdx specifying the colour component of the current block.

Outputs of this process are the reference samples p[x][y], with $x = -1 - \text{refIdx}$, $y = -1 - \text{refIdx..refH} - 1$ and $x = -\text{refIdx..refW} - 1$, $y = -1 - \text{refIdx}$.

The variable filterFlag is derived as follows:

– If all of the following conditions are true, filterFlag is set equal to 1:
  – refIdx is equal to 0;
  – nTbW * nTbH is greater than 32;
  – cIdx is equal to 0;
  – IntraSubPartitionsSplitType is equal to ISP_NO_SPLIT;
  – refFilterFlag is equal to 1;
– Otherwise, filterFlag is set equal to 0.

For the derivation of the reference samples p[x][y] the following applies:

– If filterFlag is equal to 1, the filtered sample values p[x][y] with $x = -1$, $y = -1..\text{refH} - 1$ and $x = 0..\text{refW} - 1$, $y = -1$ are derived as follows:

$$p[\,-1\,][\,-1\,] = (\,\text{refUnfilt}[\,-1\,][\,0\,] + 2 * \text{refUnfilt}[\,-1\,][\,-1\,] + \text{refUnfilt}[\,0\,][\,-1\,] + 2\,) >> 2 \quad (319)$$

$$p[\,-1\,][\,y\,] = (\,\text{refUnfilt}[\,-1\,][\,y + 1\,] + 2 * \text{refUnfilt}[\,-1\,][\,y\,] + \text{refUnfilt}[\,-1\,][\,y - 1\,] + 2\,) >> 2 \quad \text{for } y = 0..\text{refH} - 2 \quad (320)$$

$$p[\,-1\,][\,\text{refH} - 1\,] = \text{refUnfilt}[\,-1\,][\,\text{refH} - 1\,] \quad (321)$$

$$p[\,x\,][\,-1\,] = (\,\text{refUnfilt}[\,x - 1\,][\,-1\,] + 2 * \text{refUnfilt}[\,x\,][\,-1\,] + \text{refUnfilt}[\,x + 1\,][\,-1\,] + 2\,) >> 2 \quad \text{for } x = 0..\text{refW} - 2 \quad (322)$$

$$p[\,\text{refW} - 1\,][\,-1\,] = \text{refUnfilt}[\,\text{refW} - 1\,][\,-1\,] \quad (323)$$

– Otherwise, the reference samples values p[x][y] are set equal to the unfiltered sample values refUnfilt[x][y] with $x = -1 - \text{refIdx}$, $y = -1 - \text{refIdx..refH} - 1$ and $x = -\text{refIdx..refW} - 1$, $y = -1 - \text{refIdx}$.

8.4.5.2.11 Specification of INTRA_PLANAR intra prediction mode

Inputs to this process are:

– a variable nTbW specifying the transform block width,
– a variable nTbH specifying the transform block height,
– the neighbouring samples p[x][y], with $x = -1$, $y = -1..\text{nTbH}$ and $x = 0..\text{nTbW}$, $y = -1$.

Outputs of this process are the predicted samples predSamples[x][y], with $x = 0..\text{nTbW} - 1$, $y = 0..\text{nTbH} - 1$.

The values of the prediction samples predSamples[x][y], with $x = 0..\text{nTbW} - 1$ and $y = 0..\text{nTbH} - 1$, are derived as follows:

$$\text{predV}[\,x\,][\,y\,] = (\,(\,\text{nTbH} - 1 - y\,) * p[\,x\,][\,-1\,] + (\,y + 1\,) * p[\,-1\,][\,\text{nTbH}\,]\,) << \text{Log2}(\,\text{nTbW}\,) \quad (324)$$

$$\text{predH}[\,x\,][\,y\,] = (\,(\,\text{nTbW} - 1 - x\,) * p[\,-1\,][\,y\,] + (\,x + 1\,) * p[\,\text{nTbW}\,][\,-1\,]\,) << \text{Log2}(\,\text{nTbH}\,) \quad (325)$$

$$\text{predSamples}[\,x\,][\,y\,] = (\,\text{predV}[\,x\,][\,y\,] + \text{predH}[\,x\,][\,y\,] + \text{nTbW} * \text{nTbH}\,) >> (\,\text{Log2}(\,\text{nTbW}\,) + \text{Log2}(\,\text{nTbH}\,) + 1\,) \quad (326)$$

8.4.5.2.12 Specification of INTRA_DC intra prediction mode

Inputs to this process are:

– a variable nTbW specifying the transform block width,
– a variable nTbH specifying the transform block height,
– a variable refIdx specifying the intra prediction reference line index,
– the neighbouring samples p[x][y], with $x = -1 - \text{refIdx}$, $y = 0..\text{nTbH} - 1$ and $x = 0..\text{nTbW} - 1$, $y = -1 - \text{refIdx}$.

Outputs of this process are the predicted samples predSamples[x][y], with x = 0..nTbW − 1, y = 0..nTbH − 1.

The values of the prediction samples predSamples[x][y], with x = 0..nTbW − 1, y = 0..nTbH − 1, are derived by the following ordered steps:

1. A variable dcVal is derived as follows:

   – When nTbW is equal to nTbH:

$$\text{dcVal} = \left(\sum_{x'=0}^{\text{nTbW}-1} \text{p}[x'][-1-\text{refIdx}] + \sum_{y'=0}^{\text{nTbH}-1} \text{p}[-1-\text{refIdx}][y'] + \text{nTbW} \right) \gg (\text{Log2}(\text{nTbW}) + 1) \quad (327)$$

   – When nTbW is greater than nTbH:

$$\text{dcVal} = \left(\sum_{x'=0}^{\text{nTbW}-1} \text{p}[x'][-1-\text{refIdx}] + (\text{nTbW} \gg 1) \right) \gg \text{Log2}(\text{nTbW}) \quad (328)$$

   – When nTbW is less than nTbH:

$$\text{dcVal} = \left(\sum_{y'=0}^{\text{nTbH}-1} \text{p}[-1-\text{refIdx}][y'] + (\text{nTbH} \gg 1) \right) \gg \text{Log2}(\text{nTbH}) \quad (329)$$

2. The prediction samples predSamples[x][y] are derived as follows:

$$\text{predSamples}[x][y] = \text{dcVal, with } x = 0..\text{nTbW} - 1,\ y = 0..\ \text{nTbH} - 1 \quad (330)$$

8.4.5.2.13 Specification of INTRA_ANGULAR2..INTRA_ANGULAR66 intra prediction modes

Inputs to this process are:

– the intra prediction mode predModeIntra,

– a variable refIdx specifying the intra prediction reference line index,

– a variable nTbW specifying the transform block width,

– a variable nTbH specifying the transform block height,

– a variable refW specifying the reference samples width,

– a variable refH specifying the reference samples height,

– a variable nCbW specifying the coding block width,

– a variable nCbH specifying the coding block height,

– a variable refFilterFlag specifying the value of reference filter flag,

– a variable cIdx specifying the colour component of the current block,

– the neighbouring samples p[x][y], with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx.

Outputs of this process are the predicted samples predSamples[x][y], with x = 0..nTbW − 1, y = 0..nTbH − 1.

The variable nTbS is set equal to (Log2 (nTbW) + Log2 (nTbH)) >> 1.

The variable filterFlag is derived as follows:

– If one or more of the following conditions are true, filterFlag is set equal to 0:

  – refFilterFlag is equal to 1;

  – refIdx is not equal to 0;

  – IntraSubPartitionsSplitType is not equal to ISP_NO_SPLIT.

– Otherwise, the following applies:

  – The variable minDistVerHor is set equal to Min(Abs(predModeIntra − 50), Abs(predModeIntra − 18)).

  – The variable intraHorVerDistThres[nTbS] is specified in Table 23.

  – The variable filterFlag is derived as follows:

– If minDistVerHor is greater than intraHorVerDistThres[nTbS], filterFlag is set equal to 1.

– Otherwise, filterFlag is set equal to 0.

Table 23 – Specification of intraHorVerDistThres[nTbS] for various transform block sizes nTbS

	nTbS = 2	nTbS = 3	nTbS = 4	nTbS = 5	nTbS = 6
intraHorVerDistThres[nTbS]	24	14	2	0	0

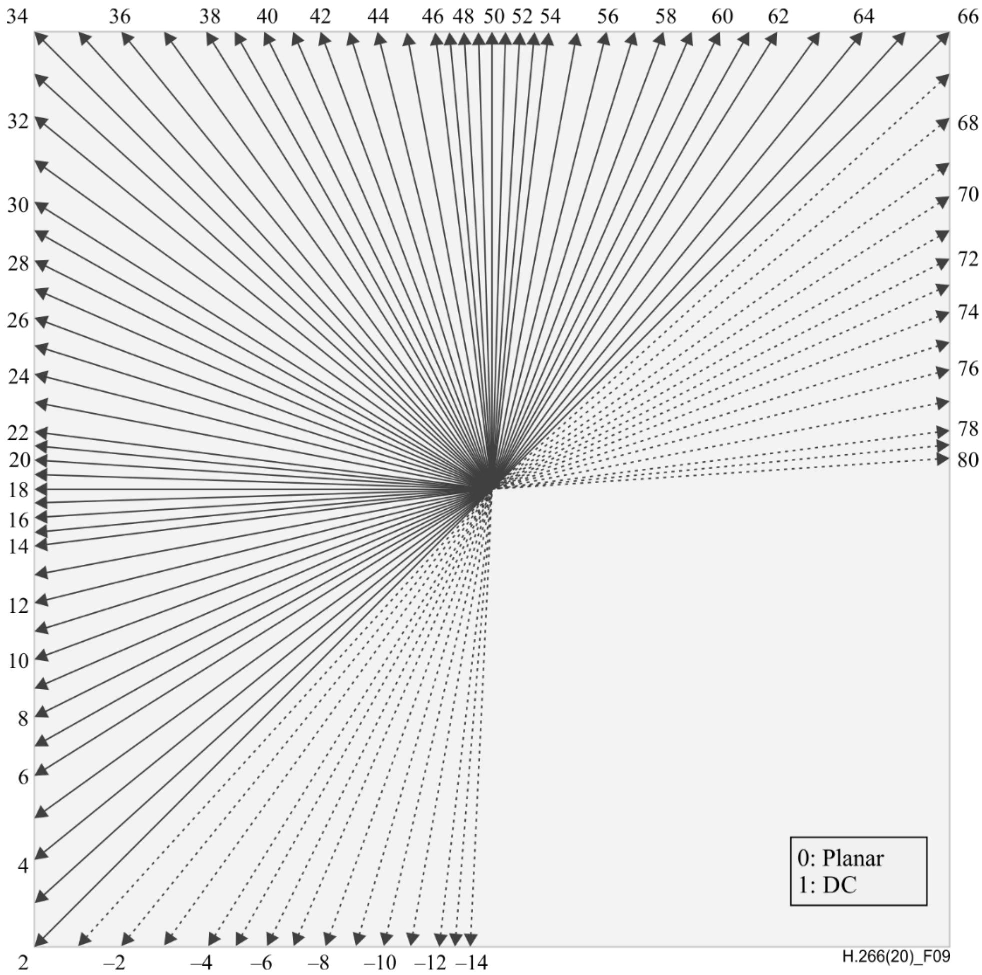

Figure 9 – Intra prediction directions (informative)

Figure 9 illustrates the 93 prediction directions, where the dashed directions are associated with the wide-angle modes that are only applied to non-square blocks.

Table 24 specifies the mapping table between predModeIntra and the angle parameter intraPredAngle.

Table 24 – Specification of intraPredAngle

predModeIntra	**−14**	**−13**	**−12**	**−11**	**−10**	**−9**	**−8**	**−7**	**−6**	**−5**	**−4**	**−3**	**−2**	**−1**	**2**	**3**	**4**
intraPredAngle	512	341	256	171	128	102	86	73	64	57	51	45	39	35	32	29	26
predModeIntra	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**
intraPredAngle	23	20	18	16	14	12	10	8	6	4	3	2	1	0	−1	−2	−3
predModeIntra	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**
intraPredAngle	−4	−6	−8	−10	−12	−14	−16	−18	−20	−23	−26	−29	−32	−29	−26	−23	−20
predModeIntra	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**
intraPredAngle	−18	−16	−14	−12	−10	−8	−6	−4	−3	−2	−1	0	1	2	3	4	6
predModeIntra	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**	**64**	**65**	**66**	**67**	**68**	**69**	**70**	**71**	**72**
intraPredAngle	8	10	12	14	16	18	20	23	26	29	32	35	39	45	51	57	64
predModeIntra	**73**	**74**	**75**	**76**	**77**	**78**	**79**	**80**									
intraPredAngle	73	86	102	128	171	256	341	512									

When intraPredAngle is not equal to 0, the inverse angle parameter invAngle is derived based on intraPredAngle as follows:

$$\text{invAngle} = \text{Round}\left(\frac{512*32}{\text{intraPredAngle}}\right) \tag{331}$$

The interpolation filter coefficients fC[phase][j] and fG[phase][j] with phase = 0..31 and j = 0..3 are specified in Table 25.

Table 25 – Specification of interpolation filter coefficients fC and fG

Fractional sample position p	fC interpolation filter coefficients				fG interpolation filter coefficients			
	fC[p][0]	fC[p][1]	fC[p][2]	fC[p][3]	fG[p][0]	fG[p][1]	fG[p][2]	fG[p][3]
0	0	64	0	0	16	32	16	0
1	−1	63	2	0	16	32	16	0
2	−2	62	4	0	15	31	17	1
3	−2	60	7	−1	15	31	17	1
4	−2	58	10	−2	14	30	18	2
5	−3	57	12	−2	14	30	18	2
6	−4	56	14	−2	13	29	19	3
7	−4	55	15	−2	13	29	19	3
8	−4	54	16	−2	12	28	20	4
9	−5	53	18	−2	12	28	20	4
10	−6	52	20	−2	11	27	21	5
11	−6	49	24	−3	11	27	21	5
12	−6	46	28	−4	10	26	22	6
13	−5	44	29	−4	10	26	22	6
14	−4	42	30	−4	9	25	23	7
15	−4	39	33	−4	9	25	23	7
16	−4	36	36	−4	8	24	24	8
17	−4	33	39	−4	8	24	24	8
18	−4	30	42	−4	7	23	25	9
19	−4	29	44	−5	7	23	25	9
20	−4	28	46	−6	6	22	26	10
21	−3	24	49	−6	6	22	26	10
22	−2	20	52	−6	5	21	27	11
23	−2	18	53	−5	5	21	27	11
24	−2	16	54	−4	4	20	28	12
25	−2	15	55	−4	4	20	28	12
26	−2	14	56	−4	3	19	29	13
27	−2	12	57	−3	3	19	29	13
28	−2	10	58	−2	2	18	30	14
29	−1	7	60	−2	2	18	30	14
30	0	4	62	−2	1	17	31	15
31	0	2	63	−1	1	17	31	15

The values of the prediction samples predSamples[x][y], with x = 0..nTbW − 1, y = 0..nTbH − 1 are derived as follows:

– If predModeIntra is greater than or equal to 34, the following ordered steps apply:

1. The reference sample array ref[x] is specified as follows:

   – The following applies:

   ref[x] = p[−1 − refIdx + x][−1 − refIdx], with x = 0..nTbW + refIdx + 1 (332)

   – If intraPredAngle is less than 0, the main reference sample array is extended as follows:

   ref[x] = p[−1 − refIdx][−1 − refIdx + Min((x * invAngle + 256) >> 9, nTbH)],
   with x = −nTbH..−1 (333)

   – Otherwise, the following applies:

   ref[x] = p[−1 − refIdx + x][−1 − refIdx], with x = nTbW + 2 + refIdx..refW + refIdx (334)

     – The additional samples ref[refW + refIdx +x] with x = 1..(Max(1, nTbW / nTbH) * refIdx + 2) are derived as follows:

$$\text{ref[refW + refIdx + x] = p[−1 + refW][−1 − refIdx]} \tag{335}$$

2. The values of the prediction samples predSamples[x][y], with x = 0..nTbW − 1, y = 0..nTbH − 1 are derived as follows:
   – The index variable iIdx and the multiplication factor iFact are derived as follows:

$$\text{iIdx = (((y + 1 + refIdx) * intraPredAngle) >> 5) + refIdx} \tag{336}$$

$$\text{iFact = ((y + 1 + refIdx) * intraPredAngle) \& 31} \tag{337}$$

   – If cIdx is equal to 0, the following applies:
      – The interpolation filter coefficients fT[j] with j = 0..3 are derived as follows:

$$\text{fT[j] = filterFlag ? fG[iFact][j] : fC[iFact][j]} \tag{338}$$

      – The value of the prediction samples predSamples[x][y] is derived as follows:

$$\text{predSamples[x][y] = Clip1(((} \textstyle\sum_{i=0}^{3} \text{fT[i]} * \text{ref[x + iIdx + i]) + 32) >> 6)} \tag{339}$$

   – Otherwise (cIdx is not equal to 0), depending on the value of iFact, the following applies:
      – If iFact is not equal to 0, the value of the prediction samples predSamples[x][y] is derived as follows:

$$\begin{aligned}&\text{predSamples[x][y] =}\\&\quad\text{((32 − iFact) * ref[x + iIdx + 1] + iFact * ref[x + iIdx + 2] + 16) >> 5}\end{aligned} \tag{340}$$

      – Otherwise, the value of the prediction samples predSamples[x][y] is derived as follows:

$$\text{predSamples[x][y] = ref[x + iIdx + 1]} \tag{341}$$

– Otherwise (predModeIntra is less than 34), the following ordered steps apply:

1. The reference sample array ref[x] is specified as follows:
   – The following applies:

$$\text{ref[x] = p[−1 − refIdx][−1 − refIdx + x], with x = 0..nTbH + refIdx + 1} \tag{342}$$

   – If intraPredAngle is less than 0, the main reference sample array is extended as follows:

$$\begin{aligned}&\text{ref[x] = p[−1 − refIdx + Min((x * invAngle + 256) >> 9, nTbW)][−1 − refIdx],}\\&\qquad\text{with x = −nTbW..−1}\end{aligned} \tag{343}$$

   – Otherwise, the following applies:

$$\text{ref[x] = p[−1 − refIdx][−1 − refIdx + x], with x = nTbH + 2 + refIdx..refH + refIdx} \tag{344}$$

      – The additional samples ref[refH + refIdx + x] with x = 1..(Max(1, nTbH / nTbW) * refIdx + 2) are derived as follows:

$$\text{ref[refH + refIdx +x] = p[−1 − refIdx][−1 + refH]} \tag{345}$$

2. The values of the prediction samples predSamples[x][y], with x = 0..nTbW − 1, y = 0..nTbH − 1 are derived as follows:
   – The index variable iIdx and the multiplication factor iFact are derived as follows:

$$\text{iIdx = (((x + 1 + refIdx) * intraPredAngle) >> 5) + refIdx} \tag{346}$$

$$\text{iFact = ((x + 1 + refIdx) * intraPredAngle) \& 31} \tag{347}$$

   – If cIdx is equal to 0, the following applies:
      – The interpolation filter coefficients fT[j] with j = 0..3 are derived as follows:

$$fT[j] = filterFlag ? fG[iFact][j] : fC[iFact][j] \quad (348)$$

– The value of the prediction samples predSamples[x][y] is derived as follows:

$$predSamples[x][y] = Clip1(((\sum_{i=0}^{3} fT[i] * ref[y + iIdx + i]) + 32) >> 6) \quad (349)$$

– Otherwise (cIdx is not equal to 0), depending on the value of iFact, the following applies:

– If iFact is not equal to 0, the value of the prediction samples predSamples[x][y] is derived as follows:

$$predSamples[x][y] = ((32 - iFact) * ref[y + iIdx + 1] + iFact * ref[y + iIdx + 2] + 16) >> 5 \quad (350)$$

– Otherwise, the value of the prediction samples predSamples[x][y] is derived as follows:

$$predSamples[x][y] = ref[y + iIdx + 1] \quad (351)$$

8.4.5.2.14 Specification of INTRA_LT_CCLM, INTRA_L_CCLM and INTRA_T_CCLM intra prediction mode

Inputs to this process are:

– the intra prediction mode predModeIntra,

– a sample location (xTbC, yTbC) of the top-left sample of the current transform block relative to the top-left sample of the current picture,

– a variable nTbW specifying the transform block width,

– a variable nTbH specifying the transform block height,

– a variable cIdx specifying the colour component of the current block,

– neighbouring chroma samples p[x][y], with x = −1, y = −1..2 * nTbH − 1 and x = 0.. 2 * nTbW − 1, y = −1.

Outputs of this process are predicted samples predSamples[x][y], with x = 0..nTbW − 1, y = 0..nTbH − 1.

The current luma location (xTbY, yTbY) is derived as follows:

$$(xTbY, yTbY) = (xTbC << (SubWidthC - 1), yTbC << (SubHeightC - 1)) \quad (352)$$

The variables availL and availT are derived as follows:

– The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xTbY, yTbY), the neighbouring luma location (xTbY − 1, yTbY), checkPredModeY set equal to FALSE, and cIdx as inputs, and the output is assigned to availL.

– The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xTbY, yTbY), the neighbouring luma location (xTbY, yTbY − 1), checkPredModeY set equal to FALSE, and cIdx as inputs, and the output is assigned to availT.

The number of available top-right neighbouring chroma samples numTopRight is derived as follows:

– The variable numTopRight is set equal to 0 and availTR is set equal to TRUE.

– When predModeIntra is equal to INTRA_T_CCLM, the following applies for x = nTbW..2 * nTbW − 1 until availTR is equal to FALSE:

  – The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xTbY, yTbY) the neighbouring luma location (xTbY + x * SubWidthC, yTbY − 1), checkPredModeY set equal to FALSE, and cIdx as inputs, and the output is assigned to availTR.

  – When availTR is equal to TRUE, numTopRight is incremented by one.

The number of available left-below neighbouring chroma samples numLeftBelow is derived as follows:

– The variable numLeftBelow is set equal to 0 and availLB is set equal to TRUE.

– When predModeIntra is equal to INTRA_L_CCLM, the following applies for y = nTbH..2 * nTbH − 1 until availLB is equal to FALSE:

– The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xTbY, yTbY), the neighbouring luma location (xTbY − 1, yTbY + y * SubHeightC), checkPredModeY set equal to FALSE, and cIdx as inputs, and the output is assigned to availLB.

– When availLB is equal to TRUE, numLeftBelow is incremented by one.

The number of available neighbouring chroma samples on the top and top-right numSampT and the number of available neighbouring chroma samples on the left and left-below numSampL are derived as follows:

– If predModeIntra is equal to INTRA_LT_CCLM, the following applies:

numSampT = availT ? nTbW : 0 (353)

numSampL = availL ? nTbH : 0 (354)

– Otherwise, the following applies:

numSampT = (availT && predModeIntra = = INTRA_T_CCLM) ?
(nTbW +Min(numTopRight, nTbH)) : 0 (355)

numSampL = (availL && predModeIntra = = INTRA_L_CCLM) ? (nTbH +
Min(numLeftBelow, nTbW)) : 0 (356)

The variable bCTUboundary is derived as follows:

bCTUboundary = ((yTbY & (CtbSizeY − 1)) = = 0) ? TRUE : FALSE (357)

The variable cntN and array pickPosN with N being replaced by L and T, are derived as follows:

– The variable numIs4N is derived as follows:

numIs4N = ((availT && availL && predModeIntra = = INTRA_LT_CCLM) ? 0 : 1) (358)

– The variable startPosN is set equal to numSampN >> (2 + numIs4N).

– The variable pickStepN is set equal to Max(1, numSampN >> (1 + numIs4N)).

– If availN is equal to TRUE and predModeIntra is equal to INTRA_LT_CCLM or INTRA_N_CCLM, the following assignments are made:

  – cntN is set equal to Min(numSampN, (1 + numIs4N) << 1).

  – pickPosN[pos] is set equal to (startPosN + pos * pickStepN), with pos = 0.. cntN − 1.

– Otherwise, cntN is set equal to 0.

The prediction samples predSamples[x][y] with x = 0..nTbW − 1, y = 0..nTbH − 1 are derived as follows:

– If both numSampL and numSampT are equal to 0, the following applies:

predSamples[x][y] = 1 << (BitDepth − 1) (359)

– Otherwise, the following ordered steps apply:

1. The collocated luma samples pY[x][y] with x = 0..nTbW * SubWidthC − 1, y= 0..nTbH * SubHeightC − 1 are set equal to the reconstructed luma samples prior to the in-loop filter process at the locations (xTbY + x, yTbY + y).

2. The neighbouring luma samples pY[x][y] are derived as follows:

   – When availL is equal to TRUE, the neighbouring luma samples pY[x][y] with x = −3..−1, y = (availT ? −1 : 0)..SubHeightC * Max(nTbH, numSampL) − 1, are set equal to the reconstructed luma samples prior to the in-loop filter process at the locations (xTbY + x, yTbY +y).

   – When availT is equal to FALSE, the neighbouring luma samples pY[x][y] with x = −2..SubWidthC * nTbW − 1, y = −2..−1, are set equal to the luma samples pY[x][0].

– When availT is equal to TRUE, the neighbouring luma samples pY[x][y] with x = (availL ? −1 : 0)..SubWidthC * Max(nTbW, numSampT) − 1, y = −3..−1, are set equal to the reconstructed luma samples prior to the in-loop filter process at the locations (xTbY+ x, yTbY + y).

– When availL is equal to FALSE, the neighbouring luma samples pY[x][y] with x = −1, y = −2..SubHeightC * nTbH − 1, are set equal to the reconstructed luma samples pY[0][y].

3. The down-sampled collocated luma samples pDsY[x][y] with x = 0..nTbW − 1, y = 0..nTbH − 1 are derived as follows:

– If both SubWidthC and SubHeightC are equal to 1, the following applies:

pDsY[x][y] = pY[x][y] (360)

– Otherwise, if SubHeightC is equal to 1, the following applies:

pDsY[x][y] = (pY[SubWidthC * x − 1][y] +
2 * pY[SubWidthC * x][y] +
pY[SubWidthC * x + 1][y] + 2) >> 2 (361)

– Otherwise (SubHeightC is not equal to 1), the following applies:

– If sps_chroma_vertical_collocated_flag is equal to 1, the following applies:

pDsY[x][y] = (pY[SubWidthC * x][SubHeightC * y − 1] +
pY[SubWidthC * x − 1][SubHeightC * y] +
4 * pY[SubWidthC * x][SubHeightC * y] +
pY[SubWidthC * x + 1][SubHeightC * y] +
pY[SubWidthC * x][SubHeightC * y + 1] + 4) >> 3 (362)

– Otherwise (sps_chroma_vertical_collocated_flag is equal to 0), the following applies:

pDsY[x][y] = (pY[SubWidthC * x − 1][SubHeightC * y] +
pY[SubWidthC * x − 1][SubHeightC * y + 1] +
2 * pY[SubWidthC * x][SubHeightC * y] +
2 * pY[SubWidthC * x][SubHeightC * y + 1] +
pY[SubWidthC * x + 1][SubHeightC * y] +
pY[SubWidthC * x + 1][SubHeightC * y + 1] + 4) >> 3 (363)

4. When numSampT is greater than 0, the selected neighbouring top chroma samples pSelC[idx] are set equal to p[pickPosT[idx]][−1] with idx = 0..cntT − 1, and the down-sampled neighbouring top luma samples pSelDsY[idx] with idx = 0..cntT − 1 are specified as follows:

– The variable x is set equal to pickPosT[idx].

– If both SubWidthC and SubHeightC are equal to 1, the following applies:

pSelDsY[idx] = pY[x][− 1] (364)

– Otherwise, the following applies:

– If SubHeightC is not equal to 1 and bCTUboundary is equal to FALSE, the following applies:

− If sps_chroma_vertical_collocated_flag is equal to 1, the following applies:

pSelDsY[idx] = (pY[SubWidthC * x][−3] +
pY[SubWidthC * x − 1][−2] +
4 * pY[SubWidthC * x][−2] +
pY[SubWidthC * x + 1][−2] +
pY[SubWidthC * x][−1] + 4) >> 3 (365)

− Otherwise (sps_chroma_vertical_collocated_flag is equal to 0), the following applies:

pSelDsY[idx] = (pY[SubWidthC * x − 1][−1] +
pY[SubWidthC * x − 1][−2] +
2* pY[SubWidthC * x][−1] +
2 * pY[SubWidthC * x][−2] + (366)

$$pY[\ SubWidthC * x + 1\ ][\ -1\ ] + pY[\ SubWidthC * x + 1\ ][\ -2\ ] + 4\ ) >> 3$$

– Otherwise (SubHeightC is equal to 1 or bCTUboundary is equal to TRUE), the following applies:

$$pSelDsY[\ idx\ ] = (\ pY[\ SubWidthC * x - 1\ ][\ -1\ ] + 2 * pY[\ SubWidthC * x\ ][\ -1\ ] + pY[\ SubWidthC * x + 1\ ][\ -1\ ] + 2\ ) >> 2 \quad (367)$$

5. When numSampL is greater than 0, the selected neighbouring left chroma samples pSelC[idx] are set equal to p[−1][pickPosL[idx − cntT]] with idx = cntT..cntT + cntL − 1, and the selected down-sampled neighbouring left luma samples pSelDsY[idx] with idx = cntT..cntT + cntL − 1 are derived as follows:

   – The variable y is set equal to pickPosL[idx − cntT].

   – If both SubWidthC and SubHeightC are equal to 1, the following applies:

$$pSelDsY[\ idx\ ] = pY[\ -1\ ][\ y\ ] \quad (368)$$

   – Otherwise, if SubHeightC is equal to 1, the following applies:

$$pSelDsY[\ idx\ ] = (\ pY[\ -1 - SubWidthC\ ][\ y\ ] + 2 * pY[\ -SubWidthC\ ][\ y\ ] + pY[\ 1 - SubWidthC\ ][\ y\ ] + 2\ ) >> 2 \quad (369)$$

   – Otherwise, the following applies:

     – If sps_chroma_vertical_collocated_flag is equal to 1, the following applies:

$$pSelDsY[\ idx\ ] = (\ pY[\ -SubWidthC\ ][\ SubHeightC * y - 1\ ] + pY[\ -1 - SubWidthC\ ][\ SubHeightC * y\ ] + 4 * pY[\ -SubWidthC\ ][\ SubHeightC * y\ ] + pY[\ 1 - SubWidthC\ ][\ SubHeightC * y\ ] + pY[\ -SubWidthC\ ][\ SubHeightC * y + 1\ ] + 4\ ) >> 3 \quad (370)$$

     – Otherwise (sps_chroma_vertical_collocated_flag is equal to 0), the following applies:

$$pSelDsY[\ idx\ ] = (\ pY[\ -1 - SubWidthC\ ][\ SubHeightC * y\ ] + pY[\ -1 - SubWidthC\ ][\ SubHeightC * y + 1\ ] + 2 * pY[\ -SubWidthC\ ][\ SubHeightC * y\ ] + 2 * pY[\ -SubWidthC\ ][\ SubHeightC * y + 1\ ] + pY[\ 1 - SubWidthC\ ][\ SubHeightC * y\ ] + pY[\ 1 - SubWidthC][\ SubHeightC * y + 1\ ] + 4\ ) >> 3 \quad (371)$$

6. The variables minY, maxY, minC and maxC are derived as follows:

   – When cntT + cntL is equal to 2, pSelComp[3] is set equal to pSelComp[0], pSelComp[2] is set equal to pSelComp[1], pSelComp[0] is set equal to pSelComp[1], and pSelComp[1] is set equal to pSelComp[3], with Comp being replaced by DsY and C.

   – The arrays minGrpIdx and maxGrpIdx are derived as follows:

$$minGrpIdx[\ 0\ ] = 0 \quad (372)$$

$$minGrpIdx[\ 1\ ] = 2 \quad (373)$$

$$maxGrpIdx[\ 0\ ] = 1 \quad (374)$$

$$maxGrpIdx[\ 1\ ] = 3 \quad (375)$$

   – When pSelDsY[minGrpIdx[0]] is greater than pSelDsY[minGrpIdx[1]], minGrpIdx[0] and minGrpIdx[1] are swapped as follows:

$$(\ minGrpIdx[\ 0\ ], minGrpIdx[\ 1\ ]\ ) = Swap(\ minGrpIdx[\ 0\ ], minGrpIdx[\ 1\ ]\ ) \quad (376)$$

– When pSelDsY[maxGrpIdx[0]] is greater than pSelDsY[maxGrpIdx[1]], maxGrpIdx[0] and maxGrpIdx[1] are swapped as follows:

$$(\text{maxGrpIdx}[\ 0\ ], \text{maxGrpIdx}[\ 1\ ]) = \text{Swap}(\text{maxGrpIdx}[\ 0\ ], \text{maxGrpIdx}[\ 1\ ]) \quad (377)$$

– When pSelDsY[minGrpIdx[0]] is greater than pSelDsY[maxGrpIdx[1]], arrays minGrpIdx and maxGrpIdx are swapped as follows:

$$(\text{minGrpIdx}, \text{maxGrpIdx}) = \text{Swap}(\text{minGrpIdx}, \text{maxGrpIdx}) \quad (378)$$

– When pSelDsY[minGrpIdx[1]] is greater than pSelDsY[maxGrpIdx[0]], minGrpIdx[1] and maxGrpIdx[0] are swapped as follows:

$$(\text{minGrpIdx}[\ 1\ ], \text{maxGrpIdx}[\ 0\ ]) = \text{Swap}(\text{minGrpIdx}[\ 1\ ], \text{maxGrpIdx}[\ 0\ ]) \quad (379)$$

– The variables maxY, maxC, minY and minC are derived as follows:

$$\text{maxY} = (\text{pSelDsY}[\ \text{maxGrpIdx}[\ 0\ ]\ ] + \text{pSelDsY}[\ \text{maxGrpIdx}[\ 1\ ]\ ] + 1) \gg 1 \quad (380)$$

$$\text{maxC} = (\text{pSelC}[\ \text{maxGrpIdx}[\ 0\ ]\ ] + \text{pSelC}[\ \text{maxGrpIdx}[\ 1\ ]\ ] + 1) \gg 1 \quad (381)$$

$$\text{minY} = (\text{pSelDsY}[\ \text{minGrpIdx}[\ 0\ ]\ ] + \text{pSelDsY}[\ \text{minGrpIdx}[\ 1\ ]\ ] + 1) \gg 1 \quad (382)$$

$$\text{minC} = (\text{pSelC}[\ \text{minGrpIdx}[\ 0\ ]\ ] + \text{pSelC}[\ \text{minGrpIdx}[\ 1\ ]\ ] + 1) \gg 1 \quad (383)$$

7. The variables a, b, and k are derived as follows:

– The variable diff is derived as follows:

$$\text{diff} = \text{maxY} - \text{minY} \quad (384)$$

– If diff is not equal to 0, the following applies:

$$\text{diffC} = \text{maxC} - \text{minC} \quad (385)$$

$$x = \text{Floor}(\text{Log2}(\text{diff})) \quad (386)$$

$$\text{normDiff} = ((\text{diff} \ll 4) \gg x) \,\&\, 15 \quad (387)$$

$$x \mathrel{+}= (\text{normDiff} \mathrel{!}= 0) \mathrel{?} 1 : 0 \quad (388)$$

$$y = \text{Abs}(\text{diffC}) > 0 \mathrel{?} \text{Floor}(\text{Log2}(\text{Abs} (\text{diffC}))) + 1 : 0 \quad (389)$$

$$a = (\text{diffC} * (\text{divSigTable}[\ \text{normDiff}\ ] \mid 8) + 2^{y-1}) \gg y \quad (390)$$

$$k = ((3 + x - y) < 1) \mathrel{?} 1 : 3 + x - y \quad (391)$$

$$a = ((3 + x - y) < 1) \mathrel{?} \text{Sign}(a) * 15 : a \quad (392)$$

$$b = \text{minC} - ((a * \text{minY}) \gg k) \quad (393)$$

where divSigTable[ ] is specified as follows:

$$\text{divSigTable}[\ ] = \{ 0, 7, 6, 5, 5, 4, 4, 3, 3, 2, 2, 1, 1, 1, 1, 0 \} \quad (394)$$

– Otherwise (diff is equal to 0), the following applies:

$$k = 0 \quad (395)$$

$$a = 0 \quad (396)$$

$$b = \text{minC} \quad (397)$$

8. The prediction samples predSamples[x][y] with x = 0..nTbW − 1, y = 0.. nTbH − 1 are derived as follows:

$$\text{predSamples}[\,x\,][\,y\,] = \text{Clip1}(\,(\,(\,\text{pDsY}[\,x\,][\,y\,] * a\,) >> k\,) + b\,) \tag{398}$$

NOTE – This process uses sps_chroma_vertical_collocated_flag. However, in order to simplify implementation, it does not use sps_chroma_horizontal_collocated_flag.

8.4.5.2.15 Position-dependent intra prediction sample filtering process

Inputs to this process are:

– the intra prediction mode predModeIntra,

– a variable nTbW specifying the transform block width,

– a variable nTbH specifying the transform block height,

– a variable refW specifying the reference samples width,

– a variable refH specifying the reference samples height,

– the predicted samples predSamples[x][y], with x = 0..nTbW − 1, y = 0..nTbH − 1,

– the neighbouring samples p[x][y], with x = −1, y = −1..refH − 1 and x = 0..refW − 1, y = −1.

Outputs of this process are the modified predicted samples predSamples[x][y] with x = 0..nTbW − 1, y = 0..nTbH − 1.

The variable nScale is derived as follows:

– If predModeIntra is greater than INTRA_ANGULAR50, nScale is set equal to Min(2, Log2(nTbH) − Floor(Log2(3 * invAngle − 2)) + 8), using invAngle as specified in clause 8.4.5.2.13.

– Otherwise, if predModeIntra is less than INTRA_ANGULAR18, not equal to INTRA_PLANAR and not equal to INTRA_DC, nScale is set equal to Min(2, Log2(nTbW) − Floor(Log2(3 * invAngle − 2)) + 8), using invAngle as specified in clause 8.4.5.2.13.

– Otherwise, nScale is set equal to ((Log2(nTbW) + Log2(nTbH) − 2) >> 2).

The reference sample arrays mainRef[x] and sideRef[y], with x = 0..refW − 1 and y = 0..refH − 1 are derived as follows:

$$\begin{aligned}&\text{mainRef}[\,x\,] = p[\,x\,][\,-1\,]\\&\text{sideRef}[\,y\,] = p[\,-1\,][\,y\,]\end{aligned} \tag{399}$$

The variables refL[x][y], refT[x][y], wT[y], and wL[x] with x = 0..nTbW − 1, y = 0..nTbH − 1 are derived as follows:

– If predModeIntra is equal to INTRA_PLANAR or INTRA_DC, the following applies:

$$\text{refL}[\,x\,][\,y\,] = p[\,-1\,][\,y\,] \tag{400}$$

$$\text{refT}[\,x\,][\,y\,] = p[\,x\,][\,-1\,] \tag{401}$$

$$\text{wT}[\,y\,] = 32 >> (\,(\,y << 1\,) >> \text{nScale}\,) \tag{402}$$

$$\text{wL}[\,x\,] = 32 >> (\,(\,x << 1\,) >> \text{nScale}\,) \tag{403}$$

– Otherwise, if predModeIntra is equal to INTRA_ANGULAR18 or INTRA_ANGULAR50, the following applies:

$$\text{refL}[\,x\,][\,y\,] = p[\,-1\,][\,y\,] - p[\,-1\,][\,-1\,] + \text{predSamples}[\,x\,][\,y\,] \tag{404}$$

$$\text{refT}[\,x\,][\,y\,] = p[\,x\,][\,-1\,] - p[\,-1\,][\,-1\,] + \text{predSamples}[\,x\,][\,y\,] \tag{405}$$

$$\text{wT}[\,y\,] = (\,\text{predModeIntra} == \text{INTRA\_ANGULAR18}\,)\ ?\ 32 >> (\,(\,y << 1\,) >> \text{nScale}\,) : 0 \tag{406}$$

$$\text{wL}[\,x\,] = (\,\text{predModeIntra} == \text{INTRA\_ANGULAR50}\,)\ ?\ 32 >> (\,(\,x << 1\,) >> \text{nScale}\,) : 0 \tag{407}$$

– Otherwise, if predModeIntra is less than INTRA_ANGULAR18 and nScale is equal to or greater than 0, the following ordered steps apply:

1. The variables dXInt[y] and dX[x][y] are derived as follows using invAngle as specified in clause 8.4.5.2.13 depending on intraPredMode:

$$dXInt[y] = ((y + 1) * invAngle + 256) >> 9$$
$$dX[x][y] = x + dXInt[y] \quad (408)$$

2. The variables refL[x][y], refT[x][y], wT[y], and wL[x] are derived as follows:

$$refL[x][y] = 0 \quad (409)$$

$$refT[x][y] = (y < (3 << nScale)) ? mainRef[dX[x][y]] : 0 \quad (410)$$

$$wT[y] = 32 >> ((y << 1) >> nScale) \quad (411)$$

$$wL[x] = 0 \quad (412)$$

– Otherwise, if predModeIntra is greater than INTRA_ANGULAR50 and nScale is equal to or greater than 0, the following ordered steps apply:

1. The variables dYInt[x] and dY[x][y] are derived as follows using invAngle as specified in clause 8.4.5.2.13 depending on intraPredMode:

$$dYInt[x] = ((x + 1) * invAngle + 256) >> 9$$
$$dY[x][y] = y + dYInt[x] \quad (413)$$

2. The variables refL[x][y], refT[x][y], wT[y], and wL[x] are derived as follows:

$$refL[x][y] = (x < (3 << nScale)) ? sideRef[dY[x][y]] : 0 \quad (414)$$

$$refT[x][y] = 0 \quad (415)$$

$$wT[y] = 0 \quad (416)$$

$$wL[x] = 32 >> ((x << 1) >> nScale) \quad (417)$$

– Otherwise, refL[x][y], refT[x][y], wT[y], and wL[x] are all set equal to 0.

The values of the modified predicted samples predSamples[x][y], with x = 0..nTbW − 1, y = 0..nTbH − 1 are derived as follows:

$$predSamples[x][y] = Clip1((refL[x][y] * wL[x] + refT[x][y] * wT[y] + (64 - wL[x] - wT[y]) * predSamples[x][y] + 32) >> 6) \quad (418)$$

8.4.5.3 Decoding process for palette mode

Inputs to this process are:

– a location (xCbComp, yCbComp) specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,

– a variable treeType specifying whether a single or a dual tree is used and if a dual tree is used, it specifies whether the current tree corresponds to the luma or chroma components,

– a variable cIdx specifying the colour component of the current block,

– two variables nCbW and nCbH specifying the width and height of the current coding block, respectively.

Output of this process is an array recSamples[x][y], with x = 0.. nCbW − 1, y = 0.. nCbH − 1 specifying reconstructed sample values for the block.

Depending on the value of treeType, the variables startComp, numComps and maxNumPalettePredictorSize are derived as follows:

– If treeType is equal to SINGLE_TREE, the following applies:

$$startComp = 0 \quad (419)$$

$$numComps = sps\_chroma\_format\_idc == 0 ? 1 : 3 \quad (420)$$

$$maxNumPalettePredictorSize = 63 \quad (421)$$

– Otherwise, if treeType is equal to DUAL_TREE_LUMA, the following applies:

$$\text{startComp} = 0 \tag{422}$$

$$\text{numComps} = 1 \tag{423}$$

$$\text{maxNumPalettePredictorSize} = 31 \tag{424}$$

– Otherwise (treeType is equal to DUAL_TREE_CHROMA), the following applies:

$$\text{startComp} = 1 \tag{425}$$

$$\text{numComps} = 2 \tag{426}$$

$$\text{maxNumPalettePredictorSize} = 31 \tag{427}$$

Depending on the value of cIdx, the variables nSubWidth and nSubHeight are derived as follows:

– If cIdx is greater than 0 and startComp is equal to 0, nSubWidth is set equal to SubWidthC and nSubHeight is set equal to SubHeightC.

– Otherwise, nSubWidth is set equal to 1 and nSubHeight is set equal to 1.

The (nCbW x nCbH) block of the reconstructed sample array recSamples at location (xCbComp, yCbComp) is represented by recSamples[x][y] with x = 0..nCbW − 1 and y = 0..nCbH − 1, and the value of recSamples[x][y] for each x in the range of 0 to nCbW − 1, inclusive, and each y in the range of 0 to nCbH − 1, inclusive, is derived as follows:

– The variables xL, yL, xCbL, and yCbL are derived as follows:

$$\text{xL} = \text{x} * \text{nSubWidth} \tag{428}$$

$$\text{yL} = \text{y} * \text{nSubHeight} \tag{429}$$

$$\text{xCbL} = \text{xCbComp} * \text{nSubWidth} \tag{430}$$

$$\text{yCbL} = \text{yCbComp} * \text{nSubHeight} \tag{431}$$

– The variable bIsEscapeSample is derived as follows:

  – If PaletteIndexMap[xCbL + xL][yCbL + yL] is equal to MaxPaletteIndex and palette_escape_val_present_flag is equal to 1, bIsEscapeSample is set equal to 1.

  – Otherwise, bIsEscapeSample is set equal to 0.

– If bIsEscapeSample is equal to 0, the following applies:

$$\text{recSamples[x][y]} = \text{CurrentPaletteEntries[cIdx][PaletteIndexMap[xCbL + xL][yCbL + yL]]} \tag{432}$$

– Otherwise (bIsEscapeSample is equal to 1), the following ordered steps apply:

1. The quantization parameter qP is derived as follows:

   – If cIdx is equal to 0, the following applies:

$$\text{qP} = \text{Max(QpPrimeTsMin, Qp}'\text{Y)} \tag{433}$$

   – Otherwise, if cIdx is equal to 1, the following applies:

$$\text{qP} = \text{Max(QpPrimeTsMin, Qp}'\text{Cb)} \tag{434}$$

   – Otherwise (cIdx is equal to 2), the following applies:

$$\text{qP} = \text{Max(QpPrimeTsMin, Qp}'\text{Cr)} \tag{435}$$

2. The list levelScale[ ] is specified as levelScale[k] = { 40, 45, 51, 57, 64, 72 } with k = 0..5.

3. The following applies:

$$\text{tmpVal} = (((\text{PaletteEscapeVal}[\text{cIdx}][\text{xCbL} + \text{xL}][\text{yCbL} + \text{yL}] * \text{levelScale}[\text{qP}\%6]) \ll (\text{qP} / 6)) + 32) \gg 6 \quad (436)$$

$$\text{recSamples}[x][y] = \text{Clip3}(0, (1 \ll \text{BitDepth}) - 1, \text{tmpVal}) \quad (437)$$

When LocalDualTreeFlag is equal to 1, the following applies:

– When treeType is equal to DUAL_TREE_LUMA, the following applies for i = 0..num_signalled_palette_entries − 1:

$$\text{CurrentPaletteEntries}[1][\text{NumPredictedPaletteEntries} + i] = 1 \ll (\text{BitDepth} - 1) \quad (438)$$

$$\text{CurrentPaletteEntries}[2][\text{NumPredictedPaletteEntries} + i] = 1 \ll (\text{BitDepth} - 1) \quad (439)$$

– The variables CurrentPaletteSize[0], startComp, numComps and maxNumPalettePredictorSize are derived as follows:

$$\text{CurrentPaletteSize}[0] = \text{CurrentPaletteSize}[\text{startComp}] \quad (440)$$

$$\text{startComp} = 0 \quad (441)$$

$$\text{numComps} = 3 \quad (442)$$

$$\text{maxNumPalettePredictorSize} = 63 \quad (443)$$

When one of the following conditions is true:

– cIdx is equal to 0 and numComps is equal to 1;

– cIdx is equal to 0 and LocalDualTreeFlag is equal to 1;

– cIdx is equal to 2 and LocalDualTreeFlag is equal to 0;

the value PredictorPaletteSize[startComp] and the array PredictorPaletteEntries are derived or modified as follows:

```
for( i = 0; i < CurrentPaletteSize[ startComp ]; i++ )
    for( cIdx = startComp; cIdx < (startComp + numComps); cIdx++ )
        newPredictorPaletteEntries[ cIdx ][ i ] = CurrentPaletteEntries[ cIdx ][ i ]
newPredictorPaletteSize = CurrentPaletteSize[ startComp ]
for( i = 0; i < PredictorPaletteSize[ startComp ]  &&  newPredictorPaletteSize < maxNumPalettePredictorSize; i++ )
    if( !PalettePredictorEntryReuseFlags[ i ] ) {
        for( cIdx = startComp; cIdx < (startComp + numComps); cIdx++ )                    (444)
            newPredictorPaletteEntries[ cIdx ][ newPredictorPaletteSize ] =
                PredictorPaletteEntries[ cIdx ][ i ]
        newPredictorPaletteSize++
    }
for( cIdx = startComp; cIdx < ( startComp + numComps ); cIdx++ )
    for( i = 0; i < newPredictorPaletteSize; i++ )
        PredictorPaletteEntries[ cIdx ][ i ] = newPredictorPaletteEntries[ cIdx ][ i ]
PredictorPaletteSize[ startComp ] = newPredictorPaletteSize
```

When sps_qtbtt_dual_tree_intra_flag is equal to 0 or sh_slice_type is not equal to I, the following applies:

$$\text{PredictorPaletteSize}[1] = \text{newPredictorPaletteSize} \quad (445)$$

It is a requirement of bitstream conformance that the value of PredictorPaletteSize[startComp] shall be in the range of 0 to maxNumPalettePredictorSize, inclusive.

8.5 Decoding process for coding units coded in inter prediction mode

8.5.1 General decoding process for coding units coded in inter prediction mode

Inputs to this process are:

– a luma location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples,

– a variable treeType specifying whether a single or a dual tree is used and if a dual tree is used, it specifies whether the current tree corresponds to the luma or chroma components.

Output of this process is a modified reconstructed picture before in-loop filtering.

The variable currPic specifies the current picture.

The derivation process for quantization parameters as specified in clause 8.7.1 is invoked with the luma location (xCb, yCb), the width of the current coding block in luma samples cbWidth and the height of the current coding block in luma samples cbHeight, and the variable treeType as inputs.

The decoding process for coding units coded in inter prediction mode consists of the following ordered steps:

1. The variable dmvrFlag is set equal to 0, the variables cbProfFlagL0 and cbProfFlagL1 are both set equal to 0, and the variable hpelIfIdx is set equal to 0.

2. The motion vector components and reference indices of the current coding unit are derived as follows:

   – If MergeGpmFlag[xCb][yCb], inter_affine_flag[xCb][yCb] and merge_subblock_flag[xCb][yCb] are all equal to 0, the following applies:

     – The derivation process for motion vector components and reference indices as specified in clause 8.5.2.1 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth and the luma coding block height cbHeight as inputs, and the luma motion vectors mvL0[0][0] and mvL1[0][0], the reference indices refIdxL0 and refIdxL1 and the prediction list utilization flags predFlagL0[0][0] and predFlagL1[0][0], the half sample interpolation filter index hpelIfIdx, and the bi-prediction weight index bcwIdx as outputs.

     – When all of the following conditions are true, dmvrFlag is set equal to 1:

       – ph_dmvr_disabled_flag is equal to 0.

       – general_merge_flag[xCb][yCb] is equal to 1.

       – both predFlagL0[0][0] and predFlagL1[0][0] are equal to 1.

       – mmvd_merge_flag[xCb][yCb] is equal to 0.

       – ciip_flag[xCb][yCb] is equal to 0.

       – DiffPicOrderCnt(currPic, RefPicList[0][refIdxL0]) is equal to DiffPicOrderCnt(RefPicList[1][refIdxL1], currPic).

       – RefPicList[0][refIdxL0] is an STRP and RefPicList[1][refIdxL1] is an STRP.

       – bcwIdx is equal to 0.

       – Both luma_weight_l0_flag[refIdxL0] and luma_weight_l1_flag[refIdxL1] are equal to 0.

       – Both chroma_weight_l0_flag[refIdxL0] and chroma_weight_l1_flag[refIdxL1] are equal to 0.

       – cbWidth is greater than or equal to 8.

       – cbHeight is greater than or equal to 8.

       – cbHeight*cbWidth is greater than or equal to 128.

       – RprConstraintsActiveFlag[0][refIdxL0] is equal to 0 and RprConstraintsActiveFlag[1][refIdxL1] is equal to 0.

     – If dmvrFlag is equal to 1, the following applies:

       – For X = 0..1, the reference picture consisting of an ordered two-dimensional array $refPicLX_L$ of luma samples and two ordered two-dimensional arrays $refPicLX_{Cb}$ and $refPicLX_{Cr}$ of chroma samples is derived by invoking the process specified in clause 8.5.6.2 with X and refIdxLX as inputs.

       – The number of luma subblocks in horizontal direction numSbX and in vertical direction numSbY, the subblock width sbWidth and the subblock height sbHeight are derived as follows:

$$\text{numSbX} = (\text{cbWidth} > 16) \;?\; (\text{cbWidth} >> 4) : 1 \quad (446)$$

$$numSbY = (cbHeight > 16) ? (cbHeight >> 4) : 1 \quad (447)$$

$$sbWidth = (cbWidth > 16) ? 16 : cbWidth \quad (448)$$

$$sbHeight = (cbHeight > 16) ? 16 : cbHeight \quad (449)$$

– For xSbIdx = 0..numSbX − 1 and ySbIdx = 0..numSbY − 1, the following applies:

– The luma motion vectors mvLX[xSbIdx][ySbIdx] and the prediction list utilization flags predFlagLX[xSbIdx][ySbIdx] with X = 0..1, and the luma location (xSb[xSbIdx][ySbIdx], ySb[xSbIdx][ySbIdx]) specifying the top-left sample of the subblock relative to the top-left luma sample of the current picture are derived as follows:

$$mvLX[xSbIdx][ySbIdx] = mvLX[0][0] \quad (450)$$

$$predFlagLX[xSbIdx][ySbIdx] = predFlagLX[0][0] \quad (451)$$

$$xSb[xSbIdx][ySbIdx] = xCb + xSbIdx * sbWidth \quad (452)$$

$$ySb[xSbIdx][ySbIdx] = yCb + ySbIdx * sbHeight \quad (453)$$

– The decoder-side motion vector refinement process specified in clause 8.5.3.1 is invoked with xSb[xSbIdx][ySbIdx], ySb[xSbIdx][ySbIdx], sbWidth, sbHeight, the motion vectors mvLX[xSbIdx][ySbIdx] and the reference picture array $refPicLX_L$ as inputs and delta motion vectors dMvLX[xSbIdx][ySbIdx] with X = 0..1, and the mimimum sum of absolute difference in decoder-side motion vector refinement process dmvrSad[xSbIdx][ySbIdx] as outputs.

– When sps_chroma_format_idc is not equal to 0, the derivation process for chroma motion vectors in clause 8.5.2.13 is invoked with mvLX[xSbIdx][ySbIdx] with X = 0..1 as inputs, and mvCLX[xSbIdx][ySbIdx] with X = 0..1 as outputs.

– Otherwise (dmvrFlag is equal to 0), the following applies:

– For X = 0..1, when sps_chroma_format_idc is not equal to 0, and treeType is equal to SINGLE_TREE, and predFlagLX[0][0] is equal to 1, the derivation process for chroma motion vectors in clause 8.5.2.13 is invoked with mvLX[0][0] as input, and mvCLX[0][0] as output.

– The number of luma subblocks in horizontal direction numSbX and in vertical direction numSbY are both set equal to 1.

– Otherwise, if MergeGpmFlag[xCb][yCb] is equal to 1, inter_affine_flag[xCb][yCb] and merge_subblock_flag[xCb][yCb] are both equal to 0, the following applies:

– The derivation process for geometric partitioning mode motion vector components and reference indices as specified in clause 8.5.4.1 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth and the luma coding block height cbHeight as inputs, and the luma motion vectors mvA and mvB, the chroma motion vectors mvCA and mvCB, the reference indices refIdxA and refIdxB and the prediction list flags predListFlagA and predListFlagB as outputs.

– The number of luma subblocks in horizontal direction numSbX and in vertical direction numSbY are both set equal to 1.

– Otherwise (inter_affine_flag[xCb][yCb] or merge_subblock_flag[xCb][yCb] is equal to 1), the derivation process for subblock motion vector components and reference indices as specified in clause 8.5.5.1 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth, the luma coding block height cbHeight as inputs, and the reference indices refIdxL0 and refIdxL1, the number of luma subblocks in horizontal direction numSbX and in vertical direction numSbY, the prediction list utilization flags predFlagLX[xSbIdx][ySbIdx], the luma motion vector array mvLX[xSbIdx][ySbIdx], and the chroma motion vector array mvCLX[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, and ySbIdx = 0..numSbY − 1, and with X = 0..1, the bi-prediction weight index bcwIdx, the prediction refinement utility flags cbProfFlagL0 and cbProfFlagL1, and motion vector difference arrays diffMvL0[xIdx][yIdx] and diffMvL1[xIdx][yIdx] with xIdx = 0.. cbWidth/numSbX − 1, yIdx = 0.. cbHeight/numSbY − 1 as outputs.

3. The arrays of luma and chroma motion vectors after decoder-side motion vector refinement, refMvLX[xSbIdx][ySbIdx] and refMvCLX[xSbIdx][ySbIdx], with X = 0..1, are derived as follows for xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1:

– If dmvrFlag is equal to 1, the derivation process for chroma motion vectors in clause 8.5.2.13 is invoked with refMvLX[xSbIdx][ySbIdx] as inputs, and refMvCLX[xSbIdx][ySbIdx] as output and the input refMvLX[xSbIdx][ySbIdx] is derived as follows:

$$\text{refMvLX[xSbIdx][ySbIdx]} = \text{mvLX[xSbIdx][ySbIdx]} + \text{dMvLX[xSbIdx][ySbIdx]} \quad (454)$$

$$\text{refMvLX[xSbIdx][ySbIdx][0]} = \text{Clip3}(-2^{17}, 2^{17} - 1, \text{refMvLX[xSbIdx][ySbIdx][0]}) \quad (455)$$

$$\text{refMvLX[xSbIdx][ySbIdx][1]} = \text{Clip3}(-2^{17}, 2^{17} - 1, \text{refMvLX[xSbIdx][ySbIdx][1]}) \quad (456)$$

– Otherwise (dmvrFlag is equal to 0), the following applies:

$$\text{refMvLX[xSbIdx][ySbIdx]} = \text{mvLX[xSbIdx][ySbIdx]} \quad (457)$$

$$\text{refMvCLX [xSbIdx][ySbIdx]} = \text{mvCLX[xSbIdx][ySbIdx]} \quad (458)$$

NOTE – The array refMvLX is stored in MvDmvrLX and used in the derivation process for collocated motion vectors in clause 8.5.2.12. The array of non-refine luma motion vectors MvLX is used in the spatial motion vector prediction and deblocking boundary filtering strength derivation processes.

4. The prediction samples of the current coding unit are derived as follows:

– If MergeGpmFlag[xCb][yCb] is equal to 0, the prediction samples of the current coding unit are derived as follows:

– The decoding process for inter blocks as specified in clause 8.5.6.1 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth and the luma coding block height cbHeight, the number of luma subblocks in horizontal direction numSbX and in vertical direction numSbY, the luma motion vectors mvL0[xSbIdx][ySbIdx] and mvL1[xSbIdx][ySbIdx], and the refined luma motion vectors refMvL0[xSbIdx][ySbIdx] and refMvL1[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, and ySbIdx = 0..numSbY − 1, the reference indices refIdxL0 and refIdxL1, the prediction list utilization flags predFlagL0[xSbIdx][ySbIdx] and predFlagL1[xSbIdx][ySbIdx], the half sample interpolation filter index hpelIfIdx, the bi-prediction weight index bcwIdx, the mimimum sum of absolute difference values in decoder-side motion vector refinement process dmvrSad[xSbIdx][ySbIdx], the decoder-side motion vector refinement flag dmvrFlag, the variable cIdx set equal to 0, the prediction refinement utility flags cbProfFlagL0 and cbProfFlagL1, and motion vector difference arrays diffMvL0[xIdx][yIdx] and diffMvL1[xIdx][yIdx] with xIdx = 0..cbWidth / numSbX − 1, and yIdx = 0..cbHeight / numSbY − 1 as inputs, and the inter prediction samples (predSamples) that are an (cbWidth)x(cbHeight) array predSamples_L of prediction luma samples as outputs.

– When sps_chroma_format_idc is not equal to 0, the decoding process for inter blocks as specified in clause 8.5.6.1 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth and the luma coding block height cbHeight, the number of luma subblocks in horizontal direction numSbX and in vertical direction numSbY, the chroma motion vectors mvCL0[xSbIdx][ySbIdx] and mvCL1[xSbIdx][ySbIdx], and the refined chroma motion vectors refMvCL0[xSbIdx][ySbIdx] and refMvCL1[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, and ySbIdx = 0..numSbY − 1, the reference indices refIdxL0 and refIdxL1, the prediction list utilization flags predFlagL0[xSbIdx][ySbIdx] and predFlagL1[xSbIdx][ySbIdx], the half sample interpolation filter index hpelIfIdx, the bi-prediction weight index bcwIdx, the mimimum sum of absolute difference values in decoder-side motion vector refinement process dmvrSad[xSbIdx][ySbIdx], the decoder-side motion vector refinement flag dmvrFlag, the variable cIdx set equal to 1, the prediction refinement utility flags cbProfFlagL0 and cbProfFlagL1, and motion vector difference arrays diffMvL0[xIdx][yIdx] and diffMvL1[xIdx][yIdx] with xIdx = 0..cbWidth / numSbX − 1, and yIdx = 0..cbHeight / numSbY − 1 as inputs, and the inter prediction samples (predSamples) that are an (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array predSamples_{Cb} of prediction chroma samples for the chroma components Cb as outputs.

– When sps_chroma_format_idc is not equal to 0, the decoding process for inter blocks as specified in clause 8.5.6.1 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth and the luma coding block height cbHeight, the number of luma subblocks in horizontal direction numSbX and in vertical direction numSbY, the chroma motion vectors mvCL0[xSbIdx][ySbIdx] and mvCL1[xSbIdx][ySbIdx], and the refined chroma motion vectors refMvCL0[xSbIdx][ySbIdx] and refMvCL1[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, and ySbIdx = 0..numSbY − 1, the reference indices refIdxL0 and refIdxL1, the prediction list utilization flags predFlagL0[xSbIdx][ySbIdx] and predFlagL1[xSbIdx][ySbIdx], the half sample interpolation filter index hpelIfIdx, the bi-prediction weight index bcwIdx, the mimimum sum of absolute difference values in decoder-side motion vector refinement process dmvrSad[xSbIdx][ySbIdx], the decoder-side motion vector refinement flag dmvrFlag, the variable cIdx set equal to 2, the prediction refinement utility flags cbProfFlagL0 and cbProfFlagL1, and motion vector difference arrays diffMvL0[xIdx][yIdx] and diffMvL1[xIdx][yIdx] with xIdx = 0..cbWidth / numSbX − 1, and yIdx = 0..cbHeight / numSbY − 1 as inputs, and the inter prediction samples (predSamples) that are an (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array $predSamples_{Cr}$ of prediction chroma samples for the chroma components Cr as outputs.

– Otherwise (MergeGpmFlag[xCb][yCb] is equal to 1), the decoding process for geometric partitioning mode inter blocks as specified in clause 8.5.7.1 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth and the luma coding block height cbHeight, the luma motion vectors mvA and mvB, the chroma motion vectors mvCA and mvCB, the reference indices refIdxA and refIdxB, and the prediction list flags predListFlagA and predListFlagB as inputs, and the inter prediction samples (predSamples) that are an (cbWidth)x(cbHeight) array $predSamples_L$ of prediction luma samples and, when sps_chroma_format_idc is not equal to 0, two (cbWidth / SubWidthC)x(cbHeight / SubHeightC) arrays $predSamples_{Cb}$ and $predSamples_{Cr}$ of prediction chroma samples, one for each of the chroma components Cb and Cr, as outputs.

5. The variables NumSbX[xCb][yCb] and NumSbY[xCb][yCb] are set equal to numSbX and numSbY, respectively.

6. The residual samples of the current coding unit are derived as follows:

– The decoding process for the residual signal of coding blocks coded in inter prediction mode as specified in clause 8.5.8 is invoked with the location (xTb0, yTb0) set equal to the luma location (xCb, yCb), the width nCbW set equal to the luma coding block width cbWidth, the height nCbH set equal to the luma coding block height cbHeight, the width nTbW set equal to the luma coding block width cbWidth, the height nTbH set equal to the luma coding block height cbHeight and the variable cIdx set equal to 0 as inputs, and the array $resSamples_L$ as output.

– When sps_chroma_format_idc is not equal to 0, the decoding process for the residual signal of coding blocks coded in inter prediction mode as specified in clause 8.5.8 is invoked with the location (xTb0, yTb0) set equal to the chroma location (xCb / SubWidthC, yCb / SubHeightC), the width nCbW set equal to the chroma coding block width cbWidth / SubWidthC, the height nCbH set equal to the chroma coding block height cbHeight / SubHeightC, the width nTbW set equal to the chroma coding block width cbWidth / SubWidthC, the height nTbH set equal to the chroma coding block height cbHeight / SubHeightC and the variable cIdx set equal to 1 as inputs, and the array $resSamples_{Cb}$ as output.

– When sps_chroma_format_idc is not equal to 0, the decoding process for the residual signal of coding blocks coded in inter prediction mode as specified in clause 8.5.8 is invoked with the location (xTb0, yTb0) set equal to the chroma location (xCb / SubWidthC, yCb / SubHeightC), the width nCbW set equal to the chroma coding block width cbWidth / SubWidthC, the height nCbH set equal to the chroma coding block height cbHeight / SubHeightC, the width nTbW set equal to the chroma coding block width cbWidth / SubWidthC, the height nTbH set equal to the chroma coding block height cbHeight / SubHeightC and the variable cIdx set equal to 2 as inputs, and the array $resSamples_{Cr}$ as output.

– When cu_act_enabled_flag[xCb][yCb] is equal to 1, the residual modification process for residual blocks using colour space conversion as specified in clause 8.7.4.6 is invoked with the variable nTbW set equal to cbWidth, the variable nTbH set equal to cbHeight, the array r_Y set equal to $resSamples_L$, the array r_{Cb} set equal to $resSamples_{Cb}$, and the array r_{Cr} set equal to $resSamples_{Cr}$ as inputs, and the output are modified versions of the arrays $resSamples_L$, $resSamples_{Cb}$ and $resSamples_{Cr}$.

7. The reconstructed samples of the current coding unit are derived as follows:

– The picture reconstruction process for a colour component as specified in clause 8.7.5.1 is invoked with the block location (xCurr, yCurr) set equal to (xCb, yCb), the block width nCurrSw set equal to cbWidth, the block height nCurrSh set equal to cbHeight, the variable cIdx set equal to 0, the (cbWidth)x(cbHeight) array predSamples set equal to $predSamples_L$ and the (cbWidth)x(cbHeight) array resSamples set equal to $resSamples_L$ as inputs, and the output is a modified reconstructed picture before in-loop filtering.

– When sps_chroma_format_idc is not equal to 0, the decoding process for the reconstructed signal of chroma coding blocks coded in inter prediction mode as specified in clause 8.5.9 is invoked with the transform block location (xTb0, yTb0) set equal to (xCb / SubWidthC, yCb / SubHeightC), the transform block width nTbW set equal to cbWidth / SubWidthC and the height nTbH set equal to cbHeight / SubHeightC, the variable cIdx set equal to 1, the (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array predSamples set equal to $predSamples_{Cb}$ and the (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array resSamples set equal to $resSamples_{Cb}$ as inputs, and the output is a modified reconstructed picture before in-loop filtering.

– When sps_chroma_format_idc is not equal to 0, the decoding process for the reconstructed signal of chroma coding blocks coded in inter prediction mode as specified in clause 8.5.9 is invoked with the transform block location (xTb0, yTb0) set equal to (xCb / SubWidthC, yCb / SubHeightC), the transform block width nTbW set equal to cbWidth / SubWidthC and the height nTbH set equal to cbHeight / SubHeightC, the variable cIdx set equal to 2, the (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array predSamples set equal to $predSamples_{Cr}$ and the (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array resSamples set equal to $resSamples_{Cr}$ as inputs, and the output is a modified reconstructed picture before in-loop filtering.

8.5.2 Derivation process for motion vector components and reference indices

8.5.2.1 General

Inputs to this process are:

– a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples.

Outputs of this process are:

– the luma motion vectors in 1/16 fractional-sample accuracy mvL0[0][0] and mvL1[0][0],

– the reference indices refIdxL0 and refIdxL1,

– the prediction list utilization flags predFlagL0[0][0] and predFlagL1[0][0],

– the half sample interpolation filter index hpelIfIdx,

– the bi-prediction weight index bcwIdx.

For the derivation of the variables mvL0[0][0] and mvL1[0][0], refIdxL0 and refIdxL1, as well as predFlagL0[0][0] and predFlagL1[0][0], the following applies:

– If general_merge_flag[xCb][yCb] is equal to 1, the derivation process for luma motion vectors for merge mode as specified in clause 8.5.2.2 is invoked with the luma location (xCb, yCb) and the variables cbWidth and cbHeight as inputs, and the output being the luma motion vectors mvL0[0][0], mvL1[0][0], the reference indices refIdxL0, refIdxL1, the prediction list utilization flags predFlagL0[0][0] and predFlagL1[0][0], the half sample interpolation filter index hpelIfIdx, the bi-prediction weight index bcwIdx and the merging candidate list mergeCandList.

– Otherwise, the following applies:

  – For X = 0..1, the following ordered steps apply with X being replaced by either 0 or 1 in the variables predFlagLX[0][0], mvLX[0][0], refIdxLX, PRED_LX, ref_idx_lX and MvdLX:

    1. The variables refIdxLX and predFlagLX[0][0] are derived as follows:

       − If inter_pred_idc[xCb][yCb] is equal to PRED_LX or PRED_BI, the following applies:

         refIdxLX = ref_idx_lX[xCb][yCb] (459)

         predFlagLX[0][0] = 1 (460)

       − Otherwise, the variables refIdxLX and predFlagLX[0][0] are specified by:

         refIdxLX = −1 (461)

         predFlagLX[0][0] = 0 (462)

    2. The variable mvdLX is derived as follows:

         mvdLX[0] = MvdLX[xCb][yCb][0] (463)

         mvdLX[1] = MvdLX[xCb][yCb][1] (464)

3. When predFlagLX[0][0] is equal to 1, the derivation process for luma motion vector prediction in clause 8.5.2.8 is invoked with the luma coding block location (xCb, yCb), the coding block width cbWidth, the coding block height cbHeight and the variable refIdxLX as inputs, and the output being mvpLX.

4. When predFlagLX[0][0] is equal to 1, the luma motion vector mvLX[0][0] is derived as follows:

$$uLX[0] = (mvpLX[0] + mvdLX[0]) \& (2^{18} - 1) \quad (465)$$

$$mvLX[0][0][0] = (uLX[0] >= 2^{17}) ? (uLX[0] - 2^{18}) : uLX[0] \quad (466)$$

$$uLX[1] = (mvpLX[1] + mvdLX[1]) \& (2^{18} - 1) \quad (467)$$

$$mvLX[0][0][1] = (uLX[1] >= 2^{17}) ? (uLX[1] - 2^{18}) : uLX[1] \quad (468)$$

NOTE – The resulting values of mvLX[0][0][0] and mvLX[0][0][1] are in the range of -2^{17} to $2^{17} - 1$, inclusive.

– The half sample interpolation filter index hpelIfIdx is derived as follows:

$$hpelIfIdx = AmvrShift == 3 ? 1 : 0 \quad (469)$$

– The bi-prediction weight index bcwIdx is set equal to bcw_idx[xCb][yCb].

When all of the following conditions are true, refIdxL1 is set equal to −1, predFlagL1 is set equal to 0, and bcwIdx is set equal to 0:

– predFlagL0[0][0] is equal to 1.

– predFlagL1[0][0] is equal to 1.

– The value of (cbWidth + cbHeight) is equal to 12.

When (xCb + cbWidth) >> Log2ParMrgLevel is greater than xCb >> Log2ParMrgLevel and (yCb + cbHeight) >> Log2ParMrgLevel is greater than yCb >> Log2ParMrgLevel, the updating process for the history-based motion vector predictor list as specified in clause 8.5.2.16 is invoked with luma motion vectors mvL0[0][0] and mvL1[0][0], reference indices refIdxL0 and refIdxL1, prediction list utilization flags predFlagL0[0][0] and predFlagL1[0][0], the bi-prediction weight index bcwIdx, and the half-sample interpolation filter index hpelIfIdx.

8.5.2.2 Derivation process for luma motion vectors for merge mode

This process is only invoked when general_merge_flag[xCb][yCb] is equal to 1, where (xCb, yCb) specify the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture.

Inputs to this process are:

– a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples.

Outputs of this process are:

– the luma motion vectors in 1/16 fractional-sample accuracy mvL0[0][0] and mvL1[0][0],

– the reference indices refIdxL0 and refIdxL1,

– the prediction list utilization flags predFlagL0[0][0] and predFlagL1[0][0],

– the half sample interpolation filter index hpelIfIdx,

– the bi-prediction weight index bcwIdx,

– the merging candidate list mergeCandList.

The bi-prediction weight index bcwIdx is set equal to 0.

The motion vectors mvL0[0][0] and mvL1[0][0], the reference indices refIdxL0 and refIdxL1 and the prediction utilization flags predFlagL0[0][0] and predFlagL1[0][0] are derived by the following ordered steps:

1. The derivation process for spatial merging candidates from neighbouring coding units as specified in clause 8.5.2.3 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth, and the luma coding block height cbHeight as inputs, and the output being the availability flags $availableFlagA_0$, $availableFlagA_1$, $availableFlagB_0$, $availableFlagB_1$ and $availableFlagB_2$, the reference indices $refIdxLXA_0$, $refIdxLXA_1$, $refIdxLXB_0$, $refIdxLXB_1$ and $refIdxLXB_2$, the prediction list utilization flags $predFlagLXA_0$, $predFlagLXA_1$, $predFlagLXB_0$, $predFlagLXB_1$ and $predFlagLXB_2$, and the motion vectors $mvLXA_0$, $mvLXA_1$, $mvLXB_0$, $mvLXB_1$ and $mvLXB_2$, with X = 0..1, the half sample interpolation filter indices $hpelIfIdxA_0$, $hpelIfIdxA_1$, $hpelIfIdxB_0$, $hpelIfIdxB_1$, $hpelIfIdxB_2$, and the bi-prediction weight indices $bcwIdxA_0$, $bcwIdxA_1$, $bcwIdxB_0$, $bcwIdxB_1$, $bcwIdxB_2$.

2. The reference indices, refIdxL0Col and refIdxL1Col, and the bi-prediction weight index bcwIdxCol for the temporal merging candidate Col are set equal to 0 and hpelIfIdxCol is set equal to 0.

3. The derivation process for temporal luma motion vector prediction as specified in clause 8.5.2.11 is invoked with the luma location (xCb, yCb), the luma coding block width cbWidth, the luma coding block height cbHeight and the variable refIdxL0Col as inputs, and the output being the availability flag availableFlagL0Col and the temporal motion vector mvL0Col. The variables availableFlagCol, predFlagL0Col and predFlagL1Col are derived as follows:

   availableFlagCol = availableFlagL0Col (470)

   predFlagL0Col = availableFlagL0Col (471)

   predFlagL1Col = 0 (472)

4. When sh_slice_type is equal to B, the derivation process for temporal luma motion vector prediction as specified in clause 8.5.2.11 is invoked with the luma location (xCb, yCb), the luma coding block width cbWidth, the luma coding block height cbHeight and the variable refIdxL1Col as inputs, and the output being the availability flag availableFlagL1Col and the temporal motion vector mvL1Col. The variables availableFlagCol and predFlagL1Col are derived as follows:

   availableFlagCol = availableFlagL0Col || availableFlagL1Col (473)

   predFlagL1Col = availableFlagL1Col (474)

5. The merging candidate list, mergeCandList, is constructed as follows:

```
i = 0
if( availableFlagB1 )
    mergeCandList[ i++ ] = B1
if( availableFlagA1 )
    mergeCandList[ i++ ] = A1
if( availableFlagB0 )
    mergeCandList[ i++ ] = B0                                   (475)
if( availableFlagA0 )
    mergeCandList[ i++ ] = A0
if( availableFlagB2 )
    mergeCandList[ i++ ] = B2
if( availableFlagCol )
    mergeCandList[ i++ ] = Col
```

6. The variable numCurrMergeCand and numOrigMergeCand are set equal to the number of merging candidates in the mergeCandList.

7. When numCurrMergeCand is less than MaxNumMergeCand − 1 and NumHmvpCand is greater than 0, the following applies:

   – The derivation process of history-based merging candidates as specified in clause 8.5.2.6 is invoked with mergeCandList and numCurrMergeCand as inputs, and modified mergeCandList and numCurrMergeCand as outputs.

   – numOrigMergeCand is set equal to numCurrMergeCand.

8. When numCurrMergeCand is less than MaxNumMergeCand and greater than 1, the following applies:

– The derivation process for pairwise average merging candidate specified in clause 8.5.2.4 is invoked with mergeCandList, the reference indices refIdxL0N and refIdxL1N, the prediction list utilization flags predFlagL0N and predFlagL1N, the motion vectors mvL0N and mvL1N, and the half sample interpolation filter index hpelIfIdxN of every candidate N in mergeCandList, and numCurrMergeCand as inputs, and the output is assigned to mergeCandList, numCurrMergeCand, the reference indices refIdxL0avgCand and refIdxL1avgCand, the prediction list utilization flags predFlagL0avgCand and predFlagL1avgCand, the motion vectors mvL0avgCand and mvL1avgCand, and the half-sample interpolation filter index hpelIfIdxavgCand of candidate avgCand being added into mergeCandList. The bi-prediction weight index bcwIdx of candidate avgCand being added into mergeCandList is set equal to 0.

– numOrigMergeCand is set equal to numCurrMergeCand.

9. The derivation process for zero motion vector merging candidates specified in clause 8.5.2.5 is invoked with the mergeCandList, the reference indices refIdxL0N and refIdxL1N, the prediction list utilization flags predFlagL0N and predFlagL1N, the motion vectors mvL0N and mvL1N of every candidate N in mergeCandList and numCurrMergeCand as inputs, and the output is assigned to mergeCandList, numCurrMergeCand, the reference indices $refIdxL0zeroCand_m$ and $refIdxL1zeroCand_m$, the prediction list utilization flags $predFlagL0zeroCand_m$ and $predFlagL1zeroCand_m$ and the motion vectors $mvL0zeroCand_m$ and $mvL1zeroCand_m$ of every new candidate $zeroCand_m$ being added into mergeCandList. The half sample interploation filter index hpelIfIdx of every new candidate $zeroCand_m$ being added into mergeCandList is set equal to 0. The bi-prediction weight index bcwIdx of every new candidate $zeroCand_m$ being added into mergeCandList is set equal to 0. The number of candidates being added, numZeroMergeCand, is set equal to (numCurrMergeCand − numOrigMergeCand). When numZeroMergeCand is greater than 0, m ranges from 0 to numZeroMergeCand − 1, inclusive.

10. The following assignments are made with N being the candidate at position merge_idx[xCb][yCb] in the merging candidate list mergeCandList (N = mergeCandList[merge_idx[xCb][yCb]]):

hpelIfIdx = hpelIfIdxN (476)

bcwIdx = bcwIdxN (477)

– For X = 0..1, the following applies:

refIdxLX = refIdxLXN (478)

predFlagLX[0][0] = predFlagLXN (479)

mvLX[0][0][0] = mvLXN[0] (480)

mvLX[0][0][1] = mvLXN[1] (481)

11. When mmvd_merge_flag[xCb][yCb] is equal to 1, the following applies:

– The derivation process for merge motion vector difference as specified in clause 8.5.2.7 is invoked with the luma location (xCb, yCb), the reference indices refIdxL0, refIdxL1 and the prediction list utilization flags predFlagL0[0][0] and predFlagL1[0][0] as inputs, and the motion vector differences mMvdL0 and mMvdL1 as outputs.

– For X = 0..1, the motion vector difference mMvdLX is added to the merge motion vectors mvLX as follows:

mvLX[0][0][0] += mMvdLX[0] (482)

mvLX[0][0][1] += mMvdLX[1] (483)

mvLX[0][0][0] = Clip3(-2^{17}, $2^{17} - 1$, mvLX[0][0][0]) (484)

mvLX[0][0] [1] = Clip3(-2^{17}, $2^{17} - 1$, mvLX[0][0][1]) (485)

8.5.2.3 Derivation process for spatial merging candidates

Inputs to this process are:

– a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples.

Outputs of this process are as follows, with X = 0..1:

– the availability flags $availableFlagA_0$, $availableFlagA_1$, $availableFlagB_0$, $availableFlagB_1$ and $availableFlagB_2$ of the neighbouring coding units,

– the reference indices $refIdxLXA_0$, $refIdxLXA_1$, $refIdxLXB_0$, $refIdxLXB_1$ and $refIdxLXB_2$ of the neighbouring coding units,

– the prediction list utilization flags $predFlagLXA_0$, $predFlagLXA_1$, $predFlagLXB_0$, $predFlagLXB_1$ and $predFlagLXB_2$ of the neighbouring coding units,

– the motion vectors in 1/16 fractional-sample accuracy $mvLXA_0$, $mvLXA_1$, $mvLXB_0$, $mvLXB_1$ and $mvLXB_2$ of the neighbouring coding units,

– the half sample interpolation filter indices $hpelIfIdxA_0$, $hpelIfIdxA_1$, $hpelIfIdxB_0$, $hpelIfIdxB_1$, and $hpelIfIdxB_2$,

– the bi-prediction weight indices $bcwIdxA_0$, $bcwIdxA_1$, $bcwIdxB_0$, $bcwIdxB_1$, and $bcwIdxB_2$.

For the derivation of $availableFlagB_1$, $refIdxLXB_1$, $predFlagLXB_1$, $mvLXB_1$, $hpelIfIdxB_1$ and $bcwIdxB_1$ the following applies:

– The luma location ($xNbB_1$, $yNbB_1$) inside the neighbouring luma coding block is set equal to (xCb + cbWidth − 1, yCb − 1).

– The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring luma location ($xNbB_1$, $yNbB_1$), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag $availableB_1$.

– When xCb >> Log2ParMrgLevel is equal to $xNbB_1$ >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to $yNbB_1$ >> Log2ParMrgLevel, $availableB_1$ is set equal to FALSE.

– The variables $availableFlagB_1$, $refIdxLXB_1$, $predFlagLXB_1$, $mvLXB_1$, $hpelIfIdxB_1$ and $bcwIdxB_1$ are derived as follows:

  – If $availableB_1$ is equal to FALSE, $availableFlagB_1$ is set equal to 0, both components of $mvLXB_1$ are set equal to 0, $refIdxLXB_1$ is set equal to −1 and $predFlagLXB_1$ is set equal to 0, with X = 0..1, $hpelIfIdxB_1$ is set equal to 0, and $bcwIdxB_1$ is set equal to 0.

  – Otherwise, $availableFlagB_1$ is set equal to 1 and the following assignments are made:

$$mvLXB_1 = MvLX[\ xNbB_1\ ][\ yNbB_1\ ] \tag{486}$$

$$refIdxLXB_1 = RefIdxLX[\ xNbB_1\ ][\ yNbB_1\ ] \tag{487}$$

$$predFlagLXB_1 = PredFlagLX[\ xNbB_1\ ][\ yNbB_1\ ] \tag{488}$$

$$hpelIfIdxB_1 = HpelIfIdx[\ xNbB_1\ ][\ yNbB_1\ ] \tag{489}$$

$$bcwIdxB_1 = BcwIdx[\ xNbB_1\ ][\ yNbB_1\ ] \tag{490}$$

For the derivation of $availableFlagA_1$, $refIdxLXA_1$, $predFlagLXA_1$, $mvLXA_1$, $hpelIfIdxA_1$ and $bcwIdxA_1$ the following applies:

– The luma location ($xNbA_1$, $yNbA_1$) inside the neighbouring luma coding block is set equal to (xCb − 1, yCb + cbHeight − 1).

– The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring luma location ($xNbA_1$, $yNbA_1$), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag $availableA_1$.

– When xCb >> Log2ParMrgLevel is equal to $xNbA_1$ >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to $yNbA_1$ >> Log2ParMrgLevel, $availableA_1$ is set equal to FALSE.

– The variables $availableFlagA_1$, $refIdxLXA_1$, $predFlagLXA_1$, $mvLXA_1$, $hpelIfIdxA_1$ and $bcwIdxA_1$ are derived as follows:

– If one or more of the following conditions are true, $availableFlagA_1$ is set equal to 0, both components of $mvLXA_1$ are set equal to 0, $refIdxLXA_1$ is set equal to −1 and $predFlagLXA_1$ is set equal to 0, with X = 0..1, $hpelIfIdxA_1$ is set equal to 0, and $bcwIdxA_1$ is set equal to 0:

  – $availableA_1$ is equal to FALSE.

  – $availableB_1$ is equal to TRUE and the luma locations ($xNbA_1$, $yNbA_1$) and ($xNbB_1$, $yNbB_1$) have the same motion vectors and the same reference indices.

– Otherwise, $availableFlagA_1$ is set equal to 1 and the following assignments are made:

$$mvLXA_1 = MvLX[xNbA_1][yNbA_1] \quad (491)$$

$$refIdxLXA_1 = RefIdxLX[xNbA_1][yNbA_1] \quad (492)$$

$$predFlagLXA_1 = PredFlagLX[xNbA_1][yNbA_1] \quad (493)$$

$$hpelIfIdxA_1 = HpelIfIdx[xNbA_1][yNbA_1] \quad (494)$$

$$bcwIdxA_1 = BcwIdx[xNbA_1][yNbA_1] \quad (495)$$

For the derivation of $availableFlagB_0$, $refIdxLXB_0$, $predFlagLXB_0$, $mvLXB_0$, $hpelIfIdxB_0$ and $bcwIdxB_0$ the following applies:

– The luma location ($xNbB_0$, $yNbB_0$) inside the neighbouring luma coding block is set equal to (xCb + cbWidth, yCb − 1).

– The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring luma location ($xNbB_0$, $yNbB_0$), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag $availableB_0$.

– When xCb >> Log2ParMrgLevel is equal to $xNbB_0$ >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to $yNbB_0$ >> Log2ParMrgLevel, $availableB_0$ is set equal to FALSE.

– The variables $availableFlagB_0$, $refIdxLXB_0$, $predFlagLXB_0$, $mvLXB_0$, $hpelIfIdxB_0$ and $bcwIdxB_0$ are derived as follows:

  – If one or more of the following conditions are true, $availableFlagB_0$ is set equal to 0, both components of $mvLXB_0$ are set equal to 0, $refIdxLXB_0$ is set equal to −1 and $predFlagLXB_0$ is set equal to 0, with X = 0..1, $hpelIfIdxB_0$ is set equal to 0, and $bcwIdxB_0$ is set equal to 0:

    – $availableB_0$ is equal to FALSE.

    – $availableB_1$ is equal to TRUE and the luma locations ($xNbB_1$, $yNbB_1$) and ($xNbB_0$, $yNbB_0$) have the same motion vectors and the same reference indices.

  – Otherwise, $availableFlagB_0$ is set equal to 1 and the following assignments are made:

$$mvLXB_0 = MvLX[xNbB_0][yNbB_0] \quad (496)$$

$$refIdxLXB_0 = RefIdxLX[xNbB_0][yNbB_0] \quad (497)$$

$$predFlagLXB_0 = PredFlagLX[xNbB_0][yNbB_0] \quad (498)$$

$$hpelIfIdxB_0 = HpelIfIdx[xNbB_0][yNbB_0] \quad (499)$$

$$bcwIdxB_0 = BcwIdx[xNbB_0][yNbB_0] \quad (500)$$

For the derivation of $availableFlagA_0$, $refIdxLXA_0$, $predFlagLXA_0$, $mvLXA_0$, $hpelIfIdxA_0$ and $bcwIdxA_0$ the following applies:

– The luma location ($xNbA_0$, $yNbA_0$) inside the neighbouring luma coding block is set equal to (xCb − 1, yCb + cbHeight).

– The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring luma location ($xNbA_0$, $yNbA_0$), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag $availableA_0$.

– When xCb >> Log2ParMrgLevel is equal to $xNbA_0$ >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to $yNbA_0$ >> Log2ParMrgLevel, $availableA_0$ is set equal to FALSE.

– The variables $availableFlagA_0$, $refIdxLXA_0$, $predFlagLXA_0$, $mvLXA_0$, $hpelIfIdxA_0$ and $bcwIdxA_0$ are derived as follows:

  – If one or more of the following conditions are true, $availableFlagA_0$ is set equal to 0, both components of $mvLXA_0$ are set equal to 0, $refIdxLXA_0$ is set equal to −1 and $predFlagLXA_0$ is set equal to 0, with X = 0..1, $hpelIfIdxA_0$ is set equal to 0, and $bcwIdxA_0$ is set equal to 0:

    – $availableA_0$ is equal to FALSE.

    – $availableA_1$ is equal to TRUE and the luma locations ($xNbA_1$, $yNbA_1$) and ($xNbA_0$, $yNbA_0$) have the same motion vectors and the same reference indices.

  – Otherwise, $availableFlagA_0$ is set equal to 1 and the following assignments are made:

$$mvLXA_0 = MvLX[\ xNbA_0\ ][\ yNbA_0\ ] \tag{501}$$

$$refIdxLXA_0 = RefIdxLX[\ xNbA_0\ ][\ yNbA_0\ ] \tag{502}$$

$$predFlagLXA_0 = PredFlagLX[\ xNbA_0\ ][\ yNbA_0\ ] \tag{503}$$

$$hpelIfIdxA_0 = HpelIfIdx[\ xNbA_0\ ][\ yNbA_0\ ] \tag{504}$$

$$bcwIdxA_0 = BcwIdx[\ xNbA_0\ ][\ yNbA_0\ ] \tag{505}$$

For the derivation of $availableFlagB_2$, $refIdxLXB_2$, $predFlagLXB_2$, $mvLXB_2$, $hpelIfIdxB_2$ and $bcwIdxB_2$ the following applies:

– The luma location ($xNbB_2$, $yNbB_2$) inside the neighbouring luma coding block is set equal to (xCb − 1, yCb − 1).

– The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring luma location ($xNbB_2$, $yNbB_2$), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag $availableB_2$.

– When xCb >> Log2ParMrgLevel is equal to $xNbB_2$ >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to $yNbB_2$ >> Log2ParMrgLevel, $availableB_2$ is set equal to FALSE.

– The variables $availableFlagB_2$, $refIdxLXB_2$, $predFlagLXB_2$, $mvLXB_2$, $hpelIfIdxB_2$ and $bcwIdxB_2$ are derived as follows:

  – If one or more of the following conditions are true, $availableFlagB_2$ is set equal to 0, both components of $mvLXB_2$ are set equal to 0, $refIdxLXB_2$ is set equal to −1 and $predFlagLXB_2$ is set equal to 0, with X = 0..1, $hpelIfIdxB_2$ is set equal to 0, and $bcwIdxB_2$ is set equal to 0:

    – $availableB_2$ is equal to FALSE.

    – $availableA_1$ is equal to TRUE and the luma locations ($xNbA_1$, $yNbA_1$) and ($xNbB_2$, $yNbB_2$) have the same motion vectors and the same reference indices.

    – $availableB_1$ is equal to TRUE and the luma locations ($xNbB_1$, $yNbB_1$) and ($xNbB_2$, $yNbB_2$) have the same motion vectors and the same reference indices.

    – $availableFlagA_0 + availableFlagA_1 + availableFlagB_0 + availableFlagB_1$ is equal to 4.

  – Otherwise, $availableFlagB_2$ is set equal to 1 and the following assignments are made:

$$mvLXB_2 = MvLX[\ xNbB_2\ ][\ yNbB_2\ ] \tag{506}$$

$$refIdxLXB_2 = RefIdxLX[\ xNbB_2\ ][\ yNbB_2\ ] \tag{507}$$

$$predFlagLXB_2 = PredFlagLX[\ xNbB_2\ ][\ yNbB_2\ ] \tag{508}$$

$$hpelIfIdxB_2 = HpelIfIdx[\ xNbB_2\ ][\ yNbB_2\ ] \tag{509}$$

$$bcwIdxB_2 = BcwIdx[xNbB_2][yNbB_2] \quad (510)$$

8.5.2.4 Derivation process for pairwise average merging candidate

Inputs to this process are:

– a merging candidate list mergeCandList,

– the reference indices refIdxL0N and refIdxL1N of every candidate N in mergeCandList,

– the prediction list utilization flags predFlagL0N and predFlagL1N of every candidate N in mergeCandList,

– the motion vectors in 1/16 fractional-sample accuracy mvL0N and mvL1N of every candidate N in mergeCandList,

– the half sample interpolation filter index hpelIfIdxN of every candidate N in mergeCandList,

– the number of elements numCurrMergeCand within mergeCandList.

Outputs of this process are:

– the merging candidate list mergeCandList,

– the number of elements numCurrMergeCand within mergeCandList,

– the reference indices refIdxL0avgCand and refIdxL1avgCand of candidate avgCand added into mergeCandList during the invocation of this process,

– the prediction list utilization flags predFlagL0avgCand and predFlagL1avgCand of candidate avgCand added into mergeCandList during the invocation of this process,

– the motion vectors in 1/16 fractional-sample accuracy mvL0avgCand and mvL1avgCand of candidate avgCand added into mergeCandList during the invocation of this process,

– the half sample interpolation filter index hpelIfIdxavgCand of every candidate avgCand added into mergeCandList during the invocation of this process.

The variable numRefLists is derived as follows:

$$numRefLists = (sh\_slice\_type == B) ? 2 : 1 \quad (511)$$

The following assignments are made, with p0Cand being the candidate at position 0 and p1Cand being the candidate at position 1 in the merging candidate list mergeCandList:

$$p0Cand = mergeCandList[0] \quad (512)$$

$$p1Cand = mergeCandList[1] \quad (513)$$

The candidate avgCand is added at the end of mergeCandList, i.e., mergeCandList[numCurrMergeCand] is set equal to avgCand, and the reference indices, the prediction list utilization flags and the motion vectors of avgCand are derived as follows and numCurrMergeCand is incremented by 1:

– For each RPL LX with X ranging from 0 to (numRefLists − 1), the following applies:

  – If predFlagLXp0Cand is equal to 1 and predFlagLXp1Cand is equal to 1, the variables refIdxLXavgCand, predFlagLXavgCand, mvLXavgCand[0], and mvLXavgCand[1] are derived as follows:

$$refIdxLXavgCand = refIdxLXp0Cand \quad (514)$$

$$predFlagLXavgCand = 1 \quad (515)$$

    – The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX[0] set equal to mvLXp0Cand[0] + mvLXp1Cand[0], mvX[1] set equal to mvLXp0Cand[1] + mvLXp1Cand[1], rightShift set equal to 1, and leftShift set equal to 0 as inputs and the rounded mvLXavgCand as output.

  – Otherwise, if predFlagLXp0Cand is equal to 1 and predFlagLXp1Cand is equal to 0, the variables refIdxLXavgCand, predFlagLXavgCand, mvLXavgCand[0], mvLXavgCand[1] are derived as follows:

$$refIdxLXavgCand = refIdxLXp0Cand \quad (516)$$

$$predFlagLXavgCand = 1 \quad (517)$$

mvLXavgCand[0] = mvLXp0Cand[0] (518)

mvLXavgCand[1] = mvLXp0Cand[1] (519)

– Otherwise, if predFlagLXp0Cand is equal to 0 and predFlagLXp1Cand is equal to 1, the variables refIdxLXavgCand, predFlagLXavgCand, mvLXavgCand[0], mvLXavgCand[1] are derived as follows:

refIdxLXavgCand = refIdxLXp1Cand (520)

predFlagLXavgCand = 1 (521)

mvLXavgCand[0] = mvLXp1Cand[0] (522)

mvLXavgCand[1] = mvLXp1Cand[1] (523)

– Otherwise, if predFlagLXp0Cand is equal to 0 and predFlagLXp1Cand is equal to 0, the variables refIdxLXavgCand, predFlagLXavgCand, mvLXavgCand[0], mvLXavgCand[1] are derived as follows:

refIdxLXavgCand = −1 (524)

predFlagLXavgCand = 0 (525)

mvLXavgCand[0] = 0 (526)

mvLXavgCand[1] = 0 (527)

– When numRefLists is equal to 1, the following applies:

refIdxL1avgCand = −1 (528)

predFlagL1avgCand = 0 (529)

– The half sample interpolation filter index hpelIfIdxavgCand is derived as follows:

  – If hpelIfIdxp0Cand is equal to hpelIfIdxp1Cand, hpelIfIdxavgCand is set equal to hpelIfIdxp0Cand.

  – Otherwise, hpelIfIdxavgCand is set equal to 0.

8.5.2.5 Derivation process for zero motion vector merging candidates

Inputs to this process are:

– a merging candidate list mergeCandList,

– the reference indices refIdxL0N and refIdxL1N of every candidate N in mergeCandList,

– the prediction list utilization flags predFlagL0N and predFlagL1N of every candidate N in mergeCandList,

– the motion vectors mvL0N and mvL1N of every candidate N in mergeCandList,

– the number of elements numCurrMergeCand within mergeCandList.

Outputs of this process are:

– the merging candidate list mergeCandList,

– the number of elements numCurrMergeCand within mergeCandList,

– the reference indices $refIdxL0zeroCand_m$ and $refIdxL1zeroCand_m$ of every new candidate $zeroCand_m$ added into mergeCandList during the invocation of this process,

– the prediction list utilization flags $predFlagL0zeroCand_m$ and $predFlagL1zeroCand_m$ of every new candidate $zeroCand_m$ added into mergeCandList during the invocation of this process,

– the motion vectors $mvL0zeroCand_m$ and $mvL1zeroCand_m$ of every new candidate $zeroCand_m$ added into mergeCandList during the invocation of this process.

The variable numRefIdx is derived as follows:

– If sh_slice_type is equal to P, numRefIdx is set equal to NumRefIdxActive[0].

– Otherwise (sh_slice_type is equal to B), numRefIdx is set equal to Min(NumRefIdxActive[0], NumRefIdxActive[1]).

When numCurrMergeCand is less than MaxNumMergeCand, the variable numInputMergeCand is set equal to numCurrMergeCand, the variable zeroIdx is set equal to 0 and the following ordered steps are repeated until numCurrMergeCand is equal to MaxNumMergeCand:

1. For the derivation of the reference indices, the prediction list utilization flags and the motion vectors of the zero motion vector merging candidate, the following applies:

   – If sh_slice_type is equal to P, the candidate $zeroCand_m$ with m equal to (numCurrMergeCand − numInputMergeCand) is added at the end of mergeCandList, i.e., mergeCandList[numCurrMergeCand] is set equal to $zeroCand_m$, and the reference indices, the prediction list utilization flags and the motion vectors of $zeroCand_m$ are derived as follows and numCurrMergeCand is incremented by 1:

$$refIdxL0zeroCand_m = (zeroIdx < numRefIdx) ? zeroIdx : 0 \tag{530}$$

$$refIdxL1zeroCand_m = -1 \tag{531}$$

$$predFlagL0zeroCand_m = 1 \tag{532}$$

$$predFlagL1zeroCand_m = 0 \tag{533}$$

$$mvL0zeroCand_m[0] = 0 \tag{534}$$

$$mvL0zeroCand_m[1] = 0 \tag{535}$$

$$mvL1zeroCand_m[0] = 0 \tag{536}$$

$$mvL1zeroCand_m[1] = 0 \tag{537}$$

$$numCurrMergeCand = numCurrMergeCand + 1 \tag{538}$$

   – Otherwise (sh_slice_type is equal to B), the candidate $zeroCand_m$ with m equal to (numCurrMergeCand − numInputMergeCand) is added at the end of mergeCandList, i.e., mergeCandList[numCurrMergeCand] is set equal to $zeroCand_m$, and the reference indices, the prediction list utilization flags and the motion vectors of $zeroCand_m$ are derived as follows and numCurrMergeCand is incremented by 1:

$$refIdxL0zeroCand_m = (zeroIdx < numRefIdx) ? zeroIdx : 0 \tag{539}$$

$$refIdxL1zeroCand_m = (zeroIdx < numRefIdx) ? zeroIdx : 0 \tag{540}$$

$$predFlagL0zeroCand_m = 1 \tag{541}$$

$$predFlagL1zeroCand_m = 1 \tag{542}$$

$$mvL0zeroCand_m[0] = 0 \tag{543}$$

$$mvL0zeroCand_m[1] = 0 \tag{544}$$

$$mvL1zeroCand_m[0] = 0 \tag{545}$$

$$mvL1zeroCand_m[1] = 0 \tag{546}$$

$$numCurrMergeCand = numCurrMergeCand + 1 \tag{547}$$

2. The variable zeroIdx is incremented by 1.

8.5.2.6 Derivation process for history-based merging candidates

Inputs to this process are:

– a merge candidate list mergeCandList,

– the number of available merging candidates in the list numCurrMergeCand.

Outputs of this process are:

– the modified merging candidate list mergeCandList,

– the modified number of merging candidates in the list numCurrMergeCand.

For each candidate in HmvpCandList[NumHmvpCand − hMvpIdx] with index hMvpIdx = 1..NumHmvpCand, the following ordered steps are repeated until numCurrMergeCand is equal to MaxNumMergeCand − 1:

1. The variable sameMotion is derived as follows:
   - If all of the following conditions are true for any merging candidate N with N being A_1 or B_1, sameMotion is set equal to TRUE:
     - hMvpIdx is less than or equal to 2.
     - The candidate HmvpCandList[NumHmvpCand − hMvpIdx] and the merging candidate N have the same motion vectors and the same reference indices.
   - Otherwise, sameMotion is set equal to FALSE.
2. When sameMotion is equal to FALSE, the candidate HmvpCandList[NumHmvpCand − hMvpIdx] is added to the merging candidate list as follows:

mergeCandList[numCurrMergeCand++] = HmvpCandList[NumHmvpCand − hMvpIdx] (548)

8.5.2.7 Derivation process for merge motion vector difference

Inputs to this process are:

– a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– reference indices refIdxL0 and refIdxL1,

– prediction list utilization flags predFlagL0 and predFlagL1.

Outputs of this process are the luma merge motion vector differences in 1/16 fractional-sample accuracy mMvdL0 and mMvdL1.

The variable currPic specifies the current picture.

The luma merge motion vector differences mMvdL0 and mMvdL1 are derived as follows:

– If both predFlagL0 and predFlagL1 are equal to 1, the following applies:

currPocDiffL0 = DiffPicOrderCnt(currPic, RefPicList[0][refIdxL0]) (549)

currPocDiffL1 = DiffPicOrderCnt(currPic, RefPicList[1][refIdxL1]) (550)

  – If currPocDiffL0 is equal to currPocDiffL1, the following applies:

mMvdL0[0] = MmvdOffset[xCb][yCb][0] (551)

mMvdL0[1] = MmvdOffset[xCb][yCb][1] (552)

mMvdL1[0] = MmvdOffset[xCb][yCb][0] (553)

mMvdL1[1] = MmvdOffset[xCb][yCb][1] (554)

  – Otherwise, if Abs(currPocDiffL0) is greater than or equal to Abs(currPocDiffL1), the following applies:

mMvdL0[0] = MmvdOffset[xCb][yCb][0] (555)

mMvdL0[1] = MmvdOffset[xCb][yCb][1] (556)

    – If RefPicList[0][refIdxL0] is not marked as "used for long-term reference" and RefPicList[1][refIdxL1] is not marked as "used for long-term reference", the following applies:

$$\text{td} = \text{Clip3}(-128, 127, \text{currPocDiffL0}) \quad (557)$$

$$\text{tb} = \text{Clip3}(-128, 127, \text{currPocDiffL1}) \quad (558)$$

$$\text{tx} = (16384 + (\text{Abs}(\text{td}) >> 1)) / \text{td} \quad (559)$$

$$\text{distScaleFactor} = \text{Clip3}(-4096, 4095, (\text{tb} * \text{tx} + 32) >> 6) \quad (560)$$

$$\text{mMvdL1}[0] = \text{Clip3}(-2^{17}, 2^{17} - 1, (\text{distScaleFactor} * \text{mMvdL0}[0] + 128 - (\text{distScaleFactor} * \text{mMvdL0}[0] >= 0)) >> 8) \quad (561)$$

$$\text{mMvdL1}[1] = \text{Clip3}(-2^{17}, 2^{17} - 1, (\text{distScaleFactor} * \text{mMvdL0}[1] + 128 - (\text{distScaleFactor} * \text{mMvdL0}[1] >= 0)) >> 8) \quad (562)$$

– Otherwise, the following applies:

$$\text{mMvdL1}[0] = \text{Sign}(\text{currPocDiffL0}) == \text{Sign}(\text{currPocDiffL1}) ? \text{mMvdL0}[0] : -\text{mMvdL0}[0] \quad (563)$$

$$\text{mMvdL1}[1] = \text{Sign}(\text{currPocDiffL0}) == \text{Sign}(\text{currPocDiffL1}) ? \text{mMvdL0}[1] : -\text{mMvdL0}[1] \quad (564)$$

– Otherwise (Abs(currPocDiffL0) is less than Abs(currPocDiffL1)), the following applies:

$$\text{mMvdL1}[0] = \text{MmvdOffset}[\text{xCb}][\text{yCb}][0] \quad (565)$$

$$\text{mMvdL1}[1] = \text{MmvdOffset}[\text{xCb}][\text{yCb}][1] \quad (566)$$

– If RefPicList[0][refIdxL0] is not marked as "used for long-term reference" and RefPicList[1][refIdxL1] is not marked as "used for long-term reference", the following applies:

$$\text{td} = \text{Clip3}(-128, 127, \text{currPocDiffL1}) \quad (567)$$

$$\text{tb} = \text{Clip3}(-128, 127, \text{currPocDiffL0}) \quad (568)$$

$$\text{tx} = (16384 + (\text{Abs}(\text{td}) >> 1)) / \text{td} \quad (569)$$

$$\text{distScaleFactor} = \text{Clip3}(-4096, 4095, (\text{tb} * \text{tx} + 32) >> 6) \quad (570)$$

$$\text{mMvdL0}[0] = \text{Clip3}(-2^{17}, 2^{17} - 1, (\text{distScaleFactor} * \text{mMvdL1}[0] + 128 - (\text{distScaleFactor} * \text{mMvdL1}[0] >= 0)) >> 8) \quad (571)$$

$$\text{mMvdL0}[1] = \text{Clip3}(-2^{17}, 2^{17} - 1, (\text{distScaleFactor} * \text{mMvdL1}[1] + 128 - (\text{distScaleFactor} * \text{mMvdL1}[1] >= 0)) >> 8) \quad (572)$$

– Otherwise, the following applies:

$$\text{mMvdL0}[0] = \text{Sign}(\text{currPocDiffL0}) == \text{Sign}(\text{currPocDiffL1}) ? \text{mMvdL1}[0] : -\text{mMvdL1}[0] \quad (573)$$

$$\text{mMvdL0}[1] = \text{Sign}(\text{currPocDiffL0}) == \text{Sign}(\text{currPocDiffL1}) ? \text{mMvdL1}[1] : -\text{mMvdL1}[1] \quad (574)$$

– Otherwise (predFlagL0 or predFlagL1 is equal to 1), the following applies for X = 0..1:

$$\text{mMvdLX}[0] = (\text{predFlagLX} == 1) ? \text{MmvdOffset}[\text{xCb}][\text{yCb}][0] : 0 \quad (575)$$

$$\text{mMvdLX}[1] = (\text{predFlagLX} == 1) ? \text{MmvdOffset}[\text{xCb}][\text{yCb}][1] : 0 \quad (576)$$

8.5.2.8 Derivation process for luma motion vector prediction

Inputs to this process are:

– a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples,

– the reference index of the current coding unit partition refIdxLX, with X being 0 or 1.

Output of this process is the prediction mvpLX in 1/16 fractional-sample accuracy of the motion vector mvLX, with X being 0 or 1.

The motion vector predictor mvpLX is derived in the following ordered steps:

1. The derivation process for motion vector predictor candidate list as specified in clause 8.5.2.9 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth, the luma coding block height cbHeight and refIdxLX, as inputs, and the motion vector predictor candidate list, mvpListLX, as output.

2. The motion vector predictor mvpLX is derived as follows:

$$\text{mvpLX} = \text{mvpListLX}[\ \text{mvp\_lX\_flag}[\ \text{xCb}\ ][\ \text{yCb}\ ]\ ] \tag{577}$$

8.5.2.9 Derivation process for motion vector predictor candidate list

Inputs to this process are:

– a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples,

– the reference index of the current coding unit partition refIdxLX, with X being 0 or 1.

Output of this process is motion vector predictor candidate list mvpListLX in 1/16 fractional-sample accuracy with X being 0 or 1.

The motion vector predictor candidate list mvpListLX is derived in the following ordered steps:

1. The derivation process for spatial motion vector predictor candidates from neighbouring coding unit partitions as specified in clause 8.5.2.10 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth, the luma coding block height cbHeight and refIdxLX as inputs, and the availability flags availableFlagLXN and the motion vectors mvLXN, with N being replaced by A or B, as output.

2. The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to mvLXN, with N being replaced by A or B, rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded mvLXN, with N being replaced by A or B, as output.

3. The availability flag availableFlagLXCol and the temporal motion vector predictor mvLXCol are derived as follows:
   - If both availableFlagLXA and availableFlagLXB are equal to 1 and mvLXA is not equal to mvLXB, availableFlagLXCol is set equal to 0.
   - Otherwise (availableFlagLXA is equal to 0, availableFlagLXB is equal to 0, or mvLXA is equal to mvLXB), the following applies:
     - The derivation process for temporal luma motion vector prediction as specified in clause 8.5.2.11 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth, the luma coding block height cbHeight and refIdxLX as inputs, and with the output being the availability flag availableFlagLXCol and the temporal motion vector predictor mvLXCol.
     - The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to mvLXCol, rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded mvLXCol as output.

4. The motion vector predictor candidate list, mvpListLX, is constructed as follows:

```
numCurrMvpCand = 0
if( availableFlagLXA ) {
    mvpListLX[ numCurrMvpCand++ ] = mvLXA
    if( availableFlagLXB  &&  ( mvLXA  !=  mvLXB ) )
        mvpListLX[ numCurrMvpCand++ ] = mvLXB                                    (578)
} else if( availableFlagLXB )
    mvpListLX[ numCurrMvpCand++ ] = mvLXB
if( numCurrMvpCand < 2  &&  availableFlagLXCol )
    mvpListLX[ numCurrMvpCand++ ] = mvLXCol
```

5. When numCurrMvpCand is less than 2 and NumHmvpCand is greater than 0, the following applies for i= 1..Min(4, NumHmvpCand) until numCurrMvpCand is equal to 2:
   – For each RPL LY with Y equal to X or (1 − X), the following applies until numCurrMvpCand is equal to 2:
     – When the reference picture corresponding to the reference index of the history-based motion vector predictor candidate HmvpCandList[i − 1] in the RPL LY is the same as the reference picture corresponding to reference index refIdxLX in the RPL LX, the following applies:
       – The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to the LY motion vector of the candidate HmvpCandList[i − 1], rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded LY motion vector of the candidate HmvpCandList[i − 1] as output is assigned to mvpListLX[numCurrMvpCand++].
6. When numCurrMvpCand is less than 2, the following applies until numCurrMvpCand is equal to 2:

mvpListLX[numCurrMvpCand][0] = 0 (579)

mvpListLX[numCurrMvpCand][1] = 0 (580)

numCurrMvpCand++ (581)

8.5.2.10 Derivation process for spatial motion vector predictor candidates

Inputs to this process are:

– a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples,

– the reference index of the current coding unit partition refIdxLX, with X being 0 or 1.

Outputs of this process are (with N being replaced by A or B):

– the motion vectors mvLXN in 1/16 fractional-sample accuracy of the neighbouring coding units,

– the availability flags availableFlagLXN of the neighbouring coding units.

Figure 10 provides an overview of spatial motion vector neighbours.

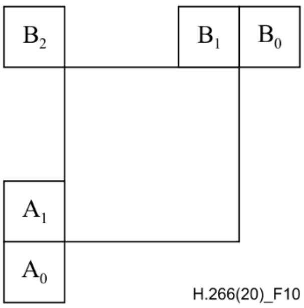

Figure 10 – Spatial motion vector neighbours (informative)

The motion vector mvLXA and the availability flag availableFlagLXA are derived in the following ordered steps:

1. The sample location ($xNbA_0$, $yNbA_0$) is set equal to (xCb − 1, yCb + cbHeight) and the sample location ($xNbA_1$, $yNbA_1$) is set equal to ($xNbA_0$, $yNbA_0$ − 1).

2. The availability flag availableFlagLXA is set equal to 0 and both components of mvLXA are set equal to 0.

3. The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring luma location ($xNbA_0$, $yNbA_0$), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag $availableA_0$.

4. The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring luma location ($xNbA_1$, $yNbA_1$), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag $availableA_1$.

5. The following applies for ($xNbA_k$, $yNbA_k$) from ($xNbA_0$, $yNbA_0$) to ($xNbA_1$, $yNbA_1$):

   – When $availableA_k$ is equal to TRUE and availableFlagLXA is equal to 0, the following applies:

     – If PredFlagLX[$xNbA_k$][$yNbA_k$] is equal to 1 and DiffPicOrderCnt(RefPicList[X][RefIdxLX[$xNbA_k$][$yNbA_k$]], RefPicList[X][refIdxLX]) is equal to 0, availableFlagLXA is set equal to 1 and the following applies:

$$mvLXA = MvLX[\ xNbA_k\ ][\ yNbA_k\ ] \tag{582}$$

     – Otherwise, when PredFlagLY[$xNbA_k$][$yNbA_k$] (with Y = !X) is equal to 1 and DiffPicOrderCnt(RefPicList[Y][RefIdxLY[$xNbA_k$][$yNbA_k$]], RefPicList[X][refIdxLX]) is equal to 0, availableFlagLXA is set equal to 1 and the following applies:

$$mvLXA = MvLY[\ xNbA_k\ ][\ yNbA_k\ ] \tag{583}$$

The motion vector mvLXB and the availability flag availableFlagLXB are derived in the following ordered steps:

1. The sample locations ($xNbB_0$, $yNbB_0$), ($xNbB_1$, $yNbB_1$) and ($xNbB_2$, $yNbB_2$) are set equal to (xCb + cbWidth, yCb − 1), (xCb + cbWidth − 1, yCb − 1) and (xCb − 1, yCb − 1), respectively.

2. The availability flag availableFlagLXB is set equal to 0 and both components of mvLXB are set equal to 0.

3. The following applies for ($xNbB_k$, $yNbB_k$) from ($xNbB_0$, $yNbB_0$) to ($xNbB_2$, $yNbB_2$):

   – The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring luma location ($xNbB_k$, $yNbB_k$), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag $availableB_k$.

   – When $availableB_k$ is equal to TRUE and availableFlagLXB is equal to 0, the following applies:

     – If PredFlagLX[$xNbB_k$][$yNbB_k$] is equal to 1, and DiffPicOrderCnt(RefPicList[X][RefIdxLX[$xNbB_k$][$yNbB_k$]], RefPicList[X][refIdxLX]) is equal to 0, availableFlagLXB is set equal to 1 and the following assignment is made:

$$mvLXB = MvLX[\ xNbB_k\ ][\ yNbB_k\ ] \tag{584}$$

     – Otherwise, when PredFlagLY[$xNbB_k$][$yNbB_k$] (with Y = !X) is equal to 1 and DiffPicOrderCnt(RefPicList[Y][RefIdxLY[$xNbB_k$][$yNbB_k$]], RefPicList[X][refIdxLX]) is equal to 0, availableFlagLXB is set equal to 1 and the following assignment is made:

$$mvLXB = MvLY[\ xNbB_k\ ][\ yNbB_k\ ] \tag{585}$$

8.5.2.11 Derivation process for temporal luma motion vector prediction

Inputs to this process are:

– a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples,

– a reference index refIdxLX, with X being 0 or 1.

Outputs of this process are:

– the motion vector prediction mvLXCol in 1/16 fractional-sample accuracy,

– the availability flag availableFlagLXCol.

The variable currCb specifies the current luma coding block at luma location (xCb, yCb).

The variables mvLXCol and availableFlagLXCol are derived as follows:

– If ph_temporal_mvp_enabled_flag is equal to 0 or (cbWidth * cbHeight) is less than or equal to 32, both components of mvLXCol are set equal to 0 and availableFlagLXCol is set equal to 0.

– Otherwise (ph_temporal_mvp_enabled_flag is equal to 1), the following ordered steps apply:

1. The bottom-right collocated motion vector and the bottom and right boundary sample locations are derived as follows:

xColBr = xCb + cbWidth (586)

yColBr = yCb + cbHeight (587)

rightBoundaryPos = sps_subpic_treated_as_pic_flag[CurrSubpicIdx] ?
  SubpicRightBoundaryPos : pps_pic_width_in_luma_samples − 1 (588)

botBoundaryPos = sps_subpic_treated_as_pic_flag[CurrSubpicIdx] ?
  SubpicBotBoundaryPos : pps_pic_height_in_luma_samples − 1 (589)

  – If yCb >> CtbLog2SizeY is equal to yColBr >> CtbLog2SizeY, yColBr is less than or equal to botBoundaryPos and xColBr is less than or equal to rightBoundaryPos, the following applies:

    – The luma location (xColCb, yColCb) is set equal to ((xColBr >> 3) << 3, (yColBr >> 3) << 3).

    – The variable colCb specifies the luma coding block covering the location (xColCb, yColCb) inside the collocated picture specified by ColPic.

    – The derivation process for collocated motion vectors as specified in clause 8.5.2.12 is invoked with currCb, colCb, (xColCb, yColCb), refIdxLX and sbFlag set equal to 0 as inputs, and the output is assigned to mvLXCol and availableFlagLXCol.

  – Otherwise, both components of mvLXCol are set equal to 0 and availableFlagLXCol is set equal to 0.

2. When availableFlagLXCol is equal to 0, the central collocated motion vector is derived as follows:

xColCtr = xCb + (cbWidth >> 1) (590)

yColCtr = yCb + (cbHeight >> 1) (591)

  – The luma location (xColCb, yColCb) is set equal to ((xColCtr >> 3) << 3, (yColCtr >> 3) << 3).

  – The variable colCb specifies the luma coding block covering the location (xColCb, yColCb) inside the collocated picture specified by ColPic.

  – The derivation process for collocated motion vectors as specified in clause 8.5.2.12 is invoked with currCb, colCb, (xColCb, yColCb), refIdxLX and sbFlag set equal to 0 as inputs, and the output is assigned to mvLXCol and availableFlagLXCol.

8.5.2.12 Derivation process for collocated motion vectors

Inputs to this process are:

– a variable currCb specifying the current coding block,

– a variable colCb specifying the collocated luma coding block inside the collocated picture specified by ColPic,

– a luma location (xColCb, yColCb) specifying the current collocated sample relative to the top-left luma sample of the collocated picture specified by ColPic,

– a reference index refIdxLX, with X being 0 or 1,

– a flag indicating a subblock temporal merging candidate sbFlag.

Outputs of this process are:

– the motion vector prediction mvLXCol in 1/16 fractional-sample accuracy,

– the availability flag availableFlagLXCol.

The variable currPic specifies the current picture.

The arrays predFlagColL0[x][y], mvL0Col[x][y] and refIdxL0Col[x][y] are set equal to PredFlagL0[x][y], MvDmvrL0[x][y] and RefIdxL0[x][y], respectively, of the collocated picture specified by ColPic, and the arrays predFlagColL1[x][y], mvL1Col[x][y] and refIdxL1Col[x][y] are set equal to PredFlagL1[x][y], MvDmvrL1[x][y] and RefIdxL1[x][y], respectively, of the collocated picture specified by ColPic.

The function LongTermRefPic(aPic, aCb, refIdx, LX), with X being 0 or 1, is defined as follows:

– If the picture with index refIdx from RPL LX of the slice containing the luma coding block aCb in the picture aPic was marked as "used for long-term reference" at the time when aPic was the current picture, LongTermRefPic(aPic, aCb, refIdx, LX) is equal to 1.

– Otherwise, LongTermRefPic(aPic, aCb, refIdx, LX) is equal to 0.

The variables mvLXCol and availableFlagLXCol are derived as follows:

– If colCb is coded in an intra, IBC, or palette prediction mode, both components of mvLXCol are set equal to 0 and availableFlagLXCol is set equal to 0.

– Otherwise, the motion vector mvCol, the reference index refIdxCol and the reference list identifier listCol are derived as follows:

  – If sbFlag is equal to 0, availableFlagLXCol is set equal to 1 and the following applies:

    – If predFlagColL0[xColCb][yColCb] is equal to 0, mvCol, refIdxCol and listCol are set equal to mvL1Col[xColCb][yColCb], refIdxL1Col[xColCb][yColCb] and L1, respectively.

    – Otherwise, if predFlagColL0[xColCb][yColCb] is equal to 1 and predFlagColL1[xColCb][yColCb] is equal to 0, mvCol, refIdxCol and listCol are set equal to mvL0Col[xColCb][yColCb], refIdxL0Col[xColCb][yColCb] and L0, respectively.

    – Otherwise (predFlagColL0[xColCb][yColCb] is equal to 1 and predFlagColL1[xColCb][yColCb] is equal to 1), the following assignments are made:

      – If NoBackwardPredFlag is equal to 1, mvCol, refIdxCol and listCol are set equal to mvLXCol[xColCb][yColCb], refIdxLXCol[xColCb][yColCb] and LX, respectively.

      – Otherwise, mvCol, refIdxCol and listCol are set equal to mvLNCol[xColCb][yColCb], refIdxLNCol[xColCb][yColCb] and LN, respectively, with N being the value of sh_collocated_from_l0_flag.

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

    – If predFlagColLX[xColCb][yColCb] is equal to 1, mvCol, refIdxCol, and listCol are set equal to mvLXCol[xColCb][yColCb], refIdxLXCol[xColCb][yColCb], and LX, respectively, availableFlagLXCol is set equal to 1.

    – Otherwise (predFlagColLX[xColCb][yColCb] is equal to 0), the following applies:

      – If NoBackwardPredFlag is equal to 1 and predFlagColLY[xColCb][yColCb] is equal to 1, mvCol, refIdxCol, and listCol are set equal to mvLYCol[xColCb][yColCb], refIdxLYCol[xColCb][yColCb] and LY, respectively, with Y being equal to 1 − X, with X being the value of X that this process is invoked for, and availableFlagLXCol is set equal to 1.

      – Otherwise, both components of mvLXCol are set equal to 0 and availableFlagLXCol is set equal to 0.

  – When availableFlagLXCol is equal to TRUE, mvLXCol and availableFlagLXCol are derived as follows:

    – If LongTermRefPic(currPic, currCb, refIdxLX, LX) is not equal to LongTermRefPic(ColPic, colCb, refIdxCol, listCol), both components of mvLXCol are set equal to 0 and availableFlagLXCol is set equal to 0.

– Otherwise, the variable availableFlagLXCol is set equal to 1, refPicList[listCol][refIdxCol] is set to be the picture with reference index refIdxCol in the RPL listCol of the slice containing the luma coding block colCb in the collocated picture specified by ColPic, and the following applies:

$$\text{colPocDiff} = \text{DiffPicOrderCnt}(\ \text{ColPic},\ \text{refPicList}[\ \text{listCol}\ ][\ \text{refIdxCol}\ ]\ ) \tag{592}$$

$$\text{currPocDiff} = \text{DiffPicOrderCnt}(\ \text{currPic},\ \text{RefPicList}[\ X\ ][\ \text{refIdxLX}\ ]\ ) \tag{593}$$

  – The temporal motion buffer compression process for collocated motion vectors as specified in clause 8.5.2.15 is invoked with mvCol as input, and the modified mvCol as output.

  – If RefPicList[X][refIdxLX] is marked as "used for long-term reference", or colPocDiff is equal to currPocDiff, mvLXCol is derived as follows:

$$\text{mvLXCol} = \text{Clip3}(\ -131072,\ 131071,\ \text{mvCol}\ ) \tag{594}$$

  – Otherwise, mvLXCol is derived as a scaled version of the motion vector mvCol as follows:

$$\text{tx} = (\ 16384 + (\ \text{Abs}(\ \text{td}\ ) \gg 1\ )\ )\ /\ \text{td} \tag{595}$$

$$\text{distScaleFactor} = \text{Clip3}(\ -4096,\ 4095,\ (\ \text{tb} * \text{tx} + 32\ ) \gg 6\ ) \tag{596}$$

$$\begin{aligned}\text{mvLXCol} = {} & \text{Clip3}(\ -131072,\ 131071,\ (\text{distScaleFactor} * \text{mvCol} + \\ & 128 - (\ \text{distScaleFactor} * \text{mvCol} >= 0\ )\ ) \gg 8\ )\end{aligned} \tag{597}$$

  where td and tb are derived as follows:

$$\text{td} = \text{Clip3}(\ -128,\ 127,\ \text{colPocDiff}\ ) \tag{598}$$

$$\text{tb} = \text{Clip3}(\ -128,\ 127,\ \text{currPocDiff}\ ) \tag{599}$$

8.5.2.13 Derivation process for chroma motion vectors

Input to this process is a luma motion vector in 1/16 fractional-sample accuracy mvLX.

Output of this process is a chroma motion vector in 1/32 fractional-sample accuracy mvCLX.

A chroma motion vector is derived from the corresponding luma motion vector.

The chroma motion vector mvCLX, is derived as follows:

$$\text{mvCLX}[\ 0\ ] = \text{mvLX}[\ 0\ ] * 2\ /\ \text{SubWidthC} \tag{600}$$

$$\text{mvCLX}[\ 1\ ] = \text{mvLX}[\ 1\ ] * 2\ /\ \text{SubHeightC} \tag{601}$$

8.5.2.14 Rounding process for motion vectors

Inputs to this process are:

– the motion vector mvX,

– the right shift parameter rightShift for rounding,

– the left shift parameter leftShift for resolution increase.

Output of this process is the rounded motion vector mvX.

For the rounding of mvX, the following applies:

$$\text{offset} = (\ \text{rightShift} == 0\ )\ ?\ 0 : (\ (\ 1 \ll (\ \text{rightShift} - 1\ )\ ) - 1\ ) \tag{602}$$

$$\text{mvX}[\ 0\ ] = \text{Sign}(\ \text{mvX}[\ 0\ ]\ ) * (\ (\ (\ \text{Abs}(\ \text{mvX}[\ 0\ ]\ ) + \text{offset}\ ) \gg \text{rightShift}\ ) \ll \text{leftShift}\ ) \tag{603}$$

$$\text{mvX}[\ 1\ ] = \text{Sign}(\ \text{mvX}[\ 1\ ]\ ) * (\ (\ (\ \text{Abs}(\ \text{mvX}[\ 1\ ]\ ) + \text{offset}\ ) \gg \text{rightShift}\ ) \ll \text{leftShift}\ ) \tag{604}$$

8.5.2.15 Temporal motion buffer compression process for collocated motion vectors

Input to this process is a motion vector mv.

Output of this process is the rounded motion vector mv.

For each motion vector component compIdx being 0 or 1, mv[compIdx] is modified as follows:

$$s = mv[\ compIdx\ ] >> 17 \quad (605)$$

$$f = Floor(\ Log2(\ (\ mv[\ compIdx\ ]\ \hat{}\ s\ )\ |\ 31\ )\ ) - 4 \quad (606)$$

$$mask = (\ -1 << f\ ) >> 1 \quad (607)$$

$$round = (\ 1 << f\ ) >> 2 \quad (608)$$

$$mv[\ compIdx\ ] = (\ mv[\ compIdx\ ] + round\ )\ \&\ mask \quad (609)$$

NOTE – This process enables storage of collocated motion vectors using a bit-reduced representation. Each signed 18-bit motion vector component can be represented in a mantissa plus exponent format with a 6-bit signed mantissa and a 4-bit exponent.

8.5.2.16 Updating process for the history-based motion vector predictor candidate list

Inputs to this process are:

– luma motion vectors in 1/16 fractional-sample accuracy mvL0 and mvL1,

– reference indices refIdxL0 and refIdxL1,

– prediction list utilization flags predFlagL0 and predFlagL1,

– the bi-prediction weight index bcwIdx,

– the half-sample interpolation filter index hpelIfIdx.

The MVP candidate hMvpCand consists of the luma motion vectors mvL0 and mvL1, the reference indices refIdxL0 and refIdxL1, the prediction list utilization flags predFlagL0 and predFlagL1, the bi-prediction weight index bcwIdx, and the half sample interpolation filter index hpelIfIdx.

The candidate list HmvpCandList is modified using the candidate hMvpCand by the following ordered steps:

1. The variable identicalCandExist is set equal to FALSE and the variable removeIdx is set equal to 0.
2. When NumHmvpCand is greater than 0, for each index hMvpIdx with hMvpIdx = 0..NumHmvpCand − 1, the following steps apply until identicalCandExist is equal to TRUE:
   – When hMvpCand and HmvpCandList[hMvpIdx] have the same motion vectors and the same reference indices, identicalCandExist is set equal to TRUE and removeIdx is set equal to hMvpIdx.
3. The candidate list HmvpCandList is updated as follows:
   – If identicalCandExist is equal to TRUE or NumHmvpCand is equal to 5, the following applies:
     – For each index i with i = (removeIdx + 1)..(NumHmvpCand − 1), HmvpCandList[i − 1] is set equal to HmvpCandList[i].
     – HmvpCandList[NumHmvpCand − 1] is set equal to hMvpCand.
   – Otherwise (identicalCandExist is equal to FALSE and NumHmvpCand is less than 5), the following applies:
     – HmvpCandList[NumHmvpCand++] is set equal to hMvpCand.

8.5.3 Decoder-side motion vector refinement process

8.5.3.1 General

Inputs to this process are:

– a luma location (xSb, ySb) specifying the top-left sample of the current subblock relative to the top-left luma sample of the current picture,

– a variable sbWidth specifying the width of the current subblock in luma samples,

– a variable sbHeight specifying the height of the current subblock in luma samples,

– the luma motion vectors in 1/16 fractional-sample accuracy mvL0 and mvL1,

– the selected luma reference picture sample arrays $refPicL0_L$ and $refPicL1_L$.

Outputs of this process are:

– delta luma motion vectors dMvL0 and dMvL1,

– a variable dmvrSad specifying the mimimum sum of absolute differences.

The variable subPelFlag is set equal to 0, the variable srRange is set equal to 2 and the integer sample offset (intOffX, intOffY) is set equal to (0, 0).

Both components of the delta luma motion vectors dMvL0 and dMvL1 are set equal to zero and modified as follows:

– For X = 0..1, the (sbWidth + 2 * srRange) x (sbHeight + 2 * srRange) array $predSamplesLX_L$ of prediction luma sample values is derived by invoking the fractional sample bilinear interpolation process specified in clause 8.5.3.2.1 with the luma location (xSb, ySb), the prediction sample block width predWidth set equal to (sbWidth + 2 * srRange), the prediction sample block height predHeight set equal to (sbHeight + 2 * srRange), the reference picture sample array $refPicLX_L$, the motion vector mvLX, and the refinement search range srRange as inputs.

– The variable minSad is derived by invoking the sum of absolute differences calculation process specified in clause 8.5.3.3 with the width sbW and height sbH of the current subblock set equal to sbWidth and sbHeight, the prediction sample arrays pL0 and pL1 set equal to $predSamplesL0_L$ and $predSamplesL1_L$, and the offset (dX, dY) set equal to (0, 0) as inputs, and minSad as output.

– The variable dmvrSad is set equal to minSad.

– When minSad is greater than or equal to sbHeight * sbWidth, the following applies:

  – The 2-D array sadArray[dX + 2][dY + 2] with dX = −2..2 and dY = −2..2 is derived by invoking the sum of absolute differences calculation process specified in clause 8.5.3.3 with the width sbW and height sbH of the current subblock set equal to sbWidth and sbHeight, the prediction sample arrays pL0 and pL1 set equal to $predSamplesL0_L$ and $predSamplesL1_L$, and the offset (dX, dY) as inputs, and sadArray[dX + 2][dY + 2] as output.

  – The integer sample offset (intOffX, intOffY) is modified by invoking the array entry selection process specified in clause 8.5.3.4 with the 2-D array sadArray[dX + 2][dY + 2] with dX = −2..2 and dY = −2..2, the best integer sample offset (intOffX, intOffY), and minSad as input, the modified best integer sample offset (intOffX, intOffY) and modified dmvrSad as outputs.

  – When the absolute value of intOffX is not equal to 2 and the absolute value of intOffY is not equal to 2, subPelFlag is set equal to 1.

  – The delta luma motion vector dMvL0 is modified as follows:

$$dMvL0[\ 0\ ] \mathrel{+}= 16 * intOffX \tag{610}$$

$$dMvL0[\ 1\ ] \mathrel{+}= 16 * intOffY \tag{611}$$

  – When subPelFlag is equal to 1, the parametric motion vector refinement process specified in clause 8.5.3.5 is invoked with the 3x3 2-D array sadArray[dX + 2][dY + 2] with dX = intOffX − 1, intOffX, intOffX + 1 and dY = intOffY − 1, intOffY, intOffY + 1, and the delta motion vector dMvL0 as inputs and the modified dMvL0 as output.

  – The delta motion vector dMvL1 is derived as follows:

$$dMvL1[\ 0\ ] = -dMvL0[\ 0\ ] \tag{612}$$

$$dMvL1[\ 1\ ] = -dMvL0[\ 1\ ] \tag{613}$$

8.5.3.2 Fractional sample bilinear interpolation process

8.5.3.2.1 General

Inputs to this process are:

– a luma location (xSb, ySb) specifying the top-left sample of the current subblock relative to the top-left luma sample of the current picture,

– a variable predWidth specifying the width of the current prediction sample block in luma samples,

– a variable predHeight specifying the height of the current prediction sample block in luma samples,

– a luma motion vector mvLX given in 1/16-luma-sample units,

– the selected reference picture sample array $refPicLX_L$,

– the refinement search range srRange.

Output of this process is:

– a (predWidth) x (predHeight) array $predSamplesLX_L$ of luma prediction sample values.

Let ($xInt_L$, $yInt_L$) be a luma location given in full-sample units and ($xFrac_L$, $yFrac_L$) be an offset given in 1/16-sample units. These variables are used only in this clause for specifying fractional-sample locations inside the reference sample array $refPicLX_L$.

For each luma sample location (x_L = 0..predWidth − 1, y_L = 0..predHeight − 1) inside the luma prediction sample array $predSamplesLX_L$, the corresponding luma prediction sample value $predSamplesLX_L[x_L][y_L]$ is derived as follows:

– The variables $xInt_L$, $yInt_L$, $xFrac_L$ and $yFrac_L$ are derived as follows:

$$xInt_L = xSb + (mvLX[0] >> 4) + x_L - srRange \quad (614)$$

$$yInt_L = ySb + (mvLX[1] >> 4) + y_L - srRange \quad (615)$$

$$xFrac_L = mvLX[0] \ \& \ 15 \quad (616)$$

$$yFrac_L = mvLX[1] \ \& \ 15 \quad (617)$$

– The luma prediction sample value $predSamplesLX_L[x_L][y_L]$ is derived by invoking the luma sample bilinear interpolation process specified in clause 8.5.3.2.2 with ($xInt_L$, $yInt_L$), ($xFrac_L$, $yFrac_L$), and $refPicLX_L$ as inputs.

8.5.3.2.2 Luma sample bilinear interpolation process

Inputs to this process are:

– a luma location in full-sample units ($xInt_L$, $yInt_L$),

– a luma location in fractional-sample units ($xFrac_L$, $yFrac_L$),

– the luma reference sample array $refPicLX_L$.

Output of this process is a predicted luma sample value $predSampleLX_L$.

The variables shift1, shift2, shift3, shift4, offset1, offset2 and offset4 are derived as follows:

$$shift1 = BitDepth - 6 \quad (618)$$

$$offset1 = 1 << (shift1 - 1) \quad (619)$$

$$shift2 = 4 \quad (620)$$

$$offset2 = 1 << (shift2 - 1) \quad (621)$$

$$shift3 = 10 - BitDepth \quad (622)$$

$$shift4 = BitDepth - 10 \quad (623)$$

$$offset4 = 1 << (shift4 - 1) \quad (624)$$

The variable picW is set equal to pps_pic_width_in_luma_samples of the reference picture refPicLX and the variable picH is set equal to pps_pic_height_in_luma_samples of the reference picture refPicLX.

The luma interpolation filter coefficients $fb_L[p]$ for each 1/16 fractional sample position p equal to $xFrac_L$ or $yFrac_L$ are specified in Table 26.

The luma locations in full-sample units ($xInt_i$, $yInt_i$) are derived as follows for i = 0..1:

– If sps_subpic_treated_as_pic_flag[CurrSubpicIdx] is equal to 1 and sps_num_subpics_minus1 for the reference picture refPicLX is greater than 0, the following applies:

$$\begin{aligned}xInt_i = {} & Clip3(SubpicLeftBoundaryPos, SubpicRightBoundaryPos, pps\_ref\_wraparound\_enabled\_flag\ ? \\ & ClipH((PpsRefWraparoundOffset) * MinCbSizeY, picW, (xInt_L + i)) : xInt_L + i)\end{aligned} \quad (625)$$

$$yInt_i = Clip3(SubpicTopBoundaryPos, SubpicBotBoundaryPos, yInt_L + i) \quad (626)$$

– Otherwise (sps_subpic_treated_as_pic_flag[CurrSubpicIdx] is equal to 0 or sps_num_subpics_minus1 for the reference picture refPicLX is equal to 0), the following applies:

$$xInt_i = Clip3(0, picW - 1, pps\_ref\_wraparound\_enabled\_flag ? ClipH((PpsRefWraparoundOffset) * MinCbSizeY, picW, (xInt_L + i)) : xInt_L + i) \quad (627)$$

$$yInt_i = Clip3(0, picH - 1, yInt_L + i) \quad (628)$$

The predicted luma sample value $predSampleLX_L$ is derived as follows:

– If both $xFrac_L$ and $yFrac_L$ are equal to 0, the value of $predSampleLX_L$ is derived as follows:

$$predSampleLX_L = BitDepth <= 10 ? (refPicLX_L[xInt_0][yInt_0] << shift3) : ((refPicLX_L[xInt_0][yInt_0] + offset4) >> shift4) \quad (629)$$

– Otherwise, if $xFrac_L$ is not equal to 0 and $yFrac_L$ is equal to 0, the value of $predSampleLX_L$ is derived as follows:

$$predSampleLX_L = ((\sum_{i=0}^{1} fb_L[xFrac_L][i] * refPicLX_L[xInt_i][yInt_0]) + offset1) >> shift1 \quad (630)$$

– Otherwise, if $xFrac_L$ is equal to 0 and $yFrac_L$ is not equal to 0, the value of $predSampleLX_L$ is derived as follows:

$$predSampleLX_L = ((\sum_{i=0}^{1} fb_L[yFrac_L][i] * refPicLX_L[xInt_0][yInt_i]) + offset1) >> shift1 \quad (631)$$

– Otherwise, if $xFrac_L$ is not equal to 0 and $yFrac_L$ is not equal to 0, the value of $predSampleLX_L$ is derived as follows:

  – The sample array temp[n] with n = 0..1, is derived as follows:

$$temp[n] = ((\sum_{i=0}^{1} fb_L[xFrac_L][i] * refPicLX_L[xInt_i][yInt_n]) + offset1) >> shift1 \quad (632)$$

  – The predicted luma sample value $predSampleLX_L$ is derived as follows:

$$predSampleLX_L = ((\sum_{i=0}^{1} fb_L[yFrac_L][i] * temp[i]) + offset2) >> shift2 \quad (633)$$

Table 26 – Specification of the luma bilinear interpolation filter coefficients fb_L[p] for each 1/16 fractional sample position p

Fractional sample position p	interpolation filter coefficients	
	fb_L[p][0]	fb_L[p][1]
1	15	1
2	14	2
3	13	3
4	12	4
5	11	5
6	10	6
7	9	7
8	8	8
9	7	9
10	6	10
11	5	11
12	4	12
13	3	13
14	2	14
15	1	15

8.5.3.3 Sum of absolute differences calculation process

Inputs to this process are:

– two variables nSbW and nSbH specifying the width and the height of the current subblock,

– two (nSbW + 4) x (nSbH + 4) arrays pL0 and pL1 containing the predicted samples for L0 and L1 respectively,

– an integer sample offset (dX, dY) in L0.

Output of this process is:

– the variable sad specifying the sum of absolute differences at the integer sample at the offset (dX, dY) in L0.

The variable sad is derived as follows:

$$\text{sad} = \sum_{x=0}^{\text{nSbW}-1} \sum_{y=0}^{\text{nSbH}/2-1} \text{Abs}(\text{pL0}[x+2+\text{dX}][2*y+2+\text{dY}] - \text{pL1}[x+2-\text{dX}][2*y+2-\text{dY}]) \quad (634)$$

When both dX and dY are equal to 0, the value of sad is modified as follows:

sad = sad − (sad >> 2) (635)

8.5.3.4 Array entry selection process

Inputs to this process are:

– a 2-D array of sum of absolute differences values sadArray[dX + 2][dY + 2] with dX = −2..2 and dY = −2..2,

– an integer sample offset (intOffX, intOffY),

– a variable minSad.

Outputs of this process are:

– the modified integer sample (intOffX, intOffY),

– a variable dmvrSad.

The following steps are applied to modify the integer sample offset (intOffX, intOffY):

```
for( dY = −2; dY <= 2; dY++ ) {
    for( dX = −2; dX <= 2; dX++ ) {
        if( sadArray[ dX + 2 ][ dY + 2 ] < minSad ) {
            minSad = sadArray[ dX + 2 ][ dY + 2 ]                    (636)
            intOffX = dX
            intOffY = dY
        }
    }
}
```

The variable dmvrSad is set equal to minSad.

8.5.3.5 Parametric motion vector refinement process

8.5.3.5.1 General

Inputs to this process are:

– a 3x3 2-D array sadArray[dX + 1][dY + 1] with dX = −1..1 and dY = −1..1,

– a delta luma motion vector dMvL0.

Output of this process is the modified delta luma motion vector dMvL0.

The variable dMvX is derived by invoking the derivation process for delta motion vector component offset specified in clause 8.5.3.5.2 with the SAD values sadMinus, sadCenter and sadPlus set equal to sadArray[0][1], sadArray[1][1], and sadArray[2][1] as inputs, and dMvX set equal to the output dMVc.

The variable dMvY is derived by invoking the derivation process for delta motion vector component offset specified in clause 8.5.3.5.2 with the SAD values sadMinus, sadCenter and sadPlus set equal to sadArray[1][0], sadArray[1][1], and sadArray[1][2] as inputs, and dMvY set equal to the output dMVc.

The delta luma motion vector dMvL0 is modified as follows:

dMvL0[0] += dMvX (637)

dMvL0[1] += dMvY (638)

NOTE – dMvC with C being X or Y is constrained to be between −8 and 8 since sadMinus, sadCenter, and sadPlus are all positive, and sadCenter is the smallest value among the three. This allows the division to be performed with up to 4 quotient bits and could be implemented using compares, shifts, and subtractions.

8.5.3.5.2 Derivation process for delta motion vector component offset

Inputs to this process are 3 SAD values sadMinus, sadCenter, and sadPlus.

Output of this process is the delta motion vector component correction offset dMvC.

The offset dMVc is derived using the following pseudo-code process:

```
denom = ( ( sadMinus + sadPlus ) − ( sadCenter << 1 ) ) << 3
if( denom = = 0 )
    dMvC = 0
else {
    if( sadMinus = = sadCenter )
        dMvC = −8
    else if( sadPlus = = sadCenter )
        dMvC = 8
    else {
        num = ( sadMinus − sadPlus ) << 4
        signNum = 0
        if( num < 0 ) {
            num = −num
            signNum = 1
        }
        quotient = 0                                                    (639)
        counter = 3
        while( counter > 0 ) {
            counter = counter − 1
            quotient = quotient << 1
            if( num >= denom ) {
                num = num − denom
                quotient = quotient + 1
            }
            denom = ( denom >> 1 )
        }
        if( signNum = = 1 )
            dMvC = −quotient
        else
            dMvC = quotient
    }
}
```

NOTE – This pseudo-code process is equivalent to an integer division of num = (sadMinus − sadPlus) << 3 by denom = (sadMinus + sadPlus − (sadCenter << 1). Given the fact that sadMinus, sadCenter, and sadPlus are all positive, and sadCenter is the smallest value among the three, the value is limited to be in the range of −8 to 8, inclusive.

8.5.4 Derivation process for geometric partitioning mode motion vector components and reference indices

8.5.4.1 General

Inputs to this process are:

– a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples.

Outputs of this process are:

– the luma motion vectors in 1/16 fractional-sample accuracy mvA and mvB,

– the chroma motion vectors in 1/32 fractional-sample accuracy mvCA and mvCB,

– the reference indices refIdxA and refIdxB,

– the prediction list flags predListFlagA and predListFlagB.

The derivation process for luma motion vectors for geometric partitioning merge mode as specified in clause 8.5.4.2 is invoked with the luma location (xCb, yCb), the variables cbWidth and cbHeight as inputs, and the output being the luma motion vectors mvA, mvB, the reference indices refIdxA, refIdxB and the prediction list flags predListFlagA and predListFlagB.

The derivation process for chroma motion vectors in clause 8.5.2.13 is invoked with mvA as input, and the output being mvCA.

The derivation process for chroma motion vectors in clause 8.5.2.13 is invoked with mvB as input, and the output being mvCB.

8.5.4.2 Derivation process for luma motion vectors for geometric partitioning merge mode

This process is only invoked when MergeGpmFlag[xCb][yCb] is equal to 1, where (xCb, yCb) specify the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture.

Inputs to this process are:

– a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples.

Outputs of this process are:

– the luma motion vectors in 1/16 fractional-sample accuracy mvA and mvB,

– the reference indices refIdxA and refIdxB,

– the prediction list flags predListFlagA and predListFlagB.

The motion vectors mvA and mvB, the reference indices refIdxA and refIdxB and the prediction list flags predListFlagA and predListFlagB are derived by the following ordered steps:

1. The derivation process for luma motion vectors for merge mode as specified in clause 8.5.2.2 is invoked with the luma location (xCb, yCb), the variables cbWidth and cbHeight as inputs, and the output being the luma motion vectors mvL0[0][0], mvL1[0][0], the reference indices refIdxL0, refIdxL1, the prediction list utilization flags predFlagL0[0][0] and predFlagL1[0][0], the bi-prediction weight index bcwIdx and the merging candidate list mergeCandList.

2. The variables m and n, being the merge index for the geometric partition 0 and 1 respectively, are derived using merge_gpm_idx0[xCb][yCb] and merge_gpm_idx1[xCb][yCb] as follows:

   m = merge_gpm_idx0[xCb][yCb] (640)

   n = merge_gpm_idx1[xCb][yCb] + ((merge_gpm_idx1[xCb][yCb] >= m) ? 1 : 0) (641)

3. Let refIdxL0M and refIdxL1M, predFlagL0M and predFlagL1M, and mvL0M and mvL1M be the reference indices, the prediction list utilization flags and the motion vectors of the merging candidate M at position m in the merging candidate list mergeCandList (M = mergeCandList[m]).

4. The variable X is set equal to (m & 0x01).

5. When predFlagLXM is equal to 0, X is set equal to (1 − X).

6. The following applies:

   mvA[0] = mvLXM[0] (642)

   mvA[1] = mvLXM[1] (643)

   refIdxA = refIdxLXM (644)

   predListFlagA = X (645)

7. Let refIdxL0N and refIdxL1N, predFlagL0N and predFlagL1N, and mvL0N and mvL1N be the reference indices, the prediction list utilization flags and the motion vectors of the merging candidate N at position m in the merging candidate list mergeCandList (N = mergeCandList[n]).

8. The variable X is set equal to (n & 0x01).

9. When predFlagLXN is equal to 0, X is set equal to (1 − X).

10. The following applies:

$$mvB[\ 0\ ] = mvLXN[\ 0\ ] \quad (646)$$

$$mvB[\ 1\ ] = mvLXN[\ 1\ ] \quad (647)$$

$$refIdxB = refIdxLXN \quad (648)$$

$$predListFlagB = X \quad (649)$$

8.5.5 Derivation process for subblock motion vector components and reference indices

8.5.5.1 General

Inputs to this process are:

– a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples.

Outputs of this process are:

– the reference indices refIdxL0 and refIdxL1,

– the number of luma subblocks in horizontal direction numSbX and in vertical direction numSbY,

– the prediction list utilization flag arrays predFlagL0[xSbIdx][ySbIdx] and predFlagL1[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbX − 1,

– the luma subblock motion vector arrays in 1/16 fractional-sample accuracy mvL0[xSbIdx][ySbIdx] and mvL1[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1,

– the chroma subblock motion vector arrays in 1/32 fractional-sample accuracy mvCL0[xSbIdx][ySbIdx] and mvCL1[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1,

– the bi-prediction weight index bcwIdx,

– the prediction refinement utilization flags cbProfFlagL0, cbProfFlagL1,

– the motion vector difference arrays diffMvL0[xIdx][yIdx] and diffMvL1[xIdx][yIdx] with xIdx = 0..cbWidth/numSbX − 1, yIdx = 0..cbHeight/numSbY − 1.

The variables cbProfFlagL0 and cbProfFlagL1 are initialized to be equal to zero.

For the derivation of the variables mvL0[xSbIdx][ySbIdx], mvL1[xSbIdx][ySbIdx], mvCL0[xSbIdx][ySbIdx] and mvCL1[xSbIdx][ySbIdx], refIdxL0, refIdxL1, numSbX, numSbY, predFlagL0[xSbIdx][ySbIdx] and predFlagL1[xSbIdx][ySbIdx], the following applies:

– If merge_subblock_flag[xCb][yCb] is equal to 1, the derivation process for motion vectors and reference indices in subblock merge mode as specified in clause 8.5.5.2 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth and the luma coding block height cbHeight as inputs, the number of luma subblocks in horizontal direction numSbX and in vertical direction numSbY, the reference indices refIdxL0, refIdxL1, the prediction list utilization flag arrays predFlagL0[xSbIdx][ySbIdx] and predFlagL1[xSbIdx][ySbIdx], the luma subblock motion vector arrays mvL0[xSbIdx][ySbIdx] and mvL1[xSbIdx][ySbIdx], and the chroma subblock motion vector arrays mvCL0[xSbIdx][ySbIdx] and mvCL1[xSbIdx][ySbIdx], with xSbIdx = 0.. numSbX − 1, ySbIdx = 0..numSbY − 1, the prediction refinement utility flags cbProfFlagL0 and cbProfFlagL1, the motion vector difference arrays diffMvL0[xIdx][yIdx] and diffMvL1[xIdx][yIdx] with xIdx = 0..cbWidth / numSbX − 1, yIdx = 0..cbHeight / numSbY − 1, and the bi-prediction weight index bcwIdx as outputs.

– Otherwise (merge_subblock_flag[xCb][yCb] is equal to 0), the following applies:

– The number of control point motion vectors numCpMv is set equal to MotionModelIdc[xCb][yCb] + 1.

– For X = 0..1, the variables refIdxLX, predFlagLX[0][0], and the control point motion vectors cpMvLX[cpIdx] with cpIdx ranging from 0 to numCpMv – 1, are derived by the following ordered steps:

1. The variables refIdxLX and predFlagLX are derived as follows:

   − If inter_pred_idc[xCb][yCb] is equal to PRED_LX or PRED_BI, the following applies:

   $$\text{refIdxLX} = \text{ref\_idx\_lX}[\ \text{xCb}\ ][\ \text{yCb}\ ] \tag{650}$$

   $$\text{predFlagLX}[\ 0\ ][\ 0\ ] = 1 \tag{651}$$

   − Otherwise, the variables refIdxLX and predFlagLX are specified as follows:

   $$\text{refIdxLX} = -1 \tag{652}$$

   $$\text{predFlagLX}[\ 0\ ][\ 0\ ] = 0 \tag{653}$$

2. The variable mvdCpLX[0] is derived as follows:

   $$\text{mvdCpLX}[\ 0\ ][\ 0\ ] = \text{MvdCpLX}[\ \text{xCb}\ ][\ \text{yCb}\ ][\ 0\ ][\ 0\ ] \tag{654}$$

   $$\text{mvdCpLX}[\ 0\ ][\ 1\ ] = \text{MvdCpLX}[\ \text{xCb}\ ][\ \text{yCb}\ ][\ 0\ ][\ 1\ ] \tag{655}$$

3. The variable mvdCpLX[cpIdx] with cpIdx ranging from 1 to numCpMv − 1, is derived as follows:

   $$\text{mvdCpLX}[\ \text{cpIdx}\ ][\ 0\ ] = \text{MvdCpLX}[\ \text{xCb}\ ][\ \text{yCb}\ ][\ \text{cpIdx}\ ][\ 0\ ] + \text{mvdCpLX}[\ 0\ ][\ 0\ ] \tag{656}$$

   $$\text{mvdCpLX}[\ \text{cpIdx}\ ][\ 1\ ] = \text{MvdCpLX}[\ \text{xCb}\ ][\ \text{yCb}\ ][\ \text{cpIdx}\ ][\ 1\ ] + \text{mvdCpLX}[\ 0\ ][\ 1\ ] \tag{657}$$

4. When predFlagLX[0][0] is equal to 1, the derivation process for luma affine control point motion vector predictors as specified in clause 8.5.5.7 is invoked with the luma coding block location (xCb, yCb), and the variables cbWidth, cbHeight, refIdxLX, and the number of control point motion vectors numCpMv as inputs, and the output being mvpCpLX[cpIdx] with cpIdx ranging from 0 to numCpMv − 1.

5. When predFlagLX[0][0] is equal to 1, the luma motion vectors cpMvLX[cpIdx] with cpIdx ranging from 0 to NumCpMv − 1, are derived as follows:

   $$\text{uLX}[\ \text{cpIdx}\ ][\ 0\ ] = (\ \text{mvpCpLX}[\ \text{cpIdx}\ ][\ 0\ ] + \text{mvdCpLX}[\ \text{cpIdx}\ ][\ 0\ ]\ )\ \&\ (\ 2^{18} - 1\ ) \tag{658}$$

   $$\text{cpMvLX}[\ \text{cpIdx}\ ][\ 0\ ] = (\text{uLX}[\ \text{cpIdx}\ ][\ 0\ ]\ \ \text{>=}\ \ 2^{17}\ )\ ?\ (\text{uLX}[\ \text{cpIdx}\ ][\ 0\ ] - 2^{18}\ ) : \text{uLX}[\ \text{cpIdx}\ ][\ 0\ ] \tag{659}$$

   $$\text{uLX}[\ \text{cpIdx}\ ][\ 1\ ] = (\ \text{mvpCpLX}[\ \text{cpIdx}\ ][\ 1\ ] + \text{mvdCpLX}[\ \text{cpIdx}\ ][\ 1\ ]\ )\ \&\ (\ 2^{18} - 1\ ) \tag{660}$$

   $$\text{cpMvLX}[\ \text{cpIdx}\ ][\ 1\ ] = (\text{uLX}[\ \text{cpIdx}\ ][\ 1\ ]\ \ \text{>=}\ \ 2^{17}\ )\ ?\ (\text{uLX}[\ \text{cpIdx}\ ][\ 1\ ] - 2^{18}\ ) : \text{uLX}[\ \text{cpIdx}\ ][\ 1\ ] \tag{661}$$

– The variables numSbX and numSbY are derived as follows:

$$\text{numSbX} = (\ \text{cbWidth}\ \ \text{>>}\ \ 2\ ) \tag{662}$$

$$\text{numSbY} = (\ \text{cbHeight}\ \ \text{>>}\ \ 2\ ) \tag{663}$$

– For xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1, the following applies:

$$\text{predFlagLX}[\ \text{xSbIdx}\ ][\ \text{ySbIdx}\ ] = \text{predFlagLX}[\ 0\ ][\ 0\ ] \tag{664}$$

– For X = 0..1, the following applies:

  – When predFlagLX[0][0] is equal to 1, the derivation process for motion vector arrays from affine control point motion vectors as specified in clause 8.5.5.9 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth, the luma coding block height cbHeight, the number of control point motion vectors numCpMv, the control point motion vectors cpMvLX[cpIdx] with cpIdx being 0..numCpMv − 1, the prediction list utilization flags predFlagL0[0][0] and predFlagL1[0][0], the reference index refIdxLX and the number of luma subblocks in horizontal direction numSbX and in vertical direction numSbY as inputs, the luma motion vector array mvLX[xSbIdx][ySbIdx], the chroma motion vector array mvCLX[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1, the prediction refinement utility flag cbProfFlagLX, and motion vector difference array diffMvLX[xIdx][yIdx] with xIdx = 0..cbWidth / numSbX − 1, yIdx = 0..cbHeight / numSbY − 1 as outputs.

– The bi-prediction weight index bcwIdx is set equal to bcw_idx[xCb][yCb].

8.5.5.2 Derivation process for motion vectors and reference indices in subblock merge mode

Inputs to this process are:

– a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– two variables cbWidth and cbHeight specifying the width and the height of the luma coding block.

Outputs of this process are:

– the number of luma subblocks in horizontal direction numSbX and in vertical direction numSbY,

– the reference indices refIdxL0 and refIdxL1,

– the prediction list utilization flag arrays predFlagL0[xSbIdx][ySbIdx] and predFlagL1[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1,

– the luma subblock motion vector arrays in 1/16 fractional-sample accuracy mvL0[xSbIdx][ySbIdx] and mvL1[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1,

– the chroma subblock motion vector arrays in 1/32 fractional-sample accuracy mvCL0[xSbIdx][ySbIdx] and mvCL1[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1,

– the prediction refinement utilization flags cbProfFlagL0 and cbProfFlagL1,

– the motion vector difference arrays diffMvL0[xIdx][yIdx] and diffMvL1[xIdx][yIdx] with xIdx = 0..cbWidth / numSbX − 1, yIdx = 0..cbHeight / numSbY − 1,

– the bi-prediction weight index bcwIdx.

The variables numSbColX, numSbColY and the subblock merging candidate list, subblockMergeCandList are derived by the following ordered steps:

1. The variables availableFlagSbCol, availableFlagA, availableFlagB, and availableFlagConstK with K = 1..6 are initialized to be equal to FALSE.

2. When sps_sbtmvp_enabled_flag is equal to 1, the following applies:

   – For the derivation of $availableFlagA_1$, $refIdxLXA_1$, $predFlagLXA_1$ and $mvLXA_1$ the following applies:

     – The luma location ($xNbA_1$, $yNbA_1$) inside the neighbouring luma coding block is set equal to (xCb − 1, yCb + cbHeight − 1).

     – The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring luma location ($xNbA_1$, $yNbA_1$), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag $availableA_1$.

     – When xCb >> Log2ParMrgLevel is equal to $xNbA_1$ >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to $yNbA_1$ >> Log2ParMrgLevel, $availableA_1$ is set equal to FALSE.

     – The variables $availableFlagA_1$, $refIdxLXA_1$, $predFlagLXA_1$ and $mvLXA_1$ are derived as follows:

       – If $availableA_1$ is equal to FALSE, $availableFlagA_1$ is set equal to 0, both components of $mvLXA_1$ are set equal to 0, $refIdxLXA_1$ is set equal to −1 and $predFlagLXA_1$ is set equal to 0, with X = 0..1, and $bcwIdxA_1$ is set equal to 0.

       – Otherwise, $availableFlagA_1$ is set equal to 1 and the following assignments are made:

$$mvLXA_1 = MvLX[xNbA_1][yNbA_1] \quad (665)$$

$$refIdxLXA_1 = RefIdxLX[xNbA_1][yNbA_1] \quad (666)$$

$$predFlagLXA_1 = PredFlagLX[xNbA_1][yNbA_1] \quad (667)$$

   – The derivation process for subblock-based temporal merging candidates as specified in clause 8.5.5.3 is invoked with the luma location (xCb, yCb), the luma coding block width cbWidth, the luma coding block height cbHeight, the availability flag $availableFlagA_1$, the reference index $refIdxLXA_1$, the prediction list utilization flag $predFlagLXA_1$, and the motion vector $mvLXA_1$ as inputs and the output being the availability flag availableFlagSbCol, the number of luma subblocks in horizontal direction numSbColX and in vertical direction numSbColY, the reference indices refIdxLXSbCol, the luma motion vectors mvLXSbCol[xSbIdx][ySbIdx] and the prediction list utilization flags predFlagLXSbCol[xSbIdx][ySbIdx] with LX being equal to L0 and L1, xSbIdx = 0..numSbColX − 1, and ySbIdx = 0..numSbColY − 1.

3. When sps_affine_enabled_flag is equal to 1, the sample locations ($xNbA_0$, $yNbA_0$), ($xNbA_1$, $yNbA_1$), ($xNbA_2$, $yNbA_2$), ($xNbB_0$, $yNbB_0$), ($xNbB_1$, $yNbB_1$), ($xNbB_2$, $yNbB_2$), and ($xNbB_3$, $yNbB_3$) are derived as follows:

$$(xNbA_0, yNbA_0) = (xCb - 1, yCb + cbHeight) \quad (668)$$

$$(xNbA_1, yNbA_1) = (xCb - 1, yCb + cbHeight - 1) \quad (669)$$

$$(xNbA_2, yNbA_2) = (xCb - 1, yCb) \quad (670)$$

$$(xNbB_0, yNbB_0) = (xCb + cbWidth, yCb - 1) \quad (671)$$

$$(xNbB_1, yNbB_1) = (xCb + cbWidth - 1, yCb - 1) \quad (672)$$

$$(xNbB_2, yNbB_2) = (xCb - 1, yCb - 1) \quad (673)$$

$$(xNbB_3, yNbB_3) = (xCb, yCb - 1) \quad (674)$$

4. When sps_affine_enabled_flag is equal to 1, the variable availableFlagA is set equal to FALSE and the following applies for ($xNbA_k$, $yNbA_k$) from ($xNbA_0$, $yNbA_0$) to ($xNbA_1$, $yNbA_1$):

   – The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring luma location ($xNbA_k$, $yNbA_k$), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag $availableA_k$.

   – When xCb >> Log2ParMrgLevel is equal to $xNbA_k$ >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to $yNbA_k$ >> Log2ParMrgLevel, $availableA_k$ is set equal to FALSE.

   – When $availableA_k$ is equal to TRUE and MotionModelIdc[$xNbA_k$][$yNbA_k$] is greater than 0 and availableFlagA is equal to FALSE, the following applies:

     – The variable availableFlagA is set equal to TRUE, motionModelIdcA is set equal to MotionModelIdc[$xNbA_k$][$yNbA_k$], (xNb, yNb) is set equal to (CbPosX[0][$xNbA_k$][$yNbA_k$], CbPosY[0][$xNbA_k$][$yNbA_k$]), nbW is set equal to CbWidth[0][$xNbA_k$][$yNbA_k$], nbH is set equal to CbHeight[0][$xNbA_k$][$yNbA_k$], numCpMv is set equal to MotionModelIdc[$xNbA_k$][$yNbA_k$] + 1, and bcwIdxA is set equal to BcwIdx[$xNbA_k$][$yNbA_k$].

     – For X = 0..1, the following applies:

       – When PredFlagLX[$xNbA_k$][$yNbA_k$] is equal to 1, the derivation process for luma affine control point motion vectors from a neighbouring block as specified in clause 8.5.5.5 is invoked with the luma coding block location (xCb, yCb), the luma coding block width and height (cbWidth, cbHeight), the neighbouring luma coding block location (xNb, yNb), the neighbouring luma coding block width and height (nbW, nbH), and the number of control point motion vectors numCpMv as input, the control point motion vector predictor candidates cpMvLXA[cpIdx] with cpIdx = 0..numCpMv − 1 as output.

       – The following assignments are made:

$$predFlagLXA = PredFlagLX[xNbA_k][yNbA_k] \quad (675)$$

$$refIdxLXA = RefIdxLX[xNbAk][yNbAk] \quad (676)$$

5. When sps_affine_enabled_flag is equal to 1, the variable availableFlagB is set equal to FALSE and the following applies for ($xNbB_k$, $yNbB_k$) from ($xNbB_0$, $yNbB_0$) to ($xNbB_2$, $yNbB_2$):

   – The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring luma location ($xNbB_k$, $yNbB_k$), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag $availableB_k$.

   – When xCb >> Log2ParMrgLevel is equal to $xNbB_k$ >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to $yNbB_k$ >> Log2ParMrgLevel, $availableB_k$ is set equal to FALSE.

– When $availableB_k$ is equal to TRUE and MotionModelIdc[$xNbB_k$][$yNbB_k$] is greater than 0 and availableFlagB is equal to FALSE, the following applies:

  – The variable availableFlagB is set equal to TRUE, motionModelIdcB is set equal to MotionModelIdc[$xNbB_k$][$yNbB_k$], (xNb, yNb) is set equal to (CbPosX[0][xNbAB][$yNbB_k$], CbPosY[0][$xNbB_k$][$yNbB_k$]), nbW is set equal to CbWidth[0][$xNbB_k$][$yNbB_k$], nbH is set equal to CbHeight[0][$xNbB_k$][$yNbB_k$], numCpMv is set equal to MotionModelIdc[$xNbB_k$][$yNbB_k$] + 1, and bcwIdxB is set equal to BcwIdx[$xNbB_k$][$yNbB_k$].

  – For X = 0..1, the following applies:

    – When PredFlagLX[$xNbB_k$][$yNbB_k$] is equal to TRUE, the derivation process for luma affine control point motion vectors from a neighbouring block as specified in clause 8.5.5.5 is invoked with the luma coding block location (xCb, yCb), the luma coding block width and height (cbWidth, cbHeight), the neighbouring luma coding block location (xNb, yNb), the neighbouring luma coding block width and height (nbW, nbH), and the number of control point motion vectors numCpMv as input, the control point motion vector predictor candidates cpMvLXB[cpIdx] with cpIdx = 0..numCpMv − 1 as output.

    – The following assignments are made:

$$\text{predFlagLXB} = \text{PredFlagLX}[\ xNbB_k\ ][\ yNbB_k\ ] \quad (677)$$

$$\text{refIdxLXB} = \text{RefIdxLX}[\ xNbB_k\ ][\ yNbB_k\ ] \quad (678)$$

6. When sps_affine_enabled_flag is equal to 1, the following applies:

   – The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring luma location ($xNbA_2$, $yNbA_2$), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag $availableA_2$.

   – When xCb >> Log2ParMrgLevel is equal to $xNbA_2$ >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to $yNbA_2$ >> Log2ParMrgLevel, $availableA_2$ is set equal to FALSE.

   – The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring luma location ($xNbB_3$, $yNbB_3$), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag $availableB_3$.

   – When xCb >> Log2ParMrgLevel is equal to $xNbB_3$ >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to $yNbB_3$ >> Log2ParMrgLevel, $availableB_3$ is set equal to FALSE.

   – The derivation process for constructed affine control point motion vector merging candidates as specified in clause 8.5.5.6 is invoked with the luma coding block location (xCb, yCb), the luma coding block width and height (cbWidth, cbHeight), the availability flags $availableA_0$, $availableA_1$, $availableA_2$, $availableB_0$, $availableB_1$, $availableB_2$ and $availableB_3$, and the sample locations ($xNbA_0$, $yNbA_0$), ($xNbA_1$, $yNbA_1$), ($xNbA_2$, $yNbA_2$), ($xNbB_0$, $yNbB_0$), ($xNbB_1$, $yNbB_1$), ($xNbB_2$, $yNbB_2$) and ($xNbB_3$, $yNbB_3$) as inputs, and the availability flags availableFlagConstK, the reference indices refIdxLXConstK, prediction list utilization flags predFlagLXConstK, motion model indices motionModelIdcConstK, bi-prediction weight indices bcwIdxConstK and cpMvLXConstK[cpIdx] with X = 0..1, K = 1..6 and cpIdx = 0..2 as outputs.

7. The initial subblock merging candidate list, subblockMergeCandList, is constructed as follows:

```
i = 0
if( availableFlagSbCol )
    subblockMergeCandList[ i++ ] = SbCol
if( availableFlagA && i < MaxNumSubblockMergeCand )
    subblockMergeCandList[ i++ ] = A
if( availableFlagB && i < MaxNumSubblockMergeCand )
    subblockMergeCandList[ i++ ] = B
if( availableFlagConst1 && i < MaxNumSubblockMergeCand )
    subblockMergeCandList[ i++ ] = Const1                                  (679)
if( availableFlagConst2 && i < MaxNumSubblockMergeCand )
    subblockMergeCandList[ i++ ] = Const2
if( availableFlagConst3 && i < MaxNumSubblockMergeCand )
    subblockMergeCandList[ i++ ] = Const3
if( availableFlagConst4 && i < MaxNumSubblockMergeCand )
```

subblockMergeCandList[i++] = Const4
if(availableFlagConst5 && i < MaxNumSubblockMergeCand)
subblockMergeCandList[i++] = Const5
if(availableFlagConst6 && i < MaxNumSubblockMergeCand)
subblockMergeCandList[i++] = Const6

8. The variables numCurrMergeCand and numOrigMergeCand are set equal to the number of merging candidates in the subblockMergeCandList.

9. When numCurrMergeCand is less than MaxNumSubblockMergeCand, the following is repeated until numCurrMergeCand is equal to MaxNumSubblockMergeCand, with mvZero[0] and mvZero[1] both being equal to 0:

   – The reference indices, the prediction list utilization flags and the motion vectors of $zeroCand_m$ with m equal to (numCurrMergeCand − numOrigMergeCand) are derived as follows:

$$refIdxL0ZeroCand_m = 0 \quad (680)$$

$$predFlagL0ZeroCand_m = 1 \quad (681)$$

$$cpMvL0ZeroCand_m[0] = mvZero \quad (682)$$

$$cpMvL0ZeroCand_m[1] = mvZero \quad (683)$$

$$refIdxL1ZeroCand_m = (sh\_slice\_type == B) ? 0 : -1 \quad (684)$$

$$predFlagL1ZeroCand_m = (sh\_slice\_type == B) ? 1 : 0 \quad (685)$$

$$cpMvL1ZeroCand_m[0] = mvZero \quad (686)$$

$$cpMvL1ZeroCand_m[1] = mvZero \quad (687)$$

$$motionModelIdcZeroCand_m = 1 \quad (688)$$

$$bcwIdxZeroCand_m = 0 \quad (689)$$

   – The candidate $zeroCand_m$ with m equal to (numCurrMergeCand − numOrigMergeCand) is added at the end of subblockMergeCandList and numCurrMergeCand is incremented by 1 as follows:

$$subblockMergeCandList[numCurrMergeCand{+}{+}] = zeroCand_m \quad (690)$$

The variables numSbX and numSbY are derived as follows:

– If subblockMergeCandList[merge_subblock_idx[xCb][yCb]] is equal to SbCol, numSbX is set equal to numSbColX, and numSbY is set equal to numSbColY.

– Otherwise, the following applies:

$$numSbX = cbWidth >> 2 \quad (691)$$

$$numSbY = cbHeight >> 2 \quad (692)$$

The variables refIdxL0, refIdxL1, predFlagL0[xSbIdx][ySbIdx], predFlagL1[xSbIdx][ySbIdx], mvL0[xSbIdx][ySbIdx], mvL1[xSbIdx][ySbIdx], mvCL0[xSbIdx][ySbIdx], and mvCL1[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1 are derived as follows:

– If subblockMergeCandList[merge_subblock_idx[xCb][yCb]] is equal to SbCol, the bi-prediction weight index bcwIdx is set equal to 0 and the following applies for X = 0..1:

$$refIdxLX = refIdxLXSbCol \quad (693)$$

  – For xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1, the following applies:

$$predFlagLX[xSbIdx][ySbIdx] = predFlagLXSbCol[xSbIdx][ySbIdx] \quad (694)$$

$$mvLX[xSbIdx][ySbIdx][0] = mvLXSbCol[xSbIdx][ySbIdx][0] \quad (695)$$

$$mvLX[xSbIdx][ySbIdx][1] = mvLXSbCol[xSbIdx][ySbIdx][1] \quad (696)$$

   – When predFlagLX[xSbIdx][ySbIdx] is equal to 1, the derivation process for chroma motion vectors in clause 8.5.2.13 is invoked with mvLX[xSbIdx][ySbIdx] as input, and the output being mvCLX[xSbIdx][ySbIdx].

– The following assignment is made for x = xCb..xCb + cbWidth − 1 and y = yCb..yCb + cbHeight − 1:

$$\text{MotionModelIdc[x][y] = 0} \quad (697)$$

– Otherwise (subblockMergeCandList[merge_subblock_idx[xCb][yCb]] is not equal to SbCol), the following applies:

  – For X = 0..1, the following assignments are made with N being the candidate at position merge_subblock_idx[xCb][yCb] in the subblock merging candidate list subblockMergeCandList (N = subblockMergeCandList[merge_subblock_idx[xCb][yCb]]):

$$\text{refIdxLX = refIdxLXN} \quad (698)$$

$$\text{predFlagLX[0][0] = predFlagLXN} \quad (699)$$

$$\text{numCpMv = motionModelIdcN + 1} \quad (700)$$

$$\text{bcwIdx = bcwIdxN} \quad (701)$$

    – For cpIdx = 0..numCpMv − 1, the following applies:

$$\text{cpMvLX[cpIdx] = cpMvLXN[cpIdx]} \quad (702)$$

  – For xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1 and X = 0..1, the following applies:

$$\text{predFlagLX[xSbIdx][ySbIdx] = predFlagLX[0][0]} \quad (703)$$

  – For X = 0..1, the following applies:

    – When predFlagLX[0][0] is equal to 1, the derivation process for motion vector arrays from affine control point motion vectors as specified in clause 8.5.5.9 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth, the luma coding block height cbHeight, the number of control point motion vectors numCpMv, the control point motion vectors cpMvLX[cpIdx] with cpIdx being 0..numCpMv − 1, the prediction list utilization flags predFlagL0[0][0] and predFlagL1[0][0], the reference index refIdxLX, and the number of luma subblocks in horizontal direction numSbX and in vertical direction numSbY as inputs, the luma subblock motion vector array mvLX[xSbIdx][ySbIdx] and the chroma subblock motion vector array mvCLX[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1, the prediction refinement utility flag cbProfFlagLX, and motion vector difference array diffMvLX[xIdx][yIdx] with xIdx = 0..cbWidth / numSbX − 1, yIdx = 0..cbHeight / numSbY − 1 as outputs.

  – The following assignment is made for x = xCb..xCb + cbWidth − 1 and y = yCb..yCb + cbHeight − 1:

$$\text{MotionModelIdc[x][y] = numCpMv − 1} \quad (704)$$

8.5.5.3 Derivation process for subblock-based temporal merging candidates

Inputs to this process are:

– a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples,

– the availability flag availableFlagA_1 of the neighbouring coding unit,

– the reference indices refIdxLXA_1 of the neighbouring coding unit with X = 0..1,

– the prediction list utilization flags predFlagLXA_1 of the neighbouring coding unit with X = 0..1,

– the motion vector in 1/16 fractional-sample accuracy mvLXA_1 of the neighbouring coding unit with X = 0..1.

Outputs of this process are:

– the availability flag availableFlagSbCol,

– the number of luma subblocks in horizontal direction numSbX and in vertical direction numSbY,

– the reference indices refIdxL0SbCol and refIdxL1SbCol,

– the luma motion vectors in 1/16 fractional-sample accuracy mvL0SbCol[xSbIdx][ySbIdx] and mvL1SbCol[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1,

– the prediction list utilization flags predFlagL0SbCol[xSbIdx][ySbIdx] and predFlagL1SbCol[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1.

The availability flag availableFlagSbCol is derived as follows:

– If one or more of the following conditions are true, availableFlagSbCol is set equal to 0:
  – ph_temporal_mvp_enabled_flag is equal to 0.
  – sps_sbtmvp_enabled_flag is equal to 0.
  – cbWidth is less than 8.
  – cbHeight is less than 8.
– Otherwise, the following ordered steps apply:
  1. The location (xCtb, yCtb) of the top-left sample of the luma coding tree block that contains the current coding block and the location (xCtrCb, yCtrCb) of the below-right center sample of the current luma coding block are derived as follows:

     xCtb = (xCb >> CtbLog2SizeY) << CtbLog2SizeY (705)

     yCtb = (yCb >> CtbLog2SizeY) << CtbLog2SizeY (706)

     xCtrCb = xCb + (cbWidth / 2) (707)

     yCtrCb = yCb + (cbHeight / 2) (708)

  2. The derivation process for subblock-based temporal merging base motion data as specified in clause 8.5.5.4 is invoked with the location (xCtb, yCtb), the location (xCtrCb, yCtrCb), the availability flag availableFlagA_1, and the prediction list utilization flag predFlagLXA_1, and the reference index refIdxLXA_1, and the motion vector mvLXA_1, with X = 0..1 as inputs and the motion vectors ctrMvLX, and the prediction list utilization flags ctrPredFlagLX of the collocated block, with X = 0..1, and the temporal motion vector tempMv as outputs.
  3. The variable availableFlagSbCol is derived as follows:
     – If both ctrPredFlagL0 and ctrPredFlagL1 are equal to 0, availableFlagSbCol is set equal to 0.
     – Otherwise, availableFlagSbCol is set equal to 1.

When availableFlagSbCol is equal to 1, the following applies:

– The variables numSbX, numSbY, sbWidth, sbHeight and refIdxLXSbCol are derived as follows:

  numSbX = cbWidth >> 3 (709)

  numSbY = cbHeight >> 3 (710)

  sbWidth = cbWidth / numSbX (711)

  sbHeight = cbHeight / numSbY (712)

  refIdxLXSbCol = 0 (713)

– For xSbIdx = 0..numSbX − 1 and ySbIdx = 0..numSbY − 1, the motion vectors mvLXSbCol[xSbIdx][ySbIdx] and prediction list utilization flags predFlagLXSbCol[xSbIdx][ySbIdx] are derived as follows:
  – The luma location (xSb, ySb) specifying the below-right center sample of the current subblock relative to the top-left luma sample of the current picture is derived as follows:

    xSb = xCb + xSbIdx * sbWidth + sbWidth / 2 (714)

    ySb = yCb + ySbIdx * sbHeight + sbHeight / 2 (715)

  – The location (xColSb, yColSb) of the collocated subblock inside ColPic is derived as follows:
    – The following applies:

$$yColSb = Clip3(yCtb,\ Min(pps\_pic\_height\_in\_luma\_samples - 1, yCtb + (1 << CtbLog2SizeY) - 1),\ ySb + tempMv[1]) \quad (716)$$

– If sps_subpic_treated_as_pic_flag[CurrSubpicIdx] is equal to 1, the following applies:

$$xColSb = Clip3(xCtb,\ Min(SubpicRightBoundaryPos, xCtb + (1 << CtbLog2SizeY) + 3),\ xSb + tempMv[0]) \quad (717)$$

– Otherwise (sps_subpic_treated_as_pic_flag[CurrSubpicIdx] is equal to 0), the following applies:

$$xColSb = Clip3(xCtb,\ Min(pps\_pic\_width\_in\_luma\_samples - 1, xCtb + (1 << CtbLog2SizeY) + 3),\ xSb + tempMv[0]) \quad (718)$$

– The variable currCb specifies the luma coding block covering the current subblock inside the current picture.

– The luma location (xColCb, yColCb) is set equal to ((xColSb >> 3) << 3, (yColSb >> 3) << 3).

– The variable colCb specifies the luma coding block covering the location (xColCb, yColCb) inside the collocated picture specified by ColPic.

– The prediction list utilization flags predFlagL0SbCol[xSbIdx][ySbIdx] and predFlagL1SbCol[xSbIdx][ySbIdx] are initialized to be equal to FALSE.

– The derivation process for collocated motion vectors as specified in clause 8.5.2.12 is invoked with currCb, colCb, (xColCb, yColCb), refIdxL0 set equal to 0 and sbFlag set equal to 1 as inputs and the output being assigned to the motion vector of the subblock mvL0SbCol[xSbIdx][ySbIdx] and predFlagL0SbCol[xSbIdx][ySbIdx].

– When sh_slice_type is equal to B, the derivation process for collocated motion vectors as specified in clause 8.5.2.12 is invoked with currCb, colCb, (xColCb, yColCb), refIdxL1 set equal to 0 and sbFlag set equal to 1 as inputs and the output being assigned to the motion vector of the subblock mvL1SbCol[xSbIdx][ySbIdx] and predFlagL1SbCol[xSbIdx][ySbIdx].

– When predFlagL0SbCol[xSbIdx][ySbIdx] and predFlagL1SbCol[xSbIdx][ySbIdx] are both equal to 0, the following applies for X = 0..1:

$$mvLXSbCol[xSbIdx][ySbIdx] = ctrMvLX \quad (719)$$

$$predFlagLXSbCol[xSbIdx][ySbIdx] = ctrPredFlagLX \quad (720)$$

8.5.5.4 Derivation process for subblock-based temporal merging base motion data

Inputs to this process are:

– the location (xCtb, yCtb) of the top-left sample of the luma coding tree block that contains the current coding block,

– the location (xCtrCb, yCtrCb) of the top-left sample of the collocated luma coding block that covers the below-right center sample,

– the availability flag $availableFlagA_1$ of the neighbouring coding unit,

– the reference indices $refIdxLXA_1$ of the neighbouring coding unit with X = 0..1,

– the prediction list utilization flags $predFlagLXA_1$ of the neighbouring coding unit with X = 0..1,

– the motion vectors in 1/16 fractional-sample accuracy $mvLXA_1$ of the neighbouring coding unit with X = 0..1.

Outputs of this process are:

– the motion vectors ctrMvL0 and ctrMvL1,

– the prediction list utilization flags ctrPredFlagL0 and ctrPredFlagL1,

– the temporal motion vector tempMv.

The variable tempMv is set as follows:

$$tempMv[0] = 0 \quad (721)$$

$$\text{tempMv}[\,1\,] = 0 \tag{722}$$

The variable currPic specifies the current picture.

When availableFlagA_1 is equal to TRUE, the following applies:

– If all of the following conditions are true, tempMv is set equal to mvL0A_1:
  – predFlagL0A_1 is equal to 1,
  – DiffPicOrderCnt(ColPic, RefPicList[0][refIdxL0A_1]) is equal to 0,
– Otherwise, if all of the following conditions are true, tempMv is set equal to mvL1A_1:
  – sh_slice_type is equal to B,
  – predFlagL1A_1 is equal to 1,
  – DiffPicOrderCnt(ColPic, RefPicList[1][refIdxL1A_1]) is equal to 0.

The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to tempMv, rightShift set equal to 4, and leftShift set equal to 0 as inputs and the rounded tempMv as output.

The location (xColCb, yColCb) of the collocated block inside ColPic is derived as follows:

  – The following applies:

$$\begin{aligned}\text{yColCb} = \text{Clip3}(&\ \text{yCtb},\\ &\text{Min}(\ \text{pps\_pic\_height\_in\_luma\_samples} - 1,\ \text{yCtb} + (\ 1 \ll \text{CtbLog2SizeY}\ ) - 1\ ),\\ &\text{yCtrCb} + \text{tempMv}[\,1\,]\ )\end{aligned} \tag{723}$$

  – If sps_subpic_treated_as_pic_flag[CurrSubpicIdx] is equal to 1, the following applies:

$$\begin{aligned}\text{xColCb} = \text{Clip3}(&\ \text{xCtb},\\ &\text{Min}(\ \text{SubpicRightBoundaryPos},\ \text{xCtb} + (\ 1 \ll \text{CtbLog2SizeY}\ ) + 3\ ),\\ &\text{xCtrCb} + \text{tempMv}[\,0\,]\ )\end{aligned} \tag{724}$$

  – Otherwise (sps_subpic_treated_as_pic_flag[CurrSubpicIdx] is equal to 0), the following applies:

$$\begin{aligned}\text{xColCb} = \text{Clip3}(&\ \text{xCtb},\\ &\text{Min}(\ \text{pps\_pic\_width\_in\_luma\_samples} - 1,\ \text{xCtb} + (\ 1 \ll \text{CtbLog2SizeY}\ ) + 3\ ),\\ &\text{xCtrCb} + \text{tempMv}[\,0\,]\ )\end{aligned} \tag{725}$$

The array colPredMode is set equal to the prediction mode array CuPredMode[0] of the collocated picture specified by ColPic.

The motion vectors ctrMvL0 and ctrMvL1, and the prediction list utilization flags ctrPredFlagL0 and ctrPredFlagL1 are derived as follows:

– If colPredMode[(xColCb >> 3) << 3][(yColCb >> 3) << 3] is equal to MODE_INTER, the following applies:
  – The variable currCb specifies the luma coding block covering (xCtrCb, yCtrCb) inside the current picture.
  – The luma location (xColCb, yColCb) is set equal to ((xColCb >> 3) << 3, (yColCb >> 3) << 3).
  – The variable colCb specifies the luma coding block covering the location (xColCb, yColCb) inside the collocated picture specified by ColPic.
  – The prediction list utilization flags ctrPredFlagL0 and ctrPredFlagL1 are initialized to be equal to FALSE.
  – The derivation process for collocated motion vectors specified in clause 8.5.2.12 is invoked with currCb, colCb, (xColCb, yColCb), refIdxL0 set equal to 0, and sbFlag set equal to 1 as inputs and the output being assigned to ctrMvL0 and ctrPredFlagL0.
  – When sh_slice_type is equal to B, the derivation process for collocated motion vectors specified in clause 8.5.2.12 is invoked with currCb, colCb, (xColCb, yColCb), refIdxL1 set equal to 0, and sbFlag set equal to 1 as inputs and the output being assigned to ctrMvL1 and ctrPredFlagL1.
– Otherwise, the following applies:

$$\text{ctrPredFlagL0} = 0 \tag{726}$$

ctrPredFlagL1 = 0 (727)

8.5.5.5 Derivation process for luma affine control point motion vectors from a neighbouring block

Inputs to this process are:

– a luma location (xCb, yCb) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– two variables cbWidth and cbHeight specifying the width and the height of the current luma coding block,

– a luma location (xNb, yNb) specifying the top-left sample of the neighbouring luma coding block relative to the top-left luma sample of the current picture,

– two variables nNbW and nNbH specifying the width and the height of the neighbouring luma coding block,

– the number of control point motion vectors numCpMv.

Outputs of this process are the luma affine control point vectors cpMvLX[cpIdx] with cpIdx = 0..numCpMv − 1 and X being 0 or 1.

The variable isCTUboundary is derived as follows:

– If all the following conditions are true, isCTUboundary is set equal to TRUE:
  – ((yNb + nNbH) % CtbSizeY) is equal to 0
  – yNb + nNbH is equal to yCb

– Otherwise, isCTUboundary is set equal to FALSE.

The variables log2NbW and log2NbH are derived as follows:

log2NbW = Log2(nNbW) (728)

log2NbH = Log2(nNbH) (729)

The variables mvScaleHor, mvScaleVer, dHorX and dVerX are derived as follows:

– If isCTUboundary is equal to TRUE, the following applies:

mvScaleHor = MvLX[xNb][yNb + nNbH − 1][0] << 7 (730)

mvScaleVer = MvLX[xNb][yNb + nNbH − 1][1] << 7 (731)

dHorX = (MvLX[xNb + nNbW − 1][yNb + nNbH − 1][0] − MvLX[xNb][yNb + nNbH − 1][0])
<< (7 − log2NbW) (732)

dVerX = (MvLX[xNb + nNbW − 1][yNb + nNbH − 1][1] − MvLX[xNb][yNb + nNbH − 1][1])
<< (7 − log2NbW) (733)

– Otherwise (isCTUboundary is equal to FALSE), the following applies:

mvScaleHor = CpMvLX[xNb][yNb][0][0] << 7 (734)

mvScaleVer = CpMvLX[xNb][yNb][0][1] << 7 (735)

dHorX = (CpMvLX[xNb + nNbW − 1][yNb][1][0] − CpMvLX[xNb][yNb][0][0])
<< (7 − log2NbW) (736)

dVerX = (CpMvLX[xNb + nNbW − 1][yNb][1][1] − CpMvLX[xNb][yNb][0][1])
<< (7 − log2NbW) (737)

The variables dHorY and dVerY are derived as follows:

– If isCTUboundary is equal to FALSE and MotionModelIdc[xNb][yNb] is equal to 2, the following applies:

dHorY = (CpMvLX[xNb][yNb + nNbH − 1][2][0] − CpMvLX[xNb][yNb][0][0])
<< (7 − log2NbH) (738)

$$dVerY = (CpMvLX[xNb][yNb + nNbH - 1][2][1] - CpMvLX[xNb][yNb][0][1]) << (7 - log2NbH) \quad (739)$$

– Otherwise (isCTUboundary is equal to TRUE or MotionModelIdc[xNb][yNb] is equal to 1), the following applies:

$$dHorY = -dVerX \quad (740)$$

$$dVerY = dHorX \quad (741)$$

The luma affine control point motion vectors cpMvLX[cpIdx] with cpIdx = 0..numCpMv − 1 are derived as follows:

– When isCTUboundary is equal to TRUE, yNb is set equal to yCb.

– The first two control point motion vectors cpMvLX[0] and cpMvLX[1] are derived as follows:

$$cpMvLX[0][0] = (mvScaleHor + dHorX * (xCb - xNb) + dHorY * (yCb - yNb)) \quad (742)$$

$$cpMvLX[0][1] = (mvScaleVer + dVerX * (xCb - xNb) + dVerY * (yCb - yNb)) \quad (743)$$

$$cpMvLX[1][0] = (mvScaleHor + dHorX * (xCb + cbWidth - xNb) + dHorY * (yCb - yNb)) \quad (744)$$

$$cpMvLX[1][1] = (mvScaleVer + dVerX * (xCb + cbWidth - xNb) + dVerY * (yCb - yNb)) \quad (745)$$

– If numCpMv is equal to 3, the third control point vector cpMvLX[2] is derived as follows:

$$cpMvLX[2][0] = (mvScaleHor + dHorX * (xCb - xNb) + dHorY * (yCb + cbHeight - yNb)) \quad (746)$$

$$cpMvLX[2][1] = (mvScaleVer + dVerX * (xCb - xNb) + dVerY * (yCb + cbHeight - yNb)) \quad (747)$$

– The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to cpMvLX[cpIdx], rightShift set equal to 7, and leftShift set equal to 0 as inputs and the rounded cpMvLX[cpIdx] as output, with cpIdx = 0..numCpMv − 1.

– The motion vectors cpMvLX[cpIdx] with cpIdx = 0..numCpMv − 1 are clipped as follows:

$$cpMvLX[cpIdx][0] = Clip3(-2^{17}, 2^{17} - 1, cpMvLX[cpIdx][0]) \quad (748)$$

$$cpMvLX[cpIdx][1] = Clip3(-2^{17}, 2^{17} - 1, cpMvLX[cpIdx][1]) \quad (749)$$

8.5.5.6 Derivation process for constructed affine control point motion vector merging candidates

Inputs to this process are:

– a luma location (xCb, yCb) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– two variables cbWidth and cbHeight specifying the width and the height of the current luma coding block,

– the availability flags $availableA_0$, $availableA_1$, $availableA_2$, $availableB_0$, $availableB_1$, $availableB_2$, $availableB_{3,}$

– the sample locations ($xNbA_0$, $yNbA_0$), ($xNbA_1$, $yNbA_1$), ($xNbA_2$, $yNbA_2$), ($xNbB_0$, $yNbB_0$), ($xNbB_1$, $yNbB_1$), ($xNbB_2$, $yNbB_2$) and ($xNbB_3$, $yNbB_3$).

Output of this process are:

– the availability flags of the constructed affine control point motion vector merging candidiates availableFlagConstK, with K = 1..6,

– the reference indices refIdxLXConstK, with K = 1..6, X = 0..1,

– the prediction list utilization flags predFlagLXConstK, with K = 1..6, X = 0..1,

– the affine motion model indices motionModelIdcConstK, with K = 1..6,

– the bi-prediction weight indices bcwIdxConstK, with K = 1..6,

– the constructed affine control point motion vectors cpMvLXConstK[cpIdx] with cpIdx = 0..2, K = 1..6 and X = 0..1.

The availability flags of the constructed affine control point motion vector merging candidates availableFlagConstK, with K = 1..6, are initialized to be equal to FALSE.

The prediction list utilization flags predFlagLXConstK, with K = 1..6, X = 0..1, are initialized to be equal to FALSE.

The first (top-left) control point motion vector cpMvLXCorner[0], reference index refIdxLXCorner[0], prediction list utilization flag predFlagLXCorner[0], bi-prediction weight index bcwIdxCorner[0] and the availability flag availableFlagCorner[0] with X = 0..1 are derived as follows:

– The availability flag availableFlagCorner[0] is set equal to FALSE.

– The following applies for (xNbTL, yNbTL) with TL being replaced by B_2, B_3, and A_2:

  – When availableTL is equal to TRUE and availableFlagCorner[0] is equal to FALSE, the following applies for X = 0..1:

refIdxLXCorner[0] = RefIdxLX[xNbTL][yNbTL] (750)

predFlagLXCorner[0] = PredFlagLX[xNbTL][yNbTL] (751)

cpMvLXCorner[0] = MvLX[xNbTL][yNbTL] (752)

bcwIdxCorner[0] = BcwIdx[xNbTL][yNbTL] (753)

availableFlagCorner[0] = TRUE (754)

The second (top-right) control point motion vector cpMvLXCorner[1], reference index refIdxLXCorner[1], prediction list utilization flag predFlagLXCorner[1], bi-prediction weight index bcwIdxCorner[1] and the availability flag availableFlagCorner[1] with X = 0..1 are derived as follows:

– The availability flag availableFlagCorner[1] is set equal to FALSE.

– The following applies for (xNbTR, yNbTR) with TR being replaced by B_1 and B_0:

  – When availableTR is equal to TRUE and availableFlagCorner[1] is equal to FALSE, the following applies for X = 0..1:

refIdxLXCorner[1] = RefIdxLX[xNbTR][yNbTR] (755)

predFlagLXCorner[1] = PredFlagLX[xNbTR][yNbTR] (756)

cpMvLXCorner[1] = MvLX[xNbTR][yNbTR] (757)

bcwIdxCorner[1] = BcwIdx[xNbTR][yNbTR] (758)

availableFlagCorner[1] = TRUE (759)

The third (bottom-left) control point motion vector cpMvLXCorner[2], reference index refIdxLXCorner[2], prediction list utilization flag predFlagLXCorner[2] and the availability flag availableFlagCorner[2] with X = 0..1 are derived as follows:

– The availability flag availableFlagCorner[2] is set equal to FALSE.

– The following applies for (xNbBL, yNbBL) with BL being replaced by A_1 and A_0:

  – When availableBL is equal to TRUE and availableFlagCorner[2] is equal to FALSE, the following applies for X = 0..1:

refIdxLXCorner[2] = RefIdxLX[xNbBL][yNbBL] (760)

predFlagLXCorner[2] = PredFlagLX[xNbBL][yNbBL] (761)

cpMvLXCorner[2] = MvLX[xNbBL][yNbBL] (762)

availableFlagCorner[2] = TRUE (763)

The fourth (collocated bottom-right) control point motion vector cpMvLXCorner[3], reference index refIdxLXCorner[3], prediction list utilization flag predFlagLXCorner[3] and the availability flag availableFlagCorner[3] with = 0..1 are derived as follows:

– The reference indices for the temporal merging candidate, refIdxLXCorner[3], with X = 0..1, are set equal to 0.

– For X = 0..1, the variables mvLXCol and availableFlagLXCol are derived as follows:

  – If ph_temporal_mvp_enabled_flag is equal to 0, both components of mvLXCol are set equal to 0 and availableFlagLXCol is set equal to 0.

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

xColBr = xCb + cbWidth (764)

yColBr = yCb + cbHeight (765)

rightBoundaryPos = sps_subpic_treated_as_pic_flag[CurrSubpicIdx] ? SubpicRightBoundaryPos :
pps_pic_width_in_luma_samples − 1 (766)

botBoundaryPos = sps_subpic_treated_as_pic_flag[CurrSubpicIdx] ? SubpicBotBoundaryPos :
pps_pic_height_in_luma_samples − 1 (767)

– If yCb >> CtbLog2SizeY is equal to yColBr >> CtbLog2SizeY, yColBr is less than or equal to botBoundaryPos and xColBr is less than or equal to rightBoundaryPos, the following applies:

  – The luma location (xColCb, yColCb) is set equal to ((xColBr >> 3) << 3, (yColBr >> 3) << 3).

  – The variable colCb specifies the luma coding block covering the location (xColCb, yColCb) inside the collocated picture specified by ColPic.

  – The derivation process for collocated motion vectors as specified in clause 8.5.2.12 is invoked with currCb, colCb, (xColCb, yColCb), refIdxLXCorner[3] and sbFlag set equal to 0 as inputs, and the output is assigned to mvLXCol and availableFlagLXCol.

– Otherwise, both components of mvLXCol are set equal to 0 and availableFlagLXCol is set equal to 0.

– The variables availableFlagCorner[3], predFlagL0Corner[3], cpMvL0Corner[3] and predFlagL1Corner[3] are derived as follows:

availableFlagCorner[3] = availableFlagL0Col (768)

predFlagL0Corner[3] = availableFlagL0Col (769)

cpMvL0Corner[3] = mvL0Col (770)

predFlagL1Corner[3] = 0 (771)

– When sh_slice_type is equal to B, the variables availableFlagCorner[3], predFlagL1Corner[3] and cpMvL1Corner[3] are derived as follows:

availableFlagCorner[3] = availableFlagL0Col | | availableFlagL1Col (772)

predFlagL1Corner[3] = availableFlagL1Col (773)

cpMvL1Corner[3] = mvL1Col (774)

When sps_6param_affine_enabled_flag is equal to 1, the first four constructed affine control point motion vector merging candidates ConstK with K = 1..4 including the availability flags availableFlagConstK, the reference indices refIdxLXConstK, the prediction list utilization flags predFlagLXConstK, the affine motion model indices motionModelIdcConstK, and the constructed affine control point motion vectors cpMvLXConstK[cpIdx] with cpIdx = 0..2 and X = 0..1 are derived as follows:

1. When availableFlagCorner[0] is equal to TRUE and availableFlagCorner[1] is equal to TRUE and availableFlagCorner[2] is equal to TRUE, the following applies:

   – For X = 0..1, the following applies:

     – The variable availableFlagLX is derived as follows:

       – If all of the following conditions are TRUE, availableFlagLX is set equal to TRUE:

         – predFlagLXCorner[0] is equal to 1;

         – predFlagLXCorner[1] is equal to 1;

         – predFlagLXCorner[2] is equal to 1;

         – refIdxLXCorner[0] is equal to refIdxLXCorner[1];

         – refIdxLXCorner[0] is equal to refIdxLXCorner[2];

       – Otherwise, availableFlagLX is set equal to FALSE.

     – When availableFlagLX is equal to TRUE, the following assignments are made:

predFlagLXConst1 = 1 (775)

refIdxLXConst1 = refIdxLXCorner[0] (776)

cpMvLXConst1[0] = cpMvLXCorner[0] (777)

$$\text{cpMvLXConst1[1] = cpMvLXCorner[1]} \tag{778}$$

$$\text{cpMvLXConst1[2] = cpMvLXCorner[2]} \tag{779}$$

– The bi-prediction weight index bcwIdxConst1 is derived as follows:
  – If both availableFlagL0 and availableFlagL1 are equal to 1, bcwIdxConst1 is set equal to bcwIdxCorner[0].
  – Otherwise, bcwIdxConst1 is set equal to 0.
– The variables availableFlagConst1 and motionModelIdcConst1 are derived as follows:
  – If availableFlagL0 or availableFlagL1 is equal to 1, availableFlagConst1 is set equal to TRUE and motionModelIdcConst1 is set equal to 2.
  – Otherwise, availableFlagConst1 is set equal to FALSE and motionModelIdcConst1 is set equal to 0.

2. When availableFlagCorner[0] is equal to TRUE and availableFlagCorner[1] is equal to TRUE and availableFlagCorner[3] is equal to TRUE, the following applies:

– For X = 0..1, the following applies:
  – The variable availableFlagLX is derived as follows:
    – If all of the following conditions are TRUE, availableFlagLX is set equal to TRUE:
      – predFlagLXCorner[0] is equal to 1;
      – predFlagLXCorner[1] is equal to 1;
      – predFlagLXCorner[3] is equal to 1;
      – refIdxLXCorner[0] is equal to refIdxLXCorner[1];
      – refIdxLXCorner[0] is equal to refIdxLXCorner[3];
    – Otherwise, availableFlagLX is set equal to FALSE.
  – When availableFlagLX is equal to TRUE, the following assignments are made:

$$\text{predFlagLXConst2} = 1 \tag{780}$$

$$\text{refIdxLXConst2 = refIdxLXCorner[0]} \tag{781}$$

$$\text{cpMvLXConst2[0] = cpMvLXCorner[0]} \tag{782}$$

$$\text{cpMvLXConst2[1] = cpMvLXCorner[1]} \tag{783}$$

$$\text{cpMvLXConst2[2] = cpMvLXCorner[3] + cpMvLXCorner[0]} - \text{cpMvLXCorner[1]} \tag{784}$$

$$\text{cpMvLXConst2[2][0] = Clip3(} -2^{17}, 2^{17} - 1, \text{cpMvLXConst2[2][0])} \tag{785}$$

$$\text{cpMvLXConst2[2][1] = Clip3(} -2^{17}, 2^{17} - 1, \text{cpMvLXConst2[2][1])} \tag{786}$$

– The bi-prediction weight index bcwIdxConst2 is derived as follows:
  – If both availableFlagL0 and availableFlagL1 are equal to 1, bcwIdxConst2 is set equal to bcwIdxCorner[0].
  – Otherwise, bcwIdxConst2 is set equal to 0.
– The variables availableFlagConst2 and motionModelIdcConst2 are derived as follows:
  – If availableFlagL0 or availableFlagL1 is equal to 1, availableFlagConst2 is set equal to TRUE and motionModelIdcConst2 is set equal to 2.
  – Otherwise, availableFlagConst2 is set equal to FALSE and motionModelIdcConst2 is set equal to 0.

3. When availableFlagCorner[0] is equal to TRUE and availableFlagCorner[2] is equal to TRUE and availableFlagCorner[3] is equal to TRUE, the following applies:

– For X = 0..1, the following applies:
  – The variable availableFlagLX is derived as follows:
    – If all of the following conditions are TRUE, availableFlagLX is set equal to TRUE:
      – predFlagLXCorner[0] is equal to 1;
      – predFlagLXCorner[2] is equal to 1;

– predFlagLXCorner[3] is equal to 1;

– refIdxLXCorner[0] is equal to refIdxLXCorner[2];

– refIdxLXCorner[0] is equal to refIdxLXCorner[3];

– Otherwise, availableFlagLX is set equal to FALSE.

– When availableFlagLX is equal to TRUE, the following assignments are made:

$$predFlagLXConst3 = 1 \tag{787}$$

$$refIdxLXConst3 = refIdxLXCorner[0] \tag{788}$$

$$cpMvLXConst3[0] = cpMvLXCorner[0] \tag{789}$$

$$cpMvLXConst3[1] = cpMvLXCorner[3] + cpMvLXCorner[0] - cpMvLXCorner[2] \tag{790}$$

$$cpMvLXConst3[1][0] = Clip3(-2^{17}, 2^{17} - 1, cpMvLXConst3[1][0]) \tag{791}$$

$$cpMvLXConst3[1][1] = Clip3(-2^{17}, 2^{17} - 1, cpMvLXConst3[1][1]) \tag{792}$$

$$cpMvLXConst3[2] = cpMvLXCorner[2] \tag{793}$$

– The bi-prediction weight index bcwIdxConst3 is derived as follows:

– If both availableFlagL0 and availableFlagL1 are equal to 1, bcwIdxConst3 is set equal to bcwIdxCorner[0].

– Otherwise, bcwIdxConst3 is set equal to 0.

– The variables availableFlagConst3 and motionModelIdcConst3 are derived as follows:

– If availableFlagL0 or availableFlagL1 is equal to 1, availableFlagConst3 is set equal to TRUE and motionModelIdcConst3 is set equal to 2.

– Otherwise, availableFlagConst3 is set equal to FALSE and motionModelIdcConst3 is set equal to 0.

4. When availableFlagCorner[1] is equal to TRUE and availableFlagCorner[2] is equal to TRUE and availableFlagCorner[3] is equal to TRUE, the following applies:

– For X = 0..1, the following applies:

– The variable availableFlagLX is derived as follows:

– If all of the following conditions are TRUE, availableFlagLX is set equal to TRUE:

– predFlagLXCorner[1] is equal to 1;

– predFlagLXCorner[2] is equal to 1;

– predFlagLXCorner[3] is equal to 1;

– refIdxLXCorner[1] is equal to refIdxLXCorner[2];

– refIdxLXCorner[1] is equal to refIdxLXCorner[3];

– Otherwise, availableFlagLX is set equal to FALSE.

– When availableFlagLX is equal to TRUE, the following assignments are made:

$$predFlagLXConst4 = 1 \tag{794}$$

$$refIdxLXConst4 = refIdxLXCorner[1] \tag{795}$$

$$cpMvLXConst4[0] = cpMvLXCorner[1] + cpMvLXCorner[2] - cpMvLXCorner[3] \tag{796}$$

$$cpMvLXConst4[0][0] = Clip3(-2^{17}, 2^{17} - 1, cpMvLXConst4[0][0]) \tag{797}$$

$$cpMvLXConst4[0][1] = Clip3(-2^{17}, 2^{17} - 1, cpMvLXConst4[0][1]) \tag{798}$$

$$cpMvLXConst4[1] = cpMvLXCorner[1] \tag{799}$$

$$cpMvLXConst4[2] = cpMvLXCorner[2] \tag{800}$$

– The bi-prediction weight index bcwIdxConst4 is derived as follows:

– If both availableFlagL0 and availableFlagL1 are equal to 1, bcwIdxConst4 is set equal to bcwIdxCorner[1].

– Otherwise, bcwIdxConst4 is set equal to 0.

– The variables availableFlagConst4 and motionModelIdcConst4 are derived as follows:

  – If availableFlagL0 or availableFlagL1 is equal to 1, availableFlagConst4 is set equal to TRUE and motionModelIdcConst4 is set equal to 2.

  – Otherwise, availableFlagConst4 is set equal to FALSE and motionModelIdcConst4 is set equal to 0.

The last two constructed affine control point motion vector merging candidates ConstK with K = 5..6 including the availability flags availableFlagConstK, the reference indices refIdxLXConstK, the prediction list utilization flags predFlagLXConstK, the affine motion model indices motionModelIdcConstK, and the constructed affine control point motion vectors cpMvLXConstK[cpIdx] with cpIdx = 0..2 and X = 0..1 are derived as follows:

5. When availableFlagCorner[0] is equal to TRUE and availableFlagCorner[1] is equal to TRUE, the following applies:

   – For X = 0..1, the following applies:

     – The variable availableFlagLX is derived as follows:

       – If all of the following conditions are TRUE, availableFlagLX is set equal to TRUE:

         – predFlagLXCorner[0] is equal to 1;

         – predFlagLXCorner[1] is equal to 1;

         – refIdxLXCorner[0] is equal to refIdxLXCorner[1];

       – Otherwise, availableFlagLX is set equal to FALSE.

     – When availableFlagLX is equal to TRUE, the following assignments are made:

$$\text{predFlagLXConst5} = 1 \tag{801}$$

$$\text{refIdxLXConst5} = \text{refIdxLXCorner}[\ 0\ ] \tag{802}$$

$$\text{cpMvLXConst5}[\ 0\ ] = \text{cpMvLXCorner}[\ 0\ ] \tag{803}$$

$$\text{cpMvLXConst5}[\ 1\ ] = \text{cpMvLXCorner}[\ 1\ ] \tag{804}$$

   – The bi-prediction weight index bcwIdxConst5 is derived as follows:

     – If both availableFlagL0 and availableFlagL1 are equal to 1, bcwIdxConst5 is set equal to bcwIdxCorner[0].

     – Otherwise, bcwIdxConst5 is set equal to 0.

   – The variables availableFlagConst5 and motionModelIdcConst5 are derived as follows:

     – If availableFlagL0 or availableFlagL1 is equal to 1, availableFlagConst5 is set equal to TRUE and motionModelIdcConst5 is set equal to 1.

     – Otherwise, availableFlagConst5 is set equal to FALSE and motionModelIdcConst5 is set equal to 0.

6. When availableFlagCorner[0] is equal to TRUE and availableFlagCorner[2] is equal to TRUE, the following applies:

   – For X = 0..1, the following applies:

     – The variable availableFlagLX is derived as follows:

       – If all of the following conditions are TRUE, availableFlagLX is set equal to TRUE:

         – predFlagLXCorner[0] is equal to 1;

         – predFlagLXCorner[2] is equal to 1;

         – refIdxLXCorner[0] is equal to refIdxLXCorner[2];

       – Otherwise, availableFlagLX is set equal to FALSE.

     – When availableFlagLX is equal to TRUE, the following applies:

       – The second control point motion vector cpMvLXCorner[1] is derived as follows:

$$\begin{aligned}\text{cpMvLXCorner}[\ 1\ ][\ 0\ ] = {} & (\ \text{cpMvLXCorner}[\ 0\ ][\ 0\ ] \ll 7\ ) + \\ & (\ (\ \text{cpMvLXCorner}[\ 2\ ][\ 1\ ] - \text{cpMvLXCorner}[\ 0\ ][\ 1\ ]\ ) \\ & \ll (\ 7 + \text{Log2}(\ \text{cbWidth}\ ) - \text{Log2}(\ \text{cbHeight}\ )\ )\ )\end{aligned} \tag{805}$$

$$\text{cpMvLXCorner}[1][1] = (\text{cpMvLXCorner}[0][1] \ll 7) - ((\text{cpMvLXCorner}[2][0] - \text{cpMvLXCorner}[0][0]) \ll (7 + \text{Log2}(\text{cbWidth}) - \text{Log2}(\text{cbHeight}))) \quad (806)$$

– The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to cpMvLXCorner[1], rightShift set equal to 7, and leftShift set equal to 0 as inputs and the rounded cpMvLXCorner[1] as output.

– The following assignments are made:

$$\text{predFlagLXConst6} = 1 \quad (807)$$

$$\text{refIdxLXConst6} = \text{refIdxLXCorner}[0] \quad (808)$$

$$\text{cpMvLXConst6}[0] = \text{cpMvLXCorner}[0] \quad (809)$$

$$\text{cpMvLXConst6}[1] = \text{cpMvLXCorner}[1] \quad (810)$$

$$\text{cpMvLXConst6}[1][0] = \text{Clip3}(-2^{17}, 2^{17} - 1, \text{cpMvLXConst6}[1][0]) \quad (811)$$

$$\text{cpMvLXConst6}[1][1] = \text{Clip3}(-2^{17}, 2^{17} - 1, \text{cpMvLXConst6}[1][1]) \quad (812)$$

– The bi-prediction weight index bcwIdxConst6 is derived as follows:

  – If both availableFlagL0 and availableFlagL1 are equal to 1, bcwIdxConst6 is set equal to bcwIdxCorner[0].

  – Otherwise, bcwIdxConst6 is set equal to 0.

– The variables availableFlagConst6 and motionModelIdcConst6 are derived as follows:

  – If availableFlagL0 or availableFlagL1 is equal to 1, availableFlagConst6 is set equal to TRUE and motionModelIdcConst6 is set equal to 1.

  – Otherwise, availableFlagConst6 is set equal to FALSE and motionModelIdcConst6 is set equal to 0.

8.5.5.7 Derivation process for luma affine control point motion vector predictors

Inputs to this process are:

– a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– two variables cbWidth and cbHeight specifying the width and the height of the current luma coding block,

– the reference index of the current coding unit refIdxLX, with X being 0 or 1,

– the number of control point motion vectors numCpMv.

Outputs of this process are the luma affine control point motion vector predictors cpMvpLX[cpIdx] with X being 0 or 1, and cpIdx = 0..numCpMv − 1.

For the derivation of the control point motion vectors predictor candidate list, cpMvpListLX, the following ordered steps apply:

1. The number of control point motion vector predictor candidates in the list numCpMvpCandLX is set equal to 0.

2. The variables availableFlagA and availableFlagB are both set equal to FALSE.

3. The sample locations ($xNbA_0$, $yNbA_0$), ($xNbA_1$, $yNbA_1$), ($xNbB_0$, $yNbB_0$), ($xNbB_1$, $yNbB_1$), and ($xNbB_2$, $yNbB_2$) are derived as follows:

$$(xNbA_0, yNbA_0) = (xCb - 1, yCb + cbHeight) \quad (813)$$

$$(xNbA_1, yNbA_1) = (xCb - 1, yCb + cbHeight - 1) \quad (814)$$

$$(xNbB_0, yNbB_0) = (xCb + cbWidth, yCb - 1) \quad (815)$$

$$(xNbB_1, yNbB_1) = (xCb + cbWidth - 1, yCb - 1) \quad (816)$$

$$(xNbB_2, yNbB_2) = (xCb - 1, yCb - 1) \quad (817)$$

4. The following applies for ($xNbA_k$, $yNbA_k$) from ($xNbA_0$, $yNbA_0$) to ($xNbA_1$, $yNbA_1$):

   – The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring luma location ($xNbA_k$, $yNbA_k$), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag availableA$_k$.

– When availableA$_k$ is equal to TRUE and MotionModelIdc[xNbA$_k$][yNbA$_k$] is greater than 0 and availableFlagA is equal to FALSE, the following applies:

  – The variable (xNb, yNb) is set equal to (CbPosX[0][xNbA$_k$][yNbA$_k$], CbPosY[0][xNbA$_k$][yNbA$_k$]), nbW is set equal to CbWidth[0][xNbA$_k$][yNbA$_k$],and nbH is set equal to CbHeight[0][xNbA$_k$][yNbA$_k$].

  – If PredFlagLX[xNbA$_k$][yNbA$_k$] is equal to 1 and DiffPicOrderCnt(RefPicList[X][RefIdxLX[xNbA$_k$][yNbA$_k$]], RefPicList[X][refIdxLX]) is equal to 0, the following applies:

    – The variable availableFlagA is set equal to TRUE.

    – The derivation process for luma affine control point motion vectors from a neighbouring block as specified in clause 8.5.5.5 is invoked with the luma coding block location (xCb, yCb), the luma coding block width and height (cbWidth, cbHeight), the neighbouring luma coding block location (xNb, yNb), the neighbouring luma coding block width and height (nbW, nbH), and the number of control point motion vectors numCpMv as input, the control point motion vector predictor candidates cpMvpLX[cpIdx] with cpIdx = 0..numCpMv − 1 as output.

    – The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to cpMvpLX[cpIdx], rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded cpMvpLX[cpIdx] with cpIdx = 0..numCpMv − 1 as output.

    – For cpIdx = 0..numCpMv − 1, the following applies:

$$\text{cpMvpListLX[numCpMvpCandLX][cpIdx] = cpMvpLX[cpIdx]} \quad (818)$$

    – The following applies:

$$\text{numCpMvpCandLX = numCpMvpCandLX + 1} \quad (819)$$

  – Otherwise, if PredFlagLY[xNbA$_k$][yNbA$_k$] (with Y = !X) is equal to 1 and DiffPicOrderCnt(RefPicList[Y][RefIdxLY[xNbA$_k$][yNbA$_k$]], RefPicList[X][refIdxLX]) is equal to 0, the following applies:

    – The variable availableFlagA is set equal to TRUE.

    – The derivation process for luma affine control point motion vectors from a neighbouring block as specified in clause 8.5.5.5 is invoked with the luma coding block location (xCb, yCb), the luma coding block width and height (cbWidth, cbHeight), the neighbouring luma coding block location (xNb, yNb), the neighbouring luma coding block width and height (nbW, nbH), and the number of control point motion vectors numCpMv as input, the control point motion vector predictor candidates cpMvpLY[cpIdx] with cpIdx = 0..numCpMv − 1 as output.

    – The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to cpMvpLY[cpIdx], rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded cpMvpLY[cpIdx] with cpIdx = 0..numCpMv − 1 as output.

    – For cpIdx = 0..numCpMv − 1, the following applies:

$$\text{cpMvpListLX[numCpMvpCandLX][cpIdx] = cpMvpLY[cpIdx]} \quad (820)$$

    – The following applies:

$$\text{numCpMvpCandLX = numCpMvpCandLX + 1} \quad (821)$$

5. The following applies for (xNbB$_k$, yNbB$_k$) from (xNbB$_0$, yNbB$_0$) to (xNbB$_2$, yNbB$_2$):

– The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring luma location (xNbB$_k$, yNbB$_k$), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag availableB$_k$.

  – When availableB$_k$ is equal to TRUE and MotionModelIdc[xNbB$_k$][yNbB$_k$] is greater than 0 and availableFlagB is equal to FALSE, the following applies:

– The variable (xNb, yNb) is set equal to (CbPosX[0][$xNbB_k$][$yNbB_k$], CbPosY[0][$xNbB_k$][$yNbB_k$]), nbW is set equal to CbWidth[0][$xNbB_k$][$yNbB_k$],and nbH is set equal to CbHeight[0][$xNbB_k$][$yNbB_k$].

– If PredFlagLX[$xNbB_k$][$yNbB_k$] is equal to 1 and DiffPicOrderCnt(RefPicList[X][RefIdxLX[$xNbB_k$][$yNbB_k$]], RefPicList[X][refIdxLX]) is equal to 0, the following applies:

  – The variable availableFlagB is set equal to TRUE.

  – The derivation process for luma affine control point motion vectors from a neighbouring block as specified in clause 8.5.5.5 is invoked with the luma coding block location (xCb, yCb), the luma coding block width and height (cbWidth, cbHeight), the neighbouring luma coding block location (xNb, yNb), the neighbouring luma coding block width and height (nbW, nbH), and the number of control point motion vectors numCpMv as input, the control point motion vector predictor candidates cpMvpLX[cpIdx] with cpIdx = 0..numCpMv − 1 as output.

  – The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to cpMvpLX[cpIdx], rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded cpMvpLX[cpIdx] with cpIdx = 0..numCpMv − 1 as output.

  – For cpIdx = 0..numCpMv − 1, the following applies:

cpMvpListLX[numCpMvpCandLX][cpIdx] = cpMvpLX[cpIdx] (822)

  – The following applies:

numCpMvpCandLX = numCpMvpCandLX + 1 (823)

– Otherwise, if PredFlagLY[$xNbB_k$][$yNbB_k$] (with Y = !X) is equal to 1 and DiffPicOrderCnt(RefPicList[Y][RefIdxLY[$xNbB_k$][$yNbB_k$]], RefPicList[X][refIdxLX]) is equal to 0, the following applies:

  – The variable availableFlagB is set equal to TRUE.

  – The derivation process for luma affine control point motion vectors from a neighbouring block as specified in clause 8.5.5.5 is invoked with the luma coding block location (xCb, yCb), the luma coding block width and height (cbWidth, cbHeight), the neighbouring luma coding block location (xNb, yNb), the neighbouring luma coding block width and height (nbW, nbH), and the number of control point motion vectors numCpMv as input, the control point motion vector predictor candidates cpMvpLY[cpIdx] with cpIdx = 0..numCpMv − 1 as output.

  – The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to cpMvpLY[cpIdx], rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded cpMvpLY[cpIdx] with cpIdx = 0..numCpMv − 1 as output.

  – For cpIdx = 0..numCpMv − 1, the following applies:

cpMvpListLX[numCpMvpCandLX][cpIdx] = cpMvpLY[cpIdx] (824)

  – The following applies:

numCpMvpCandLX = numCpMvpCandLX + 1 (825)

6. When numCpMvpCandLX is less than 2, the following applies:

   – The derivation process for constructed affine control point motion vector prediction candidate as specified in clause 8.5.5.8 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth, the luma coding block height cbHeight, and the reference index of the current coding unit refIdxLX as inputs, and the availability flag availableConsFlagLX, the availability flags availableFlagLX[cpIdx] and cpMvpLX[cpIdx] with cpIdx = 0..2 as outputs.

   – When availableConsFlagLX is equal to 1, the following applies:

     – For cpIdx = 0..numCpMv − 1, the following applies:

cpMvpListLX[numCpMvpCandLX][cpIdx] = cpMvpLX[cpIdx] (826)

     – The following applies:

numCpMvpCandLX = numCpMvpCandLX + 1 (827)

7. The following applies for nbCpIdx = 2..0:

– When numCpMvpCandLX is less than 2 and availableFlagLX[nbCpIdx] is equal to 1, the following applies:

  – For cpIdx = 0..numCpMv − 1, the following assignment is made:

$$\text{cpMvpListLX[numCpMvpCandLX][cpIdx] = cpMvpLX[nbCpIdx]} \quad (828)$$

  – The following applies:

$$\text{numCpMvpCandLX = numCpMvpCandLX + 1} \quad (829)$$

8. When numCpMvpCandLX is less than 2, the following applies:

   – The derivation process for temporal luma motion vector prediction as specified in clause 8.5.2.11 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth, the luma coding block height cbHeight and refIdxLX as inputs, and with the output being the availability flag availableFlagLXCol and the temporal motion vector predictor mvLXCol.

   – When availableFlagLXCol is equal to 1, the following applies:

     – The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to mvLXCol, rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded mvLXCol as output.

     – For cpIdx = 0..numCpMv − 1, the following applies:

$$\text{cpMvpListLX[numCpMvpCandLX][cpIdx] = mvLXCol} \quad (830)$$

     – The following applies:

$$\text{numCpMvpCandLX = numCpMvpCandLX + 1} \quad (831)$$

9. When numCpMvpCandLX is less than 2, the following is repeated until numCpMvpCandLX is equal to 2, with mvZero[0] and mvZero[1] both being equal to 0:

   – For cpIdx = 0..numCpMv − 1, the following applies:

$$\text{cpMvpListLX[numCpMvpCandLX][cpIdx] = mvZero} \quad (832)$$

   – The following applies:

$$\text{numCpMvpCandLX = numCpMvpCandLX + 1} \quad (833)$$

The affine control point motion vector predictors cpMvpLX[cpIdx] with cpIdx = 0..numCpMv − 1 are derived as follows:

$$\text{cpMvpLX[cpIdx] = cpMvpListLX[mvp\_lX\_flag[xCb][yCb]][cpIdx]} \quad (834)$$

8.5.5.8 Derivation process for constructed affine control point motion vector prediction candidates

Inputs to this process are:

– a luma location (xCb, yCb) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– two variables cbWidth and cbHeight specifying the width and the height of the current luma coding block,

– the reference index of the current coding unit refIdxLX, with X being 0 or 1.

Output of this process are:

– the availability flag of the constructed affine control point motion vector prediction candidiates availableConsFlagLX with X being 0 or 1,

– the availability flags availableFlagLX[cpIdx] with cpIdx = 0..2 and X being 0 or 1,

– the constructed affine control point motion vector prediction candidiates cpMvLX[cpIdx] with cpIdx = 0..2 and X being 0 or 1.

The first (top-left) control point motion vector cpMvLX[0] and the availability flag availableFlagLX[0] are derived in the following ordered steps:

1. The sample locations ($xNbB_2$, $yNbB_2$), ($xNbB_3$, $yNbB_3$) and ($xNbA_2$, $yNbA_2$) are set equal to (xCb − 1, yCb − 1), (xCb, yCb − 1) and (xCb − 1, yCb), respectively.

2. The availability flag availableFlagLX[0] is set equal to 0 and both components of cpMvLX[0] are set equal to 0.

3. The following applies for (xNbTL, yNbTL) with TL being replaced by B_2, B_3, and A_2:

   – The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xCb, yCb), the luma location (xNbY, yNbY) set equal to (xNbTL, yNbTL), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the coding block availability flag availableTL.

   – When availableTL is equal to TRUE and availableFlagLX[0] is equal to 0, the following applies:

     – If PredFlagLX[xNbTL][yNbTL] is equal to 1, and DiffPicOrderCnt(RefPicList[X][RefIdxLX[xNbTL][yNbTL]], RefPicList[X][refIdxLX]) is equal to 0, availableFlagLX[0] is set equal to 1 and the following assignments are made:

$$\text{cpMvLX[0] = MvLX[xNbTL][yNbTL]} \tag{835}$$

     – Otherwise, when PredFlagLY[xNbTL][yNbTL] (with Y = !X) is equal to 1 and DiffPicOrderCnt(RefPicList[Y][RefIdxLY[xNbTL][yNbTL]], RefPicList[X][refIdxLX]) is equal to 0, availableFlagLX[0] is set equal to 1 and the following assignments are made:

$$\text{cpMvLX[0] = MvLY[xNbTL][yNbTL]} \tag{836}$$

     – When availableFlagLX[0] is equal to 1, the rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to cpMvLX[0], rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded cpMvLX[0] as output.

The second (top-right) control point motion vector cpMvLX[1] and the availability flag availableFlagLX[1] are derived in the following ordered steps:

1. The sample locations ($xNbB_1$, $yNbB_1$) and ($xNbB_0$, $yNbB_0$) are set equal to (xCb + cbWidth − 1, yCb − 1) and (xCb + cbWidth, yCb − 1), respectively.

2. The availability flag availableFlagLX[1] is set equal to 0 and both components of cpMvLX[1] are set equal to 0.

3. The following applies for (xNbTR, yNbTR) with TR being replaced by B_1 and B_0:

   – The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xCb, yCb), the luma location (xNbY, yNbY) set equal to (xNbTR, yNbTR), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the coding block availability flag availableTR.

   – When availableTR is equal to TRUE and availableFlagLX[1] is equal to 0, the following applies:

     – If PredFlagLX[xNbTR][yNbTR] is equal to 1, and DiffPicOrderCnt(RefPicList[X][RefIdxLX[xNbTR][yNbTR]], RefPicList[X][refIdxLX]) is equal to 0, availableFlagLX[1] is set equal to 1 and the following assignments are made:

$$\text{cpMvLX[1] = MvLX[xNbTR][yNbTR]} \tag{837}$$

     – Otherwise, when PredFlagLY[xNbTR][yNbTR] (with Y = !X) is equal to 1 and DiffPicOrderCnt(RefPicList[Y][RefIdxLY[xNbTR][yNbTR]], RefPicList[X][refIdxLX]) is equal to 0, availableFlagLX[1] is set equal to 1 and the following assignments are made:

$$\text{cpMvLX[1] = MvLY[xNbTR][yNbTR]} \tag{838}$$

     – When availableFlagLX[1] is equal to 1, the rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to cpMvLX[1], rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded cpMvLX[1] as output.

The third (bottom-left) control point motion vector cpMvLX[2] and the availability flag availableFlagLX[2] are derived in the following ordered steps:

1. The sample locations ($xNbA_1$, $yNbA_1$) and ($xNbA_0$, $yNbA_0$) are set equal to (xCb − 1, yCb + cbHeight − 1) and (xCb − 1, yCb + cbHeight), respectively.

2. The availability flag availableFlagLX[2] is set equal to 0 and both components of cpMvLX[2] are set equal to 0.

3. The following applies for (xNbBL, yNbBL) with BL being replaced by A_1 and A_0:

   – The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xCb, yCb), the luma location (xNbY, yNbY) set equal to (xNbBL, yNbBL), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the coding block availability flag availableBL.

– When availableBL is equal to TRUE and availableFlagLX[2] is equal to 0, the following applies:

  – If PredFlagLX[xNbBL][yNbBL] is equal to 1, and DiffPicOrderCnt(RefPicList[X][RefIdxLX[xNbBL][yNbBL]], RefPicList[X][refIdxLX]) is equal to 0, availableFlagLX[2] is set equal to 1 and the following assignments are made:

$$\text{cpMvLX[2] = MvLX[xNbBL][yNbBL]} \tag{839}$$

  – Otherwise, when PredFlagLY[xNbBL][yNbBL] (with Y = !X) is equal to 1 and DiffPicOrderCnt(RefPicList[Y][RefIdxLY[xNbBL][yNbBL]], RefPicList[X][refIdxLX]) is equal to 0, availableFlagLX[2] is set equal to 1 and the following assignments are made:

$$\text{cpMvLX[2] = MvLY[xNbBL][yNbBL]} \tag{840}$$

  – When availableFlagLX[2] is equal to 1, the rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to cpMvLX[2], rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded cpMvLX[2] as output.

The variable availableConsFlagLX is derived as follows:

– If availableFlagLX[0] is equal to 1 and availableFlagLX[1] is equal to 1 and availableFlagLX[2] is equal to 1, availableConsFlagLX is set equal to 1.

– Otherwise, if availableFlagLX[0] is equal to 1, and availableFlagLX[1] is equal to 1, and MotionModelIdc[xCb][yCb] is equal to 1, availableConsFlagLX is set equal to 1.

– Otherwise, availableConsFlagLX is set equal to 0.

8.5.5.9 Derivation process for motion vector arrays from affine control point motion vectors

Inputs to this process are:

– a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– two variables cbWidth and cbHeight specifying the width and the height of the luma coding block,

– the number of control point motion vectors numCpMv,

– the control point motion vectors cpMvLX[cpIdx], with cpIdx = 0..numCpMv − 1 and X being 0 or 1,

– the prediction list utilization flags predFlagL0 and predFlagL1,

– the reference index refIdxLX and X being 0 or 1,

– the number of luma subblocks in horizontal direction numSbX and in vertical direction numSbY.

Outputs of this process are:

– the luma subblock motion vector array mvLX[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1 and X being 0 or 1,

– the chroma subblock motion vector array mvCLX[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1 and X being 0 or 1,

– the prediction refinement utilization flag cbProfFlagLX and X being 0 or 1,

– the motion vector difference array diffMvLX[xIdx][yIdx] with xIdx = 0..cbWidth / numSbX − 1, yIdx = 0..cbHeight / numSbY − 1 and X being 0 or 1.

The following applies for cpIdx = 0..numCpMv − 1, x = xCb..xCb + cbWidth − 1, and y = yCb..yCb + cbHeight − 1:

$$\text{CpMvLX[x][y][cpIdx] = cpMvLX[cpIdx]} \tag{841}$$

The variables log2CbW and log2CbH are derived as follows:

$$\text{log2CbW = Log2(cbWidth)} \tag{842}$$

$$\text{log2CbH = Log2(cbHeight)} \tag{843}$$

The variables mvScaleHor, mvScaleVer, dHorX and dVerX are derived as follows:

$$\text{mvScaleHor = cpMvLX[0][0]} \ll 7 \tag{844}$$

$$\text{mvScaleVer = cpMvLX[0][1]} \ll 7 \tag{845}$$

$$dHorX = (cpMvLX[1][0] - cpMvLX[0][0]) << (7 - log2CbW) \quad (846)$$

$$dVerX = (cpMvLX[1][1] - cpMvLX[0][1]) << (7 - log2CbW) \quad (847)$$

The variables dHorY and dVerY are derived as follows:

– If numCpMv is equal to 3, the following applies:

$$dHorY = (cpMvLX[2][0] - cpMvLX[0][0]) << (7 - log2CbH) \quad (848)$$

$$dVerY = (cpMvLX[2][1] - cpMvLX[0][1]) << (7 - log2CbH) \quad (849)$$

– Otherwise (numCpMv is equal to 2), the following applies:

$$dHorY = -dVerX \quad (850)$$

$$dVerY = dHorX \quad (851)$$

The variable fallbackModeTriggered is set equal to 1 and modified as follows:

– The variables $bxWX_4$, $bxHX_4$, $bxWX_h$, $bxHX_h$, $bxWX_v$and $bxHX_v$ are derived as follows:

$$maxW_4 = Max(0, Max(4 * (2048 + dHorX), Max(4*dHorY, 4 * (2048 + dHorX) + 4 * dHorY))) \quad (852)$$

$$minW_4 = Min(0, Min(4 * (2048 + dHorX), Min(4*dHorY, 4 * (2048 + dHorX) + 4 * dHorY))) \quad (853)$$

$$maxH_4 = Max(0, Max(4 * dVerX, Max(4* (2048 + dVerY), 4 * dVerX + 4 * (2048 + dVerY)))) \quad (854)$$

$$minH_4 = Min(0, Min(4 * dVerX, Min(4* (2048 + dVerY), 4 * dVerX + 4 * (2048 + dVerY)))) \quad (855)$$

$$bxWX_4 = ((maxW_4 - minW_4) >> 11) + 9 \quad (856)$$

$$bxHX_4 = ((maxH_4 - minH_4) >> 11) + 9 \quad (857)$$

$$bxWX_h = ((Max(0, 4 * (2048 + dHorX)) - Min(0, 4 * (2048 + dHorX))) >> 11) + 9 \quad (858)$$

$$bxHX_h = ((Max(0, 4 * dVerX) - Min(0, 4 * dVerX)) >> 11) + 9 \quad (859)$$

$$bxWX_v = ((Max(0, 4 * dHorY) - Min(0, 4 * dHorY)) >> 11) + 9 \quad (860)$$

$$bxHX_v = ((Max(0, 4 * (2048 + dVerY)) - Min(0, 4 * (2048 + dVerY))) >> 11) + 9 \quad (861)$$

– If both predFlagL0 and predFlagL1 are equal to 1, and $bxWX_4 * bxHX_4$ is less than or equal to 225, fallbackModeTriggered is set equal to 0.

– Otherwise, if either predFlagL0 or predFlagL1 is equal to 0, $bxWX_h * bxHX_h$ is less than or equal to 165, and $bxWX_v * bxHX_v$ is less than or equal to 165, fallbackModeTriggered is set equal to 0.

For xSbIdx = 0..numSbX − 1 and ySbIdx = 0..numSbY − 1, the following applies:

– The variables xPosCb and yPosCb are derived as follows:

  – If fallbackModeTriggered is equal to 1, the following applies:

$$xPosCb = (cbWidth >> 1) \quad (862)$$

$$yPosCb = (cbHeight >> 1) \quad (863)$$

  – Otherwise (fallbackModeTriggered is equal to 0), the following applies:

$$xPosCb = 2 + (xSbIdx << 2) \quad (864)$$

$$yPosCb = 2 + (ySbIdx << 2) \quad (865)$$

– The luma motion vector mvLX[xSbIdx][ySbIdx] is derived as follows :

$$mvLX[xSbIdx][ySbIdx][0] = (mvScaleHor + dHorX * xPosCb + dHorY * yPosCb) \quad (866)$$

$$mvLX[xSbIdx][ySbIdx][1] = (mvScaleVer + dVerX * xPosCb + dVerY * yPosCb) \quad (867)$$

– The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to mvLX[xSbIdx][ySbIdx], rightShift set equal to 7, and leftShift set equal to 0 as inputs and the rounded mvLX[xSbIdx][ySbIdx] as output.

– The motion vectors mvLX[xSbIdx][ySbIdx] are clipped as follows:

$$mvLX[\ xSbIdx\ ][\ ySbIdx\ ][\ 0\ ] = Clip3(\ -2^{17},\ 2^{17} - 1,\ mvLX[\ xSbIdx\ ][\ ySbIdx\ ][\ 0\ ]\ ) \quad (868)$$

$$mvLX[\ xSbIdx\ ][\ ySbIdx\ ][\ 1\ ] = Clip3(\ -2^{17},\ 2^{17} - 1,\ mvLX[\ xSbIdx\ ][\ ySbIdx\ ][\ 1\ ]\ ) \quad (869)$$

For xSbIdx = 0..numSbX − 1 and ySbIdx = 0..numSbY − 1, the following applies:

– The average luma motion vector mvAvgLX is derived as follows:

 – If both SubWidthC and SubHeightC are equal to 1, the following applies:

$$mvAvgLX = mvLX[\ xSbIdx\ ][\ ySbIdx\ ] \quad (870)$$

 – Otherwise, the following applies:

$$\begin{aligned} mvAvgLX = {} & mvLX[\ (\ xSbIdx\ >>\ (\ SubWidthC - 1\ )\ <<\ (\ SubWidthC - 1\ )\ )\ ] \\ & [\ (ySbIdx\ >>\ (\ SubHeightC - 1\ )\ <<\ (\ SubHeightC - 1\ )\ )\ ] + \\ & mvLX[\ (\ xSbIdx\ >>\ (SubWidthC - 1\ )\ <<\ (\ SubWidthC - 1\ )\ ) + (\ SubWidthC - 1\ )\ ] \\ & [\ (\ ySbIdx\ >>\ (\ SubHeightC - 1\ )\ <<\ (\ SubHeightC - 1\ )\ ) + (\ SubHeightC - 1\ )\ ] \end{aligned} \quad (871)$$

$$mvAvgLX[\ 0\ ] = (\ mvAvgLX[\ 0\ ] + 1 - (\ mvAvgLX[\ 0\ ]\ >=\ 0\ )\ )\ >>\ 1 \quad (872)$$

$$mvAvgLX[\ 1\ ] = (\ mvAvgLX[\ 1\ ] + 1 - (\ mvAvgLX[\ 1\ ]\ >=\ 0\ )\ )\ >>\ 1 \quad (873)$$

– The derivation process for chroma motion vectors in clause 8.5.2.13 is invoked with mvAvgLX as input, and the chroma motion vector mvCLX[xSbIdx][ySbIdx] as output.

The variable cbProfFlagLX is derived as follows:

– If one or more of the following conditions are true, cbProfFlagLX is set equal to FALSE:

 – ph_prof_disabled_flag is equal to 1.

 – fallbackModeTriggered is equal to 1.

 – numCpMv is equal to 2 and cpMvLX[1][0] is equal to cpMvLX[0][0] and cpMvLX[1][1] is equal to cpMvLX[0][1].

 – numCpMv is equal to 3 and cpMvLX[1][0] is equal to cpMvLX[0][0] and cpMvLX[1][1] is equal to cpMvLX[0][1] and cpMvLX[2][0] is equal to cpMvLX[0][0] and cpMvLX[2][1] is equal to cpMvLX[0][1].

 – RprConstraintsActiveFlag[X][refIdxLX] is equal to 1.

– Otherwise, cbProfFlagLX is set equal to TRUE.

When cbProfFlagLX is equal to 1, the motion vector difference array diffMvLX is derived as follows:

– The variables sbWidth and sbHeight, dmvLimit, posOffsetX and posOffsetY are derived as follows:

$$sbWidth = cbWidth\ /\ numSbX \quad (874)$$

$$sbHeight = cbHeight\ /\ numSbY \quad (875)$$

$$dmvLimit = 1\ <<\ 5 \quad (876)$$

$$posOffsetX = 6 * dHorX + 6 * dHorY \quad (877)$$

$$posOffsetY = 6 * dVerX + 6 * dVerY \quad (878)$$

– For x = 0..sbWidth − 1 and y = 0..sbHeight − 1, the following applies:

$$diffMvLX[\ x\ ][\ y\ ][\ 0\ ] = x * (\ dHorX\ <<\ 2\ ) + y * (\ dHorY\ <<\ 2\ ) - posOffsetX \quad (879)$$

$$diffMvLX[\ x\ ][\ y\ ][\ 1\ ] = x * (\ dVerX\ <<\ 2\ ) + y * (\ dVerY\ <<\ 2\ ) - posOffsetY \quad (880)$$

 – The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to diffMvLX[x][y], rightShift set equal to 8, and leftShift set equal to 0 as inputs and the rounded diffMvLX[x][y] as output.

 – The value of diffMvLX[x][y][i] is clipped as follows for i = 0..1:

$$diffMvLX[\ x\ ][\ y\ ][\ i\ ] = Clip3(\ -dmvLimit + 1,\ dmvLimit - 1,\ diffMvLX[\ x\ ][\ y\ ][\ i\ ])\quad (881)$$

8.5.6 Decoding process for inter blocks

8.5.6.1 General

This process is invoked when decoding a coding unit coded in inter prediction mode.

Inputs to this process are:

- a luma location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left luma sample of the current picture,
- a variable cbWidth specifying the width of the current coding block in luma samples,
- a variable cbHeight specifying the height of the current coding block in luma samples,
- variables numSbX and numSbY specifying the number of luma subblocks in horizontal and vertical direction,
- the motion vectors mvL0[xSbIdx][ySbIdx] and mvL1[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, and ySbIdx = 0..numSbY − 1,
- the refined motion vectors refMvL0[xSbIdx][ySbIdx] and refMvL1[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, and ySbIdx = 0..numSbY − 1,
- the reference indices refIdxL0 and refIdxL1,
- the prediction list utilization flags predFlagL0[xSbIdx][ySbIdx] and predFlagL1[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, and ySbIdx = 0..numSbY − 1,
- the half sample interpolation filter index hpelIfIdx,
- the bi-prediction weight index bcwIdx,
- the mimimum sum of absolute difference values in decoder-side motion vector refinement process dmvrSad[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX − 1, and ySbIdx = 0..numSbY − 1,
- the decoder-side motion vector refinement flag dmvrFlag,
- a variable cIdx specifying the colour component index of the current block,
- the prediction refinement utilization flag cbProfFlagL0 and cbProfFlagL1,
- a motion vector difference array diffMvL0[xIdx][yIdx] and diffMvL1[xIdx][yIdx] with xIdx = 0..cbWidth / numSbX − 1, and yIdx = 0..cbHeight / numSbY − 1.

Outputs of this process are:

- an array predSamples of prediction samples.

Let $predSamplesL0_L$, $predSamplesL1_L$ and $predSamplesIntra_L$ be (cbWidth)x(cbHeight) arrays of predicted luma sample values and, $predSamplesL0_{Cb}$, $predSamplesL1_{Cb}$, $predSamplesL0_{Cr}$ and $predSamplesL1_{Cr}$, $predSamplesIntra_{Cb,}$ and $predSamplesIntra_{Cr}$ be (cbWidth / SubWidthC)x(cbHeight / SubHeightC) arrays of predicted chroma sample values.

- The variable currPic specifies the current picture and the variable bdofFlag is derived as follows:
  - If all of the following conditions are true, bdofFlag is set equal to TRUE.
    - ph_bdof_disabled_flag is equal to 0.
    - predFlagL0[0][0] and predFlagL1[0][0] are both equal to 1.
    - DiffPicOrderCnt(currPic, RefPicList[0][refIdxL0]) is equal to DiffPicOrderCnt(RefPicList[1][refIdxL1], currPic).
    - RefPicList[0][refIdxL0] is an STRP and RefPicList[1][refIdxL1] is an STRP.
    - MotionModelIdc[xCb][yCb] is equal to 0.
    - merge_subblock_flag[xCb][yCb] is equal to 0.
    - sym_mvd_flag[xCb][yCb] is equal to 0.
    - ciip_flag[xCb][yCb] is equal to 0.
    - bcwIdx is equal to 0.
    - luma_weight_l0_flag[refIdxL0] and luma_weight_l1_flag[refIdxL1] are both equal to 0.

– chroma_weight_l0_flag[refIdxL0] and chroma_weight_l1_flag[refIdxL1] are both equal to 0.

– cbWidth is greater than or equal to 8.

– cbHeight is greater than or equal to 8.

– cbHeight * cbWidth is greater than or equal to 128.

– RprConstraintsActiveFlag[0][refIdxL0] is equal to 0 and RprConstraintsActiveFlag[1][refIdxL1] is equal to 0.

– cIdx is equal to 0.

– Otherwise, bdofFlag is set equal to FALSE.

– If numSbY is equal to 1 and numSbX is equal to 1, the following applies:

– When bdofFlag is equal to TRUE, the variables numSbY, numSbX are modified as follows:

numSbX = (cbWidth > 16) ? (cbWidth >> 4) : 1 (882)

numSbY = (cbHeight > 16) ? (cbHeight >> 4) : 1 (883)

– For X = 0..1, xSbIdx = 0..numSbX − 1 and ySbIdx = 0..numSbY − 1, the following applies:

– predFlagLX[xSbIdx][ySbIdx] is set equal to predFlagLX[0][0].

– refMvLX[xSbIdx][ySbIdx] is set equal to refMvLX[0][0].

– mvLX[xSbIdx][ySbIdx] is set equal to mvLX[0][0].

The width and the height of the current coding sublock sbWidth, sbHeight in luma samples are derived as follows:

sbWidth = cbWidth / numSbX (884)

sbHeight = cbHeight / numSbY (885)

For each subblock at subblock index (xSbIdx, ySbIdx) with xSbIdx = 0..numSbX − 1, and ySbIdx = 0..numSbY − 1, the following applies:

– The luma location (xSb, ySb) specifying the top-left sample of the current subblock relative to the top-left luma sample of the current picture is derived as follows:

(xSb, ySb) = (xCb + xSbIdx * sbWidth, yCb + ySbIdx * sbHeight) (886)

– The variable sbBdofFlag is set equal to FALSE.

– When bdofFlag is equal to TRUE, the variable sbBdofFlag is further modifed as follows:

– If dmvrFlag is equal to 1 and the variable dmvrSad[xSbIdx][ySbIdx] is less than (2 * sbWidth * sbHeight), the variable sbBdofFlag is set equal to FALSE.

– Otherwise, the variable sbBdofFlag is set equal to TRUE.

– For X = 0..1, when predFlagLX[xSbIdx][ySbIdx] is equal to 1, the following applies:

– The reference picture consisting of an ordered two-dimensional array $refPicLX_L$ of luma samples and two ordered two-dimensional arrays $refPicLX_{Cb}$ and $refPicLX_{Cr}$ of chroma samples is derived by invoking the process specified in clause 8.5.6.2 with X and refIdxLX as inputs.

– The motion vector offset mvOffset is set equal to refMvLX[xSbIdx][xSbIdx] − mvLX[xSbIdx][ySbIdx].

– If cIdx is equal to 0, the following applies:

– The array $predSamplesLX_L$ is derived by invoking the fractional sample interpolation process specified in clause 8.5.6.3 with the luma location (xSb, ySb), the subblock width sbWidth, the subblock height sbHeight in luma samples, the luma motion vector offset mvOffset, the refined luma motion vector refMvLX[xSbIdx][xSbIdx], the reference array $refPicLX_L$, the variable bdofFlag set equal to sbBdofFlag, cbProfFlagLX, dmvrFlag, hpelIfIdx, cIdx, RprConstraintsActiveFlag[X][refIdxLX], and RefPicScale[X][refIdxLX] as inputs.

– When cbProfFlagLX is equal to 1, the prediction refinement with optical flow process specified in clause 8.5.6.4 is invoked with sbWidth, sbHeight, the (sbWidth + 2)x(sbHeight + 2) array predSamplesLX$_L$ and the motion vector difference array diffMvLX[xIdx][yIdx] with xIdx = 0..cbWidth / numSbX − 1, and yIdx = 0..cbHeight / numSbY − 1 as inputs and the refined (sbWidth)x(sbHeight) array predSamplesLX$_L$ as output.

– Otherwise, if MotionModelIdc[xCb][yCb] is equal to 0, or xSbIdx % SubWidthC and ySbIdx % SubHeightC are both equal to 0, the following applies:

– If cIdx is equal to 1, the following applies:

– The array predSamplesLX$_{Cb}$ is derived by invoking the fractional sample interpolation process specified in clause 8.5.6.3 with the luma location (xSb, ySb), the subblock width sbWidth set equal to (MotionModelIdc[xCb][yCb] ? sbWidth : sbWidth / SubWidthC), the subblock height sbHeight set equal to (MotionModelIdc[xCb][yCb] ? sbHeight : sbHeight / SubHeightC), the chroma motion vector offset mvOffset, the refined chroma motion vector refMvLX[xSbIdx][ySbIdx], the reference array refPicLX$_{Cb}$, bdofFlag, cbProfFlagLX, dmvrFlag, hpelIfIdx, cIdx, RprConstraintsActiveFlag[X][refIdxLX], and RefPicScale[X][refIdxLX] as inputs.

– Otherwise (cIdx is equal to 2), the following applies:

– The array predSamplesLX$_{Cr}$ is derived by invoking the fractional sample interpolation process specified in clause 8.5.6.3 with the luma location (xSb, ySb), the subblock width sbWidth set equal to (MotionModelIdc[xCb][yCb] ? sbWidth : sbWidth / SubWidthC), the subblock height sbHeight set equal to (MotionModelIdc[xCb][yCb] ? sbHeight : sbHeight / SubHeightC), the chroma motion vector offset mvOffset, the refined chroma motion vector refMvLX[xSbIdx][xSbIdx], the reference array refPicLX$_{Cr}$, bdofFlag, cbProfFlagLX, dmvrFlag, hpelIfIdx, cIdx, RprConstraintsActiveFlag[X][refIdxLX], and RefPicScale[X][refIdxLX] as inputs.

– The array predSamples of prediction samples is derived as follows:

– The sample location (xSbInCb, ySbInCb) is set equal to (xSbIdx * sbWidth, ySbIdx * sbHeight).

– If cIdx is equal to 0, the prediction samples inside the current luma subblock, predSamples[x_L + xSbInCb][y_L + ySbInCb] with x_L = 0..sbWidth − 1 and y_L = 0..sbHeight − 1, are derived as follows:

– If sbBdofFlag is equal to TRUE, the bi-directional optical flow sample prediction process as specified in clause 8.5.6.5 is invoked with nCbW set equal to the luma subblock width sbWidth, nCbH set equal to the luma subblock height sbHeight and the sample arrays predSamplesL0$_L$ and predSamplesL1$_L$ as inputs, and predSamples[x_L + xSbInCb][y_L + ySbInCb] as outputs.

– Otherwise (sbBdofFlag is equal to FALSE), the weighted sample prediction process as specified in clause 8.5.6.6 is invoked with the luma location (xCb, yCb), the luma subblock width sbWidth, the luma subblock height sbHeight and the sample arrays predSamplesL0$_L$ and predSamplesL1$_L$, and the variables predFlagL0[xSbIdx][ySbIdx], predFlagL1[xSbIdx][ySbIdx], refIdxL0, refIdxL1, bcwIdx, dmvrFlag, and cIdx as inputs, and predSamples[x_L + xSbInCb][y_L + ySbInCb] as outputs.

– Otherwise, if MotionModelIdc[xCb][yCb] is equal to 0, or xSbIdx % SubWidthC and ySbIdx % SubHeightC are both equal to 0, the following applies:

– If cIdx is equal to 1, the prediction samples inside the current chroma component Cb subblock, predSamples[x_C + xSbInCb / SubWidthC][y_C + ySbInCb / SubHeightC] with x_C = 0..(MotionModelIdc[xCb][yCb] ? sbWidth : sbWidth / SubWidthC) − 1 and y_C = 0..(MotionModelIdc[xCb][yCb] ? sbHeight : sbHeight / SubHeightC) − 1, are derived by invoking the weighted sample prediction process specified in clause 8.5.6.6 with the luma location (xCb, yCb), nCbW set equal to (MotionModelIdc[xCb][yCb] ? sbWidth : sbWidth / SubWidthC), nCbH set equal to (MotionModelIdc[xCb][yCb] ? sbHeight : sbHeight / SubHeightC), the sample arrays predSamplesL0$_{Cb}$ and predSamplesL1$_{Cb}$, and the variables predFlagL0[xSbIdx][ySbIdx], predFlagL1[xSbIdx][ySbIdx], refIdxL0, refIdxL1, bcwIdx, dmvrFlag, and cIdx as inputs.

– Otherwise (cIdx is equal to 2), the prediction samples inside the current chroma component Cr subblock, predSamples[x_C + xSbInCb / SubWidthC][y_C + ySbInCb / SubHeightC] with x_C = 0..(MotionModelIdc[xCb][yCb] ? sbWidth : sbWidth / SubWidthC) − 1 and y_C = 0..(MotionModelIdc[xCb][yCb] ? sbHeight : sbHeight / SubHeightC) − 1, are derived by invoking the weighted sample prediction process specified in clause 8.5.6.6 with the luma location (xCb, yCb), nCbW set equal to (MotionModelIdc[xCb][yCb] ? sbWidth : sbWidth / SubWidthC), nCbH set equal to (MotionModelIdc[xCb][yCb] ? sbHeight : sbHeight / SubHeightC), the sample arrays $predSamplesL0_{Cr}$ and $predSamplesL1_{Cr}$, and the variables predFlagL0[xSbIdx][ySbIdx], predFlagL1[xSbIdx][ySbIdx], refIdxL0, refIdxL1, bcwIdx, dmvrFlag, and cIdx as inputs.

– When cIdx is equal to 0, the following assignments are made for x = 0..sbWidth − 1 and y = 0..sbHeight − 1:

MvL0[xSb + x][ySb + y] = mvL0[xSbIdx][ySbIdx] (887)

MvL1[xSb + x][ySb + y] = mvL1[xSbIdx][ySbIdx] (888)

MvDmvrL0[xSb + x][ySb + y] = refMvL0[xSbIdx][ySbIdx] (889)

MvDmvrL1[xSb + x][ySb + y] = refMvL1[xSbIdx][ySbIdx] (890)

RefIdxL0[xSb + x][ySb + y] = refIdxL0 (891)

RefIdxL1[xSb + x][ySb + y] = refIdxL1 (892)

PredFlagL0[xSb + x][ySb + y] = predFlagL0[xSbIdx][ySbIdx] (893)

PredFlagL1[xSb + x][ySb + y] = predFlagL1[xSbIdx][ySbIdx] (894)

HpelIfIdx[xSb + x][ySb + y] = hpelIfIdx (895)

BcwIdx[xSb + x][ySb + y] = bcwIdx (896)

When ciip_flag[xCb][yCb] is equal to 1, the array predSamples of prediction samples is modified as follows:

– If cIdx is equal to 0, the following applies:

  – The general intra sample prediction process as specified in clause 8.4.5.2.6 is invoked with the location (xTbCmp, yTbCmp) set equal to (xCb, yCb), the intra prediction mode predModeIntra set equal to INTRA_PLANAR, the transform block width nTbW and height nTbH set equal to cbWidth and cbHeight, the coding block width nCbW and height nCbH set equal to cbWidth and cbHeight, and the variable cIdx as inputs, and the output is assigned to the (cbWidth)x(cbHeight) array $predSamplesIntra_L$.

  – The weighted sample prediction process for combined merge and intra prediction as specified in clause 8.5.6.7 is invoked with the location (xTbCmp, yTbCmp) set equal to (xCb, yCb), the coding block width cbWidth, the coding block height cbHeight, the sample arrays predSamplesInter and predSamplesIntra set equal to predSamples and $predSamplesIntra_L$, respectively, and the colour component index cIdx as inputs, and the output is assigned to the (cbWidth)x(cbHeight) array predSamples.

– Otherwise, if cIdx is equal to 1 and cbWidth / SubWidthC is greater than or equal to 4, the following applies:

  – The general intra sample prediction process as specified in clause 8.4.5.2.6 is invoked with the location (xTbCmp, yTbCmp) set equal to (xCb / SubWidthC, yCb / SubHeightC), the intra prediction mode predModeIntra set equal to INTRA_PLANAR, the transform block width nTbW and height nTbH set equal to cbWidth / SubWidthC and cbHeight / SubHeightC, the coding block width nCbW and height nCbH set equal to cbWidth / SubWidthC and cbHeight / SubHeightC, and the variable cIdx as inputs, and the output is assigned to the (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array $predSamplesIntra_{Cb}$.

  – The weighted sample prediction process for combined merge and intra prediction as specified in clause 8.5.6.7 is invoked with the location (xTbCmp, yTbCmp) set equal to (xCb, yCb), the coding block width cbWidth / SubWidthC, the coding block height cbHeight / SubHeightC, the sample arrays predSamplesInter and predSamplesIntra set equal to $predSamples_{Cb}$ and $predSamplesIntra_{Cb}$, respectively, and the colour component index cIdx as inputs, and the output is assigned to the (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array predSamples.

– Otherwise, if cIdx is equal to 2 and cbWidth / SubWidthC is greater than or equal to 4, the following applies:

  – The general intra sample prediction process as specified in clause 8.4.5.2.6 is invoked with the location (xTbCmp, yTbCmp) set equal to (xCb / SubWidthC, yCb / SubHeightC), the intra prediction mode predModeIntra set equal to INTRA_PLANAR, the transform block width nTbW and height nTbH set equal to cbWidth / SubWidthC and cbHeight / SubHeightC, the coding block width nCbW and height nCbH set equal to cbWidth / SubWidthC and cbHeight / SubHeightC, and the variable cIdx as inputs, and the output is assigned to the (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array $predSamplesIntra_{Cr}$.

– The weighted sample prediction process for combined merge and intra prediction as specified in clause 8.5.6.7 is invoked with the location (xTbCmp, yTbCmp) set equal to (xCb, yCb), the coding block width cbWidth / SubWidthC, the coding block height cbHeight / SubHeightC, the sample arrays predSamplesInter and predSamplesIntra set equal to $predSamples_{Cr}$ and $predSamplesIntra_{Cr}$, respectively, and the colour component index cIdx as inputs, and the output is assigned to the (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array predSamples.

8.5.6.2 Reference picture selection process

Inputs to this process are:

– a value X representing a reference list being equal to either 0 or 1,

– a reference index refIdxLX.

Output of this process is a reference picture consisting of a two-dimensional array of luma samples $refPicLX_L$ and two two-dimensional arrays of chroma samples $refPicLX_{Cb}$ and $refPicLX_{Cr}$.

The output reference picture RefPicList[X][refIdxLX], where X is the value of X that this process is invoked for, consists of a pps_pic_width_in_luma_samples by pps_pic_height_in_luma_samples array of luma samples $refPicLX_L$ and two PicWidthInSamplesC by PicHeightInSamplesC arrays of chroma samples $refPicLX_{Cb}$ and $refPicLX_{Cr}$.

The reference picture sample arrays $refPicLX_L$, $refPicLX_{Cb}$ and $refPicLX_{Cr}$ correspond to decoded sample arrays S_L, S_{Cb} and S_{Cr} derived in clause 8.8 for a previously-decoded picture.

8.5.6.3 Fractional sample interpolation process

8.5.6.3.1 General

Inputs to this process are:

– a luma location (xSb, ySb) specifying the top-left sample of the current subblock relative to the top-left luma sample of the current picture,

– a variable sbWidth specifying the width of the current subblock,

– a variable sbHeight specifying the height of the current subblock,

– a motion vector offset mvOffset,

– a refined motion vector refMvLX,

– the selected reference picture sample array refPicLX,

– the bi-directional optical flow flag bdofFlag,

– the prediction refinement utilization flag cbProfFlagLX,

– the decoder-side motion vector refinement flag dmvrFlag,

– the half sample interpolation filter index hpelIfIdx,

– a variable cIdx specifying the colour component index of the current block,

– a variable refPicIsScaled indicating whether the selected reference picture requires scaling,

– a list of two scaling ratios, horizontal and vertical, scalingRatio.

Outputs of this process are:

– an (sbWidth + brdExtSize)x(sbHeight + brdExtSize) array predSamplesLX of prediction sample values.

The border extension size brdExtSize is derived as follows:

brdExtSize = (bdofFlag || cbProfFlagLX) ? 2 : 0 (897)

The variable refWraparoundEnabledFlag is set equal to (pps_ref_wraparound_enabled_flag && !refPicIsScaled).

The variable fRefLeftOffset is set equal to ((SubWidthC * pps_scaling_win_left_offset) << 10), where pps_scaling_win_left_offset is the pps_scaling_win_left_offset for the reference picture.

The variable fRefTopOffset is set equal to ((SubHeightC * pps_scaling_win_top_offset) << 10), where pps_scaling_win_top_offset is the pps_scaling_win_top_offset for the reference picture.

The (sbWidth + brdExtSize)x(sbHeight + brdExtSize) array predSamplesLX of prediction sample values is derived as follows:

– The motion vector mvLX is set equal to (refMvLX − mvOffset).

– If cIdx is equal to 0, the following applies:

  – Let ($xInt_L$, $yInt_L$) be a luma location given in full-sample units and ($xFrac_L$, $yFrac_L$) be an offset given in 1/16-sample units. These variables are used only in this clause for specifying fractional-sample locations inside the reference sample arrays refPicLX.

  – The top-left coordinate of the bounding block for reference sample padding ($xSbInt_L$, $ySbInt_L$) is set equal to (xSb + (mvLX[0] >> 4), ySb + (mvLX[1] >> 4)).

  – For each luma sample location (x_L = 0..sbWidth − 1 + brdExtSize, y_L = 0..sbHeight − 1 + brdExtSize) inside the prediction luma sample array predSamplesLX, the corresponding prediction luma sample value predSamplesLX[x_L][y_L] is derived as follows:

    – Let ($refxSb_L$, $refySb_L$) and ($refx_L$, $refy_L$) be luma locations pointed to by a motion vector (refMvLX[0], refMvLX[1]) given in 1/16-sample units. The variables $refxSb_L$, $refx_L$, $refySb_L$, and $refy_L$ are derived as follows:

$$refxSb_L = (((xSb - (SubWidthC * pps\_scaling\_win\_left\_offset)) << 4) + refMvLX[0]) * scalingRatio[0] \quad (898)$$

$$refx_L = ((Sign(refxSb_L) * ((Abs(refxSb_L) + 128) >> 8) + x_L * ((scalingRatio[0] + 8) >> 4)) + fRefLeftOffset + 32) >> 6 \quad (899)$$

$$refySb_L = (((ySb - (SubHeightC * pps\_scaling\_win\_top\_offset)) << 4) + refMvLX[1]) * scalingRatio[1] \quad (900)$$

$$refy_L = ((Sign(refySb_L) * ((Abs(refySb_L) + 128) >> 8) + y_L * ((scalingRatio[1] + 8) >> 4)) + fRefTopOffset + 32) >> 6 \quad (901)$$

    – The variables $xInt_L$, $yInt_L$, $xFrac_L$ and $yFrac_L$ are derived as follows:

$$xInt_L = refx_L >> 4 \quad (902)$$

$$yInt_L = refy_L >> 4 \quad (903)$$

$$xFrac_L = refx_L \& 15 \quad (904)$$

$$yFrac_L = refy_L \& 15 \quad (905)$$

    – The prediction luma sample value predSamplesLX[x_L][y_L] is derived as follows:

      – If bdofFlag is equal to TRUE or cbProfFlagLX is equal to TRUE, and one or more of the following conditions are true, the prediction luma sample value predSamplesLX[x_L][y_L] is derived by invoking the luma integer sample fetching process as specified in clause 8.5.6.3.3 with ($xInt_L$ + ($xFrac_L$ >> 3) − 1, $yInt_L$ + ($yFrac_L$ >> 3) − 1), refPicLX, and refWraparoundEnabledFlag as inputs:

        – x_L is equal to 0.

        – x_L is equal to sbWidth + 1.

        – y_L is equal to 0.

        – y_L is equal to sbHeight + 1.

      – Otherwise, the prediction luma sample value predSamplesLX[x_L][y_L] is derived by invoking the luma sample 8-tap interpolation filtering process as specified in clause 8.5.6.3.2 with ($xInt_L$ − (brdExtSize > 0 ? 1 : 0), $yInt_L$ − (brdExtSize > 0 ? 1 : 0)), ($xFrac_L$, $yFrac_L$), ($xSbInt_L$, $ySbInt_L$), refPicLX, hpelIfIdx, sbWidth, sbHeight, dmvrFlag, refWraparoundEnabledFlag, scalingRatio[0], scalingRatio[1], and (xSb, ySb) as inputs.

– Otherwise (cIdx 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 1/32 sample units. These variables are used only in this clause for specifying general fractional-sample locations inside the reference sample arrays refPicLX.

  – The top-left coordinate of the bounding block for reference sample padding ($xSbInt_C$, $ySbInt_C$) is set equal to ((xSb / SubWidthC) + (mvLX[0] >> 5), (ySb / SubHeightC) + (mvLX[1] >> 5)).

  – For each chroma sample location (xC = 0..sbWidth − 1, yC = 0.. sbHeight − 1) inside the prediction chroma sample arrays predSamplesLX, the corresponding prediction chroma sample value predSamplesLX[xC][yC] is derived as follows:

    – Let ($refxSb_C$, $refySb_C$) and ($refx_C$, $refy_C$) be chroma locations pointed to by a motion vector (refMvLX[0], refMvLX[1]) given in 1/32-sample units. The variables $refxSb_C$, $refySb_C$, $refx_C$ and $refy_C$ are derived as follows:

$$addX = sps\_chroma\_horizontal\_collocated\_flag\ ?\ 0 : 8 * (scalingRatio[0] - (1 \ll 14)) \tag{906}$$

$$addY = sps\_chroma\_vertical\_collocated\_flag\ ?\ 0 : 8 * (scalingRatio[1] - (1 \ll 14)) \tag{907}$$

$$\begin{aligned} refxSb_C = {} & (((xSb - (SubWidthC * pps\_scaling\_win\_left\_offset)) / SubWidthC \ll 5) + \\ & refMvLX[0]) * scalingRatio[0] + addX \end{aligned} \tag{908}$$

$$\begin{aligned} refx_C = {} & ((Sign(refxSb_C) * ((Abs(refxSb_C) + 256) \gg 9) \\ & + xC * ((scalingRatio[0] + 8) \gg 4)) + fRefLeftOffset / SubWidthC + 16) \gg 5 \end{aligned} \tag{909}$$

$$\begin{aligned} refySb_C = {} & (((ySb - (SubHeightC * pps\_scaling\_win\_top\_offset)) / SubHeightC \ll 5) + \\ & refMvLX[1]) * scalingRatio[1] + addY \end{aligned} \tag{910}$$

$$\begin{aligned} refy_C = {} & ((Sign(refySb_C) * ((Abs(refySb_C) + 256) \gg 9) \\ & + yC * ((scalingRatio[1] + 8) \gg 4)) + fRefTopOffset / SubHeightC + 16) \gg 5 \end{aligned} \tag{911}$$

    – The variables $xInt_C$, $yInt_C$, $xFrac_C$ and $yFrac_C$ are derived as follows:

$$xInt_C = refx_C \gg 5 \tag{912}$$

$$yInt_C = refy_C \gg 5 \tag{913}$$

$$xFrac_C = refx_C \& 31 \tag{914}$$

$$yFrac_C = refy_C \& 31 \tag{915}$$

    – The prediction sample value predSamplesLX[xC][yC] is derived by invoking the process specified in clause 8.5.6.3.4 with ($xInt_C$, $yInt_C$), ($xFrac_C$, $yFrac_C$), ($xSbInt_C$, $ySbInt_C$), sbWidth, sbHeight, refPicLX, dmvrFlag, refWraparoundEnabledFlag, scalingRatio[0], and scalingRatio[1] as inputs.

NOTE – Unlike the process specified in clause 8.4.5.2.14, this process uses both sps_chroma_vertical_collocated_flag and sps_chroma_horizontal_collocated_flag.

8.5.6.3.2 Luma sample interpolation filtering process

Inputs to this process are:

– a luma location in full-sample units ($xInt_L$, $yInt_L$),

– a luma location in fractional-sample units ($xFrac_L$, $yFrac_L$),

– a luma location in full-sample units ($xSbInt_L$, $ySbInt_L$) specifying the top-left sample of the bounding block for reference sample padding relative to the top-left luma sample of the reference picture,

– the luma reference sample array $refPicLX_L$,

– the half sample interpolation filter index hpelIfIdx,

– a variable sbWidth specifying the width of the current subblock,

– a variable sbHeight specifying the height of the current subblock,

– the decoder-side motion vector refinement flag dmvrFlag,

– a variable refWraparoundEnabledFlag indicating whether horizontal wrap-around motion compensation is enabled,

– a fixedpoint representation of the horizontal scaling factor scalingRatio[0],

– a fixedpoint representation of the vertical scaling factor scalingRatio[1],

– a luma location (xSb, ySb) specifying the top-left sample of the current subblock relative to the top-left luma sample of the current picture.

Output of this process is a predicted luma sample value $predSampleLX_L$.

The variables shift1, shift2 and shift3 are derived as follows:

– The variable shift1 is set equal to Min(4, BitDepth − 8), the variable shift2 is set equal to 6 and the variable shift3 is set equal to Max(2, 14 − BitDepth).

– The variable picW is set equal to pps_pic_width_in_luma_samples of the reference picture refPicLX and the variable picH is set equal to pps_pic_height_in_luma_samples of the reference picture refPicLX.

The horizontal and vertical half sample interpolation filter indices hpelHorIfIdx and hpelVerIfIdx are derived as follows:

$$\text{hpelHorIfIdx} = (\text{scalingRatio}[0] == 16384) ? \text{hpelIfIdx} : 0 \tag{916}$$

$$\text{hpelVerIfIdx} = (\text{scalingRatio}[1] == 16384) ? \text{hpelIfIdx} : 0 \tag{917}$$

The horizontal luma interpolation filter coefficients $f_{LH}[p]$ for each 1/16 fractional sample position p equal to $xFrac_L$ or $yFrac_L$ are derived as follows:

– If MotionModelIdc[xSb][ySb] is greater than 0, and sbWidth and sbHeight are both equal to 4, and scalingRatio[0] is greater than 28 672, luma interpolation filter coefficients fLH[p] are specified in Table 32.

– Otherwise, if MotionModelIdc[xSb][ySb] is greater than 0, and sbWidth and sbHeight are both equal to 4, and scalingRatio[0] is greater than 20 480, luma interpolation filter coefficients fLH[p] are specified in Table 31.

– Otherwise, if MotionModelIdc[xSb][ySb] is greater than 0, and sbWidth and sbHeight are both equal to 4, the luma interpolation filter coefficients fLH[p] are specified in Table 30.

– Otherwise, if scalingRatio[0] is greater than 28 672, luma interpolation filter coefficients $f_{LH}[p]$ are specified in Table 29.

– Otherwise, if scalingRatio[0] is greater than 20 480, luma interpolation filter coefficients $f_{LH}[p]$ are specified in Table 28.

– Otherwise, the luma interpolation filter coefficients $f_{LH}[p]$ are specified in Table 27 depending on hpelIfIdx set equal to hpelHorIfIdx.

The vertical luma interpolation filter coefficients $f_{LV}[p]$ for each 1/16 fractional sample position p equal to $yFrac_L$ are derived as follows:

– If MotionModelIdc[xSb][ySb] is greater than 0, and sbWidth and sbHeight are both equal to 4, and scalingRatio[1] is greater than 28 672, the luma interpolation filter coefficients fLV[p] are specified in Table 32.

– Otherwise, if MotionModelIdc[xSb][ySb] is greater than 0, and sbWidth and sbHeight are both equal to 4, and scalingRatio[1] is greater than 20 480, the luma interpolation filter coefficients fLV[p] are specified in Table 31.

– Otherwise, if MotionModelIdc[xSb][ySb] is greater than 0, and sbWidth and sbHeight are both equal to 4, the luma interpolation filter coefficients fLV[p] are specified in Table 30.

– Otherwise, if scalingRatio[1] is greater than 28 672, luma interpolation filter coefficients $f_{LV}[p]$ are specified in Table 29.

– Otherwise, if scalingRatio[1] is greater than 20 480, luma interpolation filter coefficients $f_{LV}[p]$ are specified in Table 28.

– Otherwise, the luma interpolation filter coefficients $f_{LV}[p]$ are specified in Table 27 depending on hpelIfIdx set equal to hpelVerIfIdx.

The luma locations in full-sample units ($xInt_i$, $yInt_i$) are derived as follows for i = 0..7:

$$xInt_i = xInt_L + i - 3 \tag{918}$$

$$\mathrm{yInt}_i = \mathrm{yInt}_L + i - 3 \tag{919}$$

– When dmvrFlag is equal to 1, the following applies:

$$\mathrm{xInt}_i = \mathrm{Clip3}(\ \mathrm{xSbInt}_L - 3, \mathrm{xSbInt}_L + \mathrm{sbWidth} - 1 + 4, \mathrm{xInt}_i\ ) \tag{920}$$

$$\mathrm{yInt}_i = \mathrm{Clip3}(\ \mathrm{ySbInt}_L - 3, \mathrm{ySbInt}_L + \mathrm{sbHeight} - 1 + 4, \mathrm{yInt}_i\ ) \tag{921}$$

– If sps_subpic_treated_as_pic_flag[CurrSubpicIdx] is equal to 1 and sps_num_subpics_minus1 for the reference picture refPicLX is greater than 0, the following applies:

$$\begin{aligned}\mathrm{xInt}_i = \mathrm{Clip3}(\ &\mathrm{SubpicLeftBoundaryPos}, \mathrm{SubpicRightBoundaryPos}, \mathrm{refWraparoundEnabledFlag}\ ?\\ &\mathrm{ClipH}(\ (\ \mathrm{PpsRefWraparoundOffset}\ ) * \mathrm{MinCbSizeY}, \mathrm{picW}, \mathrm{xInt}_i\ ) : \mathrm{xInt}_i\ )\end{aligned} \tag{922}$$

$$\mathrm{yInt}_i = \mathrm{Clip3}(\ \mathrm{SubpicTopBoundaryPos}, \mathrm{SubpicBotBoundaryPos}, \mathrm{yInt}_i\ ) \tag{923}$$

– Otherwise (sps_subpic_treated_as_pic_flag[CurrSubpicIdx] is equal to 0 or sps_num_subpics_minus1 for the reference picture refPicLX is equal to 0), the following applies:

$$\begin{aligned}\mathrm{xInt}_i = \mathrm{Clip3}(\ &0, \mathrm{picW} - 1, \mathrm{refWraparoundEnabledFlag}\ ?\\ &\mathrm{ClipH}(\ (\ \mathrm{PpsRefWraparoundOffset}\ ) * \mathrm{MinCbSizeY}, \mathrm{picW}, \mathrm{xInt}_i\ ) : \mathrm{xInt}_i\ )\end{aligned} \tag{924}$$

$$\mathrm{yInt}_i = \mathrm{Clip3}(\ 0, \mathrm{picH} - 1, \mathrm{yInt}_i\ ) \tag{925}$$

The predicted luma sample value $\mathrm{predSampleLX}_L$ is derived as follows:

– If both xFrac_L and yFrac_L are equal to 0, and both scalingRatio[0] and scalingRatio[1] are less than 20481, the value of $\mathrm{predSampleLX}_L$ is derived as follows:

$$\mathrm{predSampleLX}_L = \mathrm{refPicLX}_L[\ \mathrm{xInt}_3\ ][\ \mathrm{yInt}_3\ ] \ll \mathrm{shift3} \tag{926}$$

– Otherwise, if yFrac_L is equal to 0 and scalingRatio[1] is less than 20481, the value of $\mathrm{predSampleLX}_L$ is derived as follows:

$$\mathrm{predSampleLX}_L = (\ \textstyle\sum_{i=0}^{7} f_{LH}[\ \mathrm{xFrac}_L\ ][\ i\ ] * \mathrm{refPicLX}_L[\ \mathrm{xInt}_i\ ][\ \mathrm{yInt}_3\ ]\ ) \gg \mathrm{shift1} \tag{927}$$

– Otherwise, if xFrac_L is equal to 0 and scalingRatio[0] is less than 20481, the value of $\mathrm{predSampleLX}_L$ is derived as follows:

$$\mathrm{predSampleLX}_L = (\ \textstyle\sum_{i=0}^{7} f_{LV}[\ \mathrm{yFrac}_L\ ][\ i\ ] * \mathrm{refPicLX}_L[\ \mathrm{xInt}_3\ ][\ \mathrm{yInt}_i\ ]\ ) \gg \mathrm{shift1} \tag{928}$$

– Otherwise, the value of $\mathrm{predSampleLX}_L$ is derived as follows:

  – The sample array temp[n] with n = 0..7, is derived as follows:

$$\mathrm{temp}[\ n\ ] = (\ \textstyle\sum_{i=0}^{7} f_{LH}[\ \mathrm{xFrac}_L\ ][\ i\ ] * \mathrm{refPicLX}_L[\ \mathrm{xInt}_i\ ][\ \mathrm{yInt}_n\ ]\ ) \gg \mathrm{shift1} \tag{929}$$

  – The predicted luma sample value $\mathrm{predSampleLX}_L$ is derived as follows:

$$\mathrm{predSampleLX}_L = (\ \textstyle\sum_{i=0}^{7} f_{LV}[\ \mathrm{yFrac}_L\ ][\ i\ ] * \mathrm{temp}[\ i\ ]\ ) \gg \mathrm{shift2} \tag{930}$$

Table 27 – Specification of the luma interpolation filter coefficients f_L[p] for each 1/16 fractional sample position p

Fractional sample position p	interpolation filter coefficients							
	f_L[p][0]	f_L[p][1]	f_L[p][2]	f_L[p][3]	f_L[p][4]	f_L[p][5]	f_L[p][6]	f_L[p][7]
1	0	1	−3	63	4	−2	1	0
2	−1	2	−5	62	8	−3	1	0
3	−1	3	−8	60	13	−4	1	0
4	−1	4	−10	58	17	−5	1	0
5	−1	4	−11	52	26	−8	3	−1
6	−1	3	−9	47	31	−10	4	−1
7	−1	4	−11	45	34	−10	4	−1
8 (hpelIfIdx = = 0)	−1	4	−11	40	40	−11	4	−1
8 (hpelIfIdx = = 1)	0	3	9	20	20	9	3	0
9	−1	4	−10	34	45	−11	4	−1
10	−1	4	−10	31	47	−9	3	−1
11	−1	3	−8	26	52	−11	4	−1
12	0	1	−5	17	58	−10	4	−1
13	0	1	−4	13	60	−8	3	−1
14	0	1	−3	8	62	−5	2	−1
15	0	1	−2	4	63	−3	1	0

Table 28 – Specification of the luma interpolation filter coefficients f_L[p] for each 1/16 fractional sample position p

Fractional sample position p	interpolation filter coefficients							
	f_L[p][0]	f_L[p][1]	f_L[p][2]	f_L[p][3]	f_L[p][4]	f_L[p][5]	f_L[p][6]	f_L[p][7]
0	−1	−5	17	42	17	−5	−1	0
1	0	−5	15	41	19	−5	−1	0
2	0	−5	13	40	21	−4	−1	0
3	0	−5	11	39	24	−4	−2	1
4	0	−5	9	38	26	−3	−2	1
5	0	−5	7	38	28	−2	−3	1
6	1	−5	5	36	30	−1	−3	1
7	1	−4	3	35	32	0	−4	1
8	1	−4	2	33	33	2	−4	1
9	1	−4	0	32	35	3	−4	1
10	1	−3	−1	30	36	5	−5	1
11	1	−3	−2	28	38	7	−5	0
12	1	−2	−3	26	38	9	−5	0
13	1	−2	−4	24	39	11	−5	0
14	0	−1	−4	21	40	13	−5	0
15	0	−1	−5	19	41	15	−5	0

Table 29 – Specification of the luma interpolation filter coefficients $f_L[p]$ for each 1/16 fractional sample position

Fractional sample position p	interpolation filter coefficients							
	$f_L[p][0]$	$f_L[p][1]$	$f_L[p][2]$	$f_L[p][3]$	$f_L[p][4]$	$f_L[p][5]$	$f_L[p][6]$	$f_L[p][7]$
0	−4	2	20	28	20	2	−4	0
1	−4	0	19	29	21	5	−4	−2
2	−4	−1	18	29	22	6	−4	−2
3	−4	−1	16	29	23	7	−4	−2
4	−4	−1	16	28	24	7	−4	−2
5	−4	−1	14	28	25	8	−4	−2
6	−3	−3	14	27	26	9	−3	−3
7	−3	−1	12	28	25	10	−4	−3
8	−3	−3	11	27	27	11	−3	−3
9	−3	−4	10	25	28	12	−1	−3
10	−3	−3	9	26	27	14	−3	−3
11	−2	−4	8	25	28	14	−1	−4
12	−2	−4	7	24	28	16	−1	−4
13	−2	−4	7	23	29	16	−1	−4
14	−2	−4	6	22	29	18	−1	−4
15	−2	−4	5	21	29	19	0	−4

Table 30 – Specification of the luma interpolation filter coefficients $f_L[p]$ for each 1/16 fractional sample position p for affine motion mode

Fractional sample position p	interpolation filter coefficients							
	$f_L[p][0]$	$f_L[p][1]$	$f_L[p][2]$	$f_L[p][3]$	$f_L[p][4]$	$f_L[p][5]$	$f_L[p][6]$	$f_L[p][7]$
1	0	1	−3	63	4	−2	1	0
2	0	1	−5	62	8	−3	1	0
3	0	2	−8	60	13	−4	1	0
4	0	3	−10	58	17	−5	1	0
5	0	3	−11	52	26	−8	2	0
6	0	2	−9	47	31	−10	3	0
7	0	3	−11	45	34	−10	3	0
8	0	3	−11	40	40	−11	3	0
9	0	3	−10	34	45	−11	3	0
10	0	3	−10	31	47	−9	2	0
11	0	2	−8	26	52	−11	3	0
12	0	1	−5	17	58	−10	3	0
13	0	1	−4	13	60	−8	2	0
14	0	1	−3	8	62	−5	1	0
15	0	1	−2	4	63	−3	1	0

Table 31 – Specification of the luma interpolation filter coefficients f_L[p] for each 1/16 fractional sample position p for affine motion mode

Fractional sample position p	interpolation filter coefficients							
	f_L[p][0]	f_L[p][1]	f_L[p][2]	f_L[p][3]	f_L[p][4]	f_L[p][5]	f_L[p][6]	f_L[p][7]
0	0	−6	17	42	17	−5	−1	0
1	0	−5	15	41	19	−5	−1	0
2	0	−5	13	40	21	−4	−1	0
3	0	−5	11	39	24	−4	−1	0
4	0	−5	9	38	26	−3	−1	0
5	0	−5	7	38	28	−2	−2	0
6	0	−4	5	36	30	−1	−2	0
7	0	−3	3	35	32	0	−3	0
8	0	−3	2	33	33	2	−3	0
9	0	−3	0	32	35	3	−3	0
10	0	−2	−1	30	36	5	−4	0
11	0	−2	−2	28	38	7	−5	0
12	0	−1	−3	26	38	9	−5	0
13	0	−1	−4	24	39	11	−5	0
14	0	−1	−4	21	40	13	−5	0
15	0	−1	−5	19	41	15	−5	0

Table 32 – Specification of the luma interpolation filter coefficients f_L[p] for each 1/16 fractional sample position p for affine motion mode

Fractional sample position p	interpolation filter coefficients							
	f_L[p][0]	f_L[p][1]	f_L[p][2]	f_L[p][3]	f_L[p][4]	f_L[p][5]	f_L[p][6]	f_L[p][7]
0	0	−2	20	28	20	2	−4	0
1	0	−4	19	29	21	5	−6	0
2	0	−5	18	29	22	6	−6	0
3	0	−5	16	29	23	7	−6	0
4	0	−5	16	28	24	7	−6	0
5	0	−5	14	28	25	8	−6	0
6	0	−6	14	27	26	9	−6	0
7	0	−4	12	28	25	10	−7	0
8	0	−6	11	27	27	11	−6	0
9	0	−7	10	25	28	12	−4	0
10	0	−6	9	26	27	14	−6	0
11	0	−6	8	25	28	14	−5	0
12	0	−6	7	24	28	16	−5	0
13	0	−6	7	23	29	16	−5	0
14	0	−6	6	22	29	18	−5	0
15	0	−6	5	21	29	19	−4	0

8.5.6.3.3 Luma integer sample fetching process

Inputs to this process are:

– a luma location in full-sample units ($xInt_L$, $yInt_L$),

– the luma reference sample array $refPicLX_L$,

– a variable refWraparoundEnabledFlag indicating whether horizontal wrap-around motion compensation is enabled.

Output of this process is a predicted luma sample value $predSampleLX_L$

The variable shift is set equal to Max(2, 14 − BitDepth).

The variable picW is set equal to pps_pic_width_in_luma_samples of the reference picture refPicLX and the variable picH is set equal to pps_pic_height_in_luma_samples of the reference picture refPicLX.

The luma locations in full-sample units (xInt, yInt) are derived as follows:

– If sps_subpic_treated_as_pic_flag[CurrSubpicIdx] is equal to 1 and sps_num_subpics_minus1 for the reference picture refPicLX is greater than 0, the following applies:

$$\text{xInt} = \text{Clip3}(\ \text{SubpicLeftBoundaryPos},\ \text{SubpicRightBoundaryPos},\ \text{refWraparoundEnabledFlag}\ ?\ \text{ClipH}(\ (\ \text{PpsRefWraparoundOffset}\ ) * \text{MinCbSizeY},\ \text{picW},\ \text{xInt}_L\ ) : \text{xInt}_L\ ) \quad (931)$$

$$\text{yInt} = \text{Clip3}(\ \text{SubpicTopBoundaryPos},\ \text{SubpicBotBoundaryPos},\ \text{yInt}_L\ ) \quad (932)$$

– Otherwise (sps_subpic_treated_as_pic_flag[CurrSubpicIdx] is equal to 0 or sps_num_subpics_minus1 for the reference picture refPicLX is equal to 0), the following applies:

$$\text{xInt} = \text{Clip3}(\ 0,\ \text{picW} - 1,\ \text{refWraparoundEnabledFlag}\ ?\ \text{ClipH}(\ (\ \text{PpsRefWraparoundOffset}\ ) * \text{MinCbSizeY},\ \text{picW},\ \text{xInt}_L\ ) : \text{xInt}_L\ ) \quad (933)$$

$$\text{yInt} = \text{Clip3}(\ 0,\ \text{picH} - 1,\ \text{yInt}_L\ ) \quad (934)$$

The predicted luma sample value predSampleLX_L is derived as follows:

$$\text{predSampleLX}_L = \text{refPicLX}_L[\ \text{xInt}\ ][\ \text{yInt}\ ] \ll \text{shift} \quad (935)$$

8.5.6.3.4 Chroma sample interpolation process

Inputs to this process are:

– a chroma location in full-sample units (xInt_C, yInt_C),

– a chroma location in 1/32 fractional-sample units (xFrac_C, yFrac_C),

– a chroma location in full-sample units (xSbInt_C, ySbInt_C) specifying the top-left sample of the bounding block for reference sample padding relative to the top-left chroma sample of the reference picture,

– a variable sbWidth specifying the width of the current subblock,

– a variable sbHeight specifying the height of the current subblock,

– the chroma reference sample array refPicLX_C,

– the decoder-side motion vector refinement flag dmvrFlag,

– a variable refWraparoundEnabledFlag indicating whether horizontal wrap-around motion compensation is enabled,

– a fixedpoint representation of the horizontal scaling factor scalingRatio[0],

– a fixedpoint representation of the vertical scaling factor scalingRatio[1].

Output of this process is a predicted chroma sample value predSampleLX_C

The variables shift1, shift2 and shift3 are derived as follows:

– The variable shift1 is set equal to Min(4, BitDepth − 8), the variable shift2 is set equal to 6 and the variable shift3 is set equal to Max(2, 14 − BitDepth).

– The variable picW_C is set equal to pps_pic_width_in_luma_samples / SubWidthC of the reference picture refPicLX and the variable picH_C is set equal to pps_pic_height_in_luma_samples / SubHeightC of the reference picture refPicLX.

The horizontal chroma interpolation filter coefficients $f_{CH}[\ p\ ]$ for each 1/32 fractional sample position p equal to xFrac_C are derived as follows:

– If scalingRatio[0] is greater than 28 672, chroma interpolation filter coefficients $f_{CH}[\ p\ ]$ are specified in Table 35.

– Otherwise, if scalingRatio[0] is greater than 20 480, chroma interpolation filter coefficients $f_{CH}[\ p\ ]$ are specified in Table 34.

– Otherwise, chroma interpolation filter coefficients $f_{CH}[\ p\ ]$ are specified in Table 33.

The vertical chroma interpolation filter coefficients $f_{CV}[p]$ for each 1/32 fractional sample position p equal to $yFrac_C$ are derived as follows:

– If scalingRatio[1] is greater than 28 672, chroma interpolation filter coefficients $f_{CV}[p]$ are specified in Table 35.

– Otherwise, if scalingRatio[1] is greater than 20 480, chroma interpolation filter coefficients $f_{CV}[p]$ are specified in Table 34.

– Otherwise, chroma interpolation filter coefficients $f_{CV}[p]$ are specified in Table 33.

The variable xOffset is set equal to PpsRefWraparoundOffset * MinCbSizeY / SubWidthC.

The chroma locations in full-sample units ($xInt_i$, $yInt_i$) are derived as follows for i = 0..3:

$$xInt_i = xInt_C + i - 1 \quad (936)$$

$$yInt_i = yInt_C + i - 1 \quad (937)$$

– When dmvrFlag is equal to 1, the following applies:

$$xInt_i = \text{Clip3}(xSbInt_C - 1, xSbInt_C + sbWidth - 1 + 2, xInt_i) \quad (938)$$

$$yInt_i = \text{Clip3}(ySbInt_C - 1, ySbInt_C + sbHeight - 1 + 2, yInt_i) \quad (939)$$

– If sps_subpic_treated_as_pic_flag[CurrSubpicIdx] is equal to 1 and sps_num_subpics_minus1 for the reference picture refPicLX is greater than 0, the following applies:

$$\begin{aligned} xInt_i = \text{Clip3}(& \text{SubpicLeftBoundaryPos / SubWidthC, SubpicRightBoundaryPos / SubWidthC,} \\ & \text{refWraparoundEnabledFlag ? ClipH}(xOffset, picW_C, xInt_i) : xInt_i) \end{aligned} \quad (940)$$

$$yInt_i = \text{Clip3(SubpicTopBoundaryPos / SubHeightC, SubpicBotBoundaryPos / SubHeightC, } yInt_i) \quad (941)$$

– Otherwise (sps_subpic_treated_as_pic_flag[CurrSubpicIdx] is equal to 0 or sps_num_subpics_minus1 for the reference picture refPicLX is equal to 0), the following applies:

$$xInt_i = \text{Clip3}(0, picW_C - 1, \text{refWraparoundEnabledFlag ? ClipH}(xOffset, picW_C, xInt_i) : xInt_i) \quad (942)$$

$$yInt_i = \text{Clip3}(0, picH_C - 1, yInt_i) \quad (943)$$

The predicted chroma sample value $predSampleLX_C$ is derived as follows:

– If both $xFrac_C$ and $yFrac_C$ are equal to 0, and both scalingRatio[0] and scalingRatio[1] are less than 20481, the value of $predSampleLX_C$ is derived as follows:

$$predSampleLX_C = refPicLX_C[xInt_1][yInt_1] \ll shift3 \quad (944)$$

– Otherwise, if $yFrac_C$ is equal to 0 and scalingRatio[1] is less than 20481, the value of $predSampleLX_C$ is derived as follows:

$$predSampleLX_C = (\textstyle\sum_{i=0}^{3} f_{CH}[xFrac_C][i] * refPicLX_C[xInt_i][yInt_1]) \gg shift1 \quad (945)$$

– Otherwise, if $xFrac_C$ is equal to 0 and scalingRatio[0] is less than 20481, the value of $predSampleLX_C$ is derived as follows:

$$predSampleLX_C = (\textstyle\sum_{i=0}^{3} f_{CV}[yFrac_C][i] * refPicLX_C[xInt_1][yInt_i]) \gg shift1 \quad (946)$$

– Otherwise, the value of $predSampleLX_C$ is derived as follows:

– The sample array temp[n] with n = 0..3, is derived as follows:

$$temp[n] = (\textstyle\sum_{i=0}^{3} f_{CH}[xFrac_C][i] * refPicLX_C[xInt_i][yInt_n]) \gg shift1 \quad (947)$$

– The predicted chroma sample value $predSampleLX_C$ is derived as follows:

$$predSampleLX_C = (f_{CV}[yFrac_C][0] * temp[0] + f_{CV}[yFrac_C][1] * temp[1] + f_{CV}[yFrac_C][2] * temp[2] + f_{CV}[yFrac_C][3] * temp[3]) >> shift2 \quad (948)$$

Table 33 – Specification of the chroma interpolation filter coefficients $f_C[p]$ for each 1/32 fractional sample position p

Fractional sample position p	interpolation filter coefficients			
	$f_C[p][0]$	$f_C[p][1]$	$f_C[p][2]$	$f_C[p][3]$
1	−1	63	2	0
2	−2	62	4	0
3	−2	60	7	−1
4	−2	58	10	−2
5	−3	57	12	−2
6	−4	56	14	−2
7	−4	55	15	−2
8	−4	54	16	−2
9	−5	53	18	−2
10	−6	52	20	−2
11	−6	49	24	−3
12	−6	46	28	−4
13	−5	44	29	−4
14	−4	42	30	−4
15	−4	39	33	−4
16	−4	36	36	−4
17	−4	33	39	−4
18	−4	30	42	−4
19	−4	29	44	−5
20	−4	28	46	−6
21	−3	24	49	−6
22	−2	20	52	−6
23	−2	18	53	−5
24	−2	16	54	−4
25	−2	15	55	−4
26	−2	14	56	−4
27	−2	12	57	−3
28	−2	10	58	−2
29	−1	7	60	−2
30	0	4	62	−2
31	0	2	63	−1

Table 34 – Specification of the chroma interpolation filter coefficients $f_C[\ p\ ]$ for each 1/32 fractional sample position p for scaling factors of around 1.5x

Fractional sample position p	interpolation filter coefficients			
	$f_C[\ p\ ][\ 0\ ]$	$f_C[\ p\ ][\ 1\ ]$	$f_C[\ p\ ][\ 2\ ]$	$f_C[\ p\ ][\ 3\ ]$
0	12	40	12	0
1	11	40	13	0
2	10	40	15	−1
3	9	40	16	−1
4	8	40	17	−1
5	8	39	18	−1
6	7	39	19	−1
7	6	38	21	−1
8	5	38	22	−1
9	4	38	23	−1
10	4	37	24	−1
11	3	36	25	0
12	3	35	26	0
13	2	34	28	0
14	2	33	29	0
15	1	33	30	0
16	1	31	31	1
17	0	30	33	1
18	0	29	33	2
19	0	28	34	2
20	0	26	35	3
21	0	25	36	3
22	−1	24	37	4
23	−1	23	38	4
24	−1	22	38	5
25	−1	21	38	6
26	−1	19	39	7
27	−1	18	39	8
28	−1	17	40	8
29	−1	16	40	9
30	−1	15	40	10
31	0	13	40	11

Table 35 – Specification of the chroma interpolation filter coefficients $f_C[p]$ for each 1/32 fractional sample position p for scaling factors of around 2x

Fractional sample position p	interpolation filter coefficients			
	$f_C[p][0]$	$f_C[p][1]$	$f_C[p][2]$	$f_C[p][3]$
0	17	30	17	0
1	17	30	18	−1
2	16	30	18	0
3	16	30	18	0
4	15	30	18	1
5	14	30	18	2
6	13	29	19	3
7	13	29	19	3
8	12	29	20	3
9	11	28	21	4
10	10	28	22	4
11	10	27	22	5
12	9	27	23	5
13	9	26	24	5
14	8	26	24	6
15	7	26	25	6
16	7	25	25	7
17	6	25	26	7
18	6	24	26	8
19	5	24	26	9
20	5	23	27	9
21	5	22	27	10
22	4	22	28	10
23	4	21	28	11
24	3	20	29	12
25	3	19	29	13
26	3	19	29	13
27	2	18	30	14
28	1	18	30	15
29	0	18	30	16
30	0	18	30	16
31	−1	18	30	17

8.5.6.4 Prediction refinement with optical flow process

Inputs to this process are:

– two variables sbWidth and sbHeight specifying the width and the height of the current subblock,

– one (sbWidth + 2)x(sbHeight + 2) prediction sample array $predSamplesLX_L$,

– one (sbWidth)x(sbHeight) motion vector difference array diffMvLX.

Output of this process is the (sbWidth)x(sbHeight) array $sbSamplesLX_L$ of prediction sample values.

Variable shift1 is set equal to 6.

For x =0..sbWidth − 1, y =0..sbHeight − 1, the following ordered steps apply:

– The variables gradientH[x][y] and gradientV[x][y] are derived as follows:

$$\begin{aligned} \text{gradientH}[x][y] = {} & (\text{predSamplesLX}_L[x + 2][y + 1] \gg \text{shift1}) - \\ & (\text{predSamplesLX}_L[x][y + 1] \gg \text{shift1}) \end{aligned} \quad (949)$$

$$\text{gradientV}[\ x\ ][\ y\ ] = (\ \text{predSamplesLX}_L[\ x + 1\ ][\ y + 2\ ] \gg \text{shift1}\ ) - (\ \text{predSamplesLX}_L[\ x + 1\ ][\ y\ ] \gg \text{shift1}\ ) \quad (950)$$

– The variable dI is derived as follows:

$$\text{dI} = \text{gradientH}[\ x\ ][\ y\ ] * \text{diffMvLX}[\ x\ ][\ y\ ][\ 0\ ] + \text{gradientV}[\ x\ ][\ y\ ] * \text{diffMvLX}[\ x\ ][\ y\ ][\ 1\ ] \quad (951)$$

– Prediction sample value at location (x, y) in the subblock is derived as follows:

$$\text{dILimit} = (\ 1 \ll \text{Max}(\ 13, \text{BitDepth} + 1\ )\ ) \quad (952)$$

$$\text{sbSamplesLX}_L[\ x\ ][\ y\ ] = \text{predSamplesLX}_L[\ x + 1\ ][\ y + 1\ ] + \text{Clip3}(\ -\text{dILimit}, \text{dILimit} - 1, \text{dI}\ ) \quad (953)$$

8.5.6.5 Bi-directional optical flow prediction process

Inputs to this process are:

– two variables nCbW and nCbH specifying the width and the height of the current coding block,

– two (nCbW + 2)x(nCbH + 2) luma prediction sample arrays predSamplesL0 and predSamplesL1.

Output of this process is the (nCbW)x(nCbH) array pbSamples of luma prediction sample values.

The variables shift1, shift2, shift3, shift4, offset4, and mvRefineThres are derived as follows:

– The variable shift1 is set to be equal to 6.

– The variable shift2 is set to be equal to 4.

– The variable shift3 is set to be equal to 1.

– The variable shift4 is set equal to Max(3, 15 − BitDepth) and the variable offset4 is set equal to 1 << (shift4 − 1).

– The variable mvRefineThres is set equal to 1 << 4.

For xIdx = 0..(nCbW >> 2) − 1 and yIdx = 0..(nCbH >> 2) − 1, the following applies:

– The variable xSb is set equal to (xIdx << 2) + 1 and ySb is set equal to (yIdx << 2) + 1.

– The prediction sample values of the current subblock are derived as follows:

  – For x =xSb − 1..xSb + 4, y = ySb − 1..ySb + 4, the following ordered steps apply:

    1. The locations (h_x, v_y) for each of the corresponding sample locations (x, y) inside the prediction sample arrays are derived as follows:

$$h_x = \text{Clip3}(\ 1, \text{nCbW}, x\ ) \quad (954)$$

$$v_y = \text{Clip3}(\ 1, \text{nCbH}, y\ ) \quad (955)$$

    2. The variables gradientHL0[x][y], gradientVL0[x][y], gradientHL1[x][y] and gradientVL1[x][y] are derived as follows:

$$\text{gradientHL0}[\ x\ ][\ y\ ] = (\ \text{predSamplesL0}[\ h_x + 1\ ][\ v_y\ ] \gg \text{shift1}\ ) - (\ \text{predSamplesL0}[\ h_x - 1\ ][\ v_y\ ] \gg \text{shift1}\ ) \quad (956)$$

$$\text{gradientVL0}[\ x\ ][\ y\ ] = (\ \text{predSamplesL0}[\ h_x\ ][\ v_y + 1\ ] \gg \text{shift1}\ ) - (\ \text{predSamplesL0}[\ h_x\ ][\ v_y - 1\ ] \gg \text{shift1}\ ) \quad (957)$$

$$\text{gradientHL1}[\ x\ ][\ y\ ] = (\ \text{predSamplesL1}[\ h_x + 1\ ][\ v_y\ ] \gg \text{shift1}\ ) - (\ \text{predSamplesL1}[\ h_x - 1\ ][\ v_y\ ] \gg \text{shift1}\ ) \quad (958)$$

$$\text{gradientVL1}[\ x\ ][\ y\ ] = (\ \text{predSamplesL1}[\ h_x\ ][\ v_y + 1\ ] \gg \text{shift1}\ ) - (\ \text{predSamplesL1}[\ h_x\ ][\ v_y - 1\ ] \gg \text{shift1}\ ) \quad (959)$$

    3. The variables diff[x][y], tempH[x][y] and tempV[x][y] are derived as follows:

$$\text{diff}[\ x\ ][\ y\ ] = (\text{predSamplesL0}[\ h_x\ ][\ v_y\ ] \gg \text{shift2}\ ) - (\ \text{predSamplesL1}[\ h_x\ ][\ v_y\ ] \gg \text{shift2}\ ) \quad (960)$$

$$\text{tempH}[\ x\ ][\ y\ ] = (\text{gradientHL0}[\ x\ ][\ y\ ] + \text{gradientHL1}[\ x\ ][\ y\ ]\ ) \gg \text{shift3} \tag{961}$$

$$\text{tempV}[\ x\ ][\ y\ ] = (\text{gradientVL0}[\ x\ ][\ y\ ] + \text{gradientVL1}[\ x\ ][\ y\ ]\ ) \gg \text{shift3} \tag{962}$$

– The variables sGx2, sGy2, sGxGy, sGxdI and sGydI are derived as follows:

$$\text{sGx2} = \Sigma_i\Sigma_j\text{Abs}(\ \text{tempH}[\ \text{xSb} + i\ ][\ \text{ySb} + j\ ]\ ) \text{ with } i, j = -1..4 \tag{963}$$

$$\text{sGy2} = \Sigma_i\Sigma_j\ \text{Abs}(\ \text{tempV}[\ \text{xSb} + i\ ][\ \text{ySb} + j\ ]\ ) \text{ with } i, j = -1..4 \tag{964}$$

$$\text{sGxGy} = \Sigma_i\Sigma_j(\ \text{Sign}(\ \text{tempV}[\ \text{xSb} + i\ ][\ \text{ySb} + j\ ]\ ) * \text{tempH}[\ \text{xSb} + i\ ][\ \text{ySb} + j\ ]\ ) \text{ with } i, j = -1..4 \tag{965}$$

$$\text{sGxdI} = \Sigma_i\Sigma_j(\ -\text{Sign}(\ \text{tempH}[\ \text{xSb} + i\ ][\ \text{ySb} + j\ ]\ ) * \text{diff}[\ \text{xSb} + i\ ][\ \text{ySb} + j\ ]\ ) \text{ with } i, j = -1..4 \tag{966}$$

$$\text{sGydI} = \Sigma_i\Sigma_j(\ -\text{Sign}(\ \text{tempV}[\ \text{xSb} + i\ ][\ \text{ySb} + j\ ]\ ) * \text{diff}[\ \text{xSb} + i\ ][\ \text{ySb} + j\ ]\ ) \text{ with } i, j = -1..4 \tag{967}$$

– The horizontal and vertical motion offset of the current subblock are derived as:

$$\begin{aligned} v_x = \text{sGx2} > 0\ ?\ &\text{Clip3}(\ -\text{mvRefineThres} + 1, \text{mvRefineThres} - 1, \\ &(\ \text{sGxdI} \ll 2\ ) \gg \text{Floor}(\ \text{Log2}(\ \text{sGx2}\ )\ )\ ) : 0 \end{aligned} \tag{968}$$

$$\begin{aligned} v_y = \text{sGy2} > 0\ ?\ &\text{Clip3}(\ -\text{mvRefineThres} + 1, \text{mvRefineThres} - 1, (\ (\ \text{sGydI} \ll 2\ ) - \\ &(\ (\ v_x * \text{sGxGy}\ ) \gg 1\ )\ ) \gg \text{Floor}(\ \text{Log2}(\ \text{sGy2}\ )\ )\ ) : 0 \end{aligned} \tag{969}$$

– For x =xSb − 1..xSb + 2, y = ySb − 1..ySb + 2, the prediction sample values of the current subblock are derived as follows:

$$\begin{aligned} \text{bdofOffset} = v_x * (\ &\text{gradientHL0}[\ x + 1\ ][\ y + 1\ ] - \text{gradientHL1}[\ x + 1\ ][\ y + 1\ ]\ ) \\ &+ v_y * (\text{gradientVL0}[\ x + 1\ ][\ y + 1\ ] - \text{gradientVL1}[\ x + 1\ ][\ y + 1\ ]\ ) \end{aligned} \tag{970}$$

$$\begin{aligned} \text{pbSamples}[\ x\ ][\ y\ ] = \text{Clip3}(\ 0, (\ 2^{\text{BitDepth}}\ ) - 1, (\ &\text{predSamplesL0}[\ x + 1\ ][\ y + 1\ ] + \text{offset4} + \\ &\text{predSamplesL1}[\ x + 1\ ][\ y + 1\ ] + \text{bdofOffset}\ ) \gg \text{shift4}\ ) \end{aligned} \tag{971}$$

8.5.6.6 Weighted sample prediction process

8.5.6.6.1 General

Inputs to this process are:

– a luma location (xCb, yCb) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– two variables nCbW and nCbH specifying the width and the height of the current coding block,

– two (nCbW)x(nCbH) arrays predSamplesL0 and predSamplesL1,

– the prediction list utilization flags, predFlagL0 and predFlagL1,

– the reference indices refIdxL0 and refIdxL1,

– the bi-prediction weight index bcwIdx,

– the decoder-side motion vector refinement flag dmvrFlag,

– the variable cIdx specifying the colour component index.

Output of this process is the (nCbW)x(nCbH) array pbSamples of prediction sample values.

The variable weightedPredFlag is derived as follows:

– If sh_slice_type is equal to P, weightedPredFlag is set equal to pps_weighted_pred_flag.

– Otherwise (sh_slice_type is equal to B), weightedPredFlag is set equal to (pps_weighted_bipred_flag && !dmvrFlag).

The following applies:

– If weightedPredFlag is equal to 0 or bcwIdx is not equal to 0, the array pbSamples of the prediction samples is derived by invoking the default weighted sample prediction process as specified in clause 8.5.6.6.2 with the luma location

(xCb, yCb), the coding block width nCbW, the coding block height nCbH, two (nCbW)x(nCbH) arrays predSamplesL0 and predSamplesL1, the prediction list utilization flags predFlagL0 and predFlagL1, the bi-prediction weight index bcwIdx and the bit depth BitDepth as inputs.

– Otherwise (weightedPredFlag is equal to 1 and bcwIdx is equal to 0), the array pbSamples of the prediction samples is derived by invoking the explicit weighted sample prediction process as specified in clause 8.5.6.6.3 with the coding block width nCbW, the coding block height nCbH, two (nCbW)x(nCbH) arrays predSamplesL0 and predSamplesL1, the prediction list utilization flags predFlagL0 and predFlagL1, the reference indices refIdxL0 and refIdxL1, the colour component index cIdx and the bit depth BitDepth as inputs.

8.5.6.6.2 Default weighted sample prediction process

Inputs to this process are:

– a luma location (xCb, yCb) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– two variables nCbW and nCbH specifying the width and the height of the current coding block,

– two (nCbW)x(nCbH) arrays predSamplesL0 and predSamplesL1,

– the prediction list utilization flags predFlagL0 and predFlagL1,

– the bi-prediction weight index bcwIdx,

– the sample bit depth, bitDepth.

Output of this process is the (nCbW)x(nCbH) array pbSamples of prediction sample values.

Variables shift1, shift2, offset1, offset2, and offset3 are derived as follows:

– The variable shift1 is set equal to Max(2, 14 − bitDepth) and the variable shift2 is set equal to Max(3, 15 − bitDepth).

– The variable offset1 is set equal to 1 << (shift1 − 1).

– The variable offset2 is set equal to 1 << (shift2 − 1).

– The variable offset3 is set equal to 1 << (shift1 + 2).

Depending on the values of predFlagL0 and predFlagL1, the prediction samples pbSamples[x][y] with x = 0..nCbW − 1 and y = 0..nCbH − 1 are derived as follows:

– If predFlagL0 is equal to 1 and predFlagL1 is equal to 0, the prediction sample values are derived as follows:

$$\text{pbSamples}[\,x\,][\,y\,] = \text{Clip3}(\,0, (\,1 \ll \text{bitDepth}\,) - 1, (\,\text{predSamplesL0}[\,x\,][\,y\,] + \text{offset1}\,) \gg \text{shift1}\,) \qquad (972)$$

– Otherwise, if predFlagL0 is equal to 0 and predFlagL1 is equal to 1, the prediction sample values are derived as follows:

$$\text{pbSamples}[\,x\,][\,y\,] = \text{Clip3}(\,0, (\,1 \ll \text{bitDepth}\,) - 1, (\,\text{predSamplesL1}[\,x\,][\,y\,] + \text{offset1}\,) \gg \text{shift1}\,) \qquad (973)$$

– Otherwise (predFlagL0 is equal to 1 and predFlagL1 is equal to 1), the following applies:

  – If bcwIdx is equal to 0 or ciip_flag[xCb][yCb] is equal to 1, the prediction sample values are derived as follows:

$$\begin{aligned}\text{pbSamples}[\,x\,][\,y\,] = \text{Clip3}(\,&0, (\,1 \ll \text{bitDepth}\,) - 1,\\ &(\,\text{predSamplesL0}[\,x\,][\,y\,] + \text{predSamplesL1}[\,x\,][\,y\,] + \text{offset2}\,) \gg \text{shift2}\,)\end{aligned} \qquad (974)$$

  – Otherwise (bcwIdx is not equal to 0 and ciip_flag[xCb][yCb] is equal to 0), the following applies:

    – The variable w1 is set equal to bcwWLut[bcwIdx] with bcwWLut[k] = { 4, 5, 3, 10, −2 }.

    – The variable w0 is set equal to (8 − w1).

    – The prediction sample values are derived as follows:

$$\begin{aligned}\text{pbSamples}[\,x\,][\,y\,] = \text{Clip3}(\,&0, (\,1 \ll \text{bitDepth}\,) - 1,\\ &(\,\text{w0} * \text{predSamplesL0}[\,x\,][\,y\,] + \text{w1} * \text{predSamplesL1}[\,x\,][\,y\,] + \text{offset3}\,) \gg (\,\text{shift1} + 3\,)\,)\end{aligned} \qquad (975)$$

8.5.6.6.3 Explicit weighted sample prediction process

Inputs to this process are:

- two variables nCbW and nCbH specifying the width and the height of the current coding block,
- two (nCbW)x(nCbH) arrays predSamplesL0 and predSamplesL1,
- the prediction list utilization flags, predFlagL0 and predFlagL1,
- the reference indices, refIdxL0 and refIdxL1,
- the variable cIdx specifying the colour component index,
- the sample bit depth, bitDepth.

Output of this process is the (nCbW)x(nCbH) array pbSamples of prediction sample values.

The variable shift1 is set equal to Max(2, 14 − bitDepth).

The variables log2Wd, o0, o1, w0 and w1 are derived as follows:

- If cIdx is equal to 0 for luma samples, the following applies:

  log2Wd = luma_log2_weight_denom + shift1 (976)

  - When predFlagL0 is equal to 1, the variables w0 and o0 are derived as follows:

    w0 = LumaWeightL0[refIdxL0] (977)

    o0 = luma_offset_l0[refIdxL0] << (bitDepth − 8) (978)

  - When predFlagL1 is equal to 1, the variables w1 and o1 are derived as follows:

    w1 = LumaWeightL1[refIdxL1] (979)

    o1 = luma_offset_l1[refIdxL1] << (bitDepth − 8) (980)

- Otherwise (cIdx is not equal to 0 for chroma samples), the following applies:

  log2Wd = ChromaLog2WeightDenom + shift1 (981)

  - When predFlagL0 is equal to 1, the variables w0 and o0 are derived as follows:

    w0 = ChromaWeightL0[refIdxL0][cIdx − 1] (982)

    o0 = ChromaOffsetL0[refIdxL0][cIdx − 1] << (bitDepth − 8) (983)

  - When predFlagL1 is equal to 1, the variables w1 and o1 are derived as follows:

    w1 = ChromaWeightL1[refIdxL1][cIdx − 1] (984)

    o1 = ChromaOffsetL1[refIdxL1][cIdx − 1] << (bitDepth − 8) (985)

The prediction sample pbSamples[x][y] with x = 0..nCbW − 1 and y = 0..nCbH − 1 are derived as follows:

- If predFlagL0 is equal to 1 and predFlagL1 is equal to 0, the prediction sample values are derived as follows:

  $$\text{pbSamples[x][y]} = \text{Clip3(0, (1 << bitDepth) − 1,} \\ \text{((predSamplesL0[x][y] * w0 + } 2^{\text{log2Wd} - 1} \text{) >> log2Wd) + o0)} \quad (986)$$

- Otherwise, if predFlagL0 is equal to 0 and predFlagL1 is equal to 1, the prediction sample values are derived as follows:

  $$\text{pbSamples[x][y]} = \text{Clip3(0, (1 << bitDepth) − 1,} \\ \text{((predSamplesL1[x][y] * w1 + } 2^{\text{log2Wd} - 1} \text{) >> log2Wd) + o1)} \quad (987)$$

- Otherwise (predFlagL0 is equal to 1 and predFlagL1 is equal to 1), the prediction sample values are derived as follows:

$$\text{pbSamples}[\,x\,][\,y\,] = \text{Clip3}(\,0, (\,1 \ll \text{bitDepth}\,) - 1,\\ (\,\text{predSamplesL0}[\,x\,][\,y\,] * \text{w0} + \text{predSamplesL1}[\,x\,][\,y\,] * \text{w1} +\\ (\,(\,\text{o0} + \text{o1} + 1\,) \ll \text{log2Wd}\,)\,) \gg (\,\text{log2Wd} + 1\,)\,) \qquad (988)$$

8.5.6.7 Weighted sample prediction process for combined merge and intra prediction

Inputs to this process are:

- a luma location (xCb, yCb) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
- the width of the current coding block cbWidth,
- the height of the current coding block cbHeight,
- two (cbWidth)x(cbHeight) arrays predSamplesInter and predSamplesIntra,
- a variable cIdx specifying the colour component index.

Output of this process is the (cbWidth)x(cbHeight) array predSamplesComb of prediction sample values.

The variables scaleFactX and scaleFactY are derived as follows:

$$\text{scaleFactX} = (\,\text{cIdx} == 0 \;||\; \text{SubWidthC} == 1\,)\;?\;0 : 1 \qquad (989)$$

$$\text{scaleFactY} = (\,\text{cIdx} == 0 \;||\; \text{SubHeightC} == 1\,)\;?\;0 : 1 \qquad (990)$$

The neighbouring luma locations (xNbA, yNbA) and (xNbB, yNbB) are set equal to (xCb − 1, yCb − 1 + (cbHeight << scaleFactY)) and (xCb − 1 + (cbWidth << scaleFactX), yCb − 1), respectively.

For X being replaced by either A or B, the variables availableX and isIntraCodedNeighbourX are derived as follows:

- The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring location (xNbY, yNbY) set equal to (xNbX, yNbX), checkPredModeY set equal to FALSE, and cIdx set equal to 0 as inputs, and the output is assigned to availableX.
- The variable isIntraCodedNeighbourX is derived as follows:
  - If availableX is equal to TRUE and CuPredMode[0][xNbX][yNbX] is equal to MODE_INTRA, isIntraCodedNeighbourX is set equal to TRUE.
  - Otherwise, isIntraCodedNeighbourX is set equal to FALSE.

The weight w is derived as follows:

  - If isIntraCodedNeighbourA and isIntraCodedNeighbourB are both equal to TRUE, w is set equal to 3.
  - Otherwise, if isIntraCodedNeighbourA and isIntraCodedNeighbourB are both equal to to FALSE, w is set equal to 1.
  - Otherwise, w is set equal to 2.

When cIdx is equal to 0 and sh_lmcs_used_flag is equal to 1, predSamplesInter[x][y] with x = 0..cbWidth − 1 and y = 0..cbHeight − 1 are modified as follows:

$$\text{idxY} = \text{predSamplesInter}[\,x\,][\,y\,] \gg \text{Log2}(\,\text{OrgCW}\,)\\ \text{predSamplesInter}[\,x\,][\,y\,] = \text{Clip1}(\,\text{LmcsPivot}[\,\text{idxY}\,] +\\ (\,(\,\text{ScaleCoeff}[\,\text{idxY}\,] * (\,\text{predSamplesInter}[\,x\,][\,y\,] - \text{InputPivot}[\,\text{idxY}\,]\,) +\\ (\,1 \ll 10\,)\,) \gg 11\,)\,) \qquad (991)$$

The prediction samples predSamplesComb[x][y] with x = 0..cbWidth − 1 and y = 0..cbHeight − 1 are derived as follows:

$$\text{predSamplesComb}[\,x\,][\,y\,] = (\,w * \text{predSamplesIntra}[\,x\,][\,y\,] +\\ (\,4 - w\,) * \text{predSamplesInter}[\,x\,][\,y\,] + 2\,) \gg 2 \qquad (992)$$

8.5.7 Decoding process for geometric partitioning mode inter blocks

8.5.7.1 General

This process is invoked when decoding a coding unit with MergeGpmFlag[xCb][yCb] equal to 1.

Inputs to this process are:

– a luma location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples,

– the luma motion vectors in 1/16 fractional-sample accuracy mvA and mvB,

– the chroma motion vectors mvCA and mvCB,

– the reference indices refIdxA and refIdxB,

– the prediction list flags predListFlagA and predListFlagB.

Outputs of this process are:

– an (cbWidth)x(cbHeight) array $predSamples_L$ of luma prediction samples,

– an (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array $predSamples_{Cb}$ of chroma prediction samples for the component Cb, when sps_chroma_format_idc is not equal to 0,

– an (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array $predSamples_{Cr}$ of chroma prediction samples for the component Cr, when sps_chroma_format_idc is not equal to 0.

Let $predSamplesLA_L$ and $predSamplesLB_L$ be (cbWidth)x(cbHeight) arrays of predicted luma sample values and, when sps_chroma_format_idc is not equal to 0, $predSamplesLA_{Cb}$, $predSamplesLB_{Cb}$, $predSamplesLA_{Cr}$ and $predSamplesLB_{Cr}$ be (cbWidth / SubWidthC)x(cbHeight / SubHeightC) arrays of predicted chroma sample values.

The $predSamples_L$, $predSamples_{Cb}$ and $predSamples_{Cr}$ are derived by the following ordered steps:

1. For N being each of A and B, the following applies:
   – The reference picture consisting of an ordered two-dimensional array $refPicLN_L$ of luma samples and two ordered two-dimensional arrays $refPicLN_{Cb}$ and $refPicLN_{Cr}$ of chroma samples is derived by invoking the process specified in clause 8.5.6.2 with X set equal to predListFlagN and refIdxX set equal to refIdxN as input.
   – The array $predSamplesLN_L$ is derived by invoking the fractional sample interpolation process specified in clause 8.5.6.3 with the luma location (xCb, yCb), the luma coding block width sbWidth set equal to cbWidth, the luma coding block height sbHeight set equal to cbHeight, the motion vector offset mvOffset set equal to (0, 0), the motion vector mvLX set equal to mvN and the reference array $refPicLX_L$ set equal to $refPicLN_L$, the variable bdofFlag set equal to FALSE, the variable cbProfFlagLX set equal to FALSE, the variable dmvrFlag set equal to FALSE, the variable hpelIfIdx set equal to 0, the variable cIdx is set equal to 0, RprConstraintsActiveFlag[predListFlagN][refIdxN], and RefPicScale[predListFlagN][refIdxN] as inputs.
   – When sps_chroma_format_idc is not equal to 0, the array $predSamplesLN_{Cb}$ is derived by invoking the fractional sample interpolation process specified in clause 8.5.6.3 with the luma location (xCb, yCb), the coding block width sbWidth set equal to cbWidth / SubWidthC, the coding block height sbHeight set equal to cbHeight / SubHeightC, the motion vector offset mvOffset set equal to (0, 0), the motion vector mvLX set equal to mvCN, and the reference array $refPicLX_{Cb}$ set equal to $refPicLN_{Cb}$, the variable bdofFlag set equal to FALSE, the variable cbProfFlagLX set equal to FALSE, the variable dmvrFlag set equal to FALSE, the variable hpelIfIdx set equal to 0, the variable cIdx is set equal to 1, RprConstraintsActiveFlag[predListFlagN][refIdxN], and RefPicScale[predListFlagN][refIdxN] as inputs.
   – When sps_chroma_format_idc is not equal to 0, the array $predSamplesLN_{Cr}$ is derived by invoking the fractional sample interpolation process specified in clause 8.5.6.3 with the luma location (xCb, yCb), the coding block width sbWidth set equal to cbWidth / SubWidthC, the coding block height sbHeight set equal to cbHeight / SubHeightC, the motion vector offset mvOffset set equal to (0, 0), the motion vector mvLX set equal to mvCN, and the reference array $refPicLX_{Cr}$ set equal to $refPicLN_{Cr}$, the variable bdofFlag set equal to FALSE, the variable cbProfFlagLX set equal to FALSE, the variable dmvrFlag set equal to FALSE, the variable hpelIfIdx set equal to 0, the variable cIdx is set equal to 2, RprConstraintsActiveFlag[predListFlagN][refIdxN], and RefPicScale[predListFlagN][refIdxN] as inputs.
2. The partition angle variable angleIdx and the distance variable distanceIdx of the geometric partitioning mode are set according to the value of merge_gpm_partition_idx[xCb][yCb] as specified in Table 36.
3. The prediction samples inside the current luma coding block, $predSamples_L[x_L][y_L]$ with $x_L = 0..cbWidth - 1$ and $y_L = 0..cbHeight - 1$, are derived by invoking the weighted sample prediction process for geometric partitioning mode specified in clause 8.5.7.2 with the coding block width nCbW set equal to cbWidth, the coding block height nCbH set equal to cbHeight, the sample arrays $predSamplesLA_L$ and $predSamplesLB_L$, and the variables angleIdx, distanceIdx, and cIdx equal to 0 as inputs.

4. When sps_chroma_format_idc is not equal to 0, the prediction samples inside the current chroma component Cb coding block, $predSamples_{Cb}[x_C][y_C]$ with $x_C = 0..cbWidth / SubWidthC - 1$ and $y_C = 0..cbHeight / SubHeightC - 1$, are derived by invoking the weighted sample prediction process for geometric partitioning mode specified in clause 8.5.7.2 with the coding block width nCbW set equal to cbWidth / SubWidthC, the coding block height nCbH set equal to cbHeight / SubHeightC, the sample arrays $predSamplesLA_{Cb}$ and $predSamplesLB_{Cb}$, and the variables angleIdx, distanceIdx, and cIdx equal to 1 as inputs.

5. When sps_chroma_format_idc is not equal to 0, the prediction samples inside the current chroma component Cr coding block, $predSamples_{Cr}[x_C][y_C]$ with $x_C = 0..cbWidth / SubWidthC - 1$ and $y_C = 0..cbHeight / SubHeightC - 1$, are derived by invoking the weighted sample prediction process for geometric partitioning mode specified in clause 8.5.7.2 with the coding block width nCbW set equal to cbWidth / SubWidthC, the coding block height nCbH set equal to cbHeight / SubHeightC, the sample arrays $predSamplesLA_{Cr}$ and $predSamplesLB_{Cr}$, and the variables angleIdx, distanceIdx, and cIdx equal to 2 as inputs.

6. The motion vector storing process for merge geometric partitioning mode specified in clause 8.5.7.3 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth, the luma coding block height cbHeight, the partition angle angleIdx and the distance distanceIdx, the luma motion vectors mvA and mvB, the reference indices refIdxA and refIdxB, and the prediction list flags predListFlagA and predListFlagB as inputs.

Table 36 – Specification of angleIdx and distanceIdx based on merge_gpm_partition_idx

merge_gpm_partition_idx	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
angleIdx	0	0	2	2	2	2	3	3	3	3	4	4	4	4	5	5
distanceIdx	1	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1
merge_gpm_partition_idx	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
angleIdx	5	5	8	8	11	11	11	11	12	12	12	12	13	13	13	13
distanceIdx	2	3	1	3	0	1	2	3	0	1	2	3	0	1	2	3
merge_gpm_partition_idx	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
angleIdx	14	14	14	14	16	16	18	18	18	19	19	19	20	20	20	21
distanceIdx	0	1	2	3	1	3	1	2	3	1	2	3	1	2	3	1
merge_gpm_partition_idx	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
angleIdx	21	21	24	24	27	27	27	28	28	28	29	29	29	30	30	30
distanceIdx	2	3	1	3	1	2	3	1	2	3	1	2	3	1	2	3

8.5.7.2 Weighted sample prediction process for geometric partitioning mode

Inputs to this process are:

– two variables nCbW and nCbH specifying the width and the height of the current coding block,

– two (nCbW)x(nCbH) arrays predSamplesLA and predSamplesLB,

– a variable angleIdx specifying the angle index of the geometric partition,

– a variable distanceIdx specifying the distance index of the geometric partition,

– a variable cIdx specifying colour component index.

Output of this process is the (nCbW)x(nCbH) array pbSamples of prediction sample values.

The variables nW, nH, shift1, offset1, displacementX, displacementY, partFlip and shiftHor are derived as follows:

$$nW = (cIdx == 0) ? nCbW : nCbW * SubWidthC \tag{993}$$

$$nH = (cIdx == 0) ? nCbH : nCbH * SubHeightC \tag{994}$$

$$shift1 = Max(5, 17 - BitDepth) \tag{995}$$

$$offset1 = 1 << (shift1 - 1) \tag{996}$$

$$displacementX = angleIdx \tag{997}$$

$$displacementY = (angleIdx + 8) \% 32 \quad (998)$$

$$partFlip = (angleIdx >= 13 \&\& angleIdx <= 27) ? 0 : 1 \quad (999)$$

$$shiftHor = (angleIdx \% 16 == 8 \mid\mid (angleIdx \% 16 != 0 \&\& nH >= nW)) ? 0 : 1 \quad (1000)$$

The variables offsetX and offsetY are derived as follows:

– If shiftHor is equal to 0, the following applies:

$$offsetX = (-nW) >> 1 \quad (1001)$$

$$offsetY = ((-nH) >> 1) + (angleIdx < 16 ? (distanceIdx * nH) >> 3 : -((distanceIdx * nH) >> 3)) \quad (1002)$$

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

$$offsetX = ((-nW) >> 1) + (angleIdx < 16 ? (distanceIdx * nW) >> 3 : -((distanceIdx * nW) >> 3)) \quad (1003)$$

$$offsetY = (- nH) >> 1 \quad (1004)$$

The prediction samples pbSamples[x][y] with x = 0..nCbW − 1 and y = 0..nCbH − 1 are derived as follows:

– The variables xL and yL are derived as follows:

$$xL = (cIdx == 0) ? x : x * SubWidthC \quad (1005)$$

$$yL = (cIdx == 0) ? y : y * SubHeightC \quad (1006)$$

– The variable wValue specifying the weight of the prediction sample is derived based on the array disLut specified in Table 37 as follows:

$$weightIdx = (((xL + offsetX) << 1) + 1) * disLut[displacementX] + (((yL + offsetY) << 1) + 1) * disLut[displacementY] \quad (1007)$$

$$weightIdxL = partFlip ? 32 + weightIdx : 32 - weightIdx \quad (1008)$$

$$wValue = Clip3(0, 8, (weightIdxL + 4) >> 3) \quad (1009)$$

– The prediction sample values are derived as follows:

$$pbSamples[x][y] = Clip3(0, (1 << BitDepth) - 1, (predSamplesLA[x][y] * wValue + predSamplesLB[x][y] * (8 - wValue) + offset1) >> shift1) \quad (1010)$$

Table 37 – Specification of the geometric partitioning distance array disLut

idx	**0**	**2**	**3**	**4**	**5**	**6**	**8**	**10**	**11**	**12**	**13**	**14**
disLut[idx]	8	8	8	4	4	2	0	−2	−4	−4	−8	−8
idx	**16**	**18**	**19**	**20**	**21**	**22**	**24**	**26**	**27**	**28**	**29**	**30**
disLut[idx]	−8	−8	−8	−4	−4	−2	0	2	4	4	8	8

8.5.7.3 Motion vector storing process for geometric partitioning mode

This process is invoked when decoding a coding unit with MergeGpmFlag[xCb][yCb] equal to 1.

Inputs to this process are:

– a luma location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples,

– a variable angleIdx specifying the angle index of the geometric partition,

– a variable distanceIdx specifying the distance index of the geometric partition,

– the luma motion vectors in 1/16 fractional-sample accuracy mvA and mvB,

– the reference indices refIdxA and refIdxB,

– the prediction list flags predListFlagA and predListFlagB.

The variables numSbX and numSbY, specifying the number of 4×4 blocks in the current coding block in the horizontal and vertical directions, respectivecly, are set equal to cbWidth >> 2 and cbHeight >> 2, respectively.

The variables, displacementX, displacementY, isFlip and shiftHor are derived as follows:

$$\text{displacementX} = \text{angleIdx} \tag{1011}$$

$$\text{displacementY} = (\ \text{angleIdx} + 8\ )\ \%\ 32 \tag{1012}$$

$$\text{isFlip} = (\ \text{angleIdx} >= 13\ \&\&\ \text{angleIdx} <= 27\ )\ ?\ 1 : 0 \tag{1013}$$

$$\text{shiftHor} = (\ \text{angleIdx}\ \%\ 16 == 8\ ||\ (\ \text{angleIdx}\ \%\ 16\ != 0\ \&\&\ \text{cbHeight} >= \text{cbWidth}\ )\ )\ ?\ 0 : 1 \tag{1014}$$

The variables offsetX and offsetY are derived as follows:

– If shiftHor is equal to 0, the following applies:

$$\text{offsetX} = (\ -\text{cbWidth}\ ) >> 1 \tag{1015}$$

$$\begin{aligned}\text{offsetY} = {} & (\ (\ -\text{cbHeight}\ ) >> 1\ ) + \\ & (\ \text{angleIdx} < 16\ ?\ (\ \text{distanceIdx} * \text{cbHeight}\ ) >> 3 : -(\ (\ \text{distanceIdx} * \text{cbHeight}\ ) >> 3\ )\ )\end{aligned} \tag{1016}$$

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

$$\begin{aligned}\text{offsetX} = {} & (\ (\ -\text{cbWidth}\ ) >> 1\ ) + \\ & (\ \text{angleIdx} < 16\ ?\ (\ \text{distanceIdx} * \text{cbWidth}\ ) >> 3 : -(\ (\ \text{distanceIdx} * \text{cbWidth}\ ) >> 3\ )\ )\end{aligned} \tag{1017}$$

$$\text{offsetY} = (\ -\text{cbHeight}\ ) >> 1 \tag{1018}$$

For each 4×4 subblock at subblock index (xSbIdx, ySbIdx) with xSbIdx = 0..numSbX − 1, and ySbIdx = 0..numSbY − 1, the following applies:

– The variable motionIdx is calculated based on the array disLut specified in Table 37 as following:

$$\begin{aligned}\text{motionIdx} = {} & (\ (\ (\ 4 * \text{xSbIdx} + \text{offsetX}\ ) << 1\ ) + 5\ ) * \text{disLut}[\ \text{displacementX}\ ] + \\ & (\ (\ (\ 4 * \text{ySbIdx} + \text{offsetY}\ ) << 1\ ) + 5\ ) * \text{disLut}[\ \text{displacementY}\ ]\end{aligned} \tag{1019}$$

– The variable sType is derived as follows:

$$\text{sType} = \text{Abs}(\ \text{motionIdx}\ ) < 32\ ?\ 2 : (\ \text{motionIdx} <= 0\ ?\ (\ 1 - \text{isFlip}\ ) : \text{isFlip}\ ) \tag{1020}$$

– Depending on the value of sType, the following assignments are made:

  – If sType is equal to 0, the following applies:

$$\text{predFlagL0} = (\ \text{predListFlagA} == 0\ )\ ?\ 1 : 0 \tag{1021}$$

$$\text{predFlagL1} = (\ \text{predListFlagA} == 0\ )\ ?\ 0 : 1 \tag{1022}$$

$$\text{refIdxL0} = (\ \text{predListFlagA} == 0\ )\ ?\ \text{refIdxA} : -1 \tag{1023}$$

$$\text{refIdxL1} = (\ \text{predListFlagA} == 0\ )\ ?\ -1 : \text{refIdxA} \tag{1024}$$

$$\text{mvL0}[\ 0\ ] = (\ \text{predListFlagA} == 0\ )\ ?\ \text{mvA}[\ 0\ ] : 0 \tag{1025}$$

$$mvL0[1] = (predListFlagA == 0) ? mvA[1] : 0 \quad (1026)$$

$$mvL1[0] = (predListFlagA == 0) ? 0 : mvA[0] \quad (1027)$$

$$mvL1[1] = (predListFlagA == 0) ? 0 : mvA[1] \quad (1028)$$

– Otherwise, if sType is equal to 1 or (sType is equal to 2 and predListFlagA + predListFlagB is not equal to 1), the following applies:

$$predFlagL0 = (predListFlagB == 0) ? 1 : 0 \quad (1029)$$

$$predFlagL1 = (predListFlagB == 0) ? 0 : 1 \quad (1030)$$

$$refIdxL0 = (predListFlagB == 0) ? refIdxB : -1 \quad (1031)$$

$$refIdxL1 = (predListFlagB == 0) ? -1 : refIdxB \quad (1032)$$

$$mvL0[0] = (predListFlagB == 0) ? mvB[0] : 0 \quad (1033)$$

$$mvL0[1] = (predListFlagB == 0) ? mvB[1] : 0 \quad (1034)$$

$$mvL1[0] = (predListFlagB == 0) ? 0 : mvB[0] \quad (1035)$$

$$mvL1[1] = (predListFlagB == 0) ? 0 : mvB[1] \quad (1036)$$

– Otherwise (sType is equal to 2 and predListFlagA + predListFlagB is equal to 1), the following applies:

$$predFlagL0 = 1 \quad (1037)$$

$$predFlagL1 = 1 \quad (1038)$$

$$refIdxL0 = (predListFlagA == 0) ? refIdxA : refIdxB \quad (1039)$$

$$refIdxL1 = (predListFlagA == 0) ? refIdxB : refIdxA \quad (1040)$$

$$mvL0[0] = (predListFlagA == 0) ? mvA[0] : mvB[0] \quad (1041)$$

$$mvL0[1] = (predListFlagA == 0) ? mvA[1] : mvB[1] \quad (1042)$$

$$mvL1[0] = (predListFlagA == 0) ? mvB[0] : mvA[0] \quad (1043)$$

$$mvL1[1] = (predListFlagA == 0) ? mvB[1] : mvA[1] \quad (1044)$$

– The following assignments are made for x = xCb..xCb + 3 and y = yCb..yCb + 3:

$$MvL0[(xSbIdx << 2) + x][(ySbIdx << 2) + y] = mvL0 \quad (1045)$$

$$MvL1[(xSbIdx << 2) + x][(ySbIdx << 2) + y] = mvL1 \quad (1046)$$

$$MvDmvrL0[(xSbIdx << 2) + x][(ySbIdx << 2) + y] = mvL0 \quad (1047)$$

$$MvDmvrL1[(xSbIdx << 2) + x][(ySbIdx << 2) + y] = mvL1 \quad (1048)$$

$$RefIdxL0[(xSbIdx << 2) + x][(ySbIdx << 2) + y] = refIdxL0 \quad (1049)$$

$$RefIdxL1[(xSbIdx << 2) + x][(ySbIdx << 2) + y] = refIdxL1 \quad (1050)$$

$$PredFlagL0[(xSbIdx << 2) + x][(ySbIdx << 2) + y] = predFlagL0 \quad (1051)$$

$$PredFlagL1[(xSbIdx << 2) + x][(ySbIdx << 2) + y] = predFlagL1 \quad (1052)$$

$$BcwIdx[(xSbIdx << 2) + x][(ySbIdx << 2) + y] = 0 \quad (1053)$$

8.5.8 Decoding process for the residual signal of coding blocks coded in inter prediction mode

Inputs to this process are:

– a sample location (xTb0, yTb0) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture,
– a variable nCbW specifying the width of the current coding block,
– a variable nCbH specifying the height of the current coding block,
– a variable nTbW specifying the width of the current transform block,
– a variable nTbH specifying the height of the current transform block,
– a variable cIdx specifying the colour component of the current block.

Output of this process is an (nTbW)x(nTbH) array resSamples.

The maximum transform block width maxTbWidth and height maxTbHeight are derived as follows:

$$\text{maxTbWidth} = (\text{ cIdx } == 0\text{ }) \text{ ? MaxTbSizeY : MaxTbSizeY / SubWidthC} \quad (1054)$$

$$\text{maxTbHeight} = (\text{ cIdx } == 0\text{ }) \text{ ? MaxTbSizeY : MaxTbSizeY / SubHeightC} \quad (1055)$$

The luma sample location is derived as follows:

$$(\text{ xTbY, yTbY }) = (\text{ cIdx } == 0\text{ }) \text{ ? (xTb0, yTb0) : (xTb0 * SubWidthC, yTb0 * SubHeightC)} \quad (1056)$$

Depending on maxTbWidth and maxTbHeight, the following applies:

– If nTbW is greater than maxTbWidth or nTbH is greater than maxTbHeight, the following ordered steps apply:

1. The variables verSplitFirst, newTbW and newTbH are derived as follows:

$$\text{verSplitFirst} = (\text{ nTbW * (cIdx } == 0 \text{ ? 1 : SubWidthC) > nTbH *(cIdx } == 0 \text{ ? 1 : SubHeightC))} \;\&\&\; (\text{ nTbW > maxTbWidth })\quad (1057)$$

$$\text{newTbW} = \text{verSplitFirst ? (nTbW / 2) : nTbW} \quad (1058)$$

$$\text{newTbH} = \text{!verSplitFirst ? (nTbH / 2) : nTbH} \quad (1059)$$

2. The decoding process for the residual signal of coding blocks coded in inter prediction mode as specified in this clause is invoked with (xTb0, yTb0), nCbW, nCbH, the transform block width nTbW set equal to newTbW and the height nTbH set equal to newTbH, and cIdx as inputs, and the output is assigned to the array resSamples[x][y] with x = 0..newTbW − 1, y = 0..newTbH − 1.

3. The following applies:

   – If verSplitFirst is equal to TRUE, the decoding process for the residual signal of coding blocks coded in inter prediction mode as specified in this clause is invoked with the location (xTb0, yTb0) set equal to (xTb0 + newTbW, yTb0), nCbW, nCbH, the transform block width nTbW set equal to newTbW and the height nTbH set equal to newTbH, and cIdx as inputs, and the output is assigned to the array resSamples[x][y] with x = newTbW..nTbW − 1, y = 0..newTbH − 1.

   – Otherwise (verSplitFirst is equal to FALSE), the decoding process for the residual signal of coding blocks coded in inter prediction mode as specified in this clause is invoked with the location (xTb0, yTb0) set equal to (xTb0, yTb0 + newTbH), nCbW, nCbH, the transform block width nTbW set equal to newTbW and the height nTbH set equal to newTbH, and cIdx as inputs, and the output is assigned to the array resSamples[x][y] with x = 0..newTbW − 1, y = newTbH..nTbH − 1.

– Otherwise, if cu_sbt_flag is equal to 1, the following applies:

  – The variables sbtMinNumFourths, wPartIdx and hPartIdx are derived as follows:

$$\text{sbtMinNumFourths} = \text{cu\_sbt\_quad\_flag ? 1 : 2} \quad (1060)$$

$$\text{wPartIdx} = \text{cu\_sbt\_horizontal\_flag ? 4 : sbtMinNumFourths} \quad (1061)$$

$$\text{hPartIdx} = \text{!cu\_sbt\_horizontal\_flag ? 4 : sbtMinNumFourths} \quad (1062)$$

– The variables xPartIdx and yPartIdx are derived as follows:

  – If cu_sbt_pos_flag is equal to 0, xPartIdx and yPartIdx are set equal to 0.

  – Otherwise (cu_sbt_pos_flag is equal to 1), the variables xPartIdx and yPartIdx are derived as follows:

$$\text{xPartIdx} = \text{cu\_sbt\_horizontal\_flag} \ ? \ 0 : (4 - \text{sbtMinNumFourths}) \tag{1063}$$

$$\text{yPartIdx} = !\text{cu\_sbt\_horizontal\_flag} \ ? \ 0 : (4 - \text{sbtMinNumFourths}) \tag{1064}$$

– The variables xTbYSub, yTbYSub, xTb0Sub, yTb0Sub, nTbWSub and nTbHSub are derived as follows:

$$\text{xTbYSub} = \text{xTbY} + (\text{nTbW} * ((\text{cIdx} == 0) \ ? \ 1 : \text{SubWidthC}) * \text{xPartIdx} / 4) \tag{1065}$$

$$\text{yTbYSub} = \text{yTbY} + (\text{nTbH} * ((\text{cIdx} == 0) \ ? \ 1 : \text{SubHeightC}) * \text{yPartIdx} / 4) \tag{1066}$$

$$\text{xTb0Sub} = \text{nTbW} * \text{xPartIdx} / 4 \tag{1067}$$

$$\text{yTb0Sub} = \text{nTbH} * \text{yPartIdx} / 4 \tag{1068}$$

$$\text{nTbWSub} = \text{nTbW} * \text{wPartIdx} / 4 \tag{1069}$$

$$\text{nTbHSub} = \text{nTbH} * \text{hPartIdx} / 4 \tag{1070}$$

– The scaling and transformation process as specified in clause 8.7.2 is invoked with the luma location (xTbYSub, yTbYSub), the variable cIdx, the variable predMode set equal to MODE_INTER, nCbW, nCbH, nTbWSub and nTbHSub as inputs, and the output is an (nTbWSub)x(nTbHSub) array resTbSamples.

– The residual samples resSamples[x][y] with x = 0..nTbW − 1, y = 0..nTbH − 1 are set equal to 0.

– The residual samples resSamples[x][y] with x = xTb0Sub..xTb0Sub + nTbWSub − 1, y = yTb0Sub..yTb0Sub + nTbHSub − 1 are derived as follows:

$$\text{resSamples}[x][y] = \text{resTbSamples}[x - \text{xTb0Sub}][y - \text{yTb0Sub}] \tag{1071}$$

– Otherwise, the scaling and transformation process as specified in clause 8.7.2 is invoked with the luma location (xTbY, yTbY), the variable cIdx, the variable predMode set equal to MODE_INTER, nCbW, nCbH, the transform width nTbW and the transform height nTbH as inputs, and the output is an (nTbW)x(nTbH) array resSamples.

8.5.9 Decoding process for the reconstructed signal of chroma coding blocks coded in inter prediction mode

Inputs to this process are:

– a sample location (xTb0, yTb0) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture,

– a variable nTbW specifying the width of the current transform block,

– a variable nTbH specifying the height of the current transform block,

– a variable cIdx specifying the colour component of the current block,

– an (nTbW)x(nTbH) array predSamples specifying the prediction samples of the current block,

– an (nTbW)x(nTbH) array resSamples specifying the residual samples of the current block,

Output of this process is a modified reconstructed picture before in-loop filtering.

The maximum transform block width maxTbWidth and height maxTbHeight are derived as follows:

$$\text{maxTbWidth} = \text{MaxTbSizeY} / \text{SubWidthC} \tag{1072}$$

$$\text{maxTbHeight} = \text{MaxTbSizeY} / \text{SubHeightC} \tag{1073}$$

Depending on maxTbWidth and maxTbHeight, the following applies:

– If nTbW is greater than maxTbWidth or nTbH is greater than maxTbHeight, the following ordered steps apply:

  1. The variables verSplitFirst, newTbW and newTbH are derived as follows:

$$verSplitFirst = (nTbW * SubWidthC > nTbH *SubHeightC) \ \&\& \ (nTbW > maxTbWidth) \quad (1074)$$

$$newTbW = verSplitFirst ? (nTbW / 2) : nTbW \quad (1075)$$

$$newTbH = !verSplitFirst ? (nTbH / 2) : nTbH \quad (1076)$$

2. The decoding process for the reconstructed signal of chroma coding blocks coded in inter prediction mode as specified in this clause is invoked with (xTb0, yTb0), the transform block width nTbW set equal to newTbW and the height nTbH set equal to newTbH, cIdx, the (newTbW)x(newTbH) array predSamples set equal to predSamples[x][y], and the (newTbW)x(newTbH) array resSamples set equal to resSamples[x][y] with x = 0..newTbW − 1, y = 0..newTbH − 1 as inputs, and the output is a modified reconstructed picture before in-loop filtering.

3. The following applies:

   – If verSplitFirst is equal to TRUE, the decoding process for the reconstructed signal of chroma coding blocks coded in inter prediction mode as specified in this clause is invoked with the location (xTb0, yTb0) set equal to (xTb0 + newTbW, yTb0), the transform block width nTbW set equal to newTbW and the height nTbH set equal to newTbH, cIdx, the (newTbW)x(newTbH) array predSamples set equal to predSamples[x][y], and the (newTbW)x(newTbH) array resSamples set equal to resSamples[x][y] with x = newTbW..2 * newTbW − 1, y = 0..newTbH − 1 as inputs, and the output is a modified reconstructed picture before in-loop filtering.

   – Otherwise (verSplitFirst is equal to FALSE), the process for the reconstructed signal of chroma coding blocks coded in inter prediction mode as specified in this clause is invoked with the location (xTb0, yTb0) set equal to (xTb0, yTb0 + newTbH), the transform block width nTbW set equal to newTbW and the height nTbH set equal to newTbH, cIdx, the (newTbW)x(newTbH) array predSamples set equal to predSamples[x][y], and the (newTbW)x(newTbH) array resSamples set equal to resSamples[x][y] with x = 0..newTbW − 1, y = newTbH..2 * newTbH − 1 as inputs, and the output is a modified reconstructed picture before in-loop filtering.

– Otherwise, if cu_sbt_flag is equal to 1, the following applies:

  – The variables nTb0W, nTb0H, nTb1W, nTb1H, xTb1, and yTb1 are derived as follows:

$$nTb0W = !cu\_sbt\_horizontal\_flag ? (nTbW * SbtNumFourthsTb0 / 4) : nTbW \quad (1077)$$

$$nTb0H = !cu\_sbt\_horizontal\_flag ? nTbH : (nTbH * SbtNumFourthsTb0 / 4) \quad (1078)$$

$$nTb1W = nTbW − (!cu\_sbt\_horizontal\_flag ? nTb0W : 0) \quad (1079)$$

$$nTb1H = nTbH − (!cu\_sbt\_horizontal\_flag ? 0 : nTb0H) \quad (1080)$$

$$xTb1 = xTb0 + (!cu\_sbt\_horizontal\_flag ? nTb0W : 0) \quad (1081)$$

$$yTb1 = yTb0 + (!cu\_sbt\_horizontal\_flag ? 0 : nTb0H) \quad (1082)$$

  – The picture reconstruction process for a colour component as specified in clause 8.7.5.1 is invoked with the block location (xCurr, yCurr) set equal to (xTb0, yTb0), the block width nCurrSw set equal to nTb0W, the block height nCurrSh set equal to nTb0H, cIdx, the (nTb0W)x(nTb0H) array predSamples set equal to predSamples[x][y], and the (nTb0W)x(nTb0H) array resSamples set equal to resSamples[x][y] with x = 0..nTb0W − 1, y = 0..nTb0H − 1 as inputs, and the output is a modified reconstructed picture before in-loop filtering.

  – The picture reconstruction process for a colour component as specified in clause 8.7.5.1 is invoked with the block location (xCurr, yCurr) set equal to (xTb1, yTb1), the block width nCurrSw set equal to nTb1W, the block height nCurrSh set equal to nTb1H, cIdx, the (nTb1W)x(nTb1H) array predSamples set equal to predSamples[xTb1 − xTb0 + x][yTb1 − yTb0 + y], and the (nTb1W)x(nTb1H) array resSamples set equal to resSamples[xTb1 − xTb0 + x][yTb1 − yTb0 + y] with x = 0..nTb1W − 1, y = 0..nTb1H − 1 as inputs, and the output is a modified reconstructed picture before in-loop filtering.

– Otherwise, picture reconstruction process for a colour component as specified in clause 8.7.5.1 is invoked with the block location (xCurr, yCurr) set equal to (xTb0, yTb0), the block width nCurrSw set equal to nTbW, the block height nCurrSh set equal to nTbH, cIdx, the (nTbW)x(nTbH) array predSamples, and the (nTbW)x(nTbH) array resSamples as inputs, and the output is a modified reconstructed picture before in-loop filtering.

8.6 Decoding process for coding units coded in IBC prediction mode

8.6.1 General decoding process for coding units coded in IBC prediction mode

Inputs to this process are:

– a luma location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples,

– a variable treeType specifying whether a single or a dual tree is used and if a dual tree is used, it specifies whether the current tree corresponds to the luma or chroma components.

Output of this process is a modified reconstructed picture before in-loop filtering.

The derivation process for quantization parameters as specified in clause 8.7.1 is invoked with the luma location (xCb, yCb), the width of the current coding block in luma samples cbWidth and the height of the current coding block in luma samples cbHeight, and the variable treeType as inputs.

The variable IsGt4by4 is derived as follows:

$$\text{IsGt4by4} = (\text{cbWidth} * \text{cbHeight}) > 16 \qquad (1083)$$

The decoding process for coding units coded in IBC prediction mode consists of the following ordered steps:

1. The block vector components of the current coding unit are derived as follows:

   – The derivation process for block vector components as specified in clause 8.6.2.1 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth and the luma coding block height cbHeight as inputs, and the luma block vector bvL as output.

   – When treeType is equal to SINGLE_TREE and sps_chroma_format_idc is not equal to 0, the derivation process for chroma block vectors in clause 8.6.2.5 is invoked with luma block vector bvL as input, and chroma block vector bvC as output.

2. The prediction samples of the current coding unit are derived as follows:

   – The decoding process for IBC blocks as specified in clause 8.6.3 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth and the luma coding block height cbHeight, the luma block vector bvL, the variable cIdx set equal to 0 as inputs, and the IBC prediction samples (predSamples) that are an (cbWidth)x(cbHeight) array predSamples_L of prediction luma samples as outputs.

   – When treeType is equal to SINGLE_TREE and sps_chroma_format_idc is not equal to 0, the prediction samples of the current coding unit are derived as follows:

     – The decoding process for IBC blocks as specified in clause 8.6.3 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth and the luma coding block height cbHeight, the chroma block vector bvC and the variable cIdx set equal to 1 as inputs, and the IBC prediction samples (predSamples) that are an (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array predSamples_{Cb} of prediction chroma samples for the chroma components Cb as outputs.

     – The decoding process for IBC blocks as specified in clause 8.6.3 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth and the luma coding block height cbHeight, the chroma block vector bvC and the variable cIdx set equal to 2 as inputs, and the IBC prediction samples (predSamples) that are an (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array predSamples_{Cr} of prediction chroma samples for the chroma components Cr as outputs.

3. The residual samples of the current coding unit are derived as follows:

   – The decoding process for the residual signal of coding blocks coded in inter prediction mode as specified in clause 8.5.8 is invoked with the location (xTb0, yTb0) set equal to the luma location (xCb, yCb), the width nCbW set equal to the luma coding block width cbWidth, the height nCbH set equal to the luma coding block height cbHeight, the width nTbW set equal to the luma coding block width cbWidth, the height nTbH set equal to the luma coding block height cbHeight and the variable cIdx set equal to 0 as inputs, and the array resSamples_L as output.

   – When treeType is equal to SINGLE_TREE and sps_chroma_format_idc is not equal to 0, the decoding process for the residual signal of coding blocks coded in inter prediction mode as specified in clause 8.5.8 is invoked with the location (xTb0, yTb0) set equal to the chroma location (xCb / SubWidthC, yCb / SubHeightC), the width nCbW set equal to the chroma coding block width cbWidth / SubWidthC, the height nCbH set equal to the chroma coding block height cbHeight / SubHeightC, the width nTbW set equal to the chroma coding block width cbWidth / SubWidthC, the height nTbH set equal to the chroma coding block height cbHeight / SubHeightC and the variable cIdx set equal to 1 as inputs, and the array $resSamples_{Cb}$ as output.

– When treeType is equal to SINGLE_TREE and sps_chroma_format_idc is not equal to 0, the decoding process for the residual signal of coding blocks coded in inter prediction mode as specified in clause 8.5.8 is invoked with the location (xTb0, yTb0) set equal to the chroma location (xCb / SubWidthC, yCb / SubHeightC), the width nCbW set equal to the chroma coding block width cbWidth / SubWidthC, the height nCbH set equal to the chroma coding block height cbHeight / SubHeightC, the width nTbW set equal to the chroma coding block width cbWidth / SubWidthC, the height nTbH set equal to the chroma coding block height cbHeight / SubHeightC and the variable cIdx set equal to 2 as inputs, and the array $resSamples_{Cr}$ as output.

– When cu_act_enabled_flag[xCb][yCb] is equal to 1, the residual modification process for residual blocks using colour space conversion as specified in clause 8.7.4.6 is invoked with the variable nTbW set equal to cbWidth, the variable nTbH set equal to cbHeight, the array r_Y set equal to $resSamples_L$, the array r_{Cb} set equal to $resSamples_{Cb}$, and the array r_{Cr} set equal to $resSamples_{Cr}$ as inputs, and the output are modified versions of the arrays $resSamples_L$, $resSamples_{Cb}$ and $resSamples_{Cr}$.

4. The reconstructed samples of the current coding unit are derived as follows:

   – The picture reconstruction process for a colour component as specified in clause 8.7.5.1 is invoked with the block location (xCurr, yCurr) set equal to (xCb, yCb), the block width nCurrSw set equal to cbWidth, the block height nCurrSh set equal to cbHeight, the variable cIdx set equal to 0, the (cbWidth)x(cbHeight) array predSamples set equal to $predSamples_L$ and the (cbWidth)x(cbHeight) array resSamples set equal to $resSamples_L$ as inputs, and the output is a modified reconstructed picture before in-loop filtering.

   – When treeType is equal to SINGLE_TREE and sps_chroma_format_idc is not equal to 0, the decoding process for the reconstructed signal of chroma coding blocks coded in inter prediction mode as specified in clause 8.5.9 is invoked with the transform block location (xTb0, yTb0) set equal to (xCb / SubWidthC, yCb / SubHeightC), the transform block width nTbW set equal to cbWidth / SubWidthC, the transform block height nTbH set equal to cbHeight / SubHeightC, the variable cIdx set equal to 1, the (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array predSamples set equal to $predSamples_{Cb}$ and the (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array resSamples set equal to $resSamples_{Cb}$ as inputs, and the output is a modified reconstructed picture before in-loop filtering.

   – When treeType is equal to SINGLE_TREE and sps_chroma_format_idc is not equal to 0, the decoding process for the reconstructed signal of chroma coding blocks coded in inter prediction mode as specified in clause 8.5.9 is invoked with the transform block location (xTb0, yTb0) set equal to (xCb / SubWidthC, yCb / SubHeightC), the transform block width nTbW set equal to cbWidth / SubWidthC, the transform block height nTbH set equal to cbHeight / SubHeightC, the variable cIdx set equal to 2, the (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array predSamples set equal to $predSamples_{Cr}$ and the (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array resSamples set equal to $resSamples_{Cr}$ as inputs, and the output is a modified reconstructed picture before in-loop filtering.

8.6.2 Derivation process for block vector components for IBC blocks

8.6.2.1 General

Inputs to this process are:

– a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples.

Outputs of this process are:

– the luma block vector in 1/16 fractional-sample accuracy bvL.

The luma block vector bvL is derived as follows:

– The derivation process for IBC luma block vector prediction as specified in clause 8.6.2.2 is invoked with the luma location (xCb, yCb), the variables cbWidth and cbHeight as inputs, and the output being the luma block vector bvL.

– When general_merge_flag[xCb][yCb] is equal to 0, the following applies:

1. The variable bvd is derived as follows:

$$bvd[\ 0\ ] = MvdL0[\ xCb\ ][\ yCb\ ][\ 0\ ] \quad (1084)$$

$$bvd[\ 1\ ] = MvdL0[\ xCb\ ][\ yCb\ ][\ 1\ ] \quad (1085)$$

2. The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to bvL, rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded bvL as output.

3. The luma block vector bvL is modified as follows:

$$u[\ 0\ ] = (\ bvL[\ 0\ ] + bvd[\ 0\ ]\ )\ \&\ (\ 2^{18} - 1\ ) \quad (1086)$$

$$bvL[\ 0\ ] = (\ u[\ 0\ ]\ >=\ 2^{17}\ )\ ?\ (\ u[\ 0\ ] - 2^{18}\ ) : u[\ 0\ ] \quad (1087)$$

$$u[\ 1\ ] = (\ bvL[\ 1\ ] + bvd[\ 1\ ]\ )\ \&\ (\ 2^{18} - 1\ ) \quad (1088)$$

$$bvL[\ 1\ ] = (\ u[\ 1\ ]\ >=\ 2^{17}\ )\ ?\ (\ u[\ 1\ ] - 2^{18}\ ) : u[\ 1\ ] \quad (1089)$$

NOTE – The resulting values of bvL[0] and bvL[1] are in the range of -2^{17} to $2^{17} - 1$, inclusive.

When IsGt4by4 is equal to TRUE, the updating process for the history-based block vector predictor list as specified in clause 8.6.2.6 is invoked with luma block vector bvL.

It is a requirement of bitstream conformance that the luma block vector bvL shall obey the following constraints:

– CtbSizeY is greater than or equal to ((yCb + (bvL[1] >> 4)) & (CtbSizeY − 1)) + cbHeight.

– IbcVirBuf[0][(x + (bvL[0] >> 4)) & (IbcBufWidthY − 1)][(y + (bvL[1] >> 4)) & (CtbSizeY − 1)] shall not be equal to −1 for x = xCb..xCb + cbWidth − 1 and y = yCb..yCb + cbHeight − 1.

8.6.2.2 Derivation process for IBC luma block vector prediction

This process is only invoked when CuPredMode[0][xCb][yCb] is equal to MODE_IBC, where (xCb, yCb) specify the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture.

Inputs to this process are:

– a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples.

Outputs of this process are:

– the luma block vector in 1/16 fractional-sample accuracy bvL.

The luma block vector bvL is derived by the following ordered steps:

1. When IsGt4by4 is equal to TRUE, the derivation process for spatial block vector candidates from neighbouring coding units as specified in clause 8.6.2.3 is invoked with the luma coding block location (xCb, yCb), the luma coding block width cbWidth and height cbHeight as inputs, and the outputs being the availability flags $availableFlagA_1$, $availableFlagB_1$ and the block vectors bvA_1 and bvB_1.

2. When IsGt4by4 is equal to TRUE, the block vector candidate list, bvCandList, is constructed as follows:

$$\begin{array}{l} i = 0 \\ \text{if}(\ availableFlagA_1\ ) \\ \quad bvCandList\ [\ i{+}{+}\ ] = bvA_1 \\ \text{if}(\ availableFlagB_1\ ) \\ \quad bvCandList\ [\ i{+}{+}\ ] = bvB_1 \end{array} \quad (1090)$$

3. The variable numCurrCand is derived as follows:

   – If IsGt4by4 is equal to TRUE, numCurrCand is set equal to the number of merging candidates in the bvCandList.

– Otherwise (IsGt4by4 is equal to FALSE), numCurrCand is set equal to 0.

4. When numCurrCand is less than MaxNumIbcMergeCand and NumHmvpIbcCand is greater than 0, the derivation process of IBC history-based block vector candidates as specified in clause 8.6.2.4 is invoked with bvCandList, and numCurrCand as inputs, and modified bvCandList and numCurrCand as outputs.

5. When numCurrCand is less than MaxNumIbcMergeCand, the following applies until numCurrCand is equal to MaxNumIbcMergeCand:

   – bvCandList[numCurrCand][0] is set equal to 0.

   – bvCandList[numCurrCand][1] is set equal to 0.

   – numCurrCand is increased by 1.

6. The variable bvIdx is derived as follows:

bvIdx = general_merge_flag[xCb][yCb] ? merge_idx[xCb][yCb] : mvp_l0_flag[xCb][yCb] (1091)

7. The following assignments are made:

bvL[0] = bvCandList[bvIdx][0] (1092)

bvL[1] = bvCandList[bvIdx][1] (1093)

8.6.2.3 Derivation process for IBC spatial block vector candidates

Inputs to this process are:

– a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples.

Outputs of this process are as follows:

– the availability flags availableFlagA_1 and availableFlagB_1 of the neighbouring coding units,

– the block vectors in 1/16 fractional-sample accuracy bvA_1, and bvB_1 of the neighbouring coding units.

For the derivation of availableFlagA_1 and mvA_1 the following applies:

– The luma location (xNbA_1, yNbA_1) inside the neighbouring luma coding block is set equal to (xCb − 1, yCb + cbHeight − 1).

– The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring luma location (xNbA_1, yNbA_1), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag availableA_1.

– The variables availableFlagA_1 and bvA_1 are derived as follows:

   – If availableA_1 is equal to FALSE, availableFlagA_1 is set equal to 0 and both components of bvA_1 are set equal to 0.

   – Otherwise, availableFlagA_1 is set equal to 1 and the following assignments are made:

bvA_1 = MvL0[xNbA_1][yNbA_1] (1094)

For the derivation of availableFlagB_1 and bvB_1 the following applies:

– The luma location (xNbB_1, yNbB_1) inside the neighbouring luma coding block is set equal to (xCb + cbWidth − 1, yCb − 1).

– The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring luma location (xNbB_1, yNbB_1), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag availableB_1.

– The variables availableFlagB_1 and bvB_1 are derived as follows:

– If one or more of the following conditions are true, availableFlagB_1 is set equal to 0 and both components of bvB_1 are set equal to 0:

  – availableB_1 is equal to FALSE.

  – availableA_1 is equal to TRUE and the luma locations (xNbA_1, yNbA_1) and (xNbB_1, yNbB_1) have the same block vectors.

– Otherwise, availableFlagB_1 is set equal to 1 and the following assignments are made:

$$bvB_1 = MvL0[xNbB_1][yNbB_1] \quad (1095)$$

8.6.2.4 Derivation process for IBC history-based block vector candidates

Inputs to this process are:

– a block vector candidate list bvCandList,

– the number of available block vector candidates in the list numCurrCand.

Outputs of this process are:

– the modified block vector candidate list bvCandList,

– the modified number of motion vector candidates in the list numCurrCand.

For each candidate in HmvpIbcCandList[NumHmvpIbcCand − hMvpIdx] with index hMvpIdx = 1..NumHmvpIbcCand, the following ordered steps are repeated until numCurrCand is equal to MaxNumIbcMergeCand:

1. The variable sameMotion is derived as follows:

   – If all of the following conditions are true for any block vector candidate N with N being A_1 or B_1, sameMotion is set equal to TRUE:

     – IsGt4by4 is equal to TRUE.

     – hMvpIdx is equal to 1.

     – The candidate HmvpIbcCandList[NumHmvpIbcCand − hMvpIdx] is equal to the block vector candidate N.

   – Otherwise, sameMotion is set equal to FALSE.

2. When sameMotion is equal to FALSE, the candidate HmvpIbcCandList[NumHmvpIbcCand − hMvpIdx] is added to the block vector candidate list as follows:

$$bvCandList[numCurrCand++] = HmvpIbcCandList[NumHmvpIbcCand - hMvpIdx] \quad (1096)$$

8.6.2.5 Derivation process for chroma block vectors

Input to this process is:

– a luma block vector in 1/16 fractional-sample accuracy bvL.

Output of this process is a chroma block vector in 1/32 fractional-sample accuracy bvC.

A chroma block vector is derived from the corresponding luma block vector.

The chroma block vector bvC is derived as follows:

$$bvC[0] = (bvL[0] >> (3 + SubWidthC)) * 32 \quad (1097)$$

$$bvC[1] = (bvL[1] >> (3 + SubHeightC)) * 32 \quad (1098)$$

8.6.2.6 Updating process for the history-based block vector predictor candidate list

Inputs to this process are:

– luma block vector bvL in 1/16 fractional-sample accuracy.

The candidate list HmvpIbcCandList is modified by the following ordered steps:

1. The variable identicalCandExist is set equal to FALSE and the variable removeIdx is set equal to 0.

2. When NumHmvpIbcCand is greater than 0, for each index hMvpIdx with hMvpIdx = 0..NumHmvpIbcCand − 1, the following steps apply until identicalCandExist is equal to TRUE:
   – When bvL is equal to HmvpIbcCandList[hMvpIdx], identicalCandExist is set equal to TRUE and removeIdx is set equal to hMvpIdx.
3. The candidate list HmvpIbcCandList is updated as follows:
   – If identicalCandExist is equal to TRUE or NumHmvpIbcCand is equal to 5, the following applies:
     – For each index i with i = (removeIdx + 1)..(NumHmvpIbcCand − 1), HmvpIbcCandList[i − 1] is set equal to HmvpIbcCandList [i].
     – HmvpIbcCandList[NumHmvpIbcCand − 1] is set equal to bvL.
   – Otherwise (identicalCandExist is equal to FALSE and NumHmvpIbcCand is less than 5), the following applies:
     – HmvpIbcCandList[NumHmvpIbcCand ++] is set equal to bvL.

8.6.3 Decoding process for IBC blocks

This process is invoked when decoding a coding unit coded in IBC prediction mode.

Inputs to this process are:

– a luma location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left luma sample of the current picture,
– a variable cbWidth specifying the width of the current coding block in luma samples,
– a variable cbHeight specifying the height of the current coding block in luma samples,
– the block vector bv,
– a variable cIdx specifying the colour component index of the current block.

Outputs of this process are:

– an array predSamples of prediction samples.

When cIdx is equal to 0, for x = xCb..xCb + cbWidth − 1 and y = yCb..yCb + cbHeight − 1, the following applies:

$$xVb = (x + (bv[0] >> 4)) \& (IbcBufWidthY - 1) \quad (1099)$$

$$yVb = (y + (bv[1] >> 4)) \& (CtbSizeY - 1) \quad (1100)$$

$$predSamples[x - xCb][y - yCb] = IbcVirBuf[0][xVb][yVb] \quad (1101)$$

When cIdx is not equal to 0, for x = xCb / SubWidthC..xCb / SubWidthC + cbWidth / SubWidthC − 1 and y = yCb / SubHeightC..yCb / SubHeightC + cbHeight / SubHeightC − 1, the following applies:

$$xVb = (x + (bv[0] >> (3 + SubWidthC))) \& (IbcBufWidthC - 1) \quad (1102)$$

$$yVb = (y + (bv[1] >> (3 + SubHeightC))) \& ((CtbSizeY / SubHeightC) - 1) \quad (1103)$$

$$predSamples[x - (xCb / SubWidthC)][y - (yCb / SubHeightC)] = IbcVirBuf[cIdx][xVb][yVb] \quad (1104)$$

When cIdx is equal to 0, the following assignments are made for x = 0..cbWidth − 1 and y = 0..cbHeight − 1:

$$MvL0[xCb + x][yCb + y] = bv \quad (1105)$$

$$MvL1[xCb + x][yCb + y][0] = 0 \quad (1106)$$

$$MvL1[xCb + x][yCb + y][1] = 0 \quad (1107)$$

$$RefIdxL0[xCb + x][yCb + y] = -1 \quad (1108)$$

$$RefIdxL1[xCb + x][yCb + y] = -1 \quad (1109)$$

$$\text{PredFlagL0}[\ xCb + x\ ][\ yCb + y\ ] = 0 \tag{1110}$$

$$\text{PredFlagL1}[\ xCb + x\ ][\ yCb + y\ ] = 0 \tag{1111}$$

$$\text{BcwIdx}[\ xCb + x\ ][\ yCb + y\ ] = 0 \tag{1112}$$

8.7 Scaling, transformation and array construction process

8.7.1 Derivation process for quantization parameters

Inputs to this process are:

– a luma location (xCb, yCb) specifying the top-left luma sample of the current coding block relative to the top-left luma sample of the current picture,

– a variable cbWidth specifying the width of the current coding block in luma samples,

– a variable cbHeight specifying the height of the current coding block in luma samples,

– a variable treeType specifying whether a single tree (SINGLE_TREE) or a dual tree is used to partition the coding tree node and, when a dual tree is used, whether the luma (DUAL_TREE_LUMA) or chroma components (DUAL_TREE_CHROMA) are currently processed.

In this process, the luma quantization parameter Qp'_Y and the chroma quantization parameters Qp'_{Cb}, Qp'_{Cr} and Qp'_{CbCr} are derived.

The luma location (xQg, yQg), specifies the top-left luma sample of the current quantization group relative to the top-left luma sample of the current picture. The horizontal and vertical positions xQg and yQg are set equal to CuQgTopLeftX and CuQgTopLeftY, respectively.

NOTE – The current quantization group is a rectangular region inside a coding tree block that shares the same qP_{Y_PRED}. Its width and height are equal to the width and height of the coding tree node of which the top-left luma sample position is assigned to the variables CuQgTopLeftX and CuQgTopLeftY.

When treeType is equal to SINGLE_TREE or DUAL_TREE_LUMA, the predicted luma quantization parameter qP_{Y_PRED} is derived by the following ordered steps:

1. The variable qP_{Y_PREV} is derived as follows:
   - If one or more of the following conditions are true, qP_{Y_PREV} is set equal to $SliceQp_Y$:
     - The current quantization group is the first quantization group in a slice.
     - The current quantization group is the first quantization group in a tile.
   - Otherwise, qP_{Y_PREV} is set equal to the luma quantization parameter Qp_Y of the last luma coding unit in the previous quantization group in decoding order.

2. The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring location (xNbY, yNbY) set equal to (xQg − 1, yQg), checkPredModeY set equal to FALSE, and cIdx set equal to 0 as inputs, and the output is assigned to availableA. The variable qP_{Y_A} is derived as follows:
   - If one or more of the following conditions are true, qP_{Y_A} is set equal to qP_{Y_PREV}:
     - availableA is equal to FALSE.
     - The CTB containing the luma coding block covering the luma location (xQg − 1, yQg) is not equal to the CTB containing the current luma coding block at (xCb, yCb), i.e., one or more of the following conditions are true:
       - (xQg − 1) >> CtbLog2SizeY is not equal to (xCb) >> CtbLog2SizeY
       - (yQg) >> CtbLog2SizeY is not equal to (yCb) >> CtbLog2SizeY
   - Otherwise, qP_{Y_A} is set equal to the luma quantization parameter Qp_Y of the coding unit containing the luma coding block covering (xQg − 1, yQg).

3. The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location (xCurr, yCurr) set equal to (xCb, yCb), the neighbouring location (xNbY, yNbY) set equal to (xQg, yQg − 1), checkPredModeY set equal to FALSE, and cIdx set equal to 0 as inputs, and the output is assigned to availableB. The variable qP_{Y_B} is derived as follows:

– If one or more of the following conditions are true, qP_{Y_B} is set equal to qP_{Y_PREV}:

  – availableB is equal to FALSE.

  – The CTB containing the luma coding block covering the luma location (xQg, yQg − 1) is not the CTB containing the current luma coding block at (xCb, yCb), i.e., one or more of the following conditions are true:

    – (xQg) >> CtbLog2SizeY is not equal to (xCb) >> CtbLog2SizeY

    – (yQg − 1) >> CtbLog2SizeY is not equal to (yCb) >> CtbLog2SizeY

– Otherwise, qP_{Y_B} is set equal to the luma quantization parameter Qp_Y of the coding unit containing the luma coding block covering (xQg, yQg − 1).

4. The predicted luma quantization parameter qP_{Y_PRED} is derived as follows:

   – If all the following conditions are true, then qP_{Y_PRED} is set equal to the luma quantization parameter Qp_Y of the coding unit containing the luma coding block covering (xQg, yQg − 1):

     – availableB is equal to TRUE.

     – The current quantization group is the first quantization group in a CTB row within a tile.

   – Otherwise, qP_{Y_PRED} is derived as follows:

$$qP_{Y\_PRED} = (qP_{Y\_A} + qP_{Y\_B} + 1) >> 1 \quad (1113)$$

The variable Qp_Y is derived as follows:

$$Qp_Y = ((qP_{Y\_PRED} + CuQpDeltaVal + 64 + 2 * QpBdOffset) \% (64 + QpBdOffset)) − QpBdOffset \quad (1114)$$

The luma quantization parameter Qp'_Y is derived as follows:

$$Qp'_Y = Qp_Y + QpBdOffset \quad (1115)$$

When sps_chroma_format_idc is not equal to 0 and treeType is equal to SINGLE_TREE or DUAL_TREE_CHROMA, the following applies:

– When treeType is equal to DUAL_TREE_CHROMA, the variable Qp_Y is set equal to the luma quantization parameter Qp_Y of the luma coding unit that covers the luma location (xCb + cbWidth / 2, yCb + cbHeight / 2).

– The variables qP_{Cb}, qP_{Cr} and qP_{CbCr} are derived as follows:

$$qP_{Chroma} = Clip3(−QpBdOffset, 63, Qp_Y) \quad (1116)$$

$$qP_{Cb} = ChromaQpTable[0][qP_{Chroma}] \quad (1117)$$

$$qP_{Cr} = ChromaQpTable[1][qP_{Chroma}] \quad (1118)$$

$$qP_{CbCr} = ChromaQpTable[2][qP_{Chroma}] \quad (1119)$$

– The chroma quantization parameters for the Cb and Cr components, Qp'_{Cb} and Qp'_{Cr}, and joint Cb-Cr coding Qp'_{CbCr} are derived as follows:

$$Qp'_{Cb} = Clip3(−QpBdOffset, 63, qP_{Cb} + pps\_cb\_qp\_offset + sh\_cb\_qp\_offset + CuQpOffset_{Cb}) + QpBdOffset \quad (1120)$$

$$Qp'_{Cr} = Clip3(−QpBdOffset, 63, qP_{Cr} + pps\_cr\_qp\_offset + sh\_cr\_qp\_offset + CuQpOffset_{Cr}) + QpBdOffset \quad (1121)$$

$$Qp'_{CbCr} = Clip3(−QpBdOffset, 63, qP_{CbCr} + pps\_joint\_cbcr\_qp\_offset\_value + sh\_joint\_cbcr\_qp\_offset + CuQpOffset_{CbCr}) + QpBdOffset \quad (1122)$$

8.7.2 Scaling and transformation process

Inputs to this process are:

– a luma location (xTbY, yTbY) specifying the top-left sample of the current luma transform block relative to the top-left luma sample of the current picture,

– a variable cIdx specifying the colour component of the current block,

– a variable predMode specifying the prediction mode of the coding unit,

– a variable nCbW specifying the coding block width,

– a variable nCbH specifying the coding block height,

– a variable nTbW specifying the transform block width,

– a variable nTbH specifying the transform block height.

Output of this process is the (nTbW)x(nTbH) array of residual samples resSamples[x][y] with x = 0..nTbW − 1, y = 0..nTbH − 1.

The variable codedCIdx is derived as follows:

– If cIdx is equal to 0 or TuCResMode[xTbY][yTbY] is equal to 0, codedCIdx is set equal to cIdx.

– Otherwise, if TuCResMode[xTbY][yTbY] is equal to 1 or 2, codedCIdx is set equal to 1.

– Otherwise, codedCIdx is set equal to 2.

The variable cSign is set equal to (1 − 2 * ph_joint_cbcr_sign_flag).

The (nTbW)x(nTbH) array of residual samples resSamples is derived as follows:

1. The scaling process for transform coefficients as specified in clause 8.7.3 is invoked with the transform block location (xTbY, yTbY), the transform block width nTbW and the transform block height nTbH, the prediction mode predMode, and the colour component variable cIdx being set equal to codedCIdx as inputs, and the output is an (nTbW)x(nTbH) array of scaled transform coefficients d.

2. The (nTbW)x(nTbH) array of residual samples res is derived as follows:

   – If transform_skip_flag[xTbY][yTbY][codedCIdx] is equal to 1, the residual sample array values res[x][y] with x = 0..nTbW − 1, y = 0..nTbH − 1 are derived as follows:

     res[x][y] = d[x][y] (1123)

   – Otherwise (transform_skip_flag[xTbY][yTbY][codedCIdx] is equal to 0), the transformation process for scaled transform coefficients as specified in clause 8.7.4.1 is invoked with the transform block location (xTbY, yTbY), the coding block width nCbW and the coding block height nCbH, the transform block width nTbW and the transform block height nTbH, the colour component variable cIdx being set equal to codedCIdx and the (nTbW)x(nTbH) array of scaled transform coefficients d as inputs, and the output is an (nTbW)x(nTbH) array of residual samples res.

3. The residual samples resSamples[x][y] with x = 0..nTbW − 1, y = 0..nTbH − 1 are derived as follows:

   – If cIdx is equal to codedCIdx, the following applies:

     resSamples[x][y] = res[x][y] (1124)

   – Otherwise, if TuCResMode[xTbY][yTbY] is equal to 2, the following applies:

     resSamples[x][y] = cSign * res[x][y] (1125)

   – Otherwise, the following applies:

     resSamples[x][y] = (cSign * res[x][y]) >> 1 (1126)

8.7.3 Scaling process for transform coefficients

Inputs to this process are:

– a luma location (xTbY, yTbY) specifying the top-left sample of the current luma transform block relative to the top-left luma sample of the current picture,

– a variable nTbW specifying the transform block width,

– a variable nTbH specifying the transform block height,

– a variable predMode specifying the prediction mode of the coding unit,

– a variable cIdx specifying the colour component of the current block.

Output of this process is the (nTbW)x(nTbH) array d of scaled transform coefficients with elements d[x][y].

The quantization parameter qP and the variable QpActOffset are derived as follows:

– If cIdx is equal to 0, the following applies:

$$\text{qP} = \text{Qp}'_{\text{Y}} \quad (1127)$$

$$\text{QpActOffset} = \text{cu\_act\_enabled\_flag[xTbY][yTbY] ? } -5 : 0 \quad (1128)$$

– Otherwise, if TuCResMode[xTbY][yTbY] is equal to 2, the following applies:

$$\text{qP} = \text{Qp}'_{\text{CbCr}} \quad (1129)$$

$$\text{QpActOffset} = \text{cu\_act\_enabled\_flag[xTbY][yTbY] ? } 1 : 0 \quad (1130)$$

– Otherwise, if cIdx is equal to 1, the following applies:

$$\text{qP} = \text{Qp}'_{\text{Cb}} \quad (1131)$$

$$\text{QpActOffset} = \text{cu\_act\_enabled\_flag[xTbY][yTbY] ? } 1 : 0 \quad (1132)$$

– Otherwise (cIdx is equal to 2), the following applies:

$$\text{qP} = \text{Qp}'_{\text{Cr}} \quad (1133)$$

$$\text{QpActOffset} = \text{cu\_act\_enabled\_flag[xTbY][yTbY] ? } 3 : 0 \quad (1134)$$

The quantization parameter qP is modified and the variables rectNonTsFlag and bdShift are derived as follows:

– If transform_skip_flag[xTbY][yTbY][cIdx] is equal to 0, the following applies:

$$\text{qP} = \text{Clip3(0, 63 + QpBdOffset, qP + QpActOffset)} \quad (1135)$$

$$\text{rectNonTsFlag} = (((\text{Log2(nTbW)} + \text{Log2(nTbH)}) \,\&\, 1) == 1) \text{ ? } 1 : 0 \quad (1136)$$

$$\begin{array}{l}\text{bdShift} = \text{BitDepth} + \text{rectNonTsFlag} + \\ \quad ((\text{Log2(nTbW)} + \text{Log2(nTbH)}) / 2) + 10 - \text{Log2TransformRange} + \text{sh\_dep\_quant\_used\_flag}\end{array} \quad (1137)$$

– Otherwise (transform_skip_flag[xTbY][yTbY][cIdx] is equal to 1), the following applies:

$$\text{qP} = \text{Clip3(QpPrimeTsMin, 63 + QpBdOffset, qP + QpActOffset)} \quad (1138)$$

$$\text{rectNonTsFlag} = 0 \quad (1139)$$

$$\text{bdShift} = 10 \quad (1140)$$

The variable bdOffset is derived as follows:

$$\text{bdOffset} = (1 \ll \text{bdShift}) \gg 1 \quad (1141)$$

The list levelScale[ ][ ] is specified as levelScale[j][k] = { { 40, 45, 51, 57, 64, 72 }, { 57, 64, 72, 80, 90, 102 } } with j = 0..1, k = 0..5.

The (nTbW)x(nTbH) array dz is set equal to the (nTbW)x(nTbH) array TransCoeffLevel[xTbY][yTbY][cIdx].

For the derivation of the scaled transform coefficients d[x][y] with x = 0..nTbW − 1, y = 0..nTbH − 1, the following applies:

– The intermediate scaling factor m[x][y] is derived as follows:
  – If one or more of the following conditions are true, m[x][y] is set equal to 16:
    – sh_explicit_scaling_list_used_flag is equal to 0.
    – transform_skip_flag[xTbY][yTbY][cIdx] is equal to 1.
    – sps_scaling_matrix_for_lfnst_disabled_flag is equal to 1 and ApplyLfnstFlag[cIdx] is equal to 1.
    – sps_scaling_matrix_for_alternative_colour_space_disabled_flag is equal to 1 and sps_scaling_matrix_designated_colour_space_flag is equal to cu_act_enabled_flag[xTbY][yTbY].
  – Otherwise, the following applies:
    – The variable id is derived based on predMode, cIdx, nTbW, and nTbH as specified in Table 38 and the variable log2MatrixSize is derived as follows:

$$\text{log2MatrixSize} = (\ \text{id} < 2\ )\ ?\ 1 : (\ \text{id} < 8\ )\ ?\ 2 : 3 \tag{1142}$$

    – The scaling factor m[x][y] is derived as follows:

$$\begin{aligned} \text{m[x][y]} &= \text{ScalingMatrixRec[id][i][j]} \\ &\quad \text{with i} = (\ \text{x} \ll \text{log2MatrixSize}\ ) \gg \text{Log2(nTbW)}, \\ &\quad\quad\ \ \text{j} = (\ \text{y} \ll \text{log2MatrixSize}\ ) \gg \text{Log2(nTbH)} \end{aligned} \tag{1143}$$

    – If id is greater than 13 and both x and y are equal to 0, m[0][0] is further modified as follows:

$$\text{m[0][0]} = \text{ScalingMatrixDcRec[id} - 14\text{]} \tag{1144}$$

NOTE – A quantization matrix element m[x][y] could be zeroed out when any of the following conditions is true
– x is greater than 31
– y is greater than 31
– The decoded tu is not coded by default transform mode (i.e., transform type is not equal to 0) and x is greater than 15
– The decoded tu is not coded by default transform mode (i.e., transform type is not equal to 0) and y is greater than 15

– The scaling factor ls[x][y] is derived as follows:
  – If sh_dep_quant_used_flag is equal to 1 and transform_skip_flag[xTbY][yTbY][cIdx] is equal to 0, the following applies:

$$\text{ls[x][y]} = (\ \text{m[x][y]} * \text{levelScale[rectNonTsFlag][(qP + 1) \% 6]}\ ) \ll (\ (\ \text{qP} + 1\ ) / 6\ ) \tag{1145}$$

  – Otherwise (sh_dep_quant_used_flag is equal to 0 or transform_skip_flag[xTbY][yTbY][cIdx] is equal to 1), the following applies:

$$\text{ls[x][y]} = (\ \text{m[x][y]} * \text{levelScale[rectNonTsFlag][qP \% 6]}\ ) \ll (\ \text{qP} / 6\ ) \tag{1146}$$

– When BdpcmFlag[xTbY][yTbY][cIdx] is equal to 1, dz[x][y] is modified as follows:
  – If BdpcmDir[xTbY][yTbY][cIdx] is equal to 0 and x is greater than 0, the following applies:

$$\text{dz[x][y]} = \text{Clip3(CoeffMin, CoeffMax, dz[x} - 1\text{][y]} + \text{dz[x][y])} \tag{1147}$$

  – Otherwise, if BdpcmDir[xTbY][yTbY][cIdx] is equal to 1 and y is greater than 0, the following applies:

$$\text{dz[x][y]} = \text{Clip3(CoeffMin, CoeffMax, dz[x][y} - 1\text{]} + \text{dz[x][y])} \tag{1148}$$

– The value dnc[x][y] is derived as follows:

$$\text{dnc[x][y]} = (\ \text{dz[x][y]} * \text{ls[x][y]} + \text{bdOffset}\ ) \gg \text{bdShift} \tag{1149}$$

– The scaled transform coefficient d[x][y] is derived as follows:

$$\text{d[x][y]} = \text{Clip3(CoeffMin, CoeffMax, dnc[x][y])} \tag{1150}$$

Table 38 – Specification of the scaling matrix identifier variable id according to predMode, cIdx, nTbW, and nTbH

Max(nTbW, nTbH)		2	4	8	16	32	64
predMode = MODE_INTRA	cIdx = 0 (Y)		2	8	14	20	26
	cIdx = 1 (Cb)		3	9	15	21	21
	cIdx = 2 (Cr)		4	10	16	22	22
predMode = MODE_INTER or MODE_IBC	cIdx = 0 (Y)		5	11	17	23	27
	cIdx = 1 (Cb)	0	6	12	18	24	24
	cIdx = 2 (Cr)	1	7	13	19	25	25

8.7.4 Transformation process for scaled transform coefficients

8.7.4.1 General

Inputs to this process are:

– a luma location (xTbY, yTbY) specifying the top-left sample of the current luma transform block relative to the top-left luma sample of the current picture,

– a variable nCbW specifying the width of the current coding block,

– a variable nCbH specifying the height of the current coding block,

– a variable nTbW specifying the width of the current transform block,

– a variable nTbH specifying the height of the current transform block,

– a variable cIdx specifying the colour component of the current block,

– an (nTbW)x(nTbH) array d[x][y] of scaled transform coefficients with x = 0..nTbW − 1, y = 0..nTbH − 1.

Output of this process is the (nTbW)x(nTbH) array res[x][y] of residual samples with x = 0..nTbW − 1, y = 0..nTbH − 1.

When ApplyLfnstFlag[cIdx] is equal to 1 and transform_skip_flag[xTbY][yTbY][cIdx] is equal to 0, the following applies:

– The variables predModeIntra, nLfnstOutSize, log2LfnstSize, nLfnstSize, and nonZeroSize are derived as follows:

predModeIntra = (cIdx = = 0) ? IntraPredModeY[xTbY][yTbY] : IntraPredModeC[xTbY][yTbY]

(1151)

nLfnstOutSize = (nTbW >= 8 && nTbH >= 8) ? 48 : 16 (1152)

log2LfnstSize = (nTbW >= 8 && nTbH >= 8) ? 3 : 2 (1153)

nLfnstSize = 1 << log2LfnstSize (1154)

nonZeroSize = ((nTbW = = 4 && nTbH = = 4) | | (nTbW = = 8 && nTbH = = 8)) ? 8 : 16 (1155)

– When IntraMipFlag[xTbY][yTbY] is equal to 1 and cIdx is equal to 0, predModeIntra is set equal to INTRA_PLANAR.

– When predModeIntra is equal to either INTRA_LT_CCLM, INTRA_L_CCLM, or INTRA_T_CCLM, predModeIntra is derived as follows:

  – If IntraMipFlag[xTbY + nTbW * SubWidthC / 2][yTbY + nTbH * SubHeightC / 2] is equal to 1, predModeIntra is set equal to INTRA_PLANAR.

– Otherwise, if CuPredMode[0][xTbY + nTbW * SubWidthC / 2][yTbY + nTbH * SubHeightC / 2] is equal to MODE_IBC or MODE_PLT, predModeIntra is set equal to INTRA_DC.

– Otherwise, predModeIntra is set equal to IntraPredModeY[xTbY + nTbW * SubWidthC / 2][yTbY + nTbH * SubHeightC / 2].

– The wide angle intra prediction mode mapping process as specified in clause 8.4.5.2.7 is invoked with predModeIntra, nCbW, nCbH, nTbW, nTbH and cIdx as inputs, and the modified predModeIntra as output.

– The values of the list u[x] with x = 0..nonZeroSize − 1 are derived as follows:

xC = DiagScanOrder[2][2][x][0] (1156)

yC = DiagScanOrder[2][2][x][1] (1157)

u[x] = d[xC][yC] (1158)

– The one-dimensional low frequency non-separable transformation process as specified in clause 8.7.4.2 is invoked with the input length of the scaled transform coefficients nonZeroSize, the transform output length nTrS set equal to nLfnstOutSize, the list of scaled non-zero transform coefficients u[x] with x = 0..nonZeroSize − 1, and the intra prediction mode for LFNST set selection predModeIntra as inputs, and the list v[x] with x = 0..nLfnstOutSize − 1 as output.

– The array d[x][y] with x = 0..nLfnstSize − 1, y = 0..nLfnstSize − 1 is derived as follows:

– If predModeIntra is less than or equal to 34, the following applies:

d[x][y] = (y < 4) ? v[x + (y << log2LfnstSize)] :
((x < 4) ? v[32 + x + ((y − 4) << 2)] : d[x][y]) (1159)

– Otherwise, the following applies:

d[x][y] = (x < 4) ? v[y + (x << log2LfnstSize)] :
((y < 4) ? v[32 + y + ((x − 4) << 2)] : d[x][y]) (1160)

The variable implicitMtsEnabled is derived as follows:

– If sps_mts_enabled_flag is equal to 1 and one or more of the following conditions are true, implicitMtsEnabled is set equal to 1:

– IntraSubPartitionsSplitType is not equal to ISP_NO_SPLIT.

– cu_sbt_flag is equal to 1 and Max(nTbW, nTbH) is less than or equal to 32.

– sps_explicit_mts_intra_enabled_flag is equal to 0 and CuPredMode[0][xTbY][yTbY] is equal to MODE_INTRA and lfnst_idx is equal to 0 and IntraMipFlag[x0][y0] is equal to 0.

– Otherwise, implicitMtsEnabled is set equal to 0.

The variable trTypeHor specifying the horizontal transform kernel and the variable trTypeVer specifying the vertical transform kernel are derived as follows:

– If one or more of the following conditions are true, trTypeHor and trTypeVer are set equal to 0:

– cIdx is greater than 0.

– IntraSubPartitionsSplitType is not equal to ISP_NO_SPLIT and lfnst_idx is not equal to 0.

– Otherwise, if implicitMtsEnabled is equal to 1, the following applies:

– If cu_sbt_flag is equal to 1, trTypeHor and trTypeVer are specified in Table 40 depending on cu_sbt_horizontal_flag and cu_sbt_pos_flag.

– Otherwise (cu_sbt_flag is equal to 0), trTypeHor and trTypeVer are derived as follows:

trTypeHor = (nTbW >= 4 && nTbW <= 16) ? 1 : 0 (1161)

trTypeVer = (nTbH >= 4 && nTbH <= 16) ? 1 : 0 (1162)

– Otherwise, trTypeHor and trTypeVer are specified in Table 39 depending on mts_idx.

The variables nonZeroW and nonZeroH are derived as follows:

– If ApplyLfnstFlag[cIdx] is equal to 1, the following applies:

$$nonZeroW = (nTbW == 4 \;||\; nTbH == 4) ? 4 : 8 \quad (1163)$$

$$nonZeroH = (nTbW == 4 \;||\; nTbH == 4) ? 4 : 8 (1164)$$

– Otherwise, the following applies:

$$nonZeroW = Min(nTbW, (trTypeHor > 0) ? 16 : 32) \quad (1165)$$

$$nonZeroH = Min(nTbH, (trTypeVer > 0) ? 16 : 32) \quad (1166)$$

The (nTbW)x(nTbH) array r of residual samples is derived as follows:

1. When nTbH is greater than 1, each (vertical) column of scaled transform coefficients d[x][y] with x = 0..nonZeroW − 1, y = 0..nonZeroH − 1 is transformed to e[x][y] with x = 0..nonZeroW − 1, y = 0..nTbH − 1 by invoking the one-dimensional transformation process as specified in clause 8.7.4.4 for each column x = 0..nonZeroW − 1 with the height of the transform block nTbH, the non-zero height of the scaled transform coefficients nonZeroH, the list d[x][y] with y = 0..nonZeroH − 1 and the transform type variable trType set equal to trTypeVer as inputs, and the output is the list e[x][y] with y = 0..nTbH − 1.

2. When nTbH and nTbW are both greater than 1, the intermediate sample values g[x][y] with x = 0..nonZeroW − 1, y = 0..nTbH − 1 are derived as follows:

$$g[x][y] = Clip3(CoeffMin, CoeffMax, (e[x][y] + 64) >> 7) \quad (1167)$$

3. When nTbH is equal to 1, g[x][0] is set equal to d[x][0] for x = 0..nTbW − 1.

4. When nTbW is greater than 1, each (horizontal) row of the resulting array g[x][y] with x = 0..nonZeroW − 1, y = 0..nTbH − 1 is transformed to r[x][y] with x = 0..nTbW − 1, y = 0..nTbH − 1 by invoking the one-dimensional transformation process as specified in clause 8.7.4.4 for each row y = 0..nTbH − 1 with the width of the transform block nTbW, the non-zero width of the resulting array g[x][y] nonZeroW, the list g[x][y] with x = 0..nonZeroW − 1 and the transform type variable trType set equal to trTypeHor as inputs, and the output is the list r[x][y] with x = 0..nTbW − 1.

5. When nTbW is equal to 1, r[0][y] is set equal to e[0][y] for y = 0..nTbH − 1.

6. The nTbW x nTbH array res[x][y] of residual samples with x = 0..nTbW − 1, y = 0..nTbH − 1 is derived as follows:

$$bdShift = (nTbH > 1 \;\&\&\; nTbW > 1) ? (5 + Log2TransformRange - BitDepth) : (6 + Log2TransformRange - BitDepth) \quad (1168)$$

$$res[x][y] = (r[x][y] + (1 << (bdShift - 1))) >> bdShift \quad (1169)$$

Table 39 – Specification of trTypeHor and trTypeVer depending on mts_idx

mts_idx	**0**	**1**	**2**	**3**	**4**
trTypeHor	0	1	2	1	2
trTypeVer	0	1	1	2	2

Table 40 – Specification of trTypeHor and trTypeVer depending on cu_sbt_horizontal_flag and cu_sbt_pos_flag

cu_sbt_horizontal_flag	**cu_sbt_pos_flag**	**trTypeHor**	**trTypeVer**
0	**0**	2	1
0	**1**	1	1

1	**0**	1	2
1	**1**	1	1

8.7.4.2 Low frequency non-separable transformation process

Inputs to this process are:

– a variable nonZeroSize specifying the transform input length,

– a variable nTrS specifying the transform output length,

– a list of scaled non-zero transform coefficients x[j] with j = 0..nonZeroSize − 1,

– a variable predModeIntra specifying the intra prediction mode for LFNST set selection.

Output of this process is the list of transformed samples y[i] with i = 0..nTrS − 1.

The transformation matrix derivation process as specified in clause 8.7.4.3 is invoked with the transform output length nTrS, and the intra prediction mode for LFNST set selection predModeIntra as inputs, and the (nTrS)x(nonZeroSize) LFNST matrix lowFreqTransMatrix as output.

The list of transformed samples y[i] with i = 0..nTrS − 1 is derived as follows:

$$y[\ i\ ] = \text{Clip3}(\ \text{CoeffMin},\ \text{CoeffMax},\ (\ (\ \sum_{j=0}^{\text{nonZeroSize}-1} \text{lowFreqTransMatrix}[\ i\ ][\ j\ ] * x[\ j\ ]\ ) + 64\ ) >> 7\ ) \quad (1170)$$

8.7.4.3 Low frequency non-separable transformation matrix derivation process

Inputs to this process are:

– a variable nTrS specifying the transform output length,

– a variable predModeIntra specifying the intra prediction mode for LFNST set selection.

Output of this process is the transformation matrix lowFreqTransMatrix.

The variable lfnstTrSetIdx is specified in Table 41 depending on predModeIntra.

Table 41 – Specification of lfnstTrSetIdx

predModeIntra	lfnstTrSetIdx
predModeIntra < 0	1
0 <= predModeIntra <= 1	0
2 <= predModeIntra <= 12	1
13 <= predModeIntra <= 23	2
24 <= predModeIntra <= 44	3
45 <= predModeIntra <= 55	2
56 <= predModeIntra <= 80	1

The transformation matrix lowFreqTransMatrix is derived based on nTrS, lfnstTrSetIdx, and lfnst_idx as follows:

– If nTrS is equal to 16, lfnstTrSetIdx is equal to 0, and lfnst_idx is equal to 1, the following applies:

lowFreqTransMatrix[m][n] =

```
{
{  108  -44  -15    1  -44   19    7   -1  -11    6    2   -1    0   -1   -1    0 }
{  -40  -97   56   12  -11   29  -12   -3   18   18  -15   -3   -1   -3    2    1 }
{   25  -31   -1    7  100  -16  -29    1  -54   21   14   -4   -7    2    4    0 }
{  -32  -39  -92   51   -6  -16   36   -8    3   22   18  -15    4    1   -5    2 }
{    8   -9   33   -8  -16 -102   36   23   -4   38  -27   -5    5   16   -8   -6 }
{  -25    5   16   -3  -38   14   11   -3  -97    7   26    1   55  -10  -19    3 }
{    8    9   16    1   37   36   94  -38   -7    3  -47   11   -6  -13  -17   10 }
{    2   34   -5    1   -7   24  -25   -3    8   99  -28  -29    6  -43   21   11 }
{  -16  -27  -39 -109    6   10   16   24    3   19   10   24   -4   -7   -2   -3 }
{   -9  -10  -34    4   -9   -5  -29    5  -33  -26  -96   33   14    4   39  -14 }
{  -13    1    4   -9  -30  -17   -3  -64  -35   11   17   19  -86    6   36   14 }
{    8   -7   -5  -15    7  -30  -28  -87   31    4    4   33   61   -5  -17   22 }
{   -2   13   -6   -4   -2   28  -13  -14   -3   37  -15   -3   -2  107  -36  -24 }
{    4    9   11   31    4    9   16   19   12   33   32   94   12    0   34  -45 }
{    2   -2    8  -16    8    5   28  -17    6   -7   18  -45   40   36   97   -8 }
{    0   -2    0  -10   -1   -7   -3  -35   -1   -7   -2  -32   -6  -33  -16 -112 }
},
```

– Otherwise, if nTrS is equal to 16, lfnstTrSetIdx is equal to 0, and lfnst_idx is equal to 2, the following applies:

lowFreqTransMatrix[m][n] =

```
{
{  119  -30  -22   -3  -23   -2    3    2  -16    3    6    0   -3    2    1    0 }
{  -27 -101   31   17  -47    2   22    3   19   30   -7   -9    5    3   -5   -1 }
{    0   58   22  -15 -102    2   38    2   10  -13   -5    4   14   -1   -9    0 }
{   23    4   66  -11   22   89   -2  -26   13   -8  -38   -1   -9  -20   -2    8 }
{  -19   -5  -89    2  -26   76  -11  -17   20   13   18   -4    1  -15    3    5 }
{  -10   -1   -1    6   23   25   87   -7  -74    4   39   -5    0   -1  -20   -1 }
{  -17  -28   12   -8  -32   14  -53   -6  -68  -67   17   29    2    6   25    4 }
{    1  -24  -23    1   17   -7   52    9   50  -92  -15   27  -15  -10   -6    3 }
{   -6  -17   -2 -111    7  -17    8  -42    9   18   16   25   -4    2   -1   11 }
{    9    5   35    0    6   21   -9   34   44   -3  102   11   -7   13   11  -20 }
{    4   -5   -5  -10   15   19   -2    6    6  -12  -13    6   95   69  -29  -24 }
{   -6   -4   -9  -39    1   22    0  102  -19   19  -32   30  -16  -14   -8  -23 }
{    4   -4    7    8    4  -13  -18    5    0    0   21   22   58  -88  -54   28 }
{   -4   -7    0  -24   -7    0  -25    3   -3  -30    8  -76  -34    4  -80  -26 }
{    0    6    0   30   -6    1  -13  -23    1   20   -2   80  -44   37  -68    1 }
{    0    0   -1    5   -1   -7    1  -34   -2    3   -6   19    5  -38   11 -115 }
},
```

– Otherwise, if nTrS is equal to 16, lfnstTrSetIdx is equal to 1, and lfnst_idx is equal to 1, the following applies:

lowFreqTransMatrix[m][n] =

```
{
{ -111   39    4    3   44   11  -12   -1    7  -16   -5    2    3   -1    4    2 }
{  -47  -27   15   -1  -92   43   20   -2   20   39  -16   -5   10   -5  -13    2 }
{  -35  -23    4    4  -17  -72   32    6  -59   18   50   -6    0   40    0  -13 }
{   13   93  -27   -4  -48   13  -34    4  -52   11    1   10    3   16   -3    1 }
{  -11  -27    1    2  -47   -4  -36   10   -2  -85   14   29  -20   -2   57    4 }
{    0  -35   32   -2   26   60   -3  -17  -82    1  -30    0  -37   21    3   12 }
{  -17  -46  -92   14    7  -10  -39   29  -17   27  -28   17    1  -15  -13   17 }
{    4  -10  -23    4   16   58  -17   26   30   21   67    2  -13   59   13  -40 }
{    5  -20   32   -5    8   -3  -46   -7   -4    2  -15   24  100   44    0    5 }
{   -4   -1   38  -18   -7  -42  -63   -6   33   34  -23   15  -65   33  -20    2 }
{   -2  -10   35  -19    5    8  -44   14  -25   25   58   17    7  -84  -16  -18 }
{    5   13   18   34   11   -4   18   18    5   58   -3   42   -2  -10   85   38 }
{   -5   -7  -34  -83    2   -1   -4  -73    4   20   15  -12    4   -3   44   12 }
{    0    4   -2  -60    5    9   42   34    5  -14    9   80   -5   13  -38   37 }
{   -1    2    7  -57    3   -7    9   68   -9    6  -49  -20    6   -4   36  -64 }
{   -1    0  -12   23    1   -4   17  -53   -3    4  -21   72   -4   -8   -3  -83 }
},
```

– Otherwise, if nTrS is equal to 16, lfnstTrSetIdx is equal to 1, and lfnst_idx is equal to 2, the following applies:

lowFreqTransMatrix[m][n] =

```
{
{   88  -55    6   -3  -66   27    9   -2   11   11  -13    1   -2   -7    1    2 }
{  -58  -20   27   -2  -27   75  -29    0   47  -42  -11   11   -9   -3   19   -4 }
{  -51   23  -22    5  -63    3   37   -5    1   64  -35   -4   29  -31  -11   13 }
{  -27  -76   49   -2   40   14    9  -17  -56   36  -25    6   14    3   -6    8 }
{   19   -4  -36   22   52    7   36  -23   28  -17  -64   15   -5  -44   48    9 }
{   29   50   13  -10    1   34  -59    1  -51    4  -16   30   52  -33   24   -5 }
{  -12  -21  -74   43  -13   39   18   -5  -58  -35   27   -5   19   26    6   -5 }
{   19   38  -10   -5   28   66    0   -5   -4   19  -30  -26  -40   28  -60   37 }
{   -6   27   18   -5  -37  -18   12  -25  -44  -10  -38   37  -66   45   40   -7 }
{  -13  -28  -45  -39    0   -5  -39   69  -23   16  -12  -18  -50  -31   24   13 }
{   -1    8   24  -51  -15   -9   44   10  -28  -70  -12  -39   24  -18   -4   51 }
{   -8  -22  -17   33  -18  -45  -57  -27    0  -31  -30   29   -2  -13  -53   49 }
{    1   12   32   51   -8    8   -2  -31  -22    4   46  -39  -49  -67   14   17 }
{    4    5   24   60   -5  -14  -23   38    9    8  -34  -59   24   47   42   28 }
{   -1   -5  -20  -34    4    4  -15  -46   18   31   42   10   10   27   49   78 }
{   -3   -7  -22  -34   -5  -11  -36  -69   -1   -3  -25  -73    5    4    4  -49 }
},
```

– Otherwise, if nTrS is equal to 16, lfnstTrSetIdx is equal to 2, and lfnst_idx is equal to 1, the following applies:

lowFreqTransMatrix[m][n] =

```
{
{ -112   47   -2    2  -34   13    2    0   15   -7    1    0    8   -3   -1    0 }
{   29   -7    1   -1 -108   40    2    0  -45   13    4   -1    8   -5    1    0 }
{  -36  -87   69  -10  -17  -33   26   -2    7   14  -11    2    6    8   -7    0 }
{   28   -5    2   -2  -29   13   -2    0  103  -36   -4    1   48  -16   -4    1 }
{  -12  -24   15   -3   26   80  -61    9   15   54  -36    2    0   -4    6   -2 }
{   18   53   69  -74   14   24   28  -30   -6   -7  -11   12   -5   -7   -6    8 }
{    5   -1    2    0  -26    6    0    1   45   -9   -1    0 -113   28    8   -1 }
{  -13  -32   18   -2   15   34  -27    7  -25  -80   47   -1  -16  -50   28    2 }
{   -4  -13  -10   19   18   46   60  -48   16   33   60  -48    1    0    5   -2 }
{   15   33   63   89    8   15   25   40   -4   -8  -15   -8   -2   -6   -9   -7 }
{   -8  -24  -27   15   12   41   26  -29  -17  -50  -39   27    0   35  -67   26 }
{   -2   -6  -24   13   -1   -8   37  -22    3   18  -51   22  -23  -95   17   17 }
{   -3   -7  -16  -21   10   24   46   75    8   20   38   72    1    2    1    7 }
{    2    6   10   -3   -5  -16  -31   12    7   24   41  -16  -16  -41  -89   49 }
{    4    8   21   40   -4  -11  -28  -57    5   14   31   70    7   18   32   52 }
{    0    1    4   11   -2   -4  -13  -34    3    7   20   47   -6  -19  -42 -101 }
},
```

– Otherwise, if nTrS is equal to 16, lfnstTrSetIdx is equal to 2, and lfnst_idx is equal to 2, the following applies:

lowFreqTransMatrix[m][n] =

```
{
{  -99   39   -1    2   65  -20   -5    0  -15   -2    5   -1    0    3   -1    0 }
{   58   42  -33    3   33  -63   23   -1  -55   32    3   -5   21   -2   -8    3 }
{  -15   71  -44    5  -58  -29   25    3   62   -7   -4   -4  -19    4    0    1 }
{   46    5    4   -6   71  -12  -15    5   52  -38   13   -2  -63   23    3   -3 }
{  -14  -54  -29   29   25   -9   61  -29   27   44  -48    5  -27  -21   12    7 }
{   -3    3   69  -42  -11  -50  -26   26   24   63  -19   -5  -18  -22   12    0 }
{   17   16   -2    1   38   18  -12    0   62    1  -14    5   89  -42    8   -2 }
{   15   54   -8    6    6   60  -26   -8  -30   17  -38   22  -43  -45   42   -7 }
{   -6  -17  -55  -28    9   30   -8   58    4   34   41  -52  -16  -36  -20   16 }
{   -2   -1   -9  -79    7   11   48   44  -13  -34  -55    6   12   23   20  -11 }
{    7   29   14   -6   12   53   10  -11   14   59  -15   -3    5   71  -54   13 }
{   -5  -24  -53   15   -3  -15  -61   26    6   30  -16   23   13   56   44  -35 }
{    4    8   21   52   -1   -1   -5   29   -7  -17  -44  -84    8   20   31   39 }
{   -2  -11  -25   -4   -4  -21  -53    2   -5  -26  -64   19   -8  -19  -73   39 }
{   -3   -5  -23  -57   -2   -4  -24  -75    1    3    9  -25    6   15   41   61 }
{    1    1    7   18    1    2   16   47    2    5   24   67    3    9   25   88 }
},
```

– Otherwise, if nTrS is equal to 16, lfnstTrSetIdx is equal to 3, and lfnst_idx is equal to 1, the following applies:

lowFreqTransMatrix[m][n] =

```
{
{ -114   37    3    2  -22  -23   14    0   21  -17   -5    2    5    2   -4   -1 }
{  -19  -41   19   -2   85  -60  -11    7   17   31  -34    2  -11   19    2   -8 }
{   36  -25   18   -2  -42  -53   35    5   46  -60  -25   19    8   21  -33   -1 }
{  -27  -80   44   -3  -58    1  -29   19  -41   18  -12   -7   12  -17    7   -6 }
{  -11  -21   37  -10   44   -4   47  -12  -37  -41   58   18   10  -46  -16   31 }
{   15   47   10   -6  -16  -44   42   10  -80   25  -40   21  -23   -2    3  -14 }
{   13   25   79  -39  -13   10   31   -4   49   45   12   -8    3   -1   43    7 }
{   16   11  -26   13  -13  -74  -20   -1    5   -6   29  -47   26  -49   54    2 }
{   -8  -34  -26    7  -26  -19   29  -37    1   22   46   -9  -81   37   14   20 }
{   -6  -30  -42  -12   -3    5   57  -52   -2   37  -12    6   74   10    6  -15 }
{    5    9   -6   42  -15  -18   -9   26   15   58   14   43   23  -10  -37   75 }
{   -5  -23  -23   36    3   22   36   40   27   -4  -16   56  -25  -46   56  -24 }
{    1    3   23   73    8    5   34   46  -12    2   35  -38   26   52    2  -31 }
{   -3   -2  -21  -52    1  -10  -17   44  -19  -20   30   45   27   61   49   21 }
{   -2   -7  -33  -56   -4   -6   21   63   15   31   32  -22  -10  -26  -52  -38 }
{   -5  -12  -18  -12    8   22   38   36   -5  -15  -51  -63   -5    0   15   73 }
},
```

– Otherwise, if nTrS is equal to 16, lfnstTrSetIdx is equal to 3, and lfnst_idx is equal to 2, the following applies:

lowFreqTransMatrix[m][n] =

```
{
{ -102   22    7    2   66  -25   -6   -1  -15   14    1   -1    2   -2    1    0 }
{   12   93  -27   -6  -27  -64   36    6   13    5  -23    0   -2    6    5   -3 }
{  -59  -24   17    1  -62   -2   -3    2   83  -12  -17   -2  -24   14    7   -2 }
{  -33   23  -36   11  -21   50   35  -16  -23  -78   16   19   22   15  -30   -5 }
{    0  -38  -81   30   27    5   51  -32   24   36  -16   12  -24   -8    9    1 }
{   28   38    8   -9   62   32  -13    2   51  -32   15    5  -66   28    0   -1 }
{   11  -35   21  -17   30  -18   31   18  -11  -36  -80   12   16   49   13  -32 }
{  -13   23   22  -36  -12   64   39   25  -19   23  -36    9  -30  -58   33   -7 }
{   -9  -20  -55  -83    3   -2    1   62    8    2   27  -28    7   15  -11    5 }
{   -6   24  -38   23   -8   40  -49    0   -7    9  -25  -44   23   39   70   -3 }
{   12   17   17    0   32   27   21    2   67   11   -6  -10   89  -22  -12   16 }
{    2   -9    8   45    7   -8   27   35   -9  -31  -17  -87  -23  -22  -19   44 }
{   -1   -9   28  -24   -1  -10   49  -30   -8   -7   40    1    4   33   65   67 }
{    5  -12  -24  -17   13  -34  -32  -16   14  -67   -7    9    7  -74   49    1 }
{    2   -6   11   45    3  -10   33   55    8   -5   59    4    7   -4   44  -66 }
{   -1    1  -14   36   -1    2  -20   69    0    0  -15   72    3    4    5   65 }
},
```

– Otherwise, if nTrS is equal to 48, lfnstTrSetIdx is equal to 0, and lfnst_idx is equal to 1, the following applies:

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol0to15[m][n] with m = 0..15, n = 0..15

lowFreqTransMatrixCol0to15 =

```
{
{ -117   28   18    2    4    1    2    1   32  -18   -2    0   -1    0    0    0 }
{  -29  -91   47    1    9    0    3    0  -54   26   -8    3    0    1    0    0 }
{  -10   62  -11   -8   -2   -2   -1   -1  -95    3   32    0    4    0    2    0 }
{  -15   15  -10   -2    1    0    1    0   10  112  -20  -17   -4   -4   -1   -2 }
{   32   39   92  -44    4  -10    1   -4   26   12  -15   13   -5    2   -2    0 }
{  -10    1   50  -15    2   -3    1   -1  -28  -15   14    6    1    1    1    0 }
{    1  -33  -11  -14    7   -2    2    0   29  -12   37   -7   -4    0   -1    0 }
{    0    6   -6   21   -4    2    0    0  -20  -24 -104   30    5    5    1    2 }
{  -13  -13  -37 -101   29  -11    8   -3  -12  -15  -20    2  -11    5   -2    1 }
{    6    1  -14  -36    9   -3    2    0   10    9  -18   -1   -3    1    0    0 }
{  -12   -2  -26  -12   -9    2   -1    1   -3   30    4   34   -4    0   -1    0 }
{    0   -3    0   -4  -15    6   -3    1   -7  -15  -28  -86   19   -5    4   -1 }
{   -1    9   13    5   14   -2    2   -1   -8    3   -4  -62    4    1    1    0 }
{    6    2   -3    2   10   -1    2    0    8    3   -1  -20    0    1    0    0 }
{    6    9   -2   35  110  -22   11   -4   -2    0   -3    1  -18   12   -3    2 }
{   -1    7   -2    9  -11    5   -1    1   -7    2  -22    4  -13    0   -1    0 }
},
```

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol16to31[m − 16][n] with m = 16..31, n = 0..15

lowFreqTransMatrixCol16to31 =

```
{
{   14   -1   -3    0   -1    0    0    0    2    0    0    0    0    0    0    0 }
{   33    5   -9   -1   -2    0   -1    0   -3    3    0    0    0    0    0    0 }
{   32  -30   -4    4   -1    1    0    0    6    2   -5    0    0    0    0    0 }
{  -20  -26   31    1    0    0    0    0    2  -16   -1    6    0    1    0    0 }
{   29  -16  -22    8    0    1    0    1  -20    6    4   -3    1    0    0    0 }
{  -99   -4    9    5    5    2    2    1   44  -10  -11    1   -2    0   -1    0 }
{    6  -99    3   26   -1    5    0    2   14   30  -27   -2    1   -1    0   -1 }
{   -7  -46   10  -14    7    0    1    0    9   21    7   -6   -2   -1    0   -1 }
{  -12   10   26   12   -6    0   -1    0  -32   -2   11    3    3   -1    1    0 }
{   38   26  -13   -1   -5   -1   -1    0  102    3  -14   -1   -5   -1   -2    0 }
{  -30    3  -92   14   19    0    3    0  -11   34   21  -33    1   -2    0   -1 }
{   -5  -17  -41   42   -6    2   -1    1   -1  -40   37   13   -4    2   -1    1 }
{  -12   23   16  -11  -17    0   -1    0  -11   97   -3   -3    0   -6    0   -2 }
{   -4    4  -16    0   -2    0    1    0   34   23    6   -7   -4   -2   -1    0 }
{   -5   -4  -22    8  -25    3    0    0   -3  -21    2   -3    9   -2    1    0 }
{    0   28    0   76    4   -6    0   -2  -13    5  -76   -4   33   -1    3    0 }
},
```

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol32to47[m − 32][n] with m = 32..47, n = 0..15

lowFreqTransMatrixCol32to47 =

```
{
{    3    0   -1    0    1    0    0    0    1    0    0    0    1    0    0    0 }
{    7    2   -2    0   -1    1    0    0    2    1   -1    0    0    0    0    0 }
{    6   -3    0    0    2    0   -1    0    2   -1    0    0    1    0    0    0 }
{    1   -4    0    0    0   -3    0    1    0   -1    0    0    0   -2    0    0 }
{    1   -4   -3    2   -4    1    0    0    1   -1   -2    1   -2    0    0    0 }
{   -5    4   -3    0    8   -1   -2    0   -2    1   -1    0    4    0   -1    0 }
{   -6    6    6   -3    1    3   -3    0   -1    1    1    0    0    1   -1    0 }
{    2    2    5   -2    0    3    4   -1    0    0    1    0    0    1    2   -1 }
{   11   -5   -1    6   -4    2    1    0    3   -1    1    2   -1    0    0    0 }
{  -29   10   10    0   10   -4   -1    1   -7    1    2    1    2   -1    0    0 }
{   -9   -4   18    3    2    0    0   -2   -1   -1    3    0    0    0    0   -1 }
{  -10   13   -1   -4    4   -4    3    4   -2    2   -1   -1    1   -1    1    2 }
{  -21   -5   23    0    2   -2   -1    6   -3   -3    1    0    0    0    0    2 }
{  108   -5  -30    6  -27   10    7   -2   11   -3   -1    1   -4    1    0    1 }
{   -7    1    3   -5    3    0   -1    0    0    1    0   -1    1    0    0    0 }
{    9   18   -3  -35   -4   -1    6    1    1    2    0   -3   -1    0    2    0 }
},
```

– Otherwise, if nTrS is equal to 48, lfnstTrSetIdx is equal to 0, and lfnst_idx is equal to 2 the following applies:

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol0to15[m][n] with m = 0..15, n = 0..15

lowFreqTransMatrixCol0to15 =

```
{
{ -108   48    9    1    1    1    0    0   44   -6   -9   -1   -1    0   -1    0 }
{   55   66  -37   -5   -6   -1   -2    0   67  -30  -20    4   -2    0   -1    0 }
{    2   86  -21  -13   -4   -2   -1   -1  -88    5    6    4    5    1    1    0 }
{  -24  -21  -38   19    0    4   -1    2  -23  -89   31   20    2    3    1    1 }
{    9   20   98  -26   -3   -5    0   -2   -9  -26   15  -16    2    0    1    0 }
{  -21   -7  -37   10    2    2   -1    1  -10   69   -5   -7   -2   -2    0   -1 }
{  -10  -25    4  -17    8   -2    2   -1  -27  -17  -71   25    8    2    1    1 }
{    2    5   10   64   -9    4   -3    1   -4    8   62    3  -17    1   -2    0 }
{  -11  -15  -28  -97    6   -1    4   -1    7    3   57  -15   10   -2    0   -1 }
{    9   13   24   -6    7   -2    1   -1   16   39   20   47   -2   -2   -2    0 }
{   -7   11   12    7    2   -1    0   -1  -14   -1  -24   11    2    0    0    0 }
{    0    0    7   -6   23   -3    3   -1    5    1   18   96   13   -9   -1   -1 }
{   -2   -6   -1  -10    0    1    1    0   -7   -2  -28   20  -15    4   -3    1 }
{   -1    6  -16    0   24   -3    1   -1    2    6    6   16   18   -7    1   -1 }
{   -5   -6   -3  -19 -104   18   -4    3    0    6    0   35  -41   20   -2    2 }
{   -1   -2    0   23   -9    0   -2    0    1    1    8   -1   29    1    1    0 }
},
```

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol16to31[m − 16][n] with m = 16..31, n = 0..15

lowFreqTransMatrixCol16to31 =

```
{
{    9   -9   -1    1    0    0    0    0    3   -1    1    0    0    0    0    0  }
{  -31  -19   14    4    1    1    1    0   -6    3    5   -2    0    0    0    0  }
{   14   -5    0    3    0    0    0    0   10   -5   -2    0   -1    0    0    0  }
{  -30   26   36   -8   -2   -2    0   -1   14   18   -7   -9   -1   -1    0    0  }
{  -61   -3   -2    3    7    1    1    0   12   16   -6   -1    0   -1    0    0  }
{  -93    2   19    0    3    0    2    0   17    4    0    0   -1    0    0    0  }
{   -4  -66   28   36   -5    3    0    1  -10   20   33  -13   -8    0    0   -1  }
{   -3  -75    5  -14    1    4    0    1  -36    3   18   -4    4    0    1    0  }
{   -1  -27   13    6    1   -1    0    0  -34   -6    0    3    4    1    2    0  }
{   28   23   76   -5  -25   -3   -3   -1    6   36   -7  -39   -4   -1    0   -1  }
{  -20   48   11  -13   -5   -2    0   -1 -105  -19   17    0    6    2    3    0  }
{  -21   -7  -42   14  -24   -3    0    0   11  -47   -7    3   -5    9    1    2  }
{   -2  -32   -2  -66    3    7    1    2  -11   13  -70    5   43   -2    3    0  }
{   -3   11  -63    9    4   -5    2   -1  -22   94   -4   -6   -4   -4    1   -2  }
{   -2   10  -18   16   21    3   -2    0   -2   11    6  -10    6   -3   -1    0  }
{    3   -6   13   76   30  -11   -1   -2  -26   -8  -69    7   -9   -7    3   -1  }
},
```

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol32to47[m − 32][n] with m = 32..47, n = 0..15

lowFreqTransMatrixCol32to47 =

```
{
{    1   -1    0    0    1    0    0    0    0   -1    0    0    0    0    0    0  }
{   -7   -1    1    0   -1    1    1    0   -2   -1    1    0    0    0    0    0  }
{    6   -5    0    1    2   -1    0    0    1   -1    0    0    1    0    0    0  }
{    1    3   -2   -1    3    2   -2   -1    0    1    0    0    1    1   -1    0  }
{    2    0   -8    1    3    1   -1    1    0   -1   -2    0    1    0   -1    0  }
{    5   -4   -2    0    4   -2    0    1    0    0    0    0    2   -1    0    0  }
{    3    6   -3   -7   -1    3    3   -1    1    0   -1    0    0    1    1   -1  }
{    1   14   -2   -8   -2    1   -3    0    2    2   -1   -2    0    1   -1    0  }
{   -2    8    1    5   -2    0   -3    1    1    1    0    2   -1    0   -1    0  }
{    2   -4  -18   -3   -1   -1   -2   -2    1   -2   -2    0    0    0   -1   -1  }
{  -14    8    8    2    1    2   -1   -2    3    0   -1    0    0    0    0    0  }
{    0   -1   19   -1    1    0   -1   -6   -1    1    2    0    1    0    0   -2  }
{    8  -14   -3   43   -1    2    7   -1    1   -2    1    3   -1    1    1    0  }
{   10   23  -19   -5    0   -6   -4    6    3   -2    1    1    0   -1    0    0  }
{   -1    5   -1   -6   -1   -1   -1   -1   -1    0    0    0    0    0    0   -1  }
{  -10  -34  -25   13   -1    0   11    5    1   -1    1   -2    0    0    2    0  }
},
```

– Otherwise, if nTrS is equal to 48, lfnstTrSetIdx is equal to 1, and lfnst_idx is equal to 1 the following applies:

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol0to15[m][n] with m = 0..15, n = 0..15

lowFreqTransMatrixCol0to15 =

```
{
{  110  -49   -3   -4   -1   -1    0   -1  -38   -1   10    0    2    0    1    0  }
{  -43  -19   17   -1    3    0    1    0  -98   46   14   -1    2    0    1    0  }
{  -19   17   -7    3   -2    1   -1    0  -32  -59   29    3    4    0    2    0  }
{  -35 -103   39    1    7    0    2    0   38  -13   25   -6    1   -1    0    0  }
{    9    5   -6   -1   -1    0   -1    0   42    4   21  -11    1   -3    1   -1  }
{   -5   -5  -28    9   -3    2   -1    1  -20  -78   22   16    1    3    0    1  }
{   14   17   27  -12    1   -3    1   -1    8   19  -13    4   -2    1   -1    0  }
{    7   35   17   -4   -1    0    0    0    3    8   54  -17    1   -2    1   -1  }
{  -13  -27 -101   24   -8    6   -3    2   11   43    6   28   -6    3   -1    1  }
{  -11  -13   -3  -10    3   -1    1    0  -19  -19  -37    8    4    2    0    1  }
{   -4  -10  -24  -11    3   -2    0   -1   -6  -37  -45  -17    8   -2    2   -1  }
{   -2    1   13  -17    3   -5    1   -2    3    0  -55   22    6    1    1    0  }
{    3    1    5  -15    1   -2    1   -1    7    4   -7   29   -1    2   -1    1  }
{   -4   -8   -1  -50    6   -4    2   -2   -1    5  -22   20    6    1    0    0  }
{    5   -1   26  102  -13   12   -4    4   -4   -2  -40   -7  -23    3   -5    1  }
{   -5   -6  -27  -22  -12    0   -3    0   -5    8  -20  -83    0    0    0    0  }
},
```

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol16to31[m − 16][n] with m = 16..31, n = 0..15

lowFreqTransMatrixCol16to31 =

```
{
{   -9   13    1   -2    0    0    0    0   -4    2   -3    0    0    0    0    0  }
{   26   26  -15   -3   -2   -1   -1    0   11   -7   -9    2    0    0    0    0  }
{  -72   43   34   -9    3   -2    1   -1   13   36  -18  -10    0   -2    0   -1  }
{   -1    7    6   -7    1   -1    0    0  -13   14    2   -4    2   -1    0    0  }
{   21   70  -32  -21    0   -4   -1   -1   34  -26  -57   11    4    2    0    1  }
{   80   -6   25   -5   -4   -1   -1    0    6  -24    7   -9    0    0    0    0  }
{   48   -1   48  -15   -4   -2   -1   -1    1   60  -28  -42    5   -6    1   -2  }
{   10   14  -11  -34    4   -4    1   -1  -80   -7   -6    2   15    0    3    0  }
{   -3   14   21  -12   -7   -2   -1   -1  -23   10   -4  -12    3    0    1    0  }
{  -12  -30    3   -9    5    0    1    0  -56   -9  -47    8   21    1    4    1  }
{   17   14  -58   14   15    0    2    0  -10   34   -7   28    4   -1    1    0  }
{    8   74   21   40  -14    0   -2    0  -36   -8   11  -13  -23    1   -3    0  }
{    8    3   12  -14   -9   -1   -1    0    4   29  -15   31   10    4    1    1  }
{  -16  -15   18  -29  -11    2   -2    1   40  -45  -19  -22   31    2    4    1  }
{   -1    5    8  -23    7    2    1    1   10  -11  -13   -3   12   -3    2    0  }
{    9    7   24  -20   41    3    6    1   15   20   12   11   17   -9    1   -2  }
},
```

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol32to47[m − 32][n] with m = 32..47, n = 0..15

lowFreqTransMatrixCol32to47 =

```
{
{   -2    2    0    1   -1    1    0    0   -1    1    0    0   -1    0    0    0  }
{    9   -3   -1    2    3   -3    0    0    4   -1    0    0    2   -1    0    0  }
{    3    0  -12    3    6    1   -3    2    1   -1   -2    0    3    1   -1    1  }
{   -2   11   -6   -2   -2    4   -3    0    0    3   -2    0   -1    1   -1    0  }
{   -4  -32    5   24    1   -6   12    4   -3   -2    4   -2    0   -1    0    0  }
{   -7    3   13   -4   -3    5    1   -5   -2    3    1   -2   -1    2   -1   -2  }
{   11  -11  -51   11   -2  -10   -2   13    2   -6   -4    4   -2   -3    2    2  }
{  -16   46    1    3    2    7  -24    0    2   -2   -5    8    1   -1   -2    2  }
{    2    9  -10    0    1   -5   -4    4    2   -2    2    2    0   -2    1    0  }
{  -11  -30   10   59   -2    8   41    8    2    5    6   -7   -1    3    5   -2  }
{   23   34  -31    4   10  -22  -30   22    4  -15    9   20    2   -5    9    4  }
{  -36    6   16  -14    2   19   -4  -12   -1    0   -7   -3    0    2   -2   -1  }
{   61   22   55   14   13    3   -9  -65    1  -11  -21   -7    0    0   -1    3  }
{  -25   41    0   12    9    7  -42   12   -3  -14    2   28    5    1    6    2  }
{   -9   23    4    9   14    9  -14   -4    0  -12   -7    6    3    0    6    3  }
{  -26   -1   18   -1  -12   32    3  -18   -5   10  -25   -5   -2    1   -8   10  }
},
```

– Otherwise, if nTrS is equal to 48, lfnstTrSetIdx is equal to 1, and lfnst_idx is equal to 2 the following applies:

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol0to15[m][n] with m = 0..15, n = 0..15

lowFreqTransMatrixCol0to15 =

```
{
{   80  -49    6   -4    1   -1    1   -1  -72   36    4    0    1    0    0    0  }
{  -72   -6   17    0    3    0    1    0  -23   58  -21    2   -3    1   -1    0  }
{  -50   19  -15    4   -1    1   -1    1  -58   -2   30   -3    4   -1    2    0  }
{  -33  -43   28   -7    4   -2    2   -1  -38   11   -8    4    1    1    0    0  }
{   10   66  -21   -3   -3    0   -1    0  -53  -41   -2   16   -1    4   -1    1  }
{   18   14   13   -9    2   -2    1   -1   34   32  -31   12   -5    2   -2    1  }
{   21   66   -1    9   -4    2   -1    1  -21   41  -30  -10    0   -2    0   -1  }
{    1   -6  -24   17   -5    3   -2    1   24   10   39  -21    5   -4    2   -1  }
{    9   33  -24    1    4    0    1    0    6   50   26    1  -10    0   -2    0  }
{   -7   -9  -32   14   -3    3   -1    1  -23  -28    0   -5   -1    0    0    0  }
{    6   30   69  -18    5   -4    3   -1   -3  -11  -34  -16    9   -4    2   -1  }
{    1   -8   24   -3    7   -2    2   -1   -6  -51   -6   -4   -5    0   -1    0  }
{    4   10    4   17   -9    4   -2    1    5   14   32  -15    9   -3    2   -1  }
{   -3   -9  -23   10  -10    3   -3    1   -5  -14  -16  -27   13   -5    2   -1  }
{    2   11   22    2    9   -2    2    0   -6   -7   20  -32   -3   -4    0   -1  }
{    2   -3    8   14   -5    3   -1    1   -2  -11    5  -18    8   -3    2   -1  }
},
```

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol16to31[m − 16][n] with m = 16..31, n = 0..15

lowFreqTransMatrixCol16to31 =

```
{
{   26    0  -12    2   -2    1   -1    0   -7   -9    6    1    0    0    0    0  }
{   55  -46   -1    6   -2    1   -1    0  -22    7   17   -7    2   -1    1    0  }
{    6   57  -34    0   -2    0   -1    0   34  -48   -2   14   -4    3   -1    1  }
{  -55   24   26   -5    2   -1    1    0   15   46  -40   -1   -1    0   -1    0  }
{   36   -5   41  -20    3   -3    1   -1  -30   26  -32   -3    7   -2    2   -1  }
{   40    4   -4   -9   -3   -2   -1   -1   27  -31  -43   19   -2    3   -1    1  }
{  -35  -17   -3   26   -6    5   -2    2   56    3   18  -25   -1   -2   -1   -1  }
{   33   32  -30    4   -3   -1   -1    0   -4   13  -16  -10    0   -1    0    0  }
{  -27    1  -28  -21   16   -5    3   -2  -23   36   -2   40  -17    4   -3    1  }
{  -36  -59  -24   14    4    2    1    1  -23  -26   23   26   -3    5    0    2  }
{  -16   35  -35   30   -9    3   -2    1  -57  -13    6    4   -5    5   -1    1  }
{   38   -1    0   25    6    2    1    1   47   20   35    1  -27    1   -5    0  }
{    7   13   19   15   -8    1   -1    0    3   25   30  -18    1   -2    0   -1  }
{   -1  -13  -30   11   -5    2   -1    0   -5   -8  -22  -16   10    0    1    0  }
{   13   -5  -28    6   18   -4    3   -1  -26   27  -14    6  -20    0   -2    0  }
{   12  -23  -19   22    2    0    1    0   23   41   -7   35  -10    4   -1    1  }
},
```

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol32to47[m − 32][n] with m = 32..47, n = 0..15

lowFreqTransMatrixCol32to47 =

```
{
{    3    5   -1   -2   -2   -2   -1    1    1    1    0    0   -1   -1    0    0  }
{    9    5  -12    1   -3   -4    4    2    4    1   -2   -1   -1   -1    1    0  }
{  -10    7   21  -10    6    1  -11    0   -1   -1    4    2    3    0   -2   -1  }
{   17  -38    1   17   -3   11   15  -11    3   -1  -10    1    0    1    3    2  }
{   15   -8    1   17   -1   -2    4   -8    2    0   -1    3    0    0    0   -1  }
{    7  -49   52   10  -11   22    7  -26   -1   -6   -9    6   -2    2    4   -2  }
{  -15  -13  -27    9    9   -6   20    5   -3    2   -6   -9    3   -3    1    5  }
{   24  -26  -37   33    5  -32   55   -5   -7   22  -14  -22    1   -9   -3   13  }
{   43  -13    4  -41  -19   -2  -24   17   11   -4    8    4   -3   -3   -3   -3  }
{   10  -26   38    7  -12   11   42  -22   -5   20  -14  -15   -1   -2    1    6  }
{   28   10    4    7    0  -15    7  -10   -1    7   -2    2    1   -3    0    0  }
{   37  -37   -9  -47  -28    5    0   18    8    6    0   -8   -4   -3   -3    1  }
{   11   24   22  -11   -3   37  -13  -58   -5   12  -63   26    9  -15   11    8  }
{    0  -29  -27    6  -27  -10  -30    9   -3  -10   -7   77    9  -13   45   -8  }
{  -76  -26   -4   -7   12   51    5   24    7  -17  -16  -12   -5    4    2   13  }
{    5    7   23    5   69  -38   -8  -32  -15  -31   24   11    2   18   11  -15  }
},
```

– Otherwise, if nTrS is equal to 48, lfnstTrSetIdx is equal to 2, and lfnst_idx is equal to 1 the following applies:

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol0to15[m][n] with m = 0..15, n = 0..15

lowFreqTransMatrixCol0to15 =

```
{
{ -121   33    4    4    1    2    0    1   -1   -1    1    0    0    0    0    0  }
{    0   -2    0    0    0    0    0    0  121  -23   -7   -3   -2   -1   -1    0  }
{  -20   19   -5    2   -1    1    0    0   16    3   -2    0    0    0    0    0  }
{   32  108  -43   10   -9    3   -3    1    4   19   -7    1   -1    0    0    0  }
{   -3    0   -1    0    0    0    0    0  -29   11   -2    1    0    0    0    0  }
{   -4  -12   -3    1   -1    0    0    0   19  105  -31    7   -6    1   -2    0  }
{    7    1    2    0    0    0    0    0    4    3   -2    0    0    0    0    0  }
{   -8  -31   14   -4    3   -1    1    0    9   43    0    1   -1    0    0    0  }
{  -15  -43 -100   23  -12    6   -4    2   -6  -17  -48   10   -5    2   -1    1  }
{   -3    1    2    0    0    0    0    0   -6    3    1    0    0    0    0    0  }
{   -1   -6   -3    2   -1    0    0    0   -6  -35    9    0    2    0    0    0  }
{   -5  -14  -48    2   -5    1   -2    0   10   24   99  -17   10   -4    3   -1  }
{   -2    0    2    0    0    0    0    0   -2    0    1    0    0    0    0    0  }
{   -2  -10   -4    0    0    0    0    0    3   11   -1   -1    0    0    0    0  }
{   -2   -3  -25   -2   -3    0   -1    0   -1   -3   -1    4   -2    2    0    1  }
{    4   -4   28  103  -42   24   -9    7    1    2    4    0    3   -1    0    0  }
},
```

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol16to31[m − 16][n] with m = 16..31, n = 0..15

lowFreqTransMatrixCol16to31 =

```
{
{   24   -5   -1   -1    0    0    0    0    5   -1    0    0    0    0    0    0  }
{   17    1   -2    0    0    0    0    0  -27    4    2    0    0    0    0    0  }
{ -120   14    8    1    3    1    1    0  -18   -2    3    0    1    0    0    0  }
{   11  -30    9   -2    1   -1    0    0    0   -8    2    0    0    0    0    0  }
{   12    7   -1    0    0    0    0    0 -117   12    9    1    3    0    1    0  }
{    9   46   -6    0    0    0    0    0    8  -29    9   -3    1    0    0    0  }
{   22   -8    1   -1    0    0    0    0  -28   -9    4    0    1    0    0    0  }
{  -13 -105   17   -2    2    0    0    0   -8  -25   -3    0    0    0    0    0  }
{    1   -5   19   -6    3   -1    1    0    2    7   15   -3    1   -1    0    0  }
{    0    3   -2    0    0    0    0    0  -20    8   -2    0    0    0    0    0  }
{    1   -6   11   -2    2    0    1    0   -9 -100   17   -1    1    0    0    0  }
{    4   14   32    0    2    0    1    0   -4    0  -39    6   -4    1   -1    0  }
{   -1   -1    1   -1    0    0    0    0   -1   -4    2    0    0    0    0    0  }
{   -6  -40  -15    6   -2    1    0    0    5   57   -6    2    0    0    0    0  }
{   -7   -8  -97   17   -9    3   -3    1   -8  -26  -61   -1   -3   -1   -1   -1  }
{   -1    0   -9  -42   17   -9    3   -2   -1    1  -14    6   -4    2   -1    0  }
},
```

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol32to47[m − 32][n] with m = 32..47, n = 0..15

lowFreqTransMatrixCol32to47 =

```
{
{    3   -1    0    0    2   -1    0    0    2   -1    0    0    1    0    0    0  }
{  -12    2    1    0   -5    1    0    0   -1    0    0    0   -2    0    0    0  }
{   17   -3   -1    0    6   -1   -1    0    2    0    0    0    2    0    0    0  }
{   -7   -1    2    0   -3   -1    1    0   -2   -2    1    0    0    0    0    0  }
{  -32   -3    3    0   12   -2   -1    0    7    0    0    0    1    0    0    0  }
{   -3  -19    3    0   -4   -6    1    0    0    0    0    0    0   -1    0    0  }
{  117  -10   -8    0   32    1   -4    0    3    1   -1    0   -3    1    0    0  }
{   -7   32   -5    1   -1    4    0    0    2   -1    0    0    1    0   -1    0  }
{    4   10    5   -1    0    3    1    0   -2    1    2    0   -1    1    1    0  }
{   30   13   -3    0 -116    6   10    0  -35   -5    4    0   -3   -1    0    0  }
{  -10  -63    1    2  -17    3   -4    0   -1    9   -1    0    3    4   -1    0  }
{    2   -3   -4    0    2   -2   -2    0    0    0   -1    0    0   -1   -1    0  }
{   -8   -2   -1    1   30    4   -4    1 -102    4    8   -1  -69   -2    6   -1  }
{    1  -95   18   -6  -10  -34   -2    0   -4   17   -2    0    0    2    1    0  }
{    2   10   24   -7    5    9   19   -1    0    1    4    0   -2    0    1    0  }
{   -1   -2   -4    4    0    3    1   -1    0    2    0   -2    2    0    0    0  }
},
```

– Otherwise, if nTrS is equal to 48, lfnstTrSetIdx is equal to 2, and lfnst_idx is equal to 2 the following applies:

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol0to15[m][n] with m = 0..15, n = 0..15

lowFreqTransMatrixCol0to15 =

```
{
{   87  -41    3   -4    1   -1    0   -1  -73   28    2    1    1    1    0    0  }
{  -75    4    7    0    2    0    1    0  -41   36   -7    3   -1    1    0    0  }
{   26  -44   22   -6    4   -2    1   -1   77   24  -22    2   -4    0   -1    0  }
{  -39  -68   37   -7    6   -2    2    0   -9   56  -21    1   -2    0   -1    0  }
{   10  -20    2    0    1    0    0    0   50   -1    8   -5    1   -1    0    0  }
{  -21  -45    8   -2    3   -1    1    0   -7  -30   26   -8    3   -1    1   -1  }
{   -4   -2  -55   28   -8    5   -3    2   -2   37   43  -19    1   -2    1   -1  }
{    2   19   47  -23    6   -4    2   -1  -23  -22  -44   17   -2    2   -1    0  }
{  -19  -62   -9    3    0    0    0    0  -12  -56   27   -7    3   -1    1    0  }
{    1    9   -5    0   -1    0    0    0    0   22   -1    2    0    1    0    0  }
{    5   17   -9    0   -2    1    0    0   13   54   -2    7   -1    1    0    0  }
{    7   27   56   -2   10   -3    3   -1   -2   -6    8  -28    3   -4    1   -1  }
{    0    0   19   -4    3   -2    2   -1   -3  -13   10   -4    1    0    0    0  }
{   -3    0  -27  -80   40  -16    6   -4    4    3   31   61  -22    7   -1    1  }
{    1    2   -8    6   -1    1    0    0    2    8   -5   -1    0    0    0    0  }
{   -4  -18  -57    8   -8    1   -3    0   -5  -20  -69    7   -6    2   -2    1  }
},
```

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol16to31[m − 16][n] with m = 16..31, n = 0..15

lowFreqTransMatrixCol16to31 =

```
{
{  30  -5  -6   1  -1   0   0   0  -8  -3   3   0   0   0   0   0 }
{  72 -29  -2   0  -1   0  -1   0 -37   6   7  -2   1   0   0   0 }
{   7 -38  10   0   1   0   0   0 -51  27   4  -3   2  -1   1   0 }
{ -45   4  -3   6  -1   2   0   1  49 -13   3  -3  -1   0   0   0 }
{  66  17 -24   4  -3   1  -1   0  13 -49  15   1   0   0   0   0 }
{  -9  69 -33   5  -2   0  -1   0 -44 -31  10   7  -2   2   0   1 }
{ -47 -34 -27   5   4  -1   1   0 -39  -2  27   4  -2   1   0   0 }
{ -33   3  22  -2  -4   1  -1   0 -58 -17   6  -6   7  -1   1   0 }
{   7  -8  16  -6   4  -2   1  -1 -15  54 -23   2  -1   0   0   0 }
{ -13  17   0  -2   0  -1   0   0 -46 -10 -10   4  -1   1   0   0 }
{   4  51  -3  -6  -1  -1   0   0 -20   6 -34   9  -2   2  -1   0 }
{  -1  -4 -68  35  -5   5  -2   1   0  35  43  -4  -6   1  -1   0 }
{  -6 -37 -18  -5   2  -2   1  -1   6  -6  -7  25  -6   4  -1   1 }
{  -4  -7 -26  -6 -10   6  -4   1   3   8  14 -18  15  -5   2  -1 }
{   1  24   3   5  -1   1   0   0  -3  12   6 -10   1  -1   0   0 }
{   1   4   0  33  -7   5  -2   1   0  -9  53 -22   3  -1   0   0 }
},
```

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol32to47[m − 32][n] with m = 32..47, n = 0..15

lowFreqTransMatrixCol32to47 =

```
{
{   3   2  -1   0  -2  -1   0   0   1   1   0   0  -1   0   0   0 }
{  12   3  -4   0  -3  -2   1   0   4   0   0   0  -1   0   0   0 }
{  31  -5  -8   3 -14   0   5  -1   6   1  -3   0  -4  -1   1   0 }
{ -19   2   0   0   5   1   1   0  -2   0  -1   0   1   0   0   0 }
{ -53  34   6  -5  30  -7 -11   3 -11  -2   5   1   4   2  -1  -1 }
{  49   7   2  -6 -23  -3  -2   2   9   4   0   0  -2  -1  -1   0 }
{ -11  32  -8  -7  27 -12  -6   6 -13   0   4  -3   3  -1  -2   1 }
{ -23  40  -2   5  43 -11  -8  -1 -18  -4   5   2   4   3   0  -1 }
{ -42 -25   4   6  34   8   2  -2 -15  -1   0  -1   3   2   0   1 }
{ -80 -27  20  -4 -66  23  -2  -2  20  -3  -2   3 -14   2   3  -1 }
{  16 -52  28   1  59  15  -8  -5 -28  -7   2   2  10   3   0  -1 }
{ -14 -38 -12 -10   9   5   7   6  -9   7  -4  -3   4  -4   0   3 }
{  16  10  55 -24  15  46 -52   1  35 -43  10  12 -23  13   5  -8 }
{  -2  -4  -1  13   0   2  -4  -3   3  -1   2   1  -2   0  -2  -1 }
{  -9  -1 -25  10  45 -11  18   2  86   1 -13  -4 -65  -6   7   2 }
{   4 -27  -2  -9   5  36 -13   5  -7 -17   1   2   4   6   4  -1 }
},
```

– Otherwise, if nTrS is equal to 48, lfnstTrSetIdx is equal to 3, and lfnst_idx is equal to 1 the following applies:

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol0to15[m][n] with m = 0..15, n = 0..15

lowFreqTransMatrixCol0to15 =

```
{
{-115  37   9   2   2   1   1   0  10 -29   8   0   1   0   1   0 }
{  15  51 -18   0  -3   0  -1   0 -95   7  34  -3   5  -1   2   0 }
{  29 -22  16  -6   3  -2   1  -1  -4 -80  12  15   0   3   0   1 }
{ -36 -98  25   5   4   1   2   1 -59  11 -17   1   1   1   0   0 }
{  -6  18   3  -3  -1   0   0   0 -50  -5 -38  12   0   2   0   1 }
{   4  15  52 -13   5  -3   2  -1 -17 -45  16  24  -2   4  -1   2 }
{ -20  -7 -43   4   0   1  -1   1  -7  35   0  12  -4   1  -1   0 }
{   4  29   1  26  -5   4  -2   1 -17  -7 -73   6   6   2   1   1 }
{  12  13  10   2  -1   3  -1   1  17  -2 -46  12   7   0   2   0 }
{   5  20  90 -17   4  -3   2  -1   6  66   8  28  -7   3  -1   1 }
{  -3  -4 -34 -12   2  -1  -1   0   5  25  11  43 -10   4  -2   1 }
{  -1  -3   2  19  -2   4  -1   2   9   3 -35  22  11   1   2   0 }
{  10  -4  -6  12   5   1   1   0  11  -9 -12  -2  -7   0  -1   0 }
{   4   6  14  53  -4   4   0   2   0  -1 -20 -13   3   2  -1   1 }
{   2   9  13  37  19   6   2   2  -9  -3  -9 -28 -20  -4  -3  -1 }
{   3  -3  12  84 -12   8  -2   3   6  13  50  -1  45   1   7   0 }
},
```

lowFreqTransMatrix [m][n] = lowFreqTransMatrixCol16to31[m − 16][n] with m = 16..31, n = 0..15

lowFreqTransMatrixCol16to31 =

```
{
{   23   -8   -8    1   -1    0    0    0    3    3   -2   -1    0    0    0    0  }
{   23  -47    1    6    0    1    0    1    8    5  -12    0   -1    0    0    0  }
{   45    7  -59    7   -2    1   -1    0  -15   41   -3  -16    2   -3    0   -1  }
{    6  -13    7   -3    0    0    0    0   14   -4  -14    3   -1    0    0    0  }
{    3   67   -7  -40    3   -6    1   -3  -12  -13   65   -3  -10    0   -1    0  }
{  -87   -8  -14    7    8    1    2    0   23  -35   -6   -3    1    1    0    0  }
{  -51   -2  -57    5   15    0    4    0    7   39    5  -55    1   -7    1   -3  }
{   -5   21   -3    5   -1   -3    0   -1  -11    2  -52   -3   27   -2    5    0  }
{   16  -45   -9  -53    6    1    1    0   70   16    8   -4  -37    1   -7    0  }
{   29    5  -19   12    9   -1    1    0  -10   14   -1  -13    7    0    1    0  }
{   23   20  -40   12   21   -3    4   -1   25  -28  -10    5    8    6    0    2  }
{   -7  -65  -19  -22   11    4    2    1  -75  -18    3   -1  -10    2    0    1  }
{   33  -10   -4   18   18   -4    4   -1   28  -72    1  -49   15    2    2    1  }
{   -3    1   -5   35  -16   -6   -1   -2   46   29   13   21   37   -5    4   -1  }
{    1   18    9   28   24    6    2    2  -20   -5  -25  -33  -36    9   -2    2  }
{   -2   18  -22  -37  -13   14    0    3    1  -12   -3    2  -15   -8    1   -1  }
},
```

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol32to47[m − 32][n] with m = 32..47, n = 0..15

lowFreqTransMatrixCol32to47 =

```
{
{    4    0    0   -1    1    1    0    0    2    0    0    0    0    0    0    0  }
{    3   -3    1   -1    2    1   -2    0    1   -1    0    0    1    1   -1    0  }
{    1    0    7   -2   -3    6    1   -2    0    0    1    0   -1    2    0   -1  }
{    2    8   -3   -5    2    0    0    0    0    3    0   -1    1    0    0    0  }
{    9  -20   -5   22   -2    0    0   -1    2   -3   -2    3   -1    0    1    0  }
{    2    5  -17    0    3   -1   -1   -5    0    1   -4    0    1    0    0   -2  }
{    1  -10   41    2    4   -3   -2    3   -1   -2    7    1    1   -1   -1    0  }
{    0   27    8  -58    2   -5   25    3    0    3    0   -5    0   -2    7    0  }
{  -12   29    3   21    4    0    5   -1   -3    4    1    4    2    0    1    0  }
{    0   -6   13   -4    0   -4    1    5    0   -1   -1    1    0   -1    0    0  }
{   -4   21  -64   -8   -5   19   10  -48    3   -1   10   -3    0    4    3   -6  }
{    2  -35  -27    4    1    8  -17  -19    3    0    3   -6    0    2   -1   -2  }
{   56  -23   22   -1    4   -1  -15   26    6    4  -10    0    0    2   -3    2  }
{  -10  -53  -18    8    9   12  -41  -25   -2    2   13  -16    4    1   -5    1  }
{  -13   42    1   57  -22   -2  -25  -28    5    6   19  -12   -5   -3   -2    4  }
{   19   14   -4  -12   -4    5   17    8    2   -4   -4    4   -2    2    1    0  }
},
```

– Otherwise, if nTrS is equal to 48, lfnstTrSetIdx is equal to 3, and lfnst_idx is equal to 2 the following applies:

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol0to15[m][n] with m = 0..15, n = 0..15

lowFreqTransMatrixCol0to15 =

```
{
{  109  -26   -8   -3   -2   -1   -1    0  -50   28    2    1    0    0    0    0  }
{  -39   31   -5    2   -1    1    0    0  -95    6   18    0    4    0    1    0  }
{   29   -3   -2   -2    0    0    0    0    0  -41    9    0    2    0    1    0  }
{   18   96  -23    2   -5    1   -2    0  -10    6   10   -2    1   -1    1    0  }
{  -29  -60   16   -2    3   -1    1    0  -52    9  -17    5   -2    1   -1    1  }
{  -23   -5  -15    5   -2    1   -1    1    2   79  -13   -4   -2   -1   -1    0  }
{   -7   -3   12   -3    3   -1    1    0  -31  -62    8    7    0    2    0    1  }
{    1  -26    5    0    1    0    1    0   24   -3   43   -6    4   -2    1   -1  }
{   11   14    6   -3    1   -1    1    0   10   -7   -9    3   -2    1   -1    0  }
{  -10  -11  -47    3   -4    1   -1    0    5   28   11   -2   -1    0    0    0  }
{   -8  -24  -99   11  -10    3   -4    1   -5  -36   19  -26    4   -5    1   -2  }
{   -5    1   -1    0    1    0    0    0  -10  -14   -6    8    0    1    0    0  }
{    1   12  -20   21   -4    5   -2    2   -5   -2  -75    9   -1    2   -1    1  }
{    2   -9  -18    8   -3    3   -1    1    3  -25  -62   -6    0   -2    0   -1  }
{    4    9   39   18    0    2    0    1   -6  -16  -22  -37    5   -5    1   -2  }
{   -7   -2   15   -6    1   -1    1   -1  -11   -3   22  -14    0   -2    1   -1  }
},
```

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol16to31[m − 16][n] with m = 16..31, n = 0..15

lowFreqTransMatrixCol16to31 =

```
{
{  -18   -8    6    0    1    0    1    0    6   -2   -3    0    0    0    0    0  }
{   32  -49    5    1    1    0    0    0   27   -1  -14    2   -2    1   -1    0  }
{   86    4  -33    2   -6    1   -2    0  -32   58    1   -7    0   -2    0   -1  }
{  -14   26    2   -4    1   -1    0    0  -43   -9   35   -2    4   -1    1    0  }
{   13   56   -2   -9    0   -2    0   -1  -34  -18   41    0    3    0    1    0  }
{   -9    1    5   -1    1    0    0    0   -4   49    2  -14    1   -3    0   -1  }
{  -75    9  -45    5   -1    1   -1    0   14   35    0  -23    2   -5    1   -2  }
{   -7  -64    9   14    0    3    0    1  -12   -4    5    3   -1    1    0    0  }
{   22   21    1  -21    2   -4    1   -2   92    1   53    0   -9    1   -2    0  }
{  -12   -2  -38    2    0    1    0    0   16   38   11  -16   -1   -3    0   -2  }
{    0   25   41    5   -3    1    0    0   10   -5   -7   12    2    1    0    0  }
{  -17   -2    7   -5    3   -1    0    0  -16   13    3   31   -1    6    0    2  }
{   -1   -2  -16   -4    0   -1    0    0   -7    7  -31    0    3    0    0    0  }
{   -6  -61   14  -51    2   -6    0   -2  -19    0   40   -7  -17    0   -3    0  }
{   -5   15   63    9  -16    0   -3    0   18   42  -18   27   15    1    3    1  }
{  -18   -7   30   -9   -4    0   -1    0  -35   23   23   10  -17    1   -3    0  }
},
```

lowFreqTransMatrix[m][n] = lowFreqTransMatrixCol32to47[m − 32][n] with m = 32..47, n = 0..15

lowFreqTransMatrixCol32to47 =

```
{
{   -3    2    1   -1    0    0    0    0   -2    0    0    0    0    0    0    0  }
{    3    5   -3   -2    4    1   -1   -1    2    0    0    0    2    0    0    0  }
{  -14   -8   20    0   -2   -3    0    4   -1   -1    0    0   -1    1    0    0  }
{   14  -40    1   10    2    1  -10    1    2   -4   -1   -1    0    0   -1    0  }
{   19  -36  -10   13    3    6  -14   -1    3    1   -1   -3    1    1   -1   -1  }
{  -31  -14   56   -1   13  -37   -4   20   -2    2  -10    0    2   -4    0   -1  }
{    1   -8   32   -1    7  -12   -4   10    0    2   -6   -1    2    0    0   -2  }
{    8  -59   -3   26   14    6  -58    6   -5   17   -7  -18    3    3   -1   -5  }
{  -21  -11    1   40   -5   -4  -24    5   -4    5   -6   -5    0    0    0   -3  }
{   12   -9  -22    7   -8   60    4  -36   -6  -15   54    7    3   -7   -8   14  }
{   -1    1    9   -3   -3  -14   -3   12    2    4  -13   -2   -1    3    2   -4  }
{  -93  -15  -46   -3   23  -19    0  -47    8    4    8    3    2    3    0    0  }
{    4   11  -12    4  -12   14  -50   -1   -8   32   -4  -54    2    0   30  -15  }
{   13   -4   11    9   17    0   24    5    1  -12    4   28    0    0  -15    8  }
{   12  -34    9  -24    4   28   -2    4  -11   -4   30    2    5  -13   -4   18  }
{  -19   53    6   48  -65   12  -12   11   -8  -16   10  -21   -2  -12    6    2  }
},
```

8.7.4.4 Transformation process

Inputs to this process are:

- a variable nTbS specifying the horizontal sample size of transformed samples,
- a variable nonZeroS specifying the horizontal sample size of non-zero scaled transform coefficients,
- a list of scaled transform coefficients x[j] with j = 0..nonZeroS − 1,
- a transform kernel type variable trType.

Output of this process is the list of transformed samples y[i] with i = 0..nTbS − 1.

The transformation matrix derivation process as specified in clause 8.7.4.5 is invoked with the transform size nTbS and the transform kernel Type trType as inputs, and the transformation matrix transMatrix as output.

Depending on the value of trType, the list of transformed samples y[i] with i = 0..nTbS − 1 is derived as follows:

- If trType is equal to 0, the following transform matrix multiplication applies:

$$y[\,i\,] = \sum_{j=0}^{\mathrm{nonZeroS}-1} \mathrm{transMatrix}[\,i\,][\,j * 2^{6-\mathrm{Log2(nTbS)}}\,] * x[\,j\,] \quad \text{with } i = 0..\mathrm{nTbS}-1 \tag{1171}$$

- Otherwise (trType is equal to 1 or trType is equal to 2), the following transform matrix multiplication applies:

$$y[\,i\,] = \sum_{j=0}^{\mathrm{nonZeroS}-1} \mathrm{transMatrix}[\,i\,][\,j\,] * x[\,j\,] \quad \text{with } i = 0..\mathrm{nTbS}-1 \tag{1172}$$

8.7.4.5 Transformation matrix derivation process

Inputs to this process are:

- a variable nTbS specifying the horizontal sample size of scaled transform coefficients,
- the transformation kernel type trType.

Output of this process is the transformation matrix transMatrix.

The transformation matrix transMatrix is derived based on trType and nTbS as follows:

– If trType is equal to 0, the following applies:

transMatrix[m][n] = transMatrixCol0to15[m][n] with m = 0..15, n = 0..63 (1173)

transMatrixCol0to15 = (1174)

```
{
{  64  64  64  64  64  64  64  64  64  64  64  64  64  64  64  64 }
{  91  90  90  90  88  87  86  84  83  81  79  77  73  71  69  65 }
{  90  90  88  85  82  78  73  67  61  54  46  38  31  22  13   4 }
{  90  88  84  79  71  62  52  41  28  15   2 -11 -24 -37 -48 -59 }
{  90  87  80  70  57  43  25   9  -9 -25 -43 -57 -70 -80 -87 -90 }
{  90  84  73  59  41  20  -2 -24 -44 -62 -77 -86 -90 -90 -83 -71 }
{  90  82  67  46  22  -4 -31 -54 -73 -85 -90 -88 -78 -61 -38 -13 }
{  90  79  59  33   2 -28 -56 -77 -88 -90 -81 -62 -37  -7  24  52 }
{  89  75  50  18 -18 -50 -75 -89 -89 -75 -50 -18  18  50  75  89 }
{  88  71  41   2 -37 -69 -87 -90 -73 -44  -7  33  65  86  90  77 }
{  88  67  31 -13 -54 -82 -90 -78 -46  -4  38  73  90  85  61  22 }
{  87  62  20 -28 -69 -90 -84 -56 -11  37  73  90  81  48   2 -44 }
{  87  57   9 -43 -80 -90 -70 -25  25  70  90  80  43  -9 -57 -87 }
{  86  52  -2 -56 -87 -84 -48   7  59  88  83  44 -11 -62 -90 -81 }
{  85  46 -13 -67 -90 -73 -22  38  82  88  54  -4 -61 -90 -78 -31 }
{  84  41 -24 -77 -90 -56   7  65  91  69  11 -52 -88 -79 -28  37 }
{  83  36 -36 -83 -83 -36  36  83  83  36 -36 -83 -83 -36  36  83 }
{  83  28 -44 -88 -73 -11  59  91  62  -7 -71 -90 -48  24  81  84 }
{  82  22 -54 -90 -61  13  78  85  31 -46 -90 -67   4  73  88  38 }
{  81  15 -62 -90 -44  37  88  69  -7 -77 -84 -24  56  91  52 -28 }
{  80   9 -70 -87 -25  57  90  43 -43 -90 -57  25  87  70  -9 -80 }
{  79   2 -77 -81  -7  73  83  11 -71 -84 -15  69  86  20 -65 -87 }
{  78  -4 -82 -73  13  85  67 -22 -88 -61  31  90  54 -38 -90 -46 }
{  77 -11 -86 -62  33  90  44 -52 -90 -24  69  83   2 -81 -71  20 }
{  75 -18 -89 -50  50  89  18 -75 -75  18  89  50 -50 -89 -18  75 }
{  73 -24 -90 -37  65  81 -11 -88 -48  56  86   2 -84 -59  44  90 }
{  73 -31 -90 -22  78  67 -38 -90 -13  82  61 -46 -88  -4  85  54 }
{  71 -37 -90  -7  86  48 -62 -79  24  91  20 -81 -59  52  84 -11 }
{  70 -43 -87   9  90  25 -80 -57  57  80 -25 -90  -9  87  43 -70 }
{  69 -48 -83  24  90   2 -90 -28  81  52 -65 -71  44  84 -20 -90 }
{  67 -54 -78  38  85 -22 -90   4  90  13 -88 -31  82  46 -73 -61 }
{  65 -59 -71  52  77 -44 -81  37  84 -28 -87  20  90 -11 -90   2 }
{  64 -64 -64  64  64 -64 -64  64  64 -64 -64  64  64 -64 -64  64 }
{  62 -69 -56  73  48 -79 -41  83  33 -86 -24  88  15 -90  -7  91 }
{  61 -73 -46  82  31 -88 -13  90  -4 -90  22  85 -38 -78  54  67 }
{  59 -77 -37  87  11 -91  15  86 -41 -73  62  56 -79 -33  88   7 }
{  57 -80 -25  90  -9 -87  43  70 -70 -43  87   9 -90  25  80 -57 }
{  56 -83 -15  90 -28 -77  65  44 -87  -2  88 -41 -69  73  33 -90 }
{  54 -85  -4  88 -46 -61  82  13 -90  38  67 -78 -22  90 -31 -73 }
{  52 -87   7  83 -62 -41  90 -20 -77  71  28 -91  33  69 -79 -15 }
{  50 -89  18  75 -75 -18  89 -50 -50  89 -18 -75  75  18 -89  50 }
{  48 -90  28  65 -84   7  79 -73 -15  87 -59 -37  91 -41 -56  88 }
{  46 -90  38  54 -90  31  61 -88  22  67 -85  13  73 -82   4  78 }
{  44 -91  48  41 -90  52  37 -90  56  33 -90  59  28 -88  62  24 }
{  43 -90  57  25 -87  70   9 -80  80  -9 -70  87 -25 -57  90 -43 }
{  41 -90  65  11 -79  83 -20 -59  90 -48 -33  87 -71  -2  73 -86 }
{  38 -88  73  -4 -67  90 -46 -31  85 -78  13  61 -90  54  22 -82 }
{  37 -86  79 -20 -52  90 -69   2  65 -90  56  15 -77  87 -41 -33 }
{  36 -83  83 -36 -36  83 -83  36  36 -83  83 -36 -36  83 -83  36 }
{  33 -81  87 -48 -15  71 -90  62  -2 -59  90 -73  20  44 -86  83 }
{  31 -78  90 -61   4  54 -88  82 -38 -22  73 -90  67 -13 -46  85 }
{  28 -73  91 -71  24  33 -77  90 -69  20  37 -79  90 -65  15  41 }
{  25 -70  90 -80  43   9 -57  87 -87  57  -9 -43  80 -90  70 -25 }
{  24 -65  88 -86  59 -15 -33  71 -90  83 -52   7  41 -77  91 -79 }
{  22 -61  85 -90  73 -38  -4  46 -78  90 -82  54 -13 -31  67 -88 }
{  20 -56  81 -91  83 -59  24  15 -52  79 -90  84 -62  28  11 -48 }
{  18 -50  75 -89  89 -75  50 -18 -18  50 -75  89 -89  75 -50  18 }
{  15 -44  69 -84  91 -86  71 -48  20  11 -41  65 -83  90 -87  73 }
{  13 -38  61 -78  88 -90  85 -73  54 -31   4  22 -46  67 -82  90 }
{  11 -33  52 -69  81 -88  91 -87  79 -65  48 -28   7  15 -37  56 }
{   9 -25  43 -57  70 -80  87 -90  90 -87  80 -70  57 -43  25  -9 }
{   7 -20  33 -44  56 -65  73 -81  86 -90  91 -90  87 -83  77 -69 }
{   4 -13  22 -31  38 -46  54 -61  67 -73  78 -82  85 -88  90 -90 }
{   2  -7  11 -15  20 -24  28 -33  37 -41  44 -48  52 -56  59 -62 }
},
```

transMatrix[m][n] = transMatrixCol16to31[m − 16][n] with m = 16..31, n = 0..63 (1175)
transMatrixCol16to31 = (1176)

```
{
{  64   64   64   64   64   64   64   64   64   64   64   64   64   64   64   64 }
{  62   59   56   52   48   44   41   37   33   28   24   20   15   11    7    2 }
{  -4  -13  -22  -31  -38  -46  -54  -61  -67  -73  -78  -82  -85  -88  -90  -90 }
{ -69  -77  -83  -87  -90  -91  -90  -86  -81  -73  -65  -56  -44  -33  -20   -7 }
{ -90  -87  -80  -70  -57  -43  -25   -9    9   25   43   57   70   80   87   90 }
{ -56  -37  -15    7   28   48   65   79   87   91   88   81   69   52   33   11 }
{  13   38   61   78   88   90   85   73   54   31    4  -22  -46  -67  -82  -90 }
{  73   87   90   83   65   41   11  -20  -48  -71  -86  -91  -84  -69  -44  -15 }
{  89   75   50   18  -18  -50  -75  -89  -89  -75  -50  -18   18   50   75   89 }
{  48   11  -28  -62  -84  -90  -79  -52  -15   24   59   83   91   81   56   20 }
{ -22  -61  -85  -90  -73  -38    4   46   78   90   82   54   13  -31  -67  -88 }
{ -79  -91  -77  -41    7   52   83   90   71   33  -15  -59  -86  -88  -65  -24 }
{ -87  -57   -9   43   80   90   70   25  -25  -70  -90  -80  -43    9   57   87 }
{ -41   15   65   90   79   37  -20  -69  -90  -77  -33   24   71   91   73   28 }
{  31   78   90   61    4  -54  -88  -82  -38   22   73   90   67   13  -46  -85 }
{  83   86   44  -20  -73  -90  -59    2   62   90   71   15  -48  -87  -81  -33 }
{  83   36  -36  -83  -83  -36   36   83   83   36  -36  -83  -83  -36   36   83 }
{  33  -41  -87  -77  -15   56   90   65   -2  -69  -90  -52   20   79   86   37 }
{ -38  -88  -73   -4   67   90   46  -31  -85  -78  -13   61   90   54  -22  -82 }
{ -86  -73   -2   71   87   33  -48  -90  -59   20   83   79   11  -65  -90  -41 }
{ -80   -9   70   87   25  -57  -90  -43   43   90   57  -25  -87  -70    9   80 }
{ -24   62   88   28  -59  -90  -33   56   90   37  -52  -90  -41   48   91   44 }
{  46   90   38  -54  -90  -31   61   88   22  -67  -85  -13   73   82    4  -78 }
{  88   56  -41  -91  -37   59   87   15  -73  -79    7   84   65  -28  -90  -48 }
{  75  -18  -89  -50   50   89   18  -75  -75   18   89   50  -50  -89  -18   75 }
{  15  -79  -69   33   91   28  -71  -77   20   90   41  -62  -83    7   87   52 }
{ -54  -85    4   88   46  -61  -82   13   90   38  -67  -78   22   90   31  -73 }
{ -90  -33   73   69  -41  -88   -2   87   44  -65  -77   28   90   15  -83  -56 }
{ -70   43   87   -9  -90  -25   80   57  -57  -80   25   90    9  -87  -43   70 }
{  -7   88   33  -79  -56   62   73  -41  -86   15   91   11  -87  -37   77   59 }
{  61   73  -46  -82   31   88  -13  -90   -4   90   22  -85  -38   78   54  -67 }
{  91    7  -90  -15   88   24  -86  -33   83   41  -79  -48   73   56  -69  -62 }
{  64  -64  -64   64   64  -64  -64   64   64  -64  -64   64   64  -64  -64   64 }
{  -2  -90   11   90  -20  -87   28   84  -37  -81   44   77  -52  -71   59   65 }
{ -67  -54   78   38  -85  -22   90    4  -90   13   88  -31  -82   46   73  -61 }
{ -90   20   84  -44  -71   65   52  -81  -28   90    2  -90   24   83  -48  -69 }
{ -57   80   25  -90    9   87  -43  -70   70   43  -87   -9   90  -25  -80   57 }
{  11   84  -52  -59   81   20  -91   24   79  -62  -48   86    7  -90   37   71 }
{  73   31  -90   22   78  -67  -38   90  -13  -82   61   46  -88    4   85  -54 }
{  90  -44  -59   84    2  -86   56   48  -88   11   81  -65  -37   90  -24  -73 }
{  50  -89   18   75  -75  -18   89  -50  -50   89  -18  -75   75   18  -89   50 }
{ -20  -71   81    2  -83   69   24  -90   52   44  -90   33   62  -86   11   77 }
{ -78   -4   82  -73  -13   85  -67  -22   88  -61  -31   90  -54  -38   90  -46 }
{ -87   65   20  -86   69   15  -84   71   11  -83   73    7  -81   77    2  -79 }
{ -43   90  -57  -25   87  -70   -9   80  -80    9   70  -87   25   57  -90   43 }
{  28   52  -91   56   24  -84   77   -7  -69   88  -37  -44   90  -62  -15   81 }
{  82  -22  -54   90  -61  -13   78  -85   31   46  -90   67    4  -73   88  -38 }
{  84  -81   24   48  -90   71   -7  -62   91  -59  -11   73  -88   44   28  -83 }
{  36  -83   83  -36  -36   83  -83   36   36  -83   83  -36  -36   83  -83   36 }
{ -37  -28   79  -88   52   11  -69   91  -65    7   56  -90   77  -24  -41   84 }
{ -85   46   13  -67   90  -73   22   38  -82   88  -54   -4   61  -90   78  -31 }
{ -81   90  -62   11   44  -83   88  -59    7   48  -84   87  -56    2   52  -86 }
{ -25   70  -90   80  -43   -9   57  -87   87  -57    9   43  -80   90  -70   25 }
{  44    2  -48   81  -90   73  -37  -11   56  -84   90  -69   28   20  -62   87 }
{  88  -67   31   13  -54   82  -90   78  -46    4   38  -73   90  -85   61  -22 }
{  77  -90   86  -65   33    7  -44   73  -90   87  -69   37    2  -41   71  -88 }
{  18  -50   75  -89   89  -75   50  -18  -18   50  -75   89  -89   75  -50   18 }
{ -52   24    7  -37   62  -81   90  -88   77  -56   28    2  -33   59  -79   90 }
{ -90   82  -67   46  -22   -4   31  -54   73  -85   90  -88   78  -61   38  -13 }
{ -71   83  -90   90  -86   77  -62   44  -24    2   20  -41   59  -73   84  -90 }
{  -9   25  -43   57  -70   80  -87   90  -90   87  -80   70  -57   43  -25    9 }
{  59  -48   37  -24   11    2  -15   28  -41   52  -62   71  -79   84  -88   90 }
{  90  -90   88  -85   82  -78   73  -67   61  -54   46  -38   31  -22   13   -4 }
{  65  -69   71  -73   77  -79   81  -83   84  -86   87  -88   90  -90   90  -91 }
},
```

transMatrix[m][n] = (n & 1 ? −1 : 1) * transMatrixCol16to31[47 −m][n] (1177)
with m =32..47, n = 0..63

transMatrix[m][n] = (n & 1 ? −1 : 1) * transMatrixCol0to15[63 −m][n] (1178)
with m =48..63, n = 0..63

– Otherwise, if trType is equal to 1 and nTbS is equal to 4, the following applies:

transMatrix[m][n] = (1179)

```
{
{  29   55   74   84 }
{  74   74    0  -74 }
{  84  -29  -74   55 }
{  55  -84   74  -29 }
},
```

– Otherwise, if trType is equal to 1 and nTbS is equal to 8, the following applies:

transMatrix[m][n] = (1180)

```
{
{  17   32   46   60   71   78   85   86 }
{  46   78   86   71   32  -17  -60  -85 }
{  71   85   32  -46  -86  -60   17   78 }
{  85   46  -60  -78   17   86   32  -71 }
{  86  -17  -85   32   78  -46  -71   60 }
{  78  -71  -17   85  -60  -32   86  -46 }
{  60  -86   71  -17  -46   85  -78   32 }
{  32  -60   78  -86   85  -71   46  -17 }
},
```

– Otherwise, if trType is equal to 1 and nTbS is equal to 16, the following applies:

transMatrix[m][n] = (1181)

```
{
{   8   17   25   33   40   48   55   62   68   73   77   81   85   87   88   88 }
{  25   48   68   81   88   88   81   68   48   25    0  -25  -48  -68  -81  -88 }
{  40   73   88   85   62   25  -17  -55  -81  -88  -77  -48   -8   33   68   87 }
{  55   87   81   40  -17  -68  -88  -73  -25   33   77   88   62    8  -48  -85 }
{  68   88   48  -25  -81  -81  -25   48   88   68    0  -68  -88  -48   25   81 }
{  77   77    0  -77  -77    0   77   77    0  -77  -77    0   77   77    0  -77 }
{  85   55  -48  -87   -8   81   62  -40  -88  -17   77   68  -33  -88  -25   73 }
{  88   25  -81  -48   68   68  -48  -81   25   88    0  -88  -25   81   48  -68 }
{  88   -8  -88   17   87  -25  -85   33   81  -40  -77   48   73  -55  -68   62 }
{  87  -40  -68   73   33  -88    8   85  -48  -62   77   25  -88   17   81  -55 }
{  81  -68  -25   88  -48  -48   88  -25  -68   81    0  -81   68   25  -88   48 }
{  73  -85   25   55  -88   48   33  -87   68    8  -77   81  -17  -62   88  -40 }
{  62  -88   68   -8  -55   88  -73   17   48  -87   77  -25  -40   85  -81   33 }
{  48  -81   88  -68   25   25  -68   88  -81   48    0  -48   81  -88   68  -25 }
{  33  -62   81  -88   85  -68   40   -8  -25   55  -77   88  -87   73  -48   17 }
{  17  -33   48  -62   73  -81   87  -88   88  -85   77  -68   55  -40   25   -8 }
},
```

– Otherwise, if trType is equal to 1 and nTbS is equal to 32, the following applies:

transMatrix[m][n] = transMatrixCol0to15[m][n] with m = 0..15, n = 0..15 (1182)

transMatrixCol0to15 = (1183)

```
{
{   4    9   13   17   21   26   30   34   38   42   46   50   53   56   60   63 }
{  13   26   38   50   60   68   77   82   86   89   90   88   85   80   74   66 }
{  21   42   60   74   84   89   89   84   74   60   42   21    0  -21  -42  -60 }
{  30   56   77   87   89   80   63   38    9  -21  -50  -72  -85  -90  -84  -68 }
{  38   68   86   88   74   46    9  -30  -63  -84  -90  -78  -53  -17   21   56 }
{  46   78   90   77   42   -4  -50  -80  -90  -74  -38    9   53   82   89   72 }
{  53   85   85   53    0  -53  -85  -85  -53    0   53   85   85   53    0  -53 }
{  60   89   74   21  -42  -84  -84  -42   21   74   89   60    0  -60  -89  -74 }
{  66   90   56  -13  -74  -87  -46   26   80   84   34  -38  -85  -78  -21   50 }
{  72   86   34  -46  -89  -63   13   78   82   21  -56  -90  -53   26   84   77 }
{  77   80    9  -72  -84  -17   66   86   26  -60  -88  -34   53   90   42  -46 }
{  80   72  -17  -86  -60   34   90   46  -50  -89  -30   63   85   13  -74  -78 }
{  84   60  -42  -89  -21   74   74  -21  -89  -42   60   84    0  -84  -60   42 }
{  86   46  -63  -78   21   90   26  -77  -66   42   87    4  -85  -50   60   80 }
{  88   30  -78  -56   60   77  -34  -87    4   89   26  -80  -53   63   74  -38 }
{  90   13  -87  -26   84   38  -78  -50   72   60  -63  -68   53   77  -42  -82 }
},
```

transMatrix[m][n] = transMatrixCol16to31[m − 16][n] with m = 16..31, n = 0..15 (1184)

transMatrixCol16to31 = (1185)

```
{
{  66  68  72  74  77  78  80  82  84  85  86  87  88  89  90  90 }
{  56  46  34  21   9  −4 −17 −30 −42 −53 −63 −72 −78 −84 −87 −90 }
{ −74 −84 −89 −89 −84 −74 −60 −42 −21   0  21  42  60  74  84  89 }
{ −46 −17  13  42  66  82  90  86  74  53  26  −4 −34 −60 −78 −88 }
{  80  90  82  60  26 −13 −50 −77 −89 −85 −66 −34   4  42  72  87 }
{  34 −13 −56 −84 −88 −68 −30  17  60  85  87  66  26 −21 −63 −86 }
{ −85 −85 −53   0  53  85  85  53   0 −53 −85 −85 −53   0  53  85 }
{ −21  42  84  84  42 −21 −74 −89 −60   0  60  89  74  21 −42 −84 }
{  88  72   9 −60 −90 −63   4  68  89  53 −17 −77 −86 −42  30  82 }
{   9 −66 −88 −42  38  87  68  −4 −74 −85 −30  50  90  60 −17 −80 }
{ −90 −50  38  89  56 −30 −87 −63  21  85  68 −13 −82 −74   4  78 }
{   4  82  68 −21 −87 −56  38  90  42 −53 −88 −26  66  84   9 −77 }
{  89  21 −74 −74  21  89  42 −60 −84   0  84  60 −42 −89 −21  74 }
{ −17 −90 −30  74  68 −38 −88  −9  84  53 −56 −82  13  89  34 −72 }
{ −86   9  90  21 −82 −50  66  72 −42 −85  13  90  17 −84 −46  68 }
{  30  86 −17 −89   4  90   9 −88 −21  85  34 −80 −46  74  56 −66 }
},
```

– Otherwise, if trType is equal to 2 and nTbS is equal to 4, the following applies:

transMatrix[m][n] = (1186)

```
{
{  84  74  55  29 }
{  74   0 −74 −74 }
{  55 −74 −29  84 }
{  29 −74  84 −55 }
},
```

– Otherwise, if trType is equal to 2 and nTbS is equal to 8, the following applies:

transMatrix[m][n] = (1187)

```
{
{  86  85  78  71  60  46  32  17 }
{  85  60  17 −32 −71 −86 −78 −46 }
{  78  17 −60 −86 −46  32  85  71 }
{  71 −32 −86 −17  78  60 −46 −85 }
{  60 −71 −46  78  32 −85 −17  86 }
{  46 −86  32  60 −85  17  71 −78 }
{  32 −78  85 −46 −17  71 −86  60 }
{  17 −46  71 −85  86 −78  60 −32 }
},
```

– Otherwise, if trType is equal to 2 and nTbS is equal to 16, the following applies:

transMatrix[m][n] = (1188)

```
{
{  88  88  87  85  81  77  73  68  62  55  48  40  33  25  17   8 }
{  88  81  68  48  25   0 −25 −48 −68 −81 −88 −88 −81 −68 −48 −25 }
{  87  68  33  −8 −48 −77 −88 −81 −55 −17  25  62  85  88  73  40 }
{  85  48  −8 −62 −88 −77 −33  25  73  88  68  17 −40 −81 −87 −55 }
{  81  25 −48 −88 −68   0  68  88  48 −25 −81 −81 −25  48  88  68 }
{  77   0 −77 −77   0  77  77   0 −77 −77   0  77  77   0 −77 −77 }
{  73 −25 −88 −33  68  77 −17 −88 −40  62  81  −8 −87 −48  55  85 }
{  68 −48 −81  25  88   0 −88 −25  81  48 −68 −68  48  81 −25 −88 }
{  62 −68 −55  73  48 −77 −40  81  33 −85 −25  87  17 −88  −8  88 }
{  55 −81 −17  88 −25 −77  62  48 −85  −8  88 −33 −73  68  40 −87 }
{  48 −88  25  68 −81   0  81 −68 −25  88 −48 −48  88 −25 −68  81 }
{  40 −88  62  17 −81  77  −8 −68  87 −33 −48  88 −55 −25  85 −73 }
{  33 −81  85 −40 −25  77 −87  48  17 −73  88 −55  −8  68 −88  62 }
{  25 −68  88 −81  48   0 −48  81 −88  68 −25 −25  68 −88  81 −48 }
{  17 −48  73 −87  88 −77  55 −25  −8  40 −68  85 −88  81 −62  33 }
{   8 −25  40 −55  68 −77  85 −88  88 −87  81 −73  62 −48  33 −17 }
},
```

– Otherwise, if trType is equal to 2 and nTbS is equal to 32, the following applies:

transMatrix[m][n] = transMatrixCol0to15[m][n] with m = 0..15, n = 0..15 (1189)

transMatrixCol0to15 = (1190)

```
{
{  90   90   89   88   87   86   85   84   82   80   78   77   74   72   68   66 }
{  90   87   84   78   72   63   53   42   30   17    4   -9  -21  -34  -46  -56 }
{  89   84   74   60   42   21    0  -21  -42  -60  -74  -84  -89  -89  -84  -74 }
{  88   78   60   34    4  -26  -53  -74  -86  -90  -82  -66  -42  -13   17   46 }
{  87   72   42    4  -34  -66  -85  -89  -77  -50  -13   26   60   82   90   80 }
{  86   63   21  -26  -66  -87  -85  -60  -17   30   68   88   84   56   13  -34 }
{  85   53    0  -53  -85  -85  -53    0   53   85   85   53    0  -53  -85  -85 }
{  84   42  -21  -74  -89  -60    0   60   89   74   21  -42  -84  -84  -42   21 }
{  82   30  -42  -86  -77  -17   53   89   68    4  -63  -90  -60    9   72   88 }
{  80   17  -60  -90  -50   30   85   74    4  -68  -87  -38   42   88   66   -9 }
{  78    4  -74  -82  -13   68   85   21  -63  -87  -30   56   89   38  -50  -90 }
{  77   -9  -84  -66   26   88   53  -42  -90  -38   56   87   21  -68  -82   -4 }
{  74  -21  -89  -42   60   84    0  -84  -60   42   89   21  -74  -74   21   89 }
{  72  -34  -89  -13   82   56  -53  -84    9   88   38  -68  -74   30   90   17 }
{  68  -46  -84   17   90   13  -85  -42   72   66  -50  -82   21   90    9  -86 }
{  66  -56  -74   46   80  -34  -85   21   88   -9  -90   -4   89   17  -86  -30 }
},
```

transMatrix[m][n] = transMatrixCol16to31[m − 16][n] with m = 16..31, n = 0..15 (1191)

transMatrixCol16to31 = (1192)

```
{
{  63   60   56   53   50   46   42   38   34   30   26   21   17   13    9    4 }
{ -66  -74  -80  -85  -88  -90  -89  -86  -82  -77  -68  -60  -50  -38  -26  -13 }
{ -60  -42  -21    0   21   42   60   74   84   89   89   84   74   60   42   21 }
{  68   84   90   85   72   50   21   -9  -38  -63  -80  -89  -87  -77  -56  -30 }
{  56   21  -17  -53  -78  -90  -84  -63  -30    9   46   74   88   86   68   38 }
{ -72  -89  -82  -53   -9   38   74   90   80   50    4  -42  -77  -90  -78  -46 }
{ -53    0   53   85   85   53    0  -53  -85  -85  -53    0   53   85   85   53 }
{  74   89   60    0  -60  -89  -74  -21   42   84   84   42  -21  -74  -89  -60 }
{  50  -21  -78  -85  -38   34   84   80   26  -46  -87  -74  -13   56   90   66 }
{ -77  -84  -26   53   90   56  -21  -82  -78  -13   63   89   46  -34  -86  -72 }
{ -46   42   90   53  -34  -88  -60   26   86   66  -17  -84  -72    9   80   77 }
{  78   74  -13  -85  -63   30   89   50  -46  -90  -34   60   86   17  -72  -80 }
{  42  -60  -84    0   84   60  -42  -89  -21   74   74  -21  -89  -42   60   84 }
{ -80  -60   50   85   -4  -87  -42   66   77  -26  -90  -21   78   63  -46  -86 }
{ -38   74   63  -53  -80   26   89    4  -87  -34   77   60  -56  -78   30   88 }
{  82   42  -77  -53   68   63  -60  -72   50   78  -38  -84   26   87  -13  -90 }
},
```

8.7.4.6 Residual modification process for blocks using colour space conversion

Inputs to this process are:

– a variable nTbW specifying the block width,

– a variable nTbH specifying the block height,

– an (nTbW)x(nTbH) array of luma residual samples r_Y with elements r_Y[x][y],

– an (nTbW)x(nTbH) array of chroma residual samples r_{Cb} with elements r_{Cb}[x][y],

– an (nTbW)x(nTbH) array of chroma residual samples r_{Cr} with elements r_{Cr}[x][y].

Outputs of this process are:

– a modified (nTbW)x(nTbH) array r_Y of luma residual samples,

– a modified (nTbW)x(nTbH) array r_{Cb} of chroma residual samples,

– a modified (nTbW)x(nTbH) array r_{Cr} of chroma residual samples.

The (nTbW)x(nTbH) arrays of residual samples r_Y, r_{Cb} and r_{Cr} are modified as follows:

$$r_Y[\,x\,][\,y\,] = \text{Clip3}(\,-(\,1 \ll (\,\text{BitDepth}+1\,)\,),\,(\,1 \ll (\,\text{BitDepth}+1\,)\,)-1,\,r_Y[\,x\,][\,y\,]\,) \quad (1193)$$

$$r_{Cb}[\,x\,][\,y\,] = \text{Clip3}(\,-(\,1 \ll (\,\text{BitDepth}+1\,)\,),\,(\,1 \ll (\,\text{BitDepth}+1\,)\,)-1,\,r_{Cb}[\,x\,][\,y\,]\,) \quad (1194)$$

$$r_{Cr}[\,x\,][\,y\,] = \text{Clip3}(\,-(\,1 \ll (\,\text{BitDepth}+1\,)\,),\,(\,1 \ll (\,\text{BitDepth}+1\,)\,)-1,\,r_{Cr}[\,x\,][\,y\,]\,) \quad (1195)$$

$$\text{tmp} = r_Y[\,x\,][\,y\,] - (\,r_{Cb}[\,x\,][\,y\,] \gg 1\,) \quad (1196)$$

$$r_Y[x][y] = tmp + r_{Cb}[x][y] \quad (1197)$$

$$r_{Cb}[x][y] = tmp − (r_{Cr}[x][y] >> 1) \quad (1198)$$

$$r_{Cr}[x][y] += r_{Cb}[x][y] \quad (1199)$$

8.7.5 Picture reconstruction process

8.7.5.1 General

Inputs to this process are:

– a location (xCurr, yCurr) specifying the top-left sample of the current block relative to the top-left sample of the current picture component,

– the variables nCurrSw and nCurrSh specifying the width and height, respectively, of the current block,

– a variable cIdx specifying the colour component of the current block,

– an nCurrSw x nCurrSh array predSamples specifying the predicted samples of the current block,

– an nCurrSw x nCurrSh array resSamples specifying the residual samples of the current block.

Output of this process is a reconstructed picture sample array recSamples.

Depending on the value of the colour component cIdx, the following assignments are made:

– If cIdx is equal to 0, recSamples corresponds to the reconstructed picture sample array S_L.

– Otherwise, if cIdx is equal to 1, tuCbfChroma is set equal to (tu_cb_coded_flag[xCurr * SubWidthC][yCurr * SubHeightC] | | tu_joint_cbcr_residual_flag[xCurr * SubWidthC][yCurr * SubHeightC]), recSamples corresponds to the reconstructed chroma sample array S_{Cb}.

– Otherwise (cIdx is equal to 2), tuCbfChroma is set equal to (tu_cr_coded_flag[xCurr * SubWidthC][yCurr * SubHeightC] | | tu_joint_cbcr_residual_flag[xCurr * SubWidthC][yCurr * SubHeightC]), recSamples corresponds to the reconstructed chroma sample array S_{Cr}.

Depending on the value of sh_lmcs_used_flag, the following applies:

– If sh_lmcs_used_flag is equal to 0, the nCurrSw x nCurrSh block of the reconstructed samples recSamples at location (xCurr, yCurr) is derived as follows for i = 0..nCurrSw − 1, j = 0..nCurrSh − 1:

recSamples[xCurr + i][yCurr + j] = Clip1(predSamples[i][j] + resSamples[i][j]) (1200)

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

  – If cIdx is equal to 0, the following applies:

    – The picture reconstruction with mapping process for luma samples as specified in clause 8.7.5.2 is invoked with the luma location (xCurr, yCurr), the block width nCurrSw and height nCurrSh, the predicted luma sample array predSamples, and the residual luma sample array resSamples as inputs, and the output is the reconstructed luma sample array recSamples.

  – Otherwise (cIdx is greater than 0), the picture reconstruction with luma dependent chroma residual scaling process for chroma samples as specified in clause 8.7.5.3 is invoked with the chroma location (xCurr, yCurr), the transform block width nCurrSw and height nCurrSh, the coded block flag of the current chroma transform block tuCbfChroma, the predicted chroma sample array predSamples, and the residual chroma sample array resSamples as inputs, and the output is the reconstructed chroma sample array recSamples.

The following assignments are made for i = 0..nCurrSw − 1, j = 0..nCurrSh − 1:

xVb = (xCurr + i) % ((cIdx = = 0) ? IbcBufWidthY : IbcBufWidthC) (1201)

yVb = (yCurr + j) % ((cIdx = = 0) ? CtbSizeY : (CtbSizeY / SubHeightC)) (1202)

IbcVirBuf[cIdx][xVb][yVb] = recSamples[xCurr + i][yCurr + j] (1203)

The variables subW and subH are derived as follows:

subW = cIdx = = 0 ? 1 : SubWidthC (1204)

$$\text{subH} = \text{cIdx} == 0\ ?\ 1 : \text{SubHeightC} \tag{1205}$$

The following assignments are made for i = 0..nCurrSw * subW − 1, j = 0..nCurrSh* subH − 1:

$$\text{IsAvailable[cIdx][xCurr * subW + i][yCurr * subH + j] = TRUE} \tag{1206}$$

8.7.5.2 Picture reconstruction with mapping process for luma samples

Inputs to this process are:

– a location (xCurr, yCurr) of the top-left sample of the current block relative to the top-left sample of the current picture,

– a variable nCurrSw specifying the block width,

– a variable nCurrSh specifying the block height,

– an nCurrSw x nCurrSh array predSamples specifying the luma predicted samples of the current block,

– an nCurrSw x nCurrSh array resSamples specifying the luma residual samples of the current block.

Output of this process is a reconstructed luma picture sample array recSamples.

The nCurrSw x nCurrSh array of mapped predicted luma samples predMapSamples is derived as follows:

– If one of the following conditions is true, predMapSamples[i][j] is set equal to predSamples[i][j] for i = 0..nCurrSw − 1, j = 0..nCurrSh − 1:

  – CuPredMode[0][xCurr][yCurr] is equal to MODE_INTRA.

  – CuPredMode[0][xCurr][yCurr] is equal to MODE_IBC.

  – CuPredMode[0][xCurr][yCurr] is equal to MODE_PLT.

  – CuPredMode[0][xCurr][yCurr] is equal to MODE_INTER and ciip_flag[xCurr][yCurr] is equal to 1.

– Otherwise (CuPredMode[0][xCurr][yCurr] is equal to MODE_INTER and ciip_flag[xCurr][yCurr] is equal to 0), the following applies:

$$\begin{aligned}&\text{idxY} = \text{predSamples[i][j]} \gg \text{Log2(OrgCW)}\\&\text{predMapSamples[i][j]} = \text{LmcsPivot[idxY]} +\\&\qquad (\,(\,\text{ScaleCoeff[idxY]} * (\,\text{predSamples[i][j]} - \text{InputPivot[idxY]}\,) + (\,1 \ll 10\,)\,) \gg 11\,)\\&\quad \text{with } i = 0..\text{nCurrSw} - 1,\ j = 0..\text{nCurrSh} - 1\end{aligned} \tag{1207}$$

The reconstructed luma picture sample recSamples is derived as follows:

$$\begin{aligned}&\text{recSamples[xCurr + i][yCurr + j] = Clip1(predMapSamples[i][j] + resSamples[i][j])}\\&\quad \text{with } i = 0..\text{nCurrSw} - 1,\ j = 0..\text{nCurrSh} - 1\end{aligned} \tag{1208}$$

8.7.5.3 Picture reconstruction with luma dependent chroma residual scaling process for chroma samples

Inputs to this process are:

– a chroma location (xCurr, yCurr) of the top-left chroma sample of the current chroma transform block relative to the top-left chroma sample of the current picture,

– a variable nCurrSw specifying the chroma transform block width,

– a variable nCurrSh specifying the chroma transform block height,

– a variable tuCbfChroma specifying the coded block flag of the current chroma transform block,

– an nCurrSw x nCurrSh array predSamples specifying the chroma prediction samples of the current block,

– an nCurrSw x nCurrSh array resSamples specifying the chroma residual samples of the current block.

Output of this process is a reconstructed chroma picture sample array recSamples.

The variable sizeY is set equal to Min(CtbSizeY, 64).

The reconstructed chroma picture sample recSamples is derived as follows for i = 0..nCurrSw − 1, j = 0..nCurrSh − 1:

– If one or more of the following conditions are true, recSamples[xCurr + i][yCurr + j] is set equal to Clip1(predSamples[i][j] + resSamples[i][j]):

– ph_chroma_residual_scale_flag is equal to 0.

– sh_lmcs_used_flag is equal to 0.

– nCurrSw * nCurrSh is less than or equal to 4.

– tu_cb_coded_flag [xCurr * SubWidthC][yCurr * SubHeightC] is equal to 0, tu_cr_coded_flag[xCurr * SubWidthC][yCurr * SubHeightC] is equal to 0, and cu_act_enabled_flag[xCurr * SubWidthC][yCurr * SubHeightC] is equal to 0.

– Otherwise, the following applies:

– The current luma location (xCurrY, yCurrY) is derived as follows:

(xCurrY, yCurrY) = (xCurr * SubWidthC, yCurr * SubHeightC) (1209)

– The luma location (xCuCb, yCuCb) is specified as the top-left luma sample location of the coding unit that contains the luma sample at (xCurrY / sizeY * sizeY, yCurrY / sizeY * sizeY).

– The variables availL and availT are derived as follows:

– The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location (xCurr, yCurr) set equal to (xCuCb, yCuCb), the neighbouring luma location (xNbY, yNbY) set equal to (xCuCb − 1, yCuCb), checkPredModeY set equal to FALSE, and cIdx set equal to 0 as inputs, and the output is assigned to availL.

– The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location (xCurr, yCurr) set equal to (xCuCb, yCuCb), the neighbouring luma location (xNbY, yNbY) set equal to (xCuCb, yCuCb − 1), checkPredModeY set equal to FALSE, and cIdx set equal to 0 as inputs, and the output is assigned to availT.

– The variable currPic specifies the array of reconstructed luma samples in the current picture.

– For the derivation of the variable varScale the following ordered steps apply:

1. The variable invAvgLuma is derived as follows:

– The array recLuma[i] with i=0..(2 * sizeY − 1) and the variable cnt are derived as follows:

– The variable cnt is set equal to 0.

– When availL is equal to TRUE, the array recLuma[i] with i = 0..sizeY − 1 is set equal to currPic[xCuCb − 1][Min(yCuCb + i, pps_pic_height_in_luma_samples − 1)] with i = 0..sizeY − 1, and cnt is set equal to sizeY.

– When availT is equal to TRUE, the array recLuma[cnt + i] with i = 0..sizeY − 1 is set equal to currPic[Min(xCuCb + i, pps_pic_width_in_luma_samples − 1)][yCuCb − 1] with i = 0..sizeY − 1, and cnt is set equal to (cnt + sizeY).

– The variable invAvgLuma is derived as follows:

– If cnt is greater than 0, the following applies:

$$\text{invAvgLuma} = (\textstyle\sum_{k=0}^{cnt-1} \text{recLuma}[\ k\ ] + (\text{cnt} \gg 1)) \gg \text{Log2}(\text{cnt}) \quad (1210)$$

– Otherwise (cnt is equal to 0), the following applies:

invAvgLuma = 1 << (BitDepth − 1) (1211)

2. The variable idxYInv is derived by invoking the identification of piece-wise function index process for a luma sample as specified in clause 8.8.2.3 with the variable lumaSample set equal to invAvgLuma as the input and idxYInv as the output.

3. The variable varScale is derived as follows:

varScale = ChromaScaleCoeff[idxYInv] (1212)

– The reconstructed chroma picture sample array recSamples[xCurr + i][yCurr + j] with i = 0..nCurrSw − 1, j = 0..nCurrSh − 1 is derived as follows:

  – If tuCbfChroma is equal to 1 or cu_act_enabled_flag[xCurr * SubWidthC][yCurr * SubHeightC] is equal to 1, the following applies:

$$\text{resSamples[i][j]} = \text{Clip3}(-(1 \ll \text{BitDepth}), (1 \ll \text{BitDepth}) - 1, \text{resSamples[i][j]}) \tag{1213}$$

$$\text{recSamples[xCurr + i][yCurr + j]} = \text{Clip1}(\text{predSamples[i][j]} + \text{Sign}(\text{resSamples[i][j]}) * ((\text{Abs}(\text{resSamples[i][j]}) * \text{varScale} + (1 \ll 10)) \gg 11)) \tag{1214}$$

  – Otherwise (tuCbfChroma is equal to 0 and cu_act_enabled_flag[xCurr * SubWidthC][yCurr * SubHeightC] is equal to 0), the following applies:

$$\text{recSamples[xCurr + i][yCurr + j]} = \text{predSamples[i][j]} \tag{1215}$$

8.8 In-loop filter process

8.8.1 General

The in-loop filter process is applied as specified by the following ordered steps:

1. When sps_lmcs_enabled_flag is equal to 1, the following applies:
   – The picture inverse mapping process for luma samples as specified in clause 8.8.2.1 is invoked with the reconstructed luma sample array S_L as inputs, and the modified reconstructed luma sample array S'_L after picture inverse mapping process for luma samples as outputs.
   – The array S'_L is assigned to the array S_L (which represent the decoded picture).
2. For the deblocking filter, the following applies:
   – The deblocking filter process as specified in clause 8.8.3.1 is invoked with the reconstructed picture sample array S_L and, when sps_chroma_format_idc is not equal to 0, the arrays S_{Cb} and S_{Cr} as inputs, and the modified reconstructed picture sample array S'_L and, when sps_chroma_format_idc is not equal to 0, the arrays S'_{Cb} and S'_{Cr} after deblocking as outputs.
   – The array S'_L and, when sps_chroma_format_idc is not equal to 0, the arrays S'_{Cb} and S'_{Cr} are assigned to the array S_L and, when sps_chroma_format_idc is not equal to 0, the arrays S_{Cb} and S_{Cr} (which represent the decoded picture), respectively.
3. When sps_sao_enabled_flag is equal to 1, the following applies:
   – The sample adaptive offset process as specified in clause 8.8.4.1 is invoked with the reconstructed picture sample array S_L and, when sps_chroma_format_idc is not equal to 0, the arrays S_{Cb} and S_{Cr} as inputs, and the modified reconstructed picture sample array S'_L and, when sps_chroma_format_idc is not equal to 0, the arrays S'_{Cb} and S'_{Cr} after sample adaptive offset as outputs.
   – The array S'_L and, when sps_chroma_format_idc is not equal to 0, the arrays S'_{Cb} and S'_{Cr} are assigned to the array S_L and, when sps_chroma_format_idc is not equal to 0, the arrays S_{Cb} and S_{Cr} (which represent the decoded picture), respectively.
4. When sps_alf_enabled_flag is equal to 1, the following applies:
   – The adaptive loop filter process as specified in clause 8.8.5.1 is invoked with the reconstructed picture sample array S_L and, when sps_chroma_format_idc is not equal to 0, the arrays S_{Cb} and S_{Cr} as inputs, and the modified reconstructed picture sample array S'_L and, when sps_chroma_format_idc is not equal to 0, the arrays S'_{Cb} and S'_{Cr} after adaptive loop filter as outputs.
   – The array S'_L and, when sps_chroma_format_idc is not equal to 0, the arrays S'_{Cb} and S'_{Cr} are assigned to the array S_L and, when sps_chroma_format_idc is not equal to 0, the arrays S_{Cb} and S_{Cr} (which represent the decoded picture), respectively.

8.8.2 Picture inverse mapping process for luma samples

8.8.2.1 General

Input to this process is a reconstructed picture luma sample array S_L.

Output of this process is a modified reconstructed picture luma sample array S_L.

The inverse mapping process for a luma sample $S_L[x][y]$ with x = 0..pps_pic_width_in_luma_samples − 1, y = 0..pps_pic_height_in_luma_samples − 1 is invoked as specified in clause 8.8.2.2 with the variable lumaSample set equal to $S_L[x][y]$ as the input and the output is assigned to the luma sample $S_L[x][y]$.

8.8.2.2 Inverse mapping process for a luma sample

Input to this process is a luma sample lumaSample.

Output of this process is a modified luma sample invLumaSample.

The value of invLumaSample is derived as follows:

– If sh_lmcs_used_flag of the slice that contains the luma sample lumaSample is equal to 1, the following ordered steps apply:

1. The variable idxYInv is derived by invoking the identification of piece-wise function index process for a luma sample as specified in clause 8.8.2.3 with lumaSample as the input and idxYInv as the output.

2. The variable invSample is derived as follows:

```
invSample = InputPivot[ idxYInv ] + ( ( InvScaleCoeff[ idxYInv ] *                    (1216)
                    ( lumaSample − LmcsPivot[ idxYInv ] ) + ( 1  <<  10 ) )  >>  11 )
```

3. The inverse mapped luma sample invLumaSample is derived as follows:

```
invLumaSample = Clip1( invSample )                                                    (1217)
```

– Otherwise, invLumaSample is set equal to lumaSample.

8.8.2.3 Identification of piecewise function index process for a luma sample

Input to this process is a luma sample lumaSample.

Output of this process is an index idxYInv identifying the piece to which the luma sample lumaSample belongs.

The variable idxYInv is derived as follows:

```
for( idxYInv = lmcs_min_bin_idx; idxYInv  <=  LmcsMaxBinIdx; idxYInv++ ) {
    if( lumaSample < LmcsPivot[ idxYInv + 1 ] )                                       (1218)
        break
}
idxYInv = Min( idxYInv, 15 )
```

8.8.3 Deblocking filter process

8.8.3.1 General

Inputs to this process are the reconstructed picture prior to deblocking, i.e., the array $recPicture_L$ and, when sps_chroma_format_idc is not equal to 0, the arrays $recPicture_{Cb}$ and $recPicture_{Cr}$.

Outputs of this process are the modified reconstructed picture after deblocking, i.e., the array $recPicture_L$ and, when sps_chroma_format_idc is not equal to 0, the arrays $recPicture_{Cb}$ and $recPicture_{Cr}$.

The vertical edges in a picture are filtered first. Then the horizontal edges in a picture are filtered with samples modified by the vertical edge filtering process as input. The vertical and horizontal edges in the CTBs of each CTU are processed separately on a coding unit basis. The vertical edges of the coding blocks in a coding unit are filtered starting with the edge on the left-hand side of the coding blocks proceeding through the edges towards the right-hand side of the coding blocks in their geometrical order. The horizontal edges of the coding blocks in a coding unit are filtered starting with the edge on the top of the coding blocks proceeding through the edges towards the bottom of the coding blocks in their geometrical order.

NOTE – Although the filtering process is specified on a picture basis in this Specification, the filtering process could be implemented on a coding unit basis with an equivalent result, provided the decoder properly accounts for the processing dependency order so as to produce the same output values.

The deblocking filter process is applied to all subblock edges and transform block edges of a picture, except the following types of edges:

– Edges that are at the boundary of the picture,

– Edges that coincide with the boundaries of a subpicture with subpicture index subpicIdx and sps_loop_filter_across_subpic_enabled_flag[subpicIdx] is equal to 0,

– Edges that coincide with the virtual boundaries of the picture when VirtualBoundariesPresentFlag is equal to 1,

– Edges that coincide with tile boundaries when pps_loop_filter_across_tiles_enabled_flag is equal to 0,

– Edges that coincide with slice boundaries when pps_loop_filter_across_slices_enabled_flag is equal to 0,

– Edges that coincide with upper or left boundaries of slices with sh_deblocking_filter_disabled_flag equal to 1,

– Edges within slices with sh_deblocking_filter_disabled_flag equal to 1,

– Edges that do not correspond to 4×4 sample grid boundaries of the luma component,

– Edges that do not correspond to 8×8 sample grid boundaries of the chroma component,

– Edges within the luma component for which both sides of the edge have intra_bdpcm_luma_flag equal to 1,

– Edges within the chroma components for which both sides of the edge have intra_bdpcm_chroma_flag equal to 1,

– Edges of chroma subblocks that are not edges of the associated transform unit.

The edge type, vertical or horizontal, is represented by the variable edgeType as specified in Table 42.

Table 42 – Name of association to edgeType

edgeType	Name of edgeType
0 (vertical edge)	EDGE_VER
1 (horizontal edge)	EDGE_HOR

The following applies:

– The variable treeType is set equal to DUAL_TREE_LUMA.

– The vertical edges are filtered by invoking the deblocking filter process for one direction as specified in clause 8.8.3.2 with the variable treeType, the reconstructed picture prior to deblocking, i.e., the array $recPicture_L$ and the variable edgeType set equal to EDGE_VER as inputs, and the modified reconstructed picture after deblocking, i.e., the array $recPicture_L$ as outputs.

– The horizontal edge are filtered by invoking the deblocking filter process for one direction as specified in clause 8.8.3.2 with the variable treeType, the modified reconstructed picture after deblocking, i.e., the array $recPicture_L$ and the variable edgeType set equal to EDGE_HOR as inputs, and the modified reconstructed picture after deblocking, i.e., the array $recPicture_L$ as outputs.

– When sps_chroma_format_idc is not equal to 0, the following applies:

  – The variable treeType is set equal to DUAL_TREE_CHROMA.

  – The vertical edges are filtered by invoking the deblocking filter process for one direction as specified in clause 8.8.3.2 with the variable treeType, the reconstructed picture prior to deblocking, i.e., the arrays $recPicture_{Cb}$ and $recPicture_{Cr}$, and the variable edgeType set equal to EDGE_VER as inputs, and the modified reconstructed picture after deblocking, i.e., the arrays $recPicture_{Cb}$ and $recPicture_{Cr}$ as outputs.

  – The horizontal edge are filtered by invoking the deblocking filter process for one direction as specified in clause 8.8.3.2 with the variable treeType, the modified reconstructed picture after deblocking, i.e., the arrays $recPicture_{Cb}$ and $recPicture_{Cr}$, and the variable edgeType set equal to EDGE_HOR as inputs, and the modified reconstructed picture after deblocking, i.e., the arrays $recPicture_{Cb}$ and $recPicture_{Cr}$ as outputs.

8.8.3.2 Deblocking filter process for one direction

Inputs to this process are:

– the variable treeType specifying whether the luma (DUAL_TREE_LUMA) or chroma components (DUAL_TREE_CHROMA) are currently processed,

– when treeType is equal to DUAL_TREE_LUMA, the reconstructed picture prior to deblocking, i.e., the array $recPicture_L$,

– when sps_chroma_format_idc is not equal to 0 and treeType is equal to DUAL_TREE_CHROMA, the arrays $recPicture_{Cb}$ and $recPicture_{Cr}$,

– a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered.

Outputs of this process are the modified reconstructed picture after deblocking, i.e:

– when treeType is equal to DUAL_TREE_LUMA, the array $recPicture_L$,

– when sps_chroma_format_idc is not equal to 0 and treeType is equal to DUAL_TREE_CHROMA, the arrays $recPicture_{Cb}$ and $recPicture_{Cr}$.

The variables firstCompIdx and lastCompIdx are derived as follows:

firstCompIdx = (treeType = = DUAL_TREE_CHROMA) ? 1 : 0 (1219)

lastCompIdx = (treeType = = DUAL_TREE_LUMA | | sps_chroma_format_idc = = 0) ? 0 : 2 (1220)

When sh_deblocking_filter_disabled_flag of the current slice is equal to 0, for each coding unit and each coding block per colour component of a coding unit indicated by the colour component index cIdx ranging from firstCompIdx to lastCompIdx, inclusive, with coding block width nCbW, coding block height nCbH and location of top-left sample of the coding block (xCb, yCb), when cIdx is equal to 0, or when cIdx is not equal to 0 and edgeType is equal to EDGE_VER and xCb % 8 is equal 0, or when cIdx is not equal to 0 and edgeType is equal to EDGE_HOR and yCb % 8 is equal to 0, the edges are filtered by the following ordered steps:

1. The variable filterEdgeFlag is derived as follows:
   - If edgeType is equal to EDGE_VER and one or more of the following conditions are true, filterEdgeFlag is set equal to 0:
     - The left boundary of the current coding block is the left boundary of the picture.
     - The left boundary of the current coding block coincides with the left boundary of the current subpicture and sps_loop_filter_across_subpic_enabled_flag[CurrSubpicIdx] or sps_loop_filter_across_subpic_enabled_flag[subpicIdx] is equal to 0, where subpicIdx is the subpicture index of the subpicture for which the left boundary of the current coding block coincides with the right subpicture boundary of that subpicture.
     - The left boundary of the current coding block is the left boundary of the tile and pps_loop_filter_across_tiles_enabled_flag is equal to 0.
     - The left boundary of the current coding block is the left boundary of the slice and pps_loop_filter_across_slices_enabled_flag is equal to 0.
   - Otherwise, if edgeType is equal to EDGE_HOR and one or more of the following conditions are true, the variable filterEdgeFlag is set equal to 0:
     - The top boundary of the current coding block is the top boundary of the picture.
     - The top boundary of the current coding block coincides with the top boundary of the current subpicture and sps_loop_filter_across_subpic_enabled_flag[CurrSubpicIdx] or sps_loop_filter_across_subpic_enabled_flag[subpicIdx] is equal to 0, where subpicIdx is the subpicture index of the subpicture for which the top boundary of the current coding block coincides with the bottom subpicture boundary of that subpicture.
     - The top boundary of the current coding block is the top boundary of the tile and pps_loop_filter_across_tiles_enabled_flag is equal to 0.
     - The top boundary of the current coding block is the top boundary of the slice and pps_loop_filter_across_slices_enabled_flag is equal to 0.
   - Otherwise, filterEdgeFlag is set equal to 1.
2. All elements of the two-dimensional (nCbW)x(nCbH) array edgeIdc, maxFilterLengthQs and maxFilterLengthPs are initialized to be equal to zero.
3. The derivation process of transform block boundary specified in clause 8.8.3.3 is invoked with the location (xCb, yCb), the coding block width nCbW, the coding block height nCbH, the variable cIdx, the variable filterEdgeFlag, the array edgeIdc, the maximum filter length arrays maxFilterLengthPs and maxFilterLengthQs, and the variable edgeType as inputs, and the modified array edgeIdc, the modified maximum filter length arrays maxFilterLengthPs and maxFilterLengthQs as outputs.

4. When cIdx is equal to 0, the derivation process of subblock boundary specified in clause 8.8.3.4 is invoked with the location (xCb, yCb), the coding block width nCbW, the coding block height nCbH, the variable filterEdgeFlag, the array edgeIdc, the maximum filter length arrays maxFilterLengthPs and maxFilterLengthQs, and the variable edgeType as inputs, and the modified array edgeIdc, the modified maximum filter length arrays maxFilterLengthPs and maxFilterLengthQs as outputs.

5. The picture sample array recPicture is derived as follows:

   – If cIdx is equal to 0, recPicture is set equal to the reconstructed luma picture sample array prior to deblocking $recPicture_L$.

   – Otherwise, if cIdx is equal to 1, recPicture is set equal to the reconstructed chroma picture sample array prior to deblocking $recPicture_{Cb}$.

   – Otherwise (cIdx is equal to 2), recPicture is set equal to the reconstructed chroma picture sample array prior to deblocking $recPicture_{Cr}$.

6. The derivation process of the boundary filtering strength specified in clause 8.8.3.5 is invoked with the picture sample array recPicture, the location (xCb, yCb), the coding block width nCbW, the coding block height nCbH, the variable edgeType, the variable cIdx, and the array edgeIdc as inputs, and an (nCbW)x(nCbH) array bS as output.

7. The edge filtering process for one direction is invoked for a coding block as specified in clause 8.8.3.6 with the variable edgeType, the variable cIdx, the reconstructed picture prior to deblocking recPicture, the location (xCb, yCb), the coding block width nCbW, the coding block height nCbH, and the arrays bS, maxFilterLengthPs, and maxFilterLengthQs, as inputs, and the modified reconstructed picture recPicture as output.

8.8.3.3 Derivation process of transform block boundary

Inputs to this process are:

– a location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,

– a variable nCbW specifying the width of the current coding block,

– a variable nCbH specifying the height of the current coding block,

– a variable cIdx specifying the colour component of the current coding block,

– a variable filterEdgeFlag,

– a two-dimensional (nCbW)x(nCbH) array edgeIdc,

– two-dimensional (nCbW)x(nCbH) arrays maxFilterLengthQs and maxFilterLengthPs,

– a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered.

Outputs of this process are:

– the modified two-dimensional (nCbW)x(nCbH) array edgeIdc,

– the modified two-dimensional (nCbW)x(nCbH) arrays maxFilterLengthQs, maxFilterLengthPs.

Depending on edgeType, the arrays edgeIdc, maxFilterLengthPs and maxFilterLengthQs are derived as follows:

– The variable gridSize is set as follows:

$$gridSize = cIdx == 0 ? 4 : 8 \quad (1221)$$

– If edgeType is equal to EDGE_VER, the following applies:

  – The variable numEdges is set equal to Max(1, nCbW / gridSize).

  – For xEdge = 0..numEdges − 1 and y = 0..nCbH − 1, the following applies:

    – The horizontal position x inside the current coding block is set equal to xEdge * gridSize.

    – The value of edgeIdc[x][y] is derived as follows:

      – If x is equal to 0, edgeIdc[x][y] is set equal to filterEdgeFlag.

– Otherwise, if the transform block covering the location (xCb + x, yCb + y) does not cover the location (xCb + x − 1, yCb + y) for the colour component cIdx, edgeIdc[x][y] is set equal to 1.

– When edgeIdc[x][y] is equal to 1,the following applies:

– If cIdx is equal to 0, the following applies:

– The value of maxFilterLengthQs[x][y] is derived as follows:

– If the width in luma samples of the transform block at luma location (xCb + x, yCb + y) is equal to or less than 4 or the width in luma samples of the transform block at luma location (xCb + x − 1, yCb + y) is equal to or less than 4, maxFilterLengthQs[x][y] is set equal to 1.

– Otherwise, if the width in luma samples of the transform block at luma location (xCb + x, yCb + y) is equal to or greater than 32, maxFilterLengthQs[x][y] is set equal to 7.

– Otherwise, maxFilterLengthQs[x][y] is set equal to 3.

– The value of maxFilterLengthPs[x][y] is derived as follows:

– If the width in luma samples of the transform block at luma location (xCb + x, yCb + y) is equal to or less than 4 or the width in luma samples of the transform block at luma location (xCb + x − 1, yCb + y) is equal to or less than 4, maxFilterLengthPs[x][y] is set equal to 1.

– Otherwise, if the width in luma samples of the transform block at luma location (xCb + x − 1, yCb + y) is equal to or greater than 32, maxFilterLengthPs[x][y] is set equal to 7.

– Otherwise, maxFilterLengthPs[x][y] is set equal to 3.

– Otherwise (cIdx is not equal to 0), the values of maxFilterLengthPs[x][y] and maxFilterLengthQs[x][y] are derived as follows:

– If the width in chroma samples of the transform block at chroma location (xCb + x, yCb + y) and the width at chroma location (xCb + x − 1, yCb + y) are both equal to or greater than 8, maxFilterLengthPs[x][y] and maxFilterLengthQs[x][y] are set equal to 3.

– Otherwise, maxFilterLengthPs[x][y] and maxFilterLengthQs[x][y] are set equal to 1.

– Otherwise (edgeType is equal to EDGE_HOR), the following applies:

– The variable numEdges is set equal to Max(1, nCbH / gridSize).

– For yEdge = 0..numEdges − 1 and x = 0..nCbW − 1, the following applies:

– The vertical position y inside the current coding block is set equal to yEdge * gridSize.

– The value of edgeIdc[x][y] is derived as follows:

– If y is equal to 0, edgeIdc[x][y] is set equal to filterEdgeFlag.

– Otherwise, if the transform block covering the location (xCb + x, yCb + y) does not cover the location (xCb + x, yCb + y − 1) for the colour component cIdx, edgeIdc[x][y] is set equal to 1.

– When edgeIdc[x][y] is equal to 1,the following applies:

– If cIdx is equal to 0, the following applies:

– The value of maxFilterLengthQs[x][y] is derived as follows:

– If the height in luma samples of the transform block at luma location (xCb + x, yCb + y) is equal to or less than 4 or the height in luma samples of the transform block at luma location (xCb + x, yCb + y − 1) is equal to or less than 4, maxFilterLengthQs[x][y] is set equal to 1.

– Otherwise, if the height in luma samples of the transform block at luma location (xCb + x, yCb + y) is equal to or greater than 32, maxFilterLengthQs[x][y] is set equal to 7.

– Otherwise, maxFilterLengthQs[x][y] is set equal to 3.

– The value of maxFilterLengthPs[x][y] is derived as follows:

  – If the height in luma samples of the transform block at luma location (xCb + x, yCb + y) is equal to or less than 4 or the height in luma samples of the transform block at luma location (xCb + x, yCb + y − 1) is equal to or less than 4, maxFilterLengthPs[x][y] is set equal to 1.

  – Otherwise, if the height in luma samples of the transform block at luma location (xCb + x, yCb + y − 1) is equal to or greater than 32, maxFilterLengthPs[x][y] is set equal to 7.

  – Otherwise, maxFilterLengthPs[x][y] is set equal to 3.

– Otherwise (cIdx is not equal to 0), the values of maxFilterLengthPs[x][y] and maxFilterLengthQs[x][y] are derived as follows:

  – If the height in chroma samples of the transform block at chroma location (xCb + x, yCb + y) and the height in chroma samples of the transform block at chroma location (xCb + x, yCb + y − 1) are both equal to or greater than 8, the following applies:

    – If (yCb + y) % CtbHeightC is greater than 0, i.e., the horizontal edge do not overlap with the upper chroma CTB boundary, both maxFilterLengthPs[x][y] and maxFilterLengthQs[x][y] are set equal to 3.

    – Otherwise ((yCb + y) % CtbHeightC is equal to 0, i.e., the horizontal edge overlaps with the upper chroma CTB boundary), maxFilterLengthPs[x][y] is set equal to 1 and maxFilterLengthQs[x][y] is set equal to 3.

  – Otherwise, maxFilterLengthPs[x][y] and maxFilterLengthQs[x][y] are set equal to 1.

8.8.3.4 Derivation process of subblock boundary

Inputs to this process are:

– a location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,

– a variable nCbW specifying the width of the current coding block,

– a variable nCbH specifying the height of the current coding block,

– a variable filterEdgeFlag,

– a two-dimensional (nCbW)x(nCbH) array edgeIdc,

– two-dimensional (nCbW)x(nCbH) arrays maxFilterLengthQs and maxFilterLengthPs,

– a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered.

Outputs of this process are:

– the modified two-dimensional (nCbW)x(nCbH) array edgeIdc,

– the modified two-dimensional (nCbW)x(nCbH) arrays maxFilterLengthQs and maxFilterLengthPs.

The number of subblock in horizontal direction numSbX and in vertical direction numSbY are derived as follows:

– If inter_affine_flag[xCb][yCb] is equal to 1 or merge_subblock_flag[xCb][yCb] is equal to 1, numSbX and numSbY are set equal to NumSbX[xCb][yCb] and NumSbY[xCb][yCb], respectively.

– Otherwise, numSbX and numSbY are both set equal to 1.

Depending on the value of edgeType the following applies:

– If edgeType is equal to EDGE_VER, the following applies:

  – The variable sbW is set equal to Max(8, nCbW / numSbX).

  – The array edgeTbFlags is set equal to edgeIdc.

  – For xEdge = 0..Min(Max(1, nCbW / 8) − 1, numSbX − 1), y = 0..nCbH − 1:

– The horizontal position x inside the current coding block is set equal to xEdge *sbW.

– When x is greater than 0 or filterEdgeFlag is equal to 1, the value of edgeIdc[x][y] is modified as follows:

$$\text{edgeIdc[x][y]} = 2 \tag{1222}$$

– When edgeIdc[x][y] is equal to 1 or 2, the values of maxFilterLengthPs[x][y] and maxFilterLengthQs[x][y] are modified as follows:

  – If x is equal to 0, the following applies:

    – When numSbX is greater than 1, the following applies:

$$\text{maxFilterLengthQs[x][y]} = \text{Min(5, maxFilterLengthQs[x][y])} \tag{1223}$$

    – When InterAffineFlag[xCb − 1][yCb + y] is equal to 1 or MergeSubblockFlag[xCb − 1][yCb + y] is equal to 1, the following applies:

$$\text{maxFilterLengthPs[x][y]} = \text{Min(5, maxFilterLengthPs[x][y])} \tag{1224}$$

  – Otherwise, if edgeTbFlags[x][y] is equal to 1, the following applies:

$$\text{maxFilterLengthPs[x][y]} = \text{Min(5, maxFilterLengthPs[x][y])} \tag{1225}$$

$$\text{maxFilterLengthQs[x][y]} = \text{Min(5, maxFilterLengthQs[x][y])} \tag{1226}$$

  – Otherwise, if one or more of the following conditions are true:

    – edgeTbFlags[x − 4][y] is equal to 1,

    – edgeTbFlags[x + 4][y] is equal to 1,

  the following applies:

$$\text{maxFilterLengthPs[x][y]} = 1 \tag{1227}$$

$$\text{maxFilterLengthQs[x][y]} = 1 \tag{1228}$$

  – Otherwise, if one or more of the following conditions are true:

    – xEdge is equal to 1,

    – xEdge is equal to (nCbW / 8) − 1,

    – edgeTbFlags[x − sbW][y] is equal to 1,

    – edgeTbFlags[x + sbW][y] is equal to 1,

  the following applies:

$$\text{maxFilterLengthPs[x][y]} = 2 \tag{1229}$$

$$\text{maxFilterLengthQs[x][y]} = 2 \tag{1230}$$

  – Otherwise, the following applies:

$$\text{maxFilterLengthPs[x][y]} = 3 \tag{1231}$$

$$\text{maxFilterLengthQs[x][y]} = 3 \tag{1232}$$

– Otherwise, if edgeType is equal to EDGE_HOR, the following applies:

  – The variable sbH is set equal to Max(8, nCbH / numSbY).

  – The array edgeTbFlags is set equal to edgeIdc.

  – For yEdge = 0..Min(Max(1, nCbH / 8) − 1, numSbY − 1), x = 0..nCbW − 1:

    – The vertical position y inside the current coding block is set equal to yEdge *sbH.

– When y is greater than 0 or filterEdgeFlag is equal to 1, the value of edgeIdc[x][y] is modified as follows:

$$\text{edgeIdc}[\ x\ ][\ y\ ] = 2 \tag{1233}$$

– When edgeIdc[x][y] is equal to 1 or 2, the values of maxFilterLengthPs[x][y] and maxFilterLengthQs[x][y] are modified as follows:

– If y is equal to 0, the following applies:

– When numSbY is greater than 1, the following applies:

$$\text{maxFilterLengthQs}[\ x\ ][\ y\ ] = \text{Min}(\ 5, \text{maxFilterLengthQs}[\ x\ ][\ y\ ]\ ) \tag{1234}$$

– When InterAffineFlag[xCb + x][yCb − 1] is equal to 1 or MergeSubblockFlag[xCb + x][yCb − 1] is equal to 1, the following applies:

$$\text{maxFilterLengthPs}[\ x\ ][\ y\ ] = \text{Min}(\ 5, \text{maxFilterLengthPs}[\ x\ ][\ y\ ]\ ) \tag{1235}$$

– Otherwise, if edgeTbFlags[x][y] is equal to 1, the following applies:

$$\text{maxFilterLengthPs}[\ x\ ][\ y\ ] = \text{Min}(\ 5, \text{maxFilterLengthPs}[\ x\ ][\ y\ ]\ ) \tag{1236}$$

$$\text{maxFilterLengthQs}[\ x\ ][\ y\ ] = \text{Min}(\ 5, \text{maxFilterLengthQs}[\ x\ ][\ y\ ]\ ) \tag{1237}$$

– Otherwise, if one or more of the following conditions are true:

– edgeTbFlags[x][y − 4] is equal to 1,

– edgeTbFlags[x][y + 4] is equal to 1,

the following applies:

$$\text{maxFilterLengthPs}[\ x\ ][\ y\ ] = 1 \tag{1238}$$

$$\text{maxFilterLengthQs}[\ x\ ][\ y\ ] = 1 \tag{1239}$$

– Otherwise, if one or more of the following conditions are true:

– yEdge is equal to 1,

– yEdge is equal to (nCbH / 8) − 1,

– edgeTbFlags[x][y − sbH] is equal to 1,

– edgeTbFlags[x][y + sbH] is equal to 1,

the following applies:

$$\text{maxFilterLengthPs}[\ x\ ][\ y\ ] = 2 \tag{1240}$$

$$\text{maxFilterLengthQs}[\ x\ ][\ y\ ] = 2 \tag{1241}$$

– Otherwise, the following applies:

$$\text{maxFilterLengthPs}[\ x\ ][\ y\ ] = 3 \tag{1242}$$

$$\text{maxFilterLengthQs}[\ x\ ][\ y\ ] = 3 \tag{1243}$$

8.8.3.5 Derivation process of boundary filtering strength

Inputs to this process are:

– a picture sample array recPicture,

– a location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,

– a variable nCbW specifying the width of the current coding block,

– a variable nCbH specifying the height of the current coding block,

– a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered,

– a variable cIdx specifying the colour component of the current coding block,

– a two-dimensional (nCbW)x(nCbH) array edgeIdc.

Output of this process is a two-dimensional (nCbW)x(nCbH) array bS specifying the boundary filtering strength.

The variables gridSize, scaleWidth and scaleHeight are derived as follows:

$$\text{gridSize} = \text{cIdx} == 0\ ?\ 4 : 8 \tag{1244}$$

$$\text{scaleWidth} = (\ \text{cIdx} == 0\ )\ ?\ 1 : \text{SubWidthC} \tag{1245}$$

$$\text{scaleHeight} = (\ \text{cIdx} == 0\ )\ ?\ 1 : \text{SubHeightC} \tag{1246}$$

The variables xN and yN are derived as follows:

– If edgeType is equal to EDGE_VER, the following applies:

$$\text{xN} = \text{Max}(\ 0, (\ \text{nCbW} / \text{gridSize}\ ) - 1\ ) \tag{1247}$$

$$\text{yN} = \text{cIdx} == 0\ ?\ (\ \text{nCbH} / 4\ ) - 1 : (\ \text{nCbH} / 2\ ) - 1 \tag{1248}$$

– Otherwise (edgeType is equal to EDGE_HOR), the following applies:

$$\text{xN} = \text{cIdx} == 0\ ?\ (\ \text{nCbW} / 4\ ) - 1 : (\ \text{nCbW} / 2\ ) - 1 \tag{1249}$$

$$\text{yN} = \text{Max}(\ 0, (\ \text{nCbH} / \text{gridSize}\ ) - 1\ ) \tag{1250}$$

The variables xD_i with i = 0..xN and yD_j with j = 0..yN are derived as follows:

– If edgeType is equal to EDGE_VER, the following applies:

$$xD_i = (\ i * \text{gridSize}\ ) \tag{1251}$$

$$yD_j = \text{cIdx} == 0\ ?\ (\ j \ll 2\ ) : (\ j \ll 1\ ) \tag{1252}$$

– Otherwise (edgeType is equal to EDGE_HOR), the following applies:

$$xD_i = \text{cIdx} == 0\ ?\ (\ i \ll 2\ ) : (\ i \ll 1\ ) \tag{1253}$$

$$yD_j = j * \text{gridSize} \tag{1254}$$

For xD_i with i = 0..xN and yD_j with j = 0..yN, the following applies:

– If edgeIdc[xD_i][yD_j] is equal to 0, the variable bS[xD_i][yD_j] is set equal to 0.

– Otherwise, if edgeType is equal to EDGE_VER, VirtualBoundariesPresentFlag is equal to 1, and (xCb + xD_i) is equal to VirtualBoundaryPosX[n] / scaleWidth for any n = 0..NumVerVirtualBoundaries − 1, the variable bS[xD_i][yD_j] is set equal to 0.

– Otherwise, if edgeType is equal to EDGE_HOR, VirtualBoundariesPresentFlag is equal to 1, and (yCb + yD_j) is equal to VirtualBoundaryPosY[n] / scaleHeight for any n = 0..NumHorVirtualBoundaries − 1, the variable bS[xD_i][yD_j] is set equal to 0.

– Otherwise, the following applies:

  – The samples p_0 and q_0 are derived as follows:

    – If edgeType is equal to EDGE_VER, p_0 is recPicture[xCb + xD_i − 1][yCb + yD_j] and q_0 is recPicture[xCb + xD_i][yCb + yD_j].

    – Otherwise (edgeType is equal to EDGE_HOR), p_0 is recPicture[xCb + xD_i][yCb + yD_j − 1] and q_0 is recPicture[xCb + xD_i][yCb + yD_j].

  – The variable bS[xD_i][yD_j] is derived as follows:

– If cIdx is equal to 0 and both samples p_0 and q_0 are in a coding block with intra_bdpcm_luma_flag equal to 1, bS[xD_i][yD_j] is set equal to 0.

– Otherwise, if cIdx is greater than 0 and both samples p_0 and q_0 are in a coding block with intra_bdpcm_chroma_flag equal to 1, bS[xD_i][yD_j] is set equal to 0.

– Otherwise, if CuPredMode[cIdx = = 0 ? 0 : 1][x0][y0] is equal to MODE_INTRA or CuPredMode[cIdx = = 0 ? 0 : 1][x1][y1] is equal to MODE_INTRA, bS[xD_i][yD_j] is set equal to 2, where (x0, y0) is the luma location corresponding to the top-left sample of the coding block containing the sample p_0 and (x1, y1) is the luma location corresponding to the top-left sample of the coding block containing the sample q_0.

– Otherwise, if the sample p_0 or q_0 is in a coding block with ciip_flag equal to 1, bS[xD_i][yD_j] is set equal to 2.

– Otherwise, if the block edge is also a transform block edge and one of the following conditions is true, bS[xD_i][yD_j] is set equal to 1.

  – cIdx is equal to 0, and the sum of tu_y_coded_flag[x0][y0] and tu_y_coded_flag[x1][y1] is greater than 0, where (x0, y0) is the luma location of the top-left sample of the luma transform block containing sample p_0 and (x1, y1) is the luma location of the top-left sample of the luma transform block containing sample q_0.

  – cIdx is equal to 1, and the sum of tu_cb_coded_flag[x0][y0], tu_joint_cbcr_residual_flag[x0][y0], tu_cb_coded_flag[x1][y1], and tu_joint_cbcr_residual_flag[x1][y1] is greater than 0, where (x0, y0) is the luma location corresponding to the top-left sample of the Cb transform block containing chroma sample p_0 and (x1, y1) is the luma location corresponding the top-left sample of the Cb transform block containing chroma sample q_0.

  – cIdx is equal to 2, and the sum of tu_cr_coded_flag[x0][y0], tu_joint_cbcr_residual_flag[x0][y0], tu_cr_coded_flag[x1][y1] and tu_joint_cbcr_residual_flag[x1][y1] is greater than 0, where (x0, y0) is the luma location corresponding to the top-left sample of the Cr transform block containing chroma sample p_0 and (x1, y1) is the luma location corresponding to the top-left sample of the Cr transform block containing chroma sample q_0.

– Otherwise, if cIdx is equal to 0, edgeIdc[xD_i][yD_j] is equal to 2, and one or more of the following conditions are true, bS[xD_i][yD_j] is set equal to 1:

  – The CuPredMode[cIdx = = 0 ? 0 : 1][xp0][yp0] of the subblock containing the sample p_0 is different from the CuPredMode[cIdx = = 0 ? 0 : 1][xq0][yq0] of the subblock containing the sample q_0, where (xp0, yp0) is the luma location corresponding to the top-left sample of the coding block containing the sample p_0 and (xq0, yq0) is the luma location corresponding to the top-left sample of the coding block containing the sample q_0.

  – The subblock containing the sample p_0 and the subblock containing the sample q_0 are both coded in IBC prediction mode, and the absolute difference between the horizontal or vertical component of the block vectors used in the prediction of the two subblocks is greater than or equal to 8 in units of 1/16 luma samples.

  – For the prediction of the subblock containing the sample p_0 different reference pictures or a different number of motion vectors are used than for the prediction of the subblock containing the sample q_0.

    NOTE 1 – The determination of whether the reference pictures used for the two coding sublocks 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 RPL 0 or an index into RPL 1, and also without regard to whether the index position within an RPL is different.

    NOTE 2 – The number of motion vectors that are used for the prediction of a subblock with top-left sample covering (xSb, ySb) is equal to PredFlagL0[xSb][ySb] + PredFlagL1[xSb][ySb].

    NOTE 3 – Reference pictures and motion vectors used to predict a subblock containing a sample located at (xS, yS) refer to values of RefPicList[X][RefIdxLX[xS][yS]] and of MvLX[xS][yS]. These reference pictures and motion vectors may differ from those used for fractional sample interpolation specified in clause 8.5.6.3.

  – One motion vector is used to predict the subblock containing the sample p_0 and one motion vector is used to predict the subblock containing the sample q_0, and the absolute difference between the horizontal or vertical component of the motion vectors used is greater than or equal to 8 in units of 1/16 luma samples.

– Two motion vectors and two different reference pictures are used to predict the subblock containing the sample p_0, two motion vectors for the same two reference pictures are used to predict the subblock containing the sample q_0 and the absolute difference between the horizontal or vertical component of the two motion vectors used in the prediction of the two subblocks for the same reference picture is greater than or equal to 8 in units of 1/16 luma samples.

– Two motion vectors for the same reference picture are used to predict the subblock containing the sample p_0, two motion vectors for the same reference picture are used to predict the subblock containing the sample q_0 and both of the following conditions are true:

  – The absolute difference between the horizontal or vertical component of list 0 motion vectors used in the prediction of the two subblocks is greater than or equal to 8 in 1/16 luma samples, or the absolute difference between the horizontal or vertical component of the list 1 motion vectors used in the prediction of the two subblocks is greater than or equal to 8 in units of 1/16 luma samples.

  – The absolute difference between the horizontal or vertical component of list 0 motion vector used in the prediction of the subblock containing the sample p_0 and the list 1 motion vector used in the prediction of the subblock containing the sample q_0 is greater than or equal to 8 in units of 1/16 luma samples, or the absolute difference between the horizontal or vertical component of the list 1 motion vector used in the prediction of the subblock containing the sample p_0 and list 0 motion vector used in the prediction of the subblock containing the sample q_0 is greater than or equal to 8 in units of 1/16 luma samples.

– Otherwise, the variable bS[xD_i][yD_j] is set equal to 0.

8.8.3.6 Edge filtering process for one direction

8.8.3.6.1 General

Inputs to this process are:

– a variable edgeType specifying whether vertical edges (EDGE_VER) or horizontal edges (EDGE_HOR) are currently processed,

– a variable cIdx specifying the current colour component,

– the reconstructed picture prior to deblocking recPicture,

– a location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,

– a variable nCbW specifying the width of the current coding block,

– a variable nCbH specifying the height of the current coding block,

– the array bS specifying the boundary filtering strength,

– the arrays maxFilterLengthPs and maxFilterLengthQs.

Output of this process is the modified reconstructed picture after deblocking recPicture.

For the edge filtering process, the following applies:

– The variable gridSize is set as follows:

$$\text{gridSize} = \text{cIdx} == 0\ ?\ 4 : 8 \tag{1255}$$

– The variables subW, subH, xN, yN are derived as follows:

$$\text{subW} = \text{cIdx} == 0\ ?\ 1 : \text{SubWidthC} \tag{1256}$$

$$\text{subH} = \text{cIdx} == 0\ ?\ 1 : \text{SubHeightC} \tag{1257}$$

$$\text{xN} = \text{edgeType} == \text{EDGE\_VER}\ ?\ \text{Max}(\ 0, (\ \text{nCbW} / \text{gridSize}\ ) - 1\ ) : (\ \text{nCbW} / (\ 4 / \text{subW}\ )\ ) - 1 \tag{1258}$$

$$\text{yN} = \text{edgeType} == \text{EDGE\_VER}\ ?\ (\ \text{nCbH} / (\ 4 / \text{subH}\ )\ ) - 1 : \text{Max}(\ 0, (\ \text{nCbH} / \text{gridSize}\ ) - 1\ ) \tag{1259}$$

– The variables xD_k with k = 0..xN and yD_m with m = 0..yN are derived as follows:

$$xD_k = edgeType == EDGE\_VER\ ?\ (k * gridSize) : (k << (2 / subW)) \quad (1260)$$

$$yD_m = edgeType == EDGE\_VER\ ?\ (m << (2 / subH)) : (m * gridSize) \quad (1261)$$

– For xD_k with k = 0..xN and yD_m with m = 0..yN, the following applies:
  – When bS[xD_k][yD_m] is greater than 0, the following ordered steps apply:
    – If cIdx is equal to 0, the filtering process for edges in the luma coding block of the current coding unit consists of the following ordered steps:
      1. The decision process for luma block edges as specified in clause 8.8.3.6.2 is invoked with the luma picture sample array recPicture, the location of the luma coding block (xCb, yCb), the luma location of the block (xBl, yBl) set equal to (xD_k, yD_m), the edge direction edgeType, the boundary filtering strength bS[xD_k][yD_m], the maximum filter lengths maxFilterLengthP set equal to maxFilterLengthPs[xD_k][yD_m] and maxFilterLengthQ set equal to maxFilterLengthQs[xD_k][yD_m] as inputs, and the decisions dE, dEp and dEq, the modified maximum filter lengths maxFilterLengthP and maxFilterLengthQ, and the variable t_C as outputs.
      2. The filtering process for block edges as specified in clause 8.8.3.6.3 is invoked with the luma picture sample array recPicture, the location of the luma coding block (xCb, yCb), the luma location of the block (xBl, yBl) set equal to (xD_k, yD_m), the edge direction edgeType, the decisions dE, dEp and dEq, the maximum filter lengths maxFilterLengthP and maxFilterLengthQ, and the variable t_C as inputs, and the modified luma picture sample array recPicture as output.
    – Otherwise (cIdx is not equal to 0), the filtering process for edges in the chroma coding block of current coding unit specified by cIdx consists of the following ordered steps:
      1. The decision process for chroma block edges as specified in clause 8.8.3.6.4 is invoked with the chroma picture sample array recPicture, the location of the chroma coding block (xCb, yCb), the location of the chroma block (xBl, yBl) set equal to (xD_k, yD_m), the edge direction edgeType, the variable cIdx, the boundary filtering strength bS[xD_k][yD_m], the maximum filter lengths maxFilterLengthP set equal to maxFilterLengthPs[xD_k][yD_m] and the maximum filter lengths maxFilterLengthQ set equal to maxFilterLengthQs[xD_k][yD_m] as inputs, and the modified maximum filter lengths maxFilterLengthP and maxFilterLengthQ, and the variable t_C as outputs.
      2. When maxFilterLengthQ is greater than 0, the filtering process for chroma block edges as specified in clause 8.8.3.6.5 is invoked with the chroma picture sample array recPicture, the location of the chroma coding block (xCb, yCb), the chroma location of the block (xBl, yBl) set equal to (xD_k, yD_m), the edge direction edgeType, the variable t_C, the maximum filter lengths maxFilterLengthP and maxFilterLengthQ as inputs, and the modified chroma picture sample array recPicture as output.

8.8.3.6.2 Decision process for luma block edges

Inputs to this process are:

– a picture sample array recPicture,
– a location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,
– a location (xBl, yBl) specifying the top-left sample of the current block relative to the top-left sample of the current coding block,
– a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered,
– a variable bS specifying the boundary filtering strength,
– a variable maxFilterLengthP specifying the maximum filter length,
– a variable maxFilterLengthQ specifying the maximum filter length.

Outputs of this process are:

– the variables dE, dEp and dEq containing decisions,
– the modified filter length variables maxFilterLengthP and maxFilterLengthQ,
– the variable t_C.

The sample values $p_{i,k}$ and $q_{j,k}$ with i = 0..Max(2, maxFilterLengthP), j = 0..Max(2, maxFilterLengthQ) and k = 0 and 3 are derived as follows:

– If edgeType is equal to EDGE_VER, the following applies:

$$q_{j,k} = \text{recPicture}[\ xCb + xBl + j\ ][\ yCb + yBl + k\ ] \quad (1262)$$

$$p_{i,k} = \text{recPicture}[\ xCb + xBl - i - 1\ ][\ yCb + yBl + k\ ] \quad (1263)$$

– Otherwise (edgeType is equal to EDGE_HOR), the following applies:

$$q_{j,k} = \text{recPicture}[\ xCb + xBl + k\ ][\ yCb + yBl + j\ ] \quad (1264)$$

$$p_{i,k} = \text{recPicture}[\ xCb + xBl + k\ ][\ yCb + yBl - i - 1\ ] \quad (1265)$$

The variable qpOffset is derived as follows:

– If sps_ladf_enabled_flag is equal to 1, the following applies:

  – The variable lumaLevel of the reconstructed luma level is derived as follows:

$$\text{lumaLevel} = (\ (\ p_{0,0} + p_{0,3} + q_{0,0} + q_{0,3}\ ) >> 2\ ) \quad (1266)$$

  – The variable qpOffset is set equal to sps_ladf_lowest_interval_qp_offset and modified as follows:

```
for( i = 0; i < sps_num_ladf_intervals_minus2 + 1; i++ ) {
    if( lumaLevel > SpsLadfIntervalLowerBound[ i + 1 ] )
        qpOffset = sps_ladf_qp_offset[ i ]                    (1267)
    else
        break
}
```

– Otherwise, qpOffset is set equal to 0.

The variables Qp_Q and Qp_P are set equal to the Qp_Y values of the coding units which include the coding blocks containing the sample $q_{0,0}$ and $p_{0,0}$, respectively.

The variable qP is derived as follows:

$$qP = (\ (\ Qp_Q + Qp_P + 1\ ) >> 1\ ) + \text{qpOffset} \quad (1268)$$

The value of the variable β′ is determined as specified in Table 43 based on the quantization parameter Q derived as follows:

$$Q = \text{Clip3}(\ 0, 63, qP + (\ \text{sh\_luma\_beta\_offset\_div2} << 1\ )\ ) \quad (1269)$$

where sh_luma_beta_offset_div2 is the value of the syntax element sh_luma_beta_offset_div2 for the slice that contains sample $q_{0,0}$.

The variable β is derived as follows:

$$\beta = \beta' * (\ 1 << (\ \text{BitDepth} - 8\ )\ ) \quad (1270)$$

The value of the variable t_C' is determined as specified in Table 43 based on the quantization parameter Q derived as follows:

$$Q = \text{Clip3}(\ 0, 65, qP + 2 * (\ bS - 1\ ) + (\ \text{sh\_luma\_tc\_offset\_div2} << 1\ )\ ) \quad (1271)$$

where sh_luma_tc_offset_div2 is the value of the syntax element sh_luma_tc_offset_div2 for the slice that contains sample $q_{0,0}$.

The variable t_C is derived as follows:

– If BitDepth is less than 10, the following applies:

$$t_C = (\ t_C' + (\ 1 << (\ 9 - \text{BitDepth}\ )\ )\ ) >> (\ 10 - \text{BitDepth}\ ) \quad (1272)$$

– Otherwise (BitDepth is greater than or equal to 10), the following applies:

$$t_C = t_C' * (1 \ll (\text{BitDepth} - 10)) \quad (1273)$$

The following ordered steps apply:

1. The variables dp0, dp3, dq0 and dq3 are derived as follows:

$$\text{dp0} = \text{Abs}(p_{2,0} - 2 * p_{1,0} + p_{0,0}) \quad (1274)$$

$$\text{dp3} = \text{Abs}(p_{2,3} - 2 * p_{1,3} + p_{0,3}) \quad (1275)$$

$$\text{dq0} = \text{Abs}(q_{2,0} - 2 * q_{1,0} + q_{0,0}) \quad (1276)$$

$$\text{dq3} = \text{Abs}(q_{2,3} - 2 * q_{1,3} + q_{0,3}) \quad (1277)$$

2. When maxFilterLengthP and maxFilterLengthQ both are equal to or greater than 3 the variables sp0, sq0, spq0, sp3, sq3 and spq3 are derived as follows:

$$\text{sp0} = \text{Abs}(p_{3,0} - p_{0,0}) \quad (1278)$$

$$\text{sq0} = \text{Abs}(q_{0,0} - q_{3,0}) \quad (1279)$$

$$\text{spq0} = \text{Abs}(p_{0,0} - q_{0,0}) \quad (1280)$$

$$\text{sp3} = \text{Abs}(p_{3,3} - p_{0,3}) \quad (1281)$$

$$\text{sq3} = \text{Abs}(q_{0,3} - q_{3,3}) \quad (1282)$$

$$\text{spq3} = \text{Abs}(p_{0,3} - q_{0,3}) \quad (1283)$$

3. The variables sidePisLargeBlk and sideQisLargeBlk are set equal to 0.
4. When maxFilterLengthP is greater than 3, sidePisLargeBlk is set equal to 1.
5. When maxFilterLengthQ is greater than 3, sideQisLargeBlk is set equal to 1.
6. When edgeType is equal to EDGE_HOR and (yCb + yBl) % CtbSizeY is equal to 0, sidePisLargeBlk is set equal to 0.
7. The variables dSam0 and dSam3 are initialized to 0.
8. When sidePisLargeBlk or sideQisLargeBlk is greater than 0, the following applies:

   a. The variables dp0L, dp3L are derived and maxFilterLengthP is modified as follows:

      – If sidePisLargeBlk is equal to 1, the following applies:

$$\text{dp0L} = (\text{dp0} + \text{Abs}(p_{5,0} - 2 * p_{4,0} + p_{3,0}) + 1) \gg 1 \quad (1284)$$

$$\text{dp3L} = (\text{dp3} + \text{Abs}(p_{5,3} - 2 * p_{4,3} + p_{3,3}) + 1) \gg 1 \quad (1285)$$

      – Otherwise, the following applies:

$$\text{dp0L} = \text{dp0} \quad (1286)$$

$$\text{dp3L} = \text{dp3} \quad (1287)$$

$$\text{maxFilterLengthP} = 3 \quad (1288)$$

   b. The variables dq0L and dq3L are derived as follows:

      – If sideQisLargeBlk is equal to 1, the following applies:

$$\text{dq0L} = (\text{dq0} + \text{Abs}(q_{5,0} - 2 * q_{4,0} + q_{3,0}) + 1) \gg 1 \quad (1289)$$

$$\text{dq3L} = (\text{dq3} + \text{Abs}(q_{5,3} - 2 * q_{4,3} + q_{3,3}) + 1) \gg 1 \quad (1290)$$

– Otherwise, the following applies:

$$dq0L = dq0 \tag{1291}$$

$$dq3L = dq3 \tag{1292}$$

c. The variables sp0L and sp3L are derived as follows:

– If maxFilterLengthP is equal to 7, the following applies:

$$sp0L = sp0 + Abs(p_{7,0} - p_{6,0} - p_{5,0} + p_{4,0}) \tag{1293}$$

$$sp3L = sp3 + Abs(p_{7,3} - p_{6,3} - p_{5,3} + p_{4,3}) \tag{1294}$$

– Otherwise, the following applies:

$$sp0L = sp0 \tag{1295}$$

$$sp3L = sp3 \tag{1296}$$

d. The variables sq0L and sq3L are derived as follows:

– If maxFilterLengthQ is equal to 7, the following applies:

$$sq0L = sq0 + Abs(q_{4,0} - q_{5,0} - q_{6,0} + q_{7,0}) \tag{1297}$$

$$sq3L = sq3 + Abs(q_{4,3} - q_{5,3} - q_{6,3} + q_{7,3}) \tag{1298}$$

– Otherwise, the following applies:

$$sq0L = sq0 \tag{1299}$$

$$sq3L = sq3 \tag{1300}$$

e. The variables dpq0L, dpq3L, and dL are derived as follows:

$$dpq0L = dp0L + dq0L \tag{1301}$$

$$dpq3L = dp3L + dq3L \tag{1302}$$

$$dL = dpq0L + dpq3L \tag{1303}$$

f. When dL is less than β, the following ordered steps apply:

i. The variable dpq is set equal to 2 * dpq0L.

ii. The variable sp is set equal to sp0L, the variable sq is set equal to sq0L and the variable spq is set equal to spq0.

iii. The variables p_0, p_3, q_0, and q_3 are first initialized to 0 and then modified according to sidePisLargeBlk and sideQisLargeBlk as follows:

– When sidePisLargeBlk is equal to 1, the following applies:

$$p_3 = p_{3,0} \tag{1304}$$

$$p_0 = p_{\text{maxFilterLengthP},0} \tag{1305}$$

– When sideQisLargeBlk is equal to 1, the following applies:

$$q_3 = q_{3,0} \tag{1306}$$

$$q_0 = q_{\text{maxFilterLengthQ},0} \tag{1307}$$

iv. For the sample location (xCb + xBl, yCb + yBl), the decision process for a luma sample as specified in clause 8.8.3.6.6 is invoked with the sample values p_0, p_3, q_0, q_3, the variables dpq, sp, sq, spq, sidePisLargeBlk, sideQisLargeBlk, β and t_C as inputs, and the output is assigned to the decision dSam0.

v. The variable dpq is set equal to 2 * dpq3L.

vi. The variable sp is set equal to sp3L, the variable sq is set equal to sq3L and the variable spq is set equal to spq3.

vii. The variables p_0 p_3 q_0 and q_3 are first initialized to 0 and are then modified according to sidePisLargeBlk and sideQisLargeBlk as follows:

– When sidePisLargeBlk is equal to 1, the following applies:

$$p_3 = p_{3,3} \tag{1308}$$

$$p_0 = p_{\text{maxFilterLengthP},3} \tag{1309}$$

– When sideQisLargeBlk is equal to 1, the following applies:

$$q_3 = q_{3,3} \tag{1310}$$

$$q_0 = q_{\text{maxFilterLengthQ},3} \tag{1311}$$

viii. When edgeType is equal to EDGE_VER for the sample location (xCb + xBl, yCb + yBl + 3) or when edgeType is equal to EDGE_HOR for the sample location (xCb + xBl + 3, yCb + yBl), the decision process for a luma sample as specified in clause 8.8.3.6.6 is invoked with the sample values p_0, p_3, q_0, q_3, the variables dpq, sp, sq, spq, sidePisLargeBlk, sideQisLargeBlk, β and t_C as inputs, and the output is assigned to the decision dSam3.

9. The variables dE, dEp and dEq are derived as follows:

– If dSam0 and dSam3 are both equal to 1, the variable dE is set equal to 3, dEp is set equal to 1, and dEq is set equal to 1.

– Otherwise, the following ordered steps apply:

a. The variables dpq0, dpq3, dp, dq and d are derived as follows:

$$\text{dpq0} = \text{dp0} + \text{dq0} \tag{1312}$$

$$\text{dpq3} = \text{dp3} + \text{dq3} \tag{1313}$$

$$\text{dp} = \text{dp0} + \text{dp3} \tag{1314}$$

$$\text{dq} = \text{dq0} + \text{dq3} \tag{1315}$$

$$\text{d} = \text{dpq0} + \text{dpq3} \tag{1316}$$

b. The variables dE, dEp, dEq, sidePisLargeBlk and sideQisLargeBlk are set equal to 0.

c. When d is less than β and both maxFilterLengthP and maxFilterLengthQ are greater than 2, the following ordered steps apply:

i. The variable dpq is set equal to 2 * dpq0.

ii. The variable sp is set equal to sp0, the variable sq is set equal to sq0 and the variable spq is set equal to spq0.

iii. For the sample location (xCb + xBl, yCb + yBl), the decision process for a luma sample as specified in clause 8.8.3.6.6 is invoked with the variables p_0, p_3, q_0, q_3 all set equal to 0, the variables dpq, sp, sq, spq, sidePisLargeBlk, sideQisLargeBlk, β and t_C as inputs, and the output is assigned to the decision dSam0.

iv. The variable dpq is set equal to 2 * dpq3.

v. The variable sp is set equal to sp3, the variable sq is set equal to sq3 and the variable spq is set equal to spq3.

vi. When edgeType is equal to EDGE_VER for the sample location (xCb + xBl, yCb + yBl + 3) or when edgeType is equal to EDGE_HOR for the sample location (xCb + xBl + 3, yCb + yBl), the decision process for a sample as specified in clause 8.8.3.6.6 is invoked with the variables p_0, p_3, q_0, q_3 all set equal to 0, the variables dpq, sp, sq, spq, sidePisLargeBlk, sideQisLargeBlk, β and t_C as inputs, and the output is assigned to the decision dSam3.

d. When d is less than β, the following ordered steps apply:

i. The variable dE is set equal to 1.

ii. When dSam0 is equal to 1 and dSam3 is equal to 1, the variable dE is set equal to 2 and both maxFilterLengthP and maxFilterLengthQ are set equal to 3.

iii. When maxFilterLengthP is greater than 1, and maxFilterLengthQ is greater than 1, and dp is less than (β + (β >> 1)) >> 3, the variable dEp is set equal to 1.

iv. When maxFilterLengthP is greater than 1, and maxFilterLengthQ is greater than 1, and dq is less than (β + (β >> 1)) >> 3, the variable dEq is set equal to 1.

v. When dE is equal to 1, maxFilterLengthP is set equal to 1 + dEp and maxFilterLengthQ is set equal to 1 + dEq.

Table 43 – Derivation of threshold variables β′ and t_C′ from input Q

Q	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
β′	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6
t_C′	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Q	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
β′	7	8	9	10	11	12	13	14	15	16	17	18	20	22	24	26	28
t_C′	0	3	4	4	4	4	5	5	5	5	7	7	8	9	10	10	11
Q	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
β′	30	32	34	36	38	40	42	44	46	48	50	52	54	56	58	60	62
t_C′	13	14	15	17	19	21	24	25	29	33	36	41	45	51	57	64	71
Q	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65		
β′	64	66	68	70	72	74	76	78	80	82	84	86	88	-	-		
t_C′	80	89	100	112	125	141	157	177	198	222	250	280	314	352	395		

8.8.3.6.3 Filtering process for luma block edges

Inputs to this process are:

– a picture sample array recPicture,

– a location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,

– a location (xBl, yBl) specifying the top-left sample of the current block relative to the top-left sample of the current coding block,

– a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered,

– the variables dE, dEp and dEq containing decisions,

– the variables maxFilterLengthP and maxFilterLengthQ containing maximum filter lengths,

– the variable t_C.

Output of this process is the modified picture sample array recPicture.

Depending on the value of edgeType, the following applies:

– If edgeType is equal to EDGE_VER, the following ordered steps apply:

1. The sample values $p_{i,k}$ and $q_{j,k}$ with i = 0..maxFilterLengthP, j = 0..maxFilterLengthQ and k = 0..3 are derived as follows:

$$q_{j,k} = \text{recPicture}[\ xCb + xBl + j\ ][\ yCb + yBl + k\ ] \tag{1317}$$

$$p_{i,k} = \text{recPicture}[\ xCb + xBl - i - 1\ ][\ yCb + yBl + k\ ] \tag{1318}$$

2. When dE is not equal to 0 and dE is not equal to 3, for each sample location (xCb + xBl, yCb + yBl + k), k = 0..3, the following ordered steps apply:

   a. The filtering process for a luma sample using short filters as specified in clause 8.8.3.6.7 is invoked with the variables maxFilterLengthP, maxFilterLengthQ, the sample values $p_{i,k}$, $q_{j,k}$ with i = 0..maxFilterLengthP and j = 0..maxFilterLengthQ, the decision dE, the variables dEp and dEq and the variable t_C as inputs, and the number of filtered samples nDp and nDq from each side of the block boundary and the filtered sample values p_i' and q_j' as outputs.

   b. When nDp is greater than 0, the filtered sample values p_i' with i = 0..nDp − 1 replace the corresponding samples inside the sample array recPicture as follows:

$$\text{recPicture}[\ xCb + xBl - i - 1\ ][\ yCb + yBl + k\ ] = p_i' \tag{1319}$$

   c. When nDq is greater than 0, the filtered sample values q_j' with j = 0..nDq − 1 replace the corresponding samples inside the sample array recPicture as follows:

$$\text{recPicture}[\ xCb + xBl + j\ ][\ yCb + yBl + k\ ] = q_j' \tag{1320}$$

3. When dE is equal to 3, for each sample location (xCb + xBl, yCb + yBl + k), k = 0..3, the following ordered steps apply:

   a. The filtering process for a luma sample using long filters as specified in clause 8.8.3.6.8 is invoked with the variables maxFilterLengthP, maxFilterLengthQ, the sample values $p_{i,k}$, $q_{j,k}$ with i = 0..maxFilterLengthP and j = 0..maxFilterLengthQ, and t_C as inputs and the filtered samples values p_i' and q_j' as outputs.

   b. The filtered sample values p_i' with i = 0..maxFilterLengthP − 1 replace the corresponding samples inside the sample array recPicture as follows:

$$\text{recPicture}[\ xCb + xBl - i - 1\ ][\ yCb + yBl + k\ ] = p_i' \tag{1321}$$

   c. The filtered sample values q_j' with j = 0..maxFilterLengthQ − 1 replace the corresponding samples inside the sample array recPicture as follows:

$$\text{recPicture}[\ xCb + xBl + j\ ][\ yCb + yBl + k\ ] = q_j' \tag{1322}$$

– Otherwise (edgeType is equal to EDGE_HOR), the following ordered steps apply:

1. The sample values $p_{i,k}$ and $q_{j,k}$ with i = 0..maxFilterLengthP, j = 0..maxFilterLengthQ and k = 0..3 are derived as follows:

$$q_{j,k} = \text{recPicture}[\ xCb + xBl + k\ ][\ yCb + yBl + j\ ] \tag{1323}$$

$$p_{i,k} = \text{recPicture}[\ xCb + xBl + k\ ][\ yCb + yBl - i - 1\ ] \tag{1324}$$

2. When dE is not equal to 0 and dE is not equal to 3, for each sample location (xCb + xBl + k, yCb + yBl), k = 0..3, the following ordered steps apply:

   a. The filtering process for a luma sample using short filters as specified in clause 8.8.3.6.7 is invoked with the variables maxFilterLengthP, maxFilterLengthQ, the sample values $p_{i,k}$, $q_{i,k}$ with i = 0..maxFilterLengthP and j = 0..maxFilterLengthQ, the decision dE, the variables dEp and dEq, and the variable t_C as inputs, and the number of filtered samples nDp and nDq from each side of the block boundary and the filtered sample values p_i' and q_j' as outputs.

   b. When nDp is greater than 0, the filtered sample values p_i' with i = 0..nDp − 1 replace the corresponding samples inside the sample array recPicture as follows:

$$\text{recPicture}[\ xCb + xBl + k\ ][\ yCb + yBl - i - 1\ ] = p_i' \tag{1325}$$

c. When nDq is greater than 0, the filtered sample values q_j' with j = 0..nDq − 1 replace the corresponding samples inside the sample array recPicture as follows:

$$\text{recPicture}[\ xCb + xBl + k\ ][\ yCb + yBl + j\ ] = q_j' \tag{1326}$$

3. When dE is equal to 3, for each sample location (xCb + xBl + k, yCb + yBl), k = 0..3, the following ordered steps apply:

   a. The filtering process for a luma sample using long filters as specified in clause 8.8.3.6.8 is invoked with the variables maxFilterLengthP, maxFilterLengthQ, the sample values $p_{i,k}$, $q_{j,k}$ with i = 0..maxFilterLengthP and j = 0..maxFilterLengthQ, and the variable t_C as inputs, and the filtered sample values p_i' and q_j' as outputs.

   b. The filtered sample values p_i' with i = 0..maxFilterLengthP − 1 replace the corresponding samples inside the sample array recPicture as follows:

$$\text{recPicture}[\ xCb + xBl + k\ ][\ yCb + yBl - i - 1\ ] = p_i' \tag{1327}$$

   c. The filtered sample values q_j' with j = 0..maxFilterLengthQ − 1 replace the corresponding samples inside the sample array recPicture as follows:

$$\text{recPicture}[\ xCb + xBl + k\ ][\ yCb + yBl + j\ ] = q_j' \tag{1328}$$

8.8.3.6.4 Decision process for chroma block edges

This process is only invoked when sps_chroma_format_idc is not equal to 0.

Inputs to this process are:

– a chroma picture sample array recPicture,

– a chroma location (xCb, yCb) specifying the top-left sample of the current chroma coding block relative to the top-left chroma sample of the current picture,

– a chroma location (xBl, yBl) specifying the top-left sample of the current chroma block relative to the top-left sample of the current chroma coding block,

– a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered,

– a variable cIdx specifying the colour component index,

– a variable bS specifying the boundary filtering strength,

– a variable maxFilterLengthP specifying the maximum filter length,

– a variable maxFilterLengthQ specifying the maximum filter length.

Outputs of this process are:

– the modified filter length variables maxFilterLengthP and maxFilterLengthQ,

– the variable t_C.

The variable maxK is derived as follows:

– If edgeType is equal to EDGE_VER, the following applies:

$$\text{maxK} = (\ \text{SubHeightC} == 1\ )\ ?\ 3 : 1 \tag{1329}$$

– Otherwise (edgeType is equal to EDGE_HOR), the following applies:

$$\text{maxK} = (\ \text{SubWidthC} == 1\ )\ ?\ 3 : 1 \tag{1330}$$

The values $p_{i,k}$ and $q_{j,k}$ with i = 0..maxFilterLengthP, j = 0..maxFilterLengthQ and k = 0..maxK are derived as follows:

– If edgeType is equal to EDGE_VER, the following applies:

$$q_{j,k} = \text{recPicture}[\ xCb + xBl + j\ ][\ yCb + yBl + k\ ] \tag{1331}$$

$$p_{i,k} = \text{recPicture}[\ xCb + xBl - i - 1\ ][\ yCb + yBl + k\ ] \tag{1332}$$

$$\text{subSampleC} = \text{SubHeightC} \tag{1333}$$

– Otherwise (edgeType is equal to EDGE_HOR), the following applies:

$$q_{j,k} = \text{recPicture}[\ xCb + xBl + k\ ][\ yCb + yBl + j\ ] \quad (1334)$$

$$p_{i,k} = \text{recPicture}[\ xCb + xBl + k\ ][\ yCb + yBl - i - 1\ ] \quad (1335)$$

$$\text{subSampleC} = \text{SubWidthC} \quad (1336)$$

The variable Qp_P is derived as follows:

– The luma location (xTb_P, yTb_P) is set as the top-left luma sample position of the transform block containing the sample $p_{0,0}$, relative to the top-left luma sample of the picture.

– If TuCResMode[xTb_P][yTb_P] is equal to 2, Qp_P is set equal to Qp'_{CbCr} of the transform block containing the sample $p_{0,0}$.

– Otherwise, if cIdx is equal to 1, Qp_P is set equal to Qp'_{Cb} of the transform block containing the sample $p_{0,0}$.

– Otherwise, Qp_P is set equal to Qp'_{Cr} of the transform block containing the sample $p_{0,0}$.

The variable Qp_Q is derived as follows:

– The luma location (xTb_Q, yTb_Q) is set as the top-left luma sample position of the transform block containing the sample $q_{0,0}$, relative to the top-left luma sample of the picture.

– If TuCResMode[xTb_Q][yTb_Q] is equal to 2, Qp_Q is set equal to Qp'_{CbCr} of the transform block containing the sample $q_{0,0}$.

– Otherwise, if cIdx is equal to 1, Qp_Q is set equal to Qp'_{Cb} of the transform block containing the sample $q_{0,0}$.

– Otherwise, Qp_Q is set equal to Qp'_{Cr} of the transform block containing the sample $q_{0,0}$.

The variable Qp_C is derived as follows:

$$Qp_C = (\ Qp_Q - \text{QpBdOffset} + Qp_P - \text{QpBdOffset} + 1\ ) \gg 1 \quad (1337)$$

The value of the variable β′ is determined as specified in Table 43 based on the quantization parameter Q derived as follows:

$$\begin{aligned}&\text{sliceBetaOffsetDiv2} = (\ \text{cIdx} == 1\ ?\ \text{sh\_cb\_beta\_offset\_div2} : \text{sh\_cr\_beta\_offset\_div2}\ )\\&Q = \text{Clip3}(\ 0, 63, Qp_C + (\ \text{sliceBetaOffsetDiv2} \ll 1\ )\ )\end{aligned} \quad (1338)$$

where sh_cb_beta_offset_div2 and sh_cr_beta_offset_div2 are the values of the syntax elements sh_cb_beta_offset_div2 and sh_cr_beta_offset_div2, respectively, for the slice that contains sample $q_{0,0}$.

The variable β is derived as follows:

$$\beta = \beta' * (\ 1 \ll (\ \text{BitDepth} - 8\ )\ ) \quad (1339)$$

The value of the variable t_C' is determined as specified in Table 43 based on the chroma quantization parameter Q derived as follows:

$$\begin{aligned}&\text{sliceTcOffsetDiv2} = (\ \text{cIdx} == 1\ ?\ \text{sh\_cb\_tc\_offset\_div2} : \text{sh\_cr\_tc\_offset\_div2}\ )\\&Q = \text{Clip3}(\ 0, 65, Qp_C + 2 * (\ bS - 1\ ) + (\ \text{sliceTcOffsetDiv2} \ll 1\ )\ )\end{aligned} \quad (1340)$$

where sh_cb_tc_offset_div2 and sh_cr_tc_offset_div2 are the values of the syntax elements sh_cb_tc_offset_div2 and sh_cr_tc_offset_div2, respectively, for the slice that contains sample $q_{0,0}$.

The variable t_C is derived as follows:

– If BitDepth is less than 10, the following applies:

$$t_C = (\ t_C' + (\ 1 \ll (\ 9 - \text{BitDepth}\ )\ )\ ) \gg (\ 10 - \text{BitDepth}\ ) \quad (1341)$$

– Otherwise (BitDepth is greater than or equal to 10), the following applies:

$$t_C = t_C' * (\ 1 \ll (\ \text{BitDepth} - 10\ )\ ) \quad (1342)$$

When both maxFilterLengthP and maxFilterLengthQ are equal to 1 and bS is not equal to 2, maxFilterLengthP and maxFilterLengthQ are both set equal to 0.

When maxFilterLengthQ is equal to 3, the following ordered steps apply:

1. The variable n1 is derived as follows:

$$n1 = subSampleC == 2 ? 1 : 3 \quad (1343)$$

2. When maxFilterLengthP is equal to 1, the samples $p_{3,0}$ and $p_{2,0}$ are both set equal to $p_{1,0}$ and the samples $p_{3,n1}$, $p_{2,n1}$ are both set equal to $p_{1,n1}$.

3. The variables dpq0, dpq1, dp, dq and d are derived as follows:

$$dp0 = Abs(p_{2,0} - 2 * p_{1,0} + p_{0,0}) \quad (1344)$$

$$dp1 = Abs(p_{2,n1} - 2 * p_{1,n1} + p_{0,n1}) \quad (1345)$$

$$dq0 = Abs(q_{2,0} - 2 * q_{1,0} + q_{0,0}) \quad (1346)$$

$$dq1 = Abs(q_{2,n1} - 2 * q_{1,n1} + q_{0,n1}) \quad (1347)$$

$$dpq0 = dp0 + dq0 \quad (1348)$$

$$dpq1 = dp1 + dq1 \quad (1349)$$

$$dp = dp0 + dp1 \quad (1350)$$

$$dq = dq0 + dq1 \quad (1351)$$

$$d = dpq0 + dpq1 \quad (1352)$$

4. The variables dSam0 and dSam1 are both set equal to 0.

5. When d is less than β, the following ordered steps apply:

   a. The variable dpq is set equal to 2 * dpq0.

   b. The variable dSam0 is derived by invoking the decision process for a chroma sample as specified in clause 8.8.3.6.9 for the sample location (xCb + xBl, yCb + yBl) with sample values $p_{0,0}$, $p_{3,0}$, $q_{0,0}$, and $q_{3,0}$, the variables dpq, β and t_C as inputs, and the output is assigned to the decision dSam0.

   c. The variable dpq is set equal to 2 * dpq1.

   d. The variable dSam1 is modified as follows:

      – If edgeType is equal to EDGE_VER, for the sample location (xCb + xBl, yCb + yBl + n1), the decision process for a chroma sample as specified in clause 8.8.3.6.9 is invoked with sample values $p_{0,n1}$, $p_{3,n1}$, $q_{0,n1}$, and $q_{3,n1}$, the variables dpq, β and t_C as inputs, and the output is assigned to the decision dSam1.

      – Otherwise (edgeType is equal to EDGE_HOR), for the sample location (xCb + xBl + n1, yCb + yBl), the decision process for a chroma sample as specified in clause 8.8.3.6.9 is invoked with sample values $p_{0,n1}$, $p_{3,n1}$, $q_{0,n1}$ and $q_{3,n1}$, the variables dpq, β and t_C as inputs, and the output is assigned to the decision dSam1.

6. When dSam0 is equal to 0 or dSam1 is equal to 0, maxFilterLengthP and maxFilterLengthQ are both set equal to 1.

8.8.3.6.5 Filtering process for chroma block edges

This process is only invoked when sps_chroma_format_idc is not equal to 0.

Inputs to this process are:

– a chroma picture sample array recPicture,

– a chroma location (xCb, yCb) specifying the top-left sample of the current chroma coding block relative to the top-left chroma sample of the current picture,

– a chroma location (xBl, yBl) specifying the top-left sample of the current chroma block relative to the top-left sample of the current chroma coding block,

– a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered,

– a variable maxFilterLengthP specifying the maximum filter length,

– a variable maxFilterLengthQ specifying the maximum filter length,

– the variable tC.

Output of this process is the modified chroma picture sample array recPicture.

The variable maxK is derived as follows:

– If edgeType is equal to EDGE_VER, the following applies:

$$\text{maxK} = (\text{SubHeightC} == 1)\ ?\ 3 : 1 \tag{1353}$$

– Otherwise (edgeType is equal to EDGE_HOR), the following applies:

$$\text{maxK} = (\text{SubWidthC} == 1)\ ?\ 3 : 1 \tag{1354}$$

The values $p_{i,k}$ with i = 0..maxFilterLengthP, $q_{j,k}$ with j = 0..maxFilterLengthQ, and k = 0..maxK are derived as follows:

– If edgeType is equal to EDGE_VER, the following applies:

$$q_{j,k} = \text{recPicture}[\ \text{xCb} + \text{xBl} + j\ ][\ \text{yCb} + \text{yBl} + k\ ] \tag{1355}$$

$$p_{i,k} = \text{recPicture}[\ \text{xCb} + \text{xBl} - i - 1\ ][\ \text{yCb} + \text{yBl} + k\ ] \tag{1356}$$

– Otherwise (edgeType is equal to EDGE_HOR), the following applies:

$$q_{j,k} = \text{recPicture}[\ \text{xCb} + \text{xBl} + k\ ][\ \text{yCb} + \text{yBl} + j\ ] \tag{1357}$$

$$p_{i,k} = \text{recPicture}[\ \text{xCb} + \text{xBl} + k\ ][\ \text{yCb} + \text{yBl} - i - 1\ ] \tag{1358}$$

Depending on the value of edgeType, the following applies:

– If edgeType is equal to EDGE_VER, for each sample location (xCb + xBl, yCb + yBl + k), k = 0..maxK, the following ordered steps apply:

1. The filtering process for a chroma sample as specified in clause 8.8.3.6.10 is invoked with the variables maxFilterLengthP and maxFilterLengthQ, the sample values $p_{i,k}$, $q_{j,k}$ with i = 0..maxFilterLengthP and j = 0..maxFilterLengthQ, and the variable t_C as inputs, and the filtered sample values p_i' and q_j' with i = 0..maxFilterLengthP − 1 and j = 0..maxFilterLengthQ − 1 as outputs.

2. The filtered sample values p_i' and q_j' with i = 0..maxFilterLengthP − 1 and j = 0..maxFilterLengthQ − 1 replace the corresponding samples inside the sample array recPicture as follows:

$$\text{recPicture}[\ \text{xCb} + \text{xBl} + j\ ][\ \text{yCb} + \text{yBl} + k\ ] = q_j' \tag{1359}$$

$$\text{recPicture}[\ \text{xCb} + \text{xBl} - i - 1\ ][\ \text{yCb} + \text{yBl} + k\ ] = p_i' \tag{1360}$$

– Otherwise (edgeType is equal to EDGE_HOR), for each sample location (xCb + xBl + k, yCb + yBl), k = 0..maxK, the following ordered steps apply:

1. The filtering process for a chroma sample as specified in clause 8.8.3.6.10 is invoked with the variables maxFilterLengthP and maxFilterLengthQ, the sample values $p_{i,k}$, $q_{j,k}$, with i = 0..maxFilterLengthP and j = 0..maxFilterLengthQ, and the variable t_C as inputs, and the filtered sample values p_i' and q_j' with i = 0..maxFilterLengthP − 1 and j = 0..maxFilterLengthQ − 1 as outputs.

2. The filtered sample values p_i' and q_j' with i = 0..maxFilterLengthP − 1 and j = 0..maxFilterLengthQ − 1 replace the corresponding samples inside the sample array recPicture as follows:

$$\text{recPicture}[\ \text{xCb} + \text{xBl} + k\ ][\ \text{yCb} + \text{yBl} + j\ ] = q_j' \tag{1361}$$

$$\text{recPicture}[\ \text{xCb} + \text{xBl} + k\ ][\ \text{yCb} + \text{yBl} - i - 1\ ] = p_i' \tag{1362}$$

8.8.3.6.6 Decision process for a luma sample

Inputs to this process are:

– the sample values $p_{0,}$ p_3, q_0 and q_3,

– the variables dpq, sp, sq, spq, sidePisLargeBlk, sideQisLargeBlk, β and t_C.

Output of this process is the variable dSam containing a decision.

The variables sp and sq are modified as follows:

– When sidePisLargeBlk is equal to 1, the following applies:

$$sp = (sp + Abs(p_3 - p_0) + 1) >> 1 \quad (1363)$$

– When sideQisLargeBlk is equal to 1, the following applies:

$$sq = (sq + Abs(q_3 - q_0) + 1) >> 1 \quad (1364)$$

The variables sThr1 and sThr2 are is derived as follows:

– If sidePisLargeBlk is equal to 1 or sideQisLargeBlk is equal to 1, the following applies:

$$sThr1 = 3 * \beta >> 5 \quad (1365)$$

$$sThr2 = \beta >> 4 \quad (1366)$$

– Otherwise, the following applies:

$$sThr1 = \beta >> 3 \quad (1367)$$

$$sThr2 = \beta >> 2 \quad (1368)$$

The variable dSam is specified as follows:

– If all of the following conditions are true, dSam is set equal to 1:
  – dpq is less than sThr2,
  – sp + sq is less than sThr1,
  – spq is less than (5 * t_C + 1) >> 1.

– Otherwise, dSam is set equal to 0.

8.8.3.6.7 Filtering process for a luma sample using short filters

Inputs to this process are:

– the variables maxFilterLengthP and maxFilterLengthQ,

– the sample values p_i and q_j with i = 0..maxFilterLengthP and j = 0..maxFilterLengthQ,

– a variable dE,

– the variables dEp and dEq containing decisions to filter samples p1 and q1, respectively,

– a variable t_C.

Outputs of this process are:

– the number of filtered samples nDp and nDq,

– the filtered sample values p_i' and q_j' with i = 0..nDp − 1, j = 0..nDq − 1.

Depending on the value of dE, the following applies:

– If the variable dE is equal to 2, nDp and nDq are both set equal to 3 and the following strong filtering applies:

$$p_0' = Clip3(p_0 - 3 * t_C, p_0 + 3 * t_C, (p_2 + 2 * p_1 + 2 * p_0 + 2 * q_0 + q_1 + 4) >> 3) \quad (1369)$$

$$p_1' = Clip3(p_1 - 2 * t_C, p_1 + 2 * t_C, (p_2 + p_1 + p_0 + q_0 + 2) >> 2) \quad (1370)$$

$$p_2' = Clip3(p_2 - 1 * t_C, p_2 + 1 * t_C, (2 * p_3 + 3 * p_2 + p_1 + p_0 + q_0 + 4) >> 3) \quad (1371)$$

$$q_0' = \text{Clip3}(\ q_0 - 3 * t_C,\ q_0 + 3 * t_C,\ (\ p_1 + 2 * p_0 + 2 * q_0 + 2 * q_1 + q_2 + 4\ ) >> 3\ ) \quad (1372)$$

$$q_1' = \text{Clip3}(\ q_1 - 2 * t_C,\ q_1 + 2 * t_C,\ (\ p_0 + q_0 + q_1 + q_2 + 2\ ) >> 2\ ) \quad (1373)$$

$$q_2' = \text{Clip3}(\ q_2 - 1 * t_C,\ q_2 + 1 * t_C,\ (\ p_0 + q_0 + q_1 + 3 * q_2 + 2 * q_3 + 4\ ) >> 3\ ) \quad (1374)$$

– Otherwise, nDp and nDq are set both equal to 0 and the following weak filtering applies:
  – The following applies:

$$\Delta = (\ 9 * (\ q_0 - p_0\ ) - 3 * (\ q_1 - p_1\ ) + 8\ ) >> 4 \quad (1375)$$

  – When Abs(Δ) is less than t_C * 10, the following ordered steps apply:
    – The filtered sample values p_0' and q_0' are specified as follows:

$$\Delta = \text{Clip3}(\ -t_C,\ t_C,\ \Delta\ ) \quad (1376)$$

$$p_0' = \text{Clip1}(\ p_0 + \Delta\ ) \quad (1377)$$

$$q_0' = \text{Clip1}(\ q_0 - \Delta\ ) \quad (1378)$$

    – When dEp is equal to 1, the filtered sample value p_1' is specified as follows:

$$\Delta p = \text{Clip3}(\ -(\ t_C >> 1\ ),\ t_C >> 1,\ (\ (\ (\ p_2 + p_0 + 1\ ) >> 1\ ) - p_1 + \Delta\ ) >> 1\ ) \quad (1379)$$

$$p_1' = \text{Clip1}(\ p_1 + \Delta p\ ) \quad (1380)$$

    – When dEq is equal to 1, the filtered sample value q_1' is specified as follows:

$$\Delta q = \text{Clip3}(\ -(\ t_C >> 1\ ),\ t_C >> 1,\ (\ (\ (\ q_2 + q_0 + 1\ ) >> 1\ ) - q_1 - \Delta\ ) >> 1\ ) \quad (1381)$$

$$q_1' = \text{Clip1}(\ q_1 + \Delta q\ ) \quad (1382)$$

    – nDp is set equal to dEp + 1 and nDq is set equal to dEq + 1.

When nDp is greater than 0 and pred_mode_plt_flag of the coding unit that includes the coding block containing the sample p_0 is equal to 1, nDp is set equal to 0.

When nDq is greater than 0 and pred_mode_plt_flag of the coding unit that includes the coding block containing the sample q_0 is equal to 1, nDq is set equal to 0.

8.8.3.6.8 Filtering process for a luma sample using long filters

Inputs to this process are:

– the variables maxFilterLengthP and maxFilterLengthQ,

– the sample values p_i and q_j with i = 0..maxFilterLengthP and j = 0..maxFilterLengthQ,

– a variable t_C.

Outputs of this process are:

– the filtered sample values p_i' and q_j' with i = 0..maxFilterLengthP − 1, j = 0..maxFilterLengthQ − 1.

The variable refMiddle is derived as follows:

– If maxFilterLengthP is equal to maxFilterLengthQ and maxFilterLengthP is equal to 5, the following applies:

$$\text{refMiddle} = (\ p_4 + p_3 + 2 * (\ p_2 + p_1 + p_0 + q_0 + q_1 + q_2\ ) + q_3 + q_4 + 8\ ) >> 4 \quad (1383)$$

– Otherwise, if maxFilterLengthP is equal to maxFilterLengthQ and maxFilterLengthP is not equal to 5, the following applies:

$$\text{refMiddle} = (\ p_6 + p_5 + p_4 + p_3 + p_2 + p_1 + 2 * (\ p_0 + q_0\ ) + q_1 + q_2 + q_3 + q_4 + q_5 + q_6 + 8\ ) >> 4 \quad (1384)$$

– Otherwise, if one of the following conditions is true:

– maxFilterLengthQ is equal to 7 and maxFilterLengthP is equal to 5,

– maxFilterLengthQ is equal to 5 and maxFilterLengthP is equal to 7,

the following applies:

$$refMiddle = (p_5 + p_4 + p_3 + p_2 + 2 * (p_1 + p_0 + q_0 + q_1) + q_2 + q_3 + q_4 + q_5 + 8) >> 4 \quad (1385)$$

– Otherwise, if one of the following conditions is true:

– maxFilterLengthQ is equal to 5 and maxFilterLengthP is equal to 3,

– maxFilterLengthQ is equal to 3 and maxFilterLengthP is equal to 5,

the following applies:

$$refMiddle = (p_3 + p_2 + p_1 + p_0 + q_0 + q_1 + q_2 + q_3 + 4) >> 3 \quad (1386)$$

– Otherwise, if maxFilterLengthQ is equal to 7 and maxFilterLengthP is equal to 3, the following applies:

$$refMiddle = (2 * (p_2 + p_1 + p_0 + q_0) + p_0 + p_1 + q_1 + q_2 + q_3 + q_4 + q_5 + q_6 + 8) >> 4 \quad (1387)$$

– Otherwise, the following applies:

$$refMiddle = (p_6 + p_5 + p_4 + p_3 + p_2 + p_1 + 2 * (q_2 + q_1 + q_0 + p_0) + q_0 + q_1 + 8) >> 4 \quad (1388)$$

The variables refP and refQ are derived as follows:

$$refP = (p_{maxFilterLengthP} + p_{maxFilterLengthP-1} + 1) >> 1 \quad (1389)$$

$$refQ = (q_{maxFilterLengthQ} + q_{maxFilterLengthQ-1} + 1) >> 1 \quad (1390)$$

The variables f_i and t_CPD_i are defined as follows:

– If maxFilterLengthP is equal to 7, the following applies:

$$f_{0..6} = \{ 59, 50, 41, 32, 23, 14, 5 \} \quad (1391)$$

$$t_CPD_{0..6} = \{ 6, 5, 4, 3, 2, 1, 1 \} \quad (1392)$$

– Otherwise, if maxFilterLengthP is equal to 5, the following applies:

$$f_{0..4} = \{ 58, 45, 32, 19, 6 \} \quad (1393)$$

$$t_CPD_{0..4} = \{ 6, 5, 4, 3, 2 \} \quad (1394)$$

– Otherwise, the following applies:

$$f_{0..2} = \{ 53, 32, 11 \} \quad (1395)$$

$$t_CPD_{0..2} = \{ 6, 4, 2 \} \quad (1396)$$

The variables g_j and t_CQD_j are defined as follows:

– If maxFilterLengthQ is equal to 7, the following applies:

$$g_{0..6} = \{ 59, 50, 41, 32, 23, 14, 5 \} \quad (1397)$$

$$t_CQD_{0..6} = \{ 6, 5, 4, 3, 2, 1, 1 \} \quad (1398)$$

– Otherwise, if maxFilterLengthQ is equal to 5, the following applies:

$$g_{0..4} = \{ 58, 45, 32, 19, 6 \} \quad (1399)$$

$$t_CQD_{0..4} = \{ 6, 5, 4, 3, 2 \} \quad (1400)$$

– Otherwise, the following applies:

$$g_{0..2} = \{ 53, 32, 11 \} \quad (1401)$$

$$t_C QD_{0..2} = \{ 6, 4, 2 \} \quad (1402)$$

The filtered sample values p_i' and q_j' with i = 0..maxFilterLengthP − 1 and j = 0..maxFilterLengthQ − 1 are derived as follows:

$$p_i' = \text{Clip3}(p_i - (t_C * t_C PD_i >> 1), p_i + (t_C * t_C PD_i >> 1), (\text{refMiddle} * f_i + \text{refP} * (64 - f_i) + 32) >> 6) \quad (1403)$$

$$q_j' = \text{Clip3}(q_j - (t_C * t_C QD_j >> 1), q_j + (t_C * t_C QD_j >> 1), (\text{refMiddle} * g_j + \text{refQ} * (64 - g_j) + 32) >> 6) \quad (1404)$$

When pred_mode_plt_flag of the coding unit that includes the coding block containing the sample p_i is equal to 1, the filtered sample value, p_i' is substituted by the corresponding input sample value p_i with i = 0..maxFilterLengthP − 1.

When pred_mode_plt_flag of the coding unit that includes the coding block containing the sample q_i is equal to 1, the filtered sample value, q_i' is substituted by the corresponding input sample value q_j with j = 0..maxFilterLengthQ − 1.

8.8.3.6.9 Decision process for a chroma sample

Inputs to this process are:

– the sample values p_0, p_3, q_0 and q_3,

– the variables dpq, β and t_C.

Output of this process is the variable dSam containing a decision.

The variable dSam is specified as follows:

– If all of the following conditions are true, dSam is set equal to 1:

  – dpq is less than (β >> 2),

  – Abs($p_3 - p_0$) + Abs($q_0 - q_3$) is less than (β >> 3),

  – Abs($p_0 - q_0$) is less than (5 * t_C + 1) >> 1.

– Otherwise, dSam is set equal to 0.

8.8.3.6.10 Filtering process for a chroma sample

This process is only invoked when sps_chroma_format_idc is not equal to 0.

Inputs to this process are:

– the variables maxFilterLengthP and maxFilterLengthQ,

– the chroma sample values p_i and q_j with i = 0..maxFilterLengthP and j = 0..maxFilterLengthQ,

– a variable t_C.

Outputs of this process are the filtered sample values p_i' and q_j' with i = 0..maxFilterLengthP − 1 and j = 0..maxFilterLengthQ − 1.

The filtered sample values p_i' and q_j' with i = 0..maxFilterLengthP − 1 and j = 0..maxFilterLengthQ − 1 are derived as follows:

– If both of maxFilterLengthP and maxFilterLengthQ are equal to 3, the following strong filtering applies:

$$p_0' = \text{Clip3}(p_0 - t_C, p_0 + t_C, (p_3 + p_2 + p_1 + 2 * p_0 + q_0 + q_1 + q_2 + 4) >> 3) \quad (1405)$$

$$p_1' = \text{Clip3}(p_1 - t_C, p_1 + t_C, (2 * p_3 + p_2 + 2 * p_1 + p_0 + q_0 + q_1 + 4) >> 3) \quad (1406)$$

$$p_2' = \text{Clip3}(p_2 - t_C, p_2 + t_C, (3 * p_3 + 2 * p_2 + p_1 + p_0 + q_0 + 4) >> 3) \quad (1407)$$

$$q_0' = \text{Clip3}(q_0 - t_C, q_0 + t_C, (p_2 + p_1 + p_0 + 2 * q_0 + q_1 + q_2 + q_3 + 4) >> 3) \quad (1408)$$

$$q_1' = \text{Clip3}(q_1 - t_C, q_1 + t_C, (p_1 + p_0 + q_0 + 2 * q_1 + q_2 + 2 * q_3 + 4) >> 3) \quad (1409)$$

$$q_2' = \text{Clip3}(\ q_2 - t_C,\ q_2 + t_C,\ (\ p_0 + q_0 + q_1 + 2 * q_2 + 3 * q_3 + 4\ ) >> 3\ ) \quad (1410)$$

– Otherwise, if the variable maxFilterLengthP is equal to 1 and maxFilterLengthQ is equal to 3, the following filtering applies:

$$p_0' = \text{Clip3}(\ p_0 - t_C,\ p_0 + t_C,\ (\ 3 * p_1 + 2 * p_0 + q_0 + q_1 + q_2 + 4\ ) >> 3\ ) \quad (1411)$$

$$q_0' = \text{Clip3}(\ q_0 - t_C,\ q_0 + t_C,\ (\ 2 * p_1 + p_0 + 2 * q_0 + q_1 + q_2 + q_3 + 4\ ) >> 3\ ) \quad (1412)$$

$$q_1' = \text{Clip3}(\ q_1 - t_C,\ q_1 + t_C,\ (\ p_1 + p_0 + q_0 + 2 * q_1 + q_2 + 2 * q_3 + 4\ ) >> 3\ ) \quad (1413)$$

$$q_2' = \text{Clip3}(\ q_2 - t_C,\ q_2 + t_C,\ (\ p_0 + q_0 + q_1 + 2 * q_2 + 3 * q_3 + 4\ ) >> 3\ ) \quad (1414)$$

– Otherwise, the following weak filtering applies:

$$\Delta = \text{Clip3}(\ -t_C,\ t_C,\ (\ (\ (\ (\ q_0 - p_0\ ) << 2\ ) + p_1 - q_1 + 4\ ) >> 3\ )\ ) \quad (1415)$$

$$p_0' = \text{Clip1}(\ p_0 + \Delta\ ) \quad (1416)$$

$$q_0' = \text{Clip1}(\ q_0 - \Delta\ ) \quad (1417)$$

When pred_mode_plt_flag of the coding unit that includes the coding block containing the sample p_i is equal to 1, the filtered sample value, p_i' is substituted by the corresponding input sample value p_i with i = 0..maxFilterLengthP − 1.

When pred_mode_plt_flag of the coding unit that includes the coding block containing the sample q_i is equal to 1, the filtered sample value, q_i' is substituted by the corresponding input sample value q_i with i = 0..maxFilterLengthQ − 1.

8.8.4 Sample adaptive offset process

8.8.4.1 General

Inputs to this process are the reconstructed picture sample array prior to sample adaptive offset recPicture_L and, when sps_chroma_format_idc is not equal to 0, the arrays recPicture_{Cb} and recPicture_{Cr}.

Outputs of this process are the modified reconstructed picture sample array after sample adaptive offset saoPicture_L and, when sps_chroma_format_idc is not equal to 0, the arrays saoPicture_{Cb} and saoPicture_{Cr}.

This process is performed on a CTB basis after the completion of the deblocking filter process for the decoded picture.

The sample values in the modified reconstructed picture sample array saoPicture_L and, when sps_chroma_format_idc is not equal to 0, the arrays saoPicture_{Cb} and saoPicture_{Cr} are initially set equal to the sample values in the reconstructed picture sample array recPicture_L and, when sps_chroma_format_idc is not equal to 0, the arrays recPicture_{Cb} and recPicture_{Cr}, respectively.

For every CTU with CTB location (rx, ry), where rx = 0..PicWidthInCtbsY − 1 and ry = 0..PicHeightInCtbsY − 1, the following applies:

– When sh_sao_luma_used_flag of the current slice is equal to 1, the CTB modification process as specified in clause 8.8.4.2 is invoked with recPicture set equal to recPicture_L, cIdx set equal to 0, (rx, ry), and both nCtbSw and nCtbSh set equal to CtbSizeY as inputs, and the modified luma picture sample array saoPicture_L as output.

– When sps_chroma_format_idc is not equal to 0 and sh_sao_chroma_used_flag of the current slice is equal to 1, the CTB modification process as specified in clause 8.8.4.2 is invoked with recPicture set equal to recPicture_{Cb}, cIdx set equal to 1, (rx, ry), nCtbSw set equal to (1 << CtbLog2SizeY) / SubWidthC and nCtbSh set equal to (1 << CtbLog2SizeY) / SubHeightC as inputs, and the modified chroma picture sample array saoPicture_{Cb} as output.

– When sps_chroma_format_idc is not equal to 0 and sh_sao_chroma_used_flag of the current slice is equal to 1, the CTB modification process as specified in clause 8.8.4.2 is invoked with recPicture set equal to recPicture_{Cr}, cIdx set equal to 2, (rx, ry), nCtbSw set equal to (1 << CtbLog2SizeY) / SubWidthC and nCtbSh set equal to (1 << CtbLog2SizeY) / SubHeightC as inputs, and the modified chroma picture sample array saoPicture_{Cr} as output.

8.8.4.2 CTB modification process

Inputs to this process are:

– the picture sample array recPicture for the colour component cIdx,

– a variable cIdx specifying the colour component index,

– a pair of variables (rx, ry) specifying the CTB location,

– the CTB width nCtbSw and height nCtbSh.

Output of this process is a modified picture sample array saoPicture for the colour component cIdx.

The variables scaleWidth and scaleHeight are derived as follows:

$$\text{scaleWidth} = (\text{cIdx} == 0) ? 1 : \text{SubWidthC} \quad (1418)$$

$$\text{scaleHeight} = (\text{cIdx} == 0) ? 1 : \text{SubHeightC} \quad (1419)$$

The location (xCtb, yCtb), specifying the top-left sample of the current CTB for the colour component cIdx relative to the top-left sample of the current picture component cIdx, is derived as follows:

$$(\text{xCtb}, \text{yCtb}) = (\text{rx} * \text{nCtbSw}, \text{ry} * \text{nCtbSh}) \quad (1420)$$

The sample locations inside the current CTB are derived as follows:

$$(xS_i, yS_j) = (\text{xCtb} + i, \text{yCtb} + j) \quad (1421)$$

$$(xY_i, yY_j) = (\text{cIdx} == 0) ? (xS_i, yS_j) : (xS_i * \text{SubWidthC}, yS_j * \text{SubHeightC}) \quad (1422)$$

For all sample locations (xS_i, yS_j) and (xY_i, yY_j) with i = 0..nCtbSw − 1 and j = 0..nCtbSh − 1, the following applies:

– If SaoTypeIdx[cIdx][rx][ry] is equal to 0, saoPicture[xS_i][yS_j] is not modified.

– Otherwise, if SaoTypeIdx[cIdx][rx][ry] is equal to 2, the following ordered steps apply:

1. The values of hPos[k] and vPos[k] for k = 0..1 are specified in Table 44 based on SaoEoClass[cIdx][rx][ry].

2. The variable edgeIdx is derived as follows:

   – The modified sample locations (xS_{ik}', yS_{jk}') and (xY_{ik}', yY_{jk}') are derived as follows:

$$(xS_{ik}', yS_{jk}') = (xS_i + \text{hPos}[k], yS_j + \text{vPos}[k]) \quad (1423)$$

$$(xY_{ik}', yY_{jk}') = (\text{cIdx} == 0) ? (xS_{ik}', yS_{jk}') : (xS_{ik}' * \text{SubWidthC}, yS_{jk}' * \text{SubHeightC}) \quad (1424)$$

   – If one or more of the following conditions are true, edgeIdx is set equal to 0:

     – The sample at location (xS_{ik}', yS_{jk}') for any k = 0..1 is outside the picture boundaries.

     – The sample at location (xS_{ik}', yS_{jk}') for any k = 0..1 belongs to a different subpicture and sps_loop_filter_across_subpic_enabled_flag[CurrSubpicIdx] for the subpicture to which the sample recPicture[xS_i][yS_j] belongs to is equal to 0.

     – pps_loop_filter_across_slices_enabled_flag is equal to 0 and the sample at location (xS_{ik}', yS_{jk}') for any k = 0..1 belongs to a different slice.

     – pps_loop_filter_across_tiles_enabled_flag is equal to 0 and the sample at location (xS_{ik}', yS_{jk}') for any k = 0..1 belongs to a different tile.

     – VirtualBoundariesPresentFlag is equal to 1 and xS_i is equal to (VirtualBoundaryPosX[n] / scaleWidth) − 1 for any n = 0..NumVerVirtualBoundaries − 1 and SaoEoClass[cIdx][rx][ry] is not equal to 1.

     – VirtualBoundariesPresentFlag is equal to 1 and xS_i is equal to VirtualBoundaryPosX[n] / scaleWidth for any n = 0..NumVerVirtualBoundaries − 1 and SaoEoClass[cIdx][rx][ry] is not equal to 1.

     – VirtualBoundariesPresentFlag is equal to 1 and yS_j is equal to (VirtualBoundaryPosY[n] / scaleHeight) − 1 for any n = 0..NumHorVirtualBoundaries − 1 and SaoEoClass[cIdx][rx][ry] is not equal to 0.

     – VirtualBoundariesPresentFlag is equal to 1 and yS_j is equal to VirtualBoundaryPosY[n] / scaleHeight for any n = 0..NumHorVirtualBoundaries − 1 and SaoEoClass[cIdx][rx][ry] is not equal to 0.

   – Otherwise, edgeIdx is derived as follows:

– The following applies:

$$\text{edgeIdx} = 2 + \text{Sign}(\ \text{recPicture}[\ xS_i\ ][\ yS_j\ ] - \text{recPicture}[\ xS_i + \text{hPos}[\ 0\ ]\ ][\ yS_j + \text{vPos}[\ 0\ ]\ ]\ ) + \text{Sign}(\ \text{recPicture}[\ xS_i\ ][\ yS_j\ ] - \text{recPicture}[\ xS_i + \text{hPos}[\ 1\ ]\ ][\ yS_j + \text{vPos}[\ 1\ ]\ ]\ ) \quad (1425)$$

– When edgeIdx is equal to 0, 1, or 2, edgeIdx is modified as follows:

$$\text{edgeIdx} = (\ \text{edgeIdx} == 2\ )\ ?\ 0 : (\ \text{edgeIdx} + 1\ ) \quad (1426)$$

3. The modified picture sample array saoPicture[xS_i][yS_j] is derived as follows:

$$\text{saoPicture}[\ xS_i\ ][\ yS_j\ ] = \text{Clip3}(\ 0, (\ 1 \ll \text{BitDepth}\ ) - 1, \text{recPicture}[\ xS_i\ ][\ yS_j\ ] + \text{SaoOffsetVal}[\ \text{cIdx}\ ][\ \text{rx}\ ][\ \text{ry}\ ][\ \text{edgeIdx}\ ]\ ) \quad (1427)$$

– Otherwise (SaoTypeIdx[cIdx][rx][ry] is equal to 1), the following ordered steps apply:

1. The variable bandShift is set equal to BitDepth − 5.
2. The variable saoLeftClass is set equal to sao_band_position[cIdx][rx][ry].
3. The list bandTable is defined with 32 elements and all elements are initially set equal to 0. Then, four of its elements (indicating the starting position of bands for explicit offsets) are modified as follows:

```
for( k = 0; k < 4; k++ )
    bandTable[ ( k + saoLeftClass ) & 31 ] = k + 1          (1428)
```

4. The variable bandIdx is set equal to bandTable[recPicture[xS_i][yS_j] >> bandShift].
5. The modified picture sample array saoPicture[xS_i][yS_j] is derived as follows:

$$\text{saoPicture}[\ xS_i\ ][\ yS_j\ ] = \text{Clip3}(\ 0, (\ 1 \ll \text{BitDepth}\ ) - 1, \text{recPicture}[\ xS_i\ ][\ yS_j\ ] + \text{SaoOffsetVal}[\ \text{cIdx}\ ][\ \text{rx}\ ][\ \text{ry}\ ][\ \text{bandIdx}\ ]\ ) \quad (1429)$$

Table 44 – Specification of hPos and vPos according to the sample adaptive offset class

SaoEoClass[cIdx][rx][ry]	0	1	2	3
hPos[0]	−1	0	−1	1
hPos[1]	1	0	1	−1
vPos[0]	0	−1	−1	−1
vPos[1]	0	1	1	1

8.8.5 Adaptive loop filter process

8.8.5.1 General

Inputs of this process are the reconstructed picture sample array prior to adaptive loop filter recPicture_L and, when sps_chroma_format_idc is not equal to 0, the arrays recPicture_{Cb} and recPicture_{Cr}.

Outputs of this process are the modified reconstructed picture sample array after adaptive loop filter alfPicture_L and, when sps_chroma_format_idc is not equal to 0, the arrays ccAlfPicture_{Cb} and ccAlfPicture_{Cr}.

The sample values in the modified reconstructed picture sample array after adaptive loop filter alfPicture_L and, when sps_chroma_format_idc is not equal to 0, the arrays alfPicture_{Cb} and alfPicture_{Cr} are initially set equal to the sample values in the reconstructed picture sample array prior to adaptive loop filter recPicture_L and, when sps_chroma_format_idc is not equal to 0, the arrays recPicture_{Cb} and recPicture_{Cr}, respectively.

The following ordered steps apply:

– For every coding tree unit with luma coding tree block location (rx, ry), where rx = 0..PicWidthInCtbsY − 1 and ry = 0..PicHeightInCtbsY − 1, the following applies:
  – When alf_ctb_flag[0][rx][ry] is equal to 1, the coding tree block filtering process for luma samples as specified in clause 8.8.5.2 is invoked with recPicture_L, alfPicture_L, and the luma coding tree block location (xCtb, yCtb) set

equal to (rx << CtbLog2SizeY, ry << CtbLog2SizeY) as inputs, and the output is the modified filtered picture $alfPicture_L$.

– When sps_chroma_format_idc is not equal to 0 and alf_ctb_flag[1][rx][ry] is equal to 1, the coding tree block filtering process for chroma samples as specified in clause 8.8.5.4 is invoked with recPicture set equal to $recPicture_{Cb}$, alfPicture set equal to $alfPicture_{Cb}$, the chroma coding tree block location (xCtbC, yCtbC) set equal to ((rx << CtbLog2SizeY) / SubWidthC, (ry << CtbLog2SizeY) / SubHeightC), and the alternative chroma filter index altIdx set equal to alf_ctb_filter_alt_idx[0][rx][ry] as inputs, and the output is the modified filtered picture $alfPicture_{Cb}$.

– When sps_chroma_format_idc is not equal to 0 and alf_ctb_flag[2][rx][ry] is equal to 1, the coding tree block filtering process for chroma samples as specified in clause 8.8.5.4 is invoked with recPicture set equal to $recPicture_{Cr}$, alfPicture set equal to $alfPicture_{Cr}$, the chroma coding tree block location (xCtbC, yCtbC) set equal to ((rx << CtbLog2SizeY) / SubWidthC, (ry << CtbLog2SizeY) / SubHeightC), and the alternative chroma filter index altIdx set equal to alf_ctb_filter_alt_idx[1][rx][ry] as inputs, and the output is the modified filtered picture $alfPicture_{Cr}$.

– When sps_chroma_format_idc is not equal to 0, the sample values in the arrays $ccAlfPicture_{Cb}$ and $ccAlfPicture_{Cr}$ are set equal to the sample values in the arrays $alfPicture_{Cb}$ and $alfPicture_{Cr}$, respectively.

– For every coding tree unit with luma coding tree block location (rx, ry), where rx = 0..PicWidthInCtbsY − 1 and ry = 0..PicHeightInCtbsY − 1, the following applies:

  – When sps_chroma_format_idc is not equal to 0 and alf_ctb_cc_cb_idc[rx][ry] is not equal to 0, the cross-component filtering process as specified in clause 8.8.5.7 is invoked with $recPicture_L$ set equal to $recPicture_L$, $alfPicture_C$ set equal to $alfPicture_{Cb}$, the chroma coding tree block location (xCtbC, yCtbC) set equal to ((rx << CtbLog2SizeY) / SubWidthC, (ry << CtbLog2SizeY) / SubHeightC), ccAlfWidth set equal to (1 << CtbLog2SizeY) / SubWidthC, ccAlfHeight set equal to (1 << CtbLog2SizeY) / SubHeightC, and the cross-component filter coefficients CcAlfCoeff[j] set equal to $CcAlfApsCoeff_{Cb}$[sh_alf_cc_cb_aps_id][alf_ctb_cc_cb_idc[rx][ry] − 1][j], with j = 0..6, as inputs, and the output is the modified filtered picture $ccAlfPicture_{Cb}$.

  – When sps_chroma_format_idc is not equal to 0 and alf_ctb_cc_cr_idc[rx][ry] is not equal to 0, the cross-component filtering process as specified in clause 8.8.5.7 is invoked with $recPicture_L$ set equal to $recPicture_L$, $alfPicture_C$ set equal to $alfPicture_{Cr}$, the chroma coding tree block location (xCtbC, yCtbC) set equal to ((rx << CtbLog2SizeY) / SubWidthC, (ry << CtbLog2SizeY) / SubHeightC), ccAlfWidth set equal to (1 << CtbLog2SizeY) / SubWidthC, ccAlfHeight set equal to (1 << CtbLog2SizeY) / SubHeightC, and the cross-component filter coefficients CcAlfCoeff[j] set equal to $CcAlfApsCoeff_{Cr}$[sh_alf_cc_cr_aps_id][alf_ctb_cc_cr_idc[rx][ry] − 1][j], with j = 0..6, as inputs, and the output is the modified filtered picture $ccAlfPicture_{Cr}$.

8.8.5.2 Coding tree block filtering process for luma samples

Inputs of this process are:

– a reconstructed luma picture sample array recPicture prior to the adaptive loop filtering process,

– a filtered reconstructed luma picture sample array $alfPicture_L$,

– a luma location (xCtb, yCtb) specifying the top-left sample of the current luma coding tree block relative to the top-left sample of the current picture.

Output of this process is the modified filtered reconstructed luma picture sample array $alfPicture_L$.

The derivation process for filter index specified in clause 8.8.5.3 is invoked with the location (xCtb, yCtb) and the reconstructed luma picture sample array recPicture as inputs, and filtIdx[x][y] and transposeIdx[x][y] with x, y = 0..CtbSizeY − 1 as outputs.

For the derivation of the filtered reconstructed luma samples $alfPicture_L$[xCtb + x][yCtb + y], each reconstructed luma sample inside the current luma coding tree block recPicture[xCtb + x][yCtb + y] is filtered as follows with x, y = 0..CtbSizeY − 1:

– The array of luma filter coefficients f[j] and the array of luma clipping values c[j] corresponding to the filter specified by filtIdx[x][y] is derived as follows with j = 0..11:

  – If AlfCtbFiltSetIdxY[xCtb >> CtbLog2SizeY][yCtb >> CtbLog2SizeY] is less than 16, the following applies:

    i = AlfCtbFiltSetIdxY[xCtb >> CtbLog2SizeY][yCtb >> CtbLog2SizeY] (1430)

$$f[j] = AlfFixFiltCoeff[AlfClassToFiltMap[i][filtIdx[x][y]]][j] \tag{1431}$$

$$c[j] = 2^{BitDepth} \tag{1432}$$

– Otherwise (AlfCtbFiltSetIdxY[xCtb >> CtbLog2SizeY][yCtb >> CtbLog2SizeY] is greater than or equal to 16), the following applies:

$$i = sh\_alf\_aps\_id\_luma[AlfCtbFiltSetIdxY[xCtb >> CtbLog2SizeY][yCtb >> CtbLog2SizeY] - 16] \tag{1433}$$

$$f[j] = AlfCoeff_L[i][filtIdx[x][y]][j] \tag{1434}$$

$$c[j] = AlfClip_L[i][filtIdx[x][y]][j] \tag{1435}$$

– The luma filter coefficients and clipping values index idx are derived depending on transposeIdx[x][y] as follows:

– If transposeIdx[x][y] is equal to 1, the following applies:

$$idx[ ] = \{ 9, 4, 10, 8, 1, 5, 11, 7, 3, 0, 2, 6 \} \tag{1436}$$

– Otherwise, if transposeIdx[x][y] is equal to 2, the following applies:

$$idx[ ] = \{ 0, 3, 2, 1, 8, 7, 6, 5, 4, 9, 10, 11 \} \tag{1437}$$

– Otherwise, if transposeIdx[x][y] is equal to 3, the following applies:

$$idx[ ] = \{ 9, 8, 10, 4, 3, 7, 11, 5, 1, 0, 2, 6 \} \tag{1438}$$

– Otherwise, the following applies:

$$idx[ ] = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 \} \tag{1439}$$

– The variables clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag are derived by invoking the ALF boundary position derivation process as specified in clause 8.8.5.5 with (xCtb, yCtb), (x, y) and the variable vbOffset set equal to 4 as inputs.

– The locations ($h_{x+i,\,y+j}$, v_{y+j}) for each of the corresponding luma samples (x, y) inside the given array recPicture of luma samples with i, j = −3..3 are derived as follows:

$$h_{x+i,\,y+j} = Clip3(0, pps\_pic\_width\_in\_luma\_samples - 1, xCtb + x + i) \tag{1440}$$

$$v_{y+j} = Clip3(0, pps\_pic\_height\_in\_luma\_samples - 1, yCtb + y + j) \tag{1441}$$

– The location ($h_{x+i,\,y+j}$, v_{y+j}) is modified by invoking the ALF sample padding process as specified in clause 8.8.5.6 with (xCtb, yCtb), ($h_{x+i,\,y+j}$, v_{y+j}), the variable isChroma set equal to 0, clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag as inputs.

– The variable applyAlfLineBufBoundary is derived as follows:

– If the bottom boundary of the current coding tree block is the bottom boundary of current picture and pps_pic_height_in_luma_samples − yCtb <= CtbSizeY − 4, applyAlfLineBufBoundary is set equal to 0.

– Otherwise, applyAlfLineBufBoundary is set equal to 1.

– The vertical sample position offsets y1, y2, y3 and the variable alfShiftY are specified in Table 45 according to the vertical luma sample position y and applyAlfLineBufBoundary.

– The variable curr is derived as follows:

$$curr = recPicture[h_{x,\,y}][v_y] \tag{1442}$$

– The variable sum is derived as follows:

$$
\begin{aligned}
\text{sum} = {} & \text{f[idx[0]]} * (\ \text{Clip3}(-\text{c[idx[0]], c[idx[0]]}, \text{recPicture[} h_{x,\, y+y3} \text{][} v_{y+y3} \text{]} - \text{curr}) + \\
& \qquad \text{Clip3}(-\text{c[idx[0]], c[idx[0]]}, \text{recPicture[} h_{x,\, y-y3} \text{][} v_{y-y3} \text{]} - \text{curr})) + \\
& \text{f[idx[1]]} * (\ \text{Clip3}(-\text{c[idx[1]], c[idx[1]]}, \text{recPicture[} h_{x+1,\, y+y2} \text{][} v_{y+y2} \text{]} - \text{curr}) + \\
& \qquad \text{Clip3}(-\text{c[idx[1]], c[idx[1]]}, \text{recPicture[} h_{x-1,\, y-y2} \text{][} v_{y-y2} \text{]} - \text{curr})) + \\
& \text{f[idx[2]]} * (\ \text{Clip3}(-\text{c[idx[2]], c[idx[2]]}, \text{recPicture[} h_{x,\, y+y2} \text{][} v_{y+y2} \text{]} - \text{curr}) + \\
& \qquad \text{Clip3}(-\text{c[idx[2]], c[idx[2]]}, \text{recPicture[} h_{x,\, y-y2} \text{][} v_{y-y2} \text{]} - \text{curr})) + \\
& \text{f[idx[3]]} * (\ \text{Clip3}(-\text{c[idx[3]], c[idx[3]]}, \text{recPicture[} h_{x-1,\, y+y2} \text{][} v_{y+y2} \text{]} - \text{curr}) + \\
& \qquad \text{Clip3}(-\text{c[idx[3]], c[idx[3]]}, \text{recPicture[} h_{x+1,\, y-y2} \text{][} v_{y-y2} \text{]} - \text{curr})) + \\
& \text{f[idx[4]]} * (\ \text{Clip3}(-\text{c[idx[4]], c[idx[4]]}, \text{recPicture[} h_{x+2,\, y+y1} \text{][} v_{y+y1} \text{]} - \text{curr}) + \\
& \qquad \text{Clip3}(-\text{c[idx[4]], c[idx[4]]}, \text{recPicture[} h_{x-2,\, y-y1} \text{][} v_{y-y1} \text{]} - \text{curr})) + \\
& \text{f[idx[5]]} * (\ \text{Clip3}(-\text{c[idx[5]], c[idx[5]]}, \text{recPicture[} h_{x+1,\, y+y1} \text{][} v_{y+y1} \text{]} - \text{curr}) + \\
& \qquad \text{Clip3}(-\text{c[idx[5]], c[idx[5]]}, \text{recPicture[} h_{x-1,\, y-y1} \text{][} v_{y-y1} \text{]} - \text{curr})) + \\
& \text{f[idx[6]]} * (\ \text{Clip3}(-\text{c[idx[6]], c[idx[6]]}, \text{recPicture[} h_{x,\, y+y1} \text{][} v_{y+y1} \text{]} - \text{curr}) + \\
& \qquad \text{Clip3}(-\text{c[idx[6]], c[idx[6]]}, \text{recPicture[} h_{x,\, y-y1} \text{][} v_{y-y1} \text{]} - \text{curr})) + \\
& \text{f[idx[7]]} * (\ \text{Clip3}(-\text{c[idx[7]], c[idx[7]]}, \text{recPicture[} h_{x-1,\, y+y1} \text{][} v_{y+y1} \text{]} - \text{curr}) + \\
& \qquad \text{Clip3}(-\text{c[idx[7]], c[idx[7]]}, \text{recPicture[} h_{x+1,\, y-y1} \text{][} v_{y-y1} \text{]} - \text{curr})) + \\
& \text{f[idx[8]]} * (\ \text{Clip3}(-\text{c[idx[8]], c[idx[8]]}, \text{recPicture[} h_{x-2,\, y+y1} \text{][} v_{y+y1} \text{]} - \text{curr}) + \\
& \qquad \text{Clip3}(-\text{c[idx[8]], c[idx[8]]}, \text{recPicture[} h_{x+2,\, y-y1} \text{][} v_{y-y1} \text{]} - \text{curr})) + \\
& \text{f[idx[9]]} * (\ \text{Clip3}(-\text{c[idx[9]], c[idx[9]]}, \text{recPicture[} h_{x+3,\, y} \text{][} v_{y} \text{]} - \text{curr}) + \\
& \qquad \text{Clip3}(-\text{c[idx[9]], c[idx[9]]}, \text{recPicture[} h_{x-3,\, y} \text{][} v_{y} \text{]} - \text{curr})) + \\
& \text{f[idx[10]]} * (\ \text{Clip3}(-\text{c[idx[10]], c[idx[10]]}, \text{recPicture[} h_{x+2,\, y} \text{][} v_{y} \text{]} - \text{curr}) + \\
& \qquad \text{Clip3}(-\text{c[idx[10]], c[idx[10]]}, \text{recPicture[} h_{x-2,\, y} \text{][} v_{y} \text{]} - \text{curr})) + \\
& \text{f[idx[11]]} * (\ \text{Clip3}(-\text{c[idx[11]], c[idx[11]]}, \text{recPicture[} h_{x+1,\, y} \text{][} v_{y} \text{]} - \text{curr}) + \\
& \qquad \text{Clip3}(-\text{c[idx[11]], c[idx[11]]}, \text{recPicture[} h_{x-1,\, y} \text{][} v_{y} \text{]} - \text{curr}))
\end{aligned} \tag{1443}
$$

$$\text{sum} = \text{curr} + ((\text{sum} + (1 \ll (\text{alfShiftY} - 1))) \gg \text{alfShiftY}) \tag{1444}$$

– The modified filtered reconstructed luma picture sample alfPicture_L[xCtb + x][yCtb + y] is derived as follows:

$$\text{alfPicture}_L[\ \text{xCtb} + x\ ][\ \text{yCtb} + y\ ] = \text{Clip3}(0, (1 \ll \text{BitDepth}) - 1, \text{sum}) \tag{1445}$$

Table 45 – Specification of y1, y2, y3 and alfShiftY according to the vertical luma sample position y and applyAlfLineBufBoundary

Condition	alfShiftY	y1	y2	y3
(y = = CtbSizeY − 5 || y = = CtbSizeY − 4) && (applyAlfLineBufBoundary = = 1)	10	0	0	0
(y = = CtbSizeY − 6 || y = = CtbSizeY − 3) && (applyAlfLineBufBoundary = = 1)	7	1	1	1
(y = = CtbSizeY − 7 || y = = CtbSizeY − 2) && (applyAlfLineBufBoundary = = 1)	7	1	2	2
Otherwise	7	1	2	3

8.8.5.3 Derivation process for ALF transpose and filter index for luma samples

Inputs of this process are:

– a luma location (xCtb, yCtb) specifying the top-left sample of the current luma coding tree block relative to the top-left sample of the current picture,

– a reconstructed luma picture sample array recPicture prior to the adaptive loop filtering process.

Outputs of this process are:

– the classification filter index array filtIdx[x][y] with x, y = 0..CtbSizeY − 1,

– the transpose index array transposeIdx[x][y] with x, y = 0..CtbSizeY − 1.

The variables ac[x][y], sumH[x][y], sumV[x][y], sumD0[x][y], sumD1[x][y] and sumOfHV[x][y] with x, y = 0..(CtbSizeY − 1) >> 2 are derived as follows:

– The variables x4 and y4 are set as (x << 2) and (y << 2), respectively.

– The variables minY, maxY, and ac[x][y] are derived as follows:

  – If y4 is equal to (CtbSizeY − 8) and one of the following conditions is true, minY is set equal to −2, maxY is set equal to 3, and ac[x][y] is set equal to 3:

    – The bottom boundary of the current coding tree block is the bottom boundary of the picture and pps_pic_height_in_luma_samples − yCtb > CtbSizeY − 4.

    – The bottom boundary of the current coding tree block is not the bottom boundary of the picture.

  – Otherwise, if y4 is equal to (CtbSizeY − 4) and one of the following conditions is true, minY is set equal to 0, maxY is set equal to 5, and ac[x][y] is set equal to 3:

    – The bottom boundary of the current coding tree block is the bottom boundary of the picture and pps_pic_height_in_luma_samples − yCtb > CtbSizeY − 4.

    – The bottom boundary of the current coding tree block is not the bottom boundary of the picture.

  – Otherwise, minY is set equal to −2 and maxY is set equal to 5, and ac[x][y] is set equal to 2.

– The variables clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag are derived by invoking the ALF boundary position derivation process as specified in clause 8.8.5.5 with (xCtb, yCtb), (x4, y4) and the variable vbOffset set equal to 4 as inputs.

– The locations ($h_{x4 + i, y4 + j}$, $v_{y4 + j}$) for each of the corresponding luma samples inside the given array recPicture of luma samples with i, j = −3..6 are derived as follows:

$$h_{x4 + i, y4 + j} = \text{Clip3}(0, \text{pps\_pic\_width\_in\_luma\_samples} - 1, \text{xCtb} + \text{x4} + i) \quad (1446)$$

$$v_{y4 + j} = \text{Clip3}(0, \text{pps\_pic\_height\_in\_luma\_samples} - 1, \text{yCtb} + \text{y4} + j) \quad (1447)$$

  – The location ($h_{x4 + i, y4 + j}$, $v_{y4 + j}$) is modified by invoking the ALF sample padding process as specified in clause 8.8.5.6 with (xCtb, yCtb), ($h_{x4 + i, y4 + j}$, $v_{y4 + j}$), the variable isChroma set equal to 0, clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag as inputs.

– The variables filtH[i][j], filtV[i][j], filtD0[i][j] and filtD1[i][j] with i, j = −2..5 are derived as follows:

  – If both i and j are even numbers or both i and j are not even numbers, the following applies:

$$\begin{aligned}\text{filtH}[i][j] = \text{Abs}((&\text{recPicture}[h_{x4 + i, y4 + j}][v_{y4 + j}] << 1) - \\ &\text{recPicture}[h_{x4 + i - 1, y4 + j}][v_{y4 + j}] - \\ &\text{recPicture}[h_{x4 + i + 1, y4 + j}][v_{y4 + j}])\end{aligned} \quad (1448)$$

$$\begin{aligned}\text{filtV}[i][j] = \text{Abs}((&\text{recPicture}[h_{x4 + i, y4 + j}][v_{y4 + j}] << 1) - \\ &\text{recPicture}[h_{x4 + i, y4 + j - 1}][v_{y4 + j - 1}] - \\ &\text{recPicture}[h_{x4 + i, y4 + j + 1}][v_{y4 + j + 1}])\end{aligned} \quad (1449)$$

$$\begin{aligned}\text{filtD0}[i][j] = \text{Abs}((&\text{recPicture}[h_{x4 + i, y4 + j}][v_{y4 + j}] << 1) - \\ &\text{recPicture}[h_{x4 + i - 1, y4 + j - 1}][v_{y4 + j - 1}] - \\ &\text{recPicture}[h_{x4 + i + 1, y4 + j + 1}][v_{y4 + j + 1}])\end{aligned} \quad (1450)$$

$$\begin{aligned}\text{filtD1}[i][j] = \text{Abs}((&\text{recPicture}[h_{x4 + i, y4 + j}][v_{y4 + j}] << 1) - \\ &\text{recPicture}[h_{x4 + i + 1, y4 + j - 1}][v_{y4 + j - 1}] - \\ &\text{recPicture}[h_{x4 + i - 1, y4 + j + 1}][v_{y4 + j + 1}])\end{aligned} \quad (1451)$$

  – Otherwise, filtH[i][j], filtV[i][j], filtD0[i][j] and filtD1[i][j] are set equal to 0.

– The variables sumH[x][y], sumV[x][y], sumD0[x][y], sumD1[x][y] and sumOfHV[x][y] are derived as follows:

$$\text{sumH}[x][y] = \Sigma_i \Sigma_j \text{ filtH}[i][j], \text{ with } i = -2..5, j = \text{minY..maxY} \quad (1452)$$

$$\text{sumV}[x][y] = \Sigma_i \Sigma_j \text{ filtV}[i][j], \text{ with } i = -2..5, j = \text{minY..maxY} \quad (1453)$$

$$\text{sumD0}[x][y] = \Sigma_i \Sigma_j \text{ filtD0}[i][j], \text{ with } i = -2..5, j = \text{minY..maxY} \quad (1454)$$

$$sumD1[x][y] = \Sigma_i\Sigma_j filtD1[i][j], with i = -2..5, j = minY..maxY \quad (1455)$$

$$sumOfHV[x][y] = sumH[x][y] + sumV[x][y] \quad (1456)$$

The classification filter index array filtIdx and transpose index array transposeIdx are derived by the following steps:

1. The variables dir1[x][y], dir2[x][y] and dirS[x][y] with x, y = 0..CtbSizeY − 1 are derived as follows:
   – The variables hv1, hv0 and dirHV are derived as follows:
     – If sumV[x >> 2][y >> 2] is greater than sumH[x >> 2][y >> 2], the following applies:

$$hv1 = sumV[x >> 2][y >> 2] \quad (1457)$$

$$hv0 = sumH[x >> 2][y >> 2] \quad (1458)$$

$$dirHV = 1 \quad (1459)$$

     – Otherwise, the following applies:

$$hv1 = sumH[x >> 2][y >> 2] \quad (1460)$$

$$hv0 = sumV[x >> 2][y >> 2] \quad (1461)$$

$$dirHV = 3 \quad (1462)$$

   – The variables d1, d0 and dirD are derived as follows:
     – If sumD0[x >> 2][y >> 2] is greater than sumD1[x >> 2][y >> 2], the following applies:

$$d1 = sumD0[x >> 2][y >> 2] \quad (1463)$$

$$d0 = sumD1[x >> 2][y >> 2] \quad (1464)$$

$$dirD = 0 \quad (1465)$$

     – Otherwise, the following applies:

$$d1 = sumD1[x >> 2][y >> 2] \quad (1466)$$

$$d0 = sumD0[x >> 2][y >> 2] \quad (1467)$$

$$dirD = 2 \quad (1468)$$

   – The variables hvd1, hvd0, are derived as follows:

$$hvd1 = (d1 * hv0 > hv1 * d0) ? d1 : hv1 \quad (1469)$$

$$hvd0 = (d1 * hv0 > hv1 * d0) ? d0 : hv0 \quad (1470)$$

   – The variables dirS[x][y], dir1[x][y] and dir2[x][y] derived as follows:

$$dir1[x][y] = (d1 * hv0 > hv1 * d0) ? dirD : dirHV \quad (1471)$$

$$dir2[x][y] = (d1 * hv0 > hv1 * d0) ? dirHV : dirD \quad (1472)$$

$$dirS[x][y] = (hvd1 *2 > 9 * hvd0) ? 2 : ((hvd1 > 2 * hvd0) ? 1 : 0) \quad (1473)$$

2. The variable avgVar[x][y] with x, y = 0..CtbSizeY − 1 is derived as follows:

$$varTab[ ] = \{ 0, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 4 \} \quad (1474)$$

$$avgVar[x][y] = varTab[Clip3(0, 15, (sumOfHV[x >> 2][y >> 2] * ac[x >> 2][y >> 2]) >> (BitDepth - 1))] \quad (1475)$$

3. The classification filter index array filtIdx[x][y] and the transpose index array transposeIdx[x][y] with x, y = 0..CtbSizeY − 1 are derived as follows:

$$transposeTable[ ] = \{ 0, 1, 0, 2, 2, 3, 1, 3 \}$$

$$transposeIdx[x][y] = transposeTable[dir1[x][y] * 2 + (dir2[x][y] >> 1)]$$

$\text{filtIdx}[\ x\ ][\ y\ ] = \text{avgVar}[\ x\ ][\ y\ ]$

– When dirS[x][y] is not equal to 0, filtIdx[x][y] is modified as follows:

$$\text{filtIdx}[\ x\ ][\ y\ ] \mathrel{+}= (\ (\ (\ \text{dir1}[\ x\ ][\ y\ ]\ \&\ 0x1\ )\ \ll\ 1\ ) + \text{dirS}[\ x\ ][\ y\ ]\ ) * 5 \tag{1476}$$

8.8.5.4 Coding tree block filtering process for chroma samples

Inputs of this process are:

– a reconstructed chroma picture sample array recPicture prior to the adaptive loop filtering process,

– a filtered reconstructed chroma picture sample array alfPicture,

– a chroma location (xCtbC, yCtbC) specifying the top-left sample of the current chroma coding tree block relative to the top-left sample of the current picture,

– an alternative chroma filter index altIdx.

Output of this process is the modified filtered reconstructed chroma picture sample array alfPicture.

The width and height of the current chroma coding tree block ctbWidthC and ctbHeightC is derived as follows:

$$\text{ctbWidthC} = \text{CtbSizeY} / \text{SubWidthC} \tag{1477}$$

$$\text{ctbHeightC} = \text{CtbSizeY} / \text{SubHeightC} \tag{1478}$$

For the derivation of the filtered reconstructed chroma samples alfPicture[xCtbC + x][yCtbC + y], each reconstructed chroma sample inside the current chroma coding tree block recPicture[xCtbC + x][yCtbC + y] is filtered as follows with x = 0..ctbWidthC − 1, y = 0..ctbHeightC − 1:

– The variables clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag are derived by invoking the ALF boundary position derivation process as specified in clause 8.8.5.5 with (xCtbC * SubWidthC, yCtbC * SubHeightC), (x * SubWidthC, y *SubHeightC) and the variable vbOffset set equal to 2 * SubHeightC as inputs.

– The locations ($h_{x+i,\,y+j}$, v_{y+j}) for each of the corresponding chroma samples (x, y) inside the given array recPicture of chroma samples with i, j = −2..2 are derived as follows:

$$h_{x+i,\,y+j} = \text{Clip3}(\ 0,\ \text{pps\_pic\_width\_in\_luma\_samples} / \text{SubWidthC} - 1,\ \text{xCtbC} + x + i\ ) \tag{1479}$$

$$v_{y+j} = \text{Clip3}(\ 0,\ \text{pps\_pic\_height\_in\_luma\_samples} / \text{SubHeightC} - 1,\ \text{yCtbC} + y + j\ ) \tag{1480}$$

– The location ($h_{x+i,\,y+j}$, v_{y+j}) is modified by invoking the ALF sample padding process as specified in clause 8.8.5.6 with (xCtbC * SubWidthC, yCtbC * SubHeightC), ($h_{x+i,\,y+j}$, v_{y+j}), the variable isChroma set equal to 1, clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag as inputs.

– The variable applyAlfLineBufBoundary is derived as follows:

  – If the bottom boundary of the current coding tree block is the bottom boundary of the picture and pps_pic_height_in_luma_samples − (yCtbC * SubHeightC) < CtbSizeY − 4, applyAlfLineBufBoundary is set equal to 0.

  – Otherwise, applyAlfLineBufBoundary is set equal to 1.

– The vertical sample position offsets y1, y2 and the variable alfShiftC are specified in Table 46 according to the vertical chroma sample position y and applyAlfLineBufBoundary.

– The variable curr is derived as follows:

$$\text{curr} = \text{recPicture}[\ h_{x,\,y}\ ][\ v_y\ ] \tag{1481}$$

– The array of chroma filter coefficients f[j] and the array of chroma clipping values c[j] is derived as follows with j = 0..5:

$$f[\ j\ ] = \text{AlfCoeff}_C[\ \text{sh\_alf\_aps\_id\_chroma}\ ][\ \text{altIdx}\ ][\ j\ ] \tag{1482}$$

$$c[\ j\ ] = \text{AlfClip}_C[\ \text{sh\_alf\_aps\_id\_chroma}\ ][\ \text{altIdx}\ ][\ j\ ] \tag{1483}$$

– The variable sum is derived as follows:

$$\begin{aligned}\text{sum} = {} & \text{f[0] * (Clip3(} -\text{c[0], c[0], recPicture[} h_{x,\, y + y2} \text{][} v_{y + y2} \text{]} - \text{curr) +} \\ & \quad \text{Clip3(} -\text{c[0], c[0], recPicture[} h_{x,\, y - y2} \text{][} v_{y - y2} \text{]} - \text{curr)) +} \\ & \text{f[1] * (Clip3(} -\text{c[1], c[1], recPicture[} h_{x + 1,\, y + y1} \text{][} v_{y + y1} \text{]} - \text{curr) +} \\ & \quad \text{Clip3(} -\text{c[1], c[1], recPicture[} h_{x - 1,\, y - y1} \text{][} v_{y - y1} \text{]} - \text{curr)) +} \\ & \text{f[2] * (Clip3(} -\text{c[2], c[2], recPicture[} h_{x,\, y + y1} \text{][} v_{y + y1} \text{]} - \text{curr) +} \\ & \quad \text{Clip3(} -\text{c[2], c[2], recPicture[} h_{x,\, y - y1} \text{][} v_{y - y1} \text{]} - \text{curr)) +} \\ & \text{f[3] * (Clip3(} -\text{c[3], c[3], recPicture[} h_{x - 1,\, y + y1} \text{][} v_{y + y1} \text{]} - \text{curr) +} \\ & \quad \text{Clip3(} -\text{c[3], c[3], recPicture[} h_{x + 1,\, y - y1} \text{][} v_{y - y1} \text{]} - \text{curr)) +} \\ & \text{f[4] * (Clip3(} -\text{c[4], c[4], recPicture[} h_{x + 2,\, y} \text{][} v_{y} \text{]} - \text{curr) +} \\ & \quad \text{Clip3(} -\text{c[4], c[4], recPicture[} h_{x - 2,\, y} \text{][} v_{y} \text{]} - \text{curr)) +} \\ & \text{f[5] * (Clip3(} -\text{c[5], c[5], recPicture[} h_{x + 1,\, y} \text{][} v_{y} \text{]} - \text{curr) +} \\ & \quad \text{Clip3(} -\text{c[5], c[5], recPicture[} h_{x - 1,\, y} \text{][} v_{y} \text{]} - \text{curr))}\end{aligned} \tag{1484}$$

$$\text{sum} = \text{curr} + \text{((sum + (1 << (alfShiftC} - \text{1))) >> alfShiftC)} \tag{1485}$$

– The modified filtered reconstructed chroma picture sample alfPicture[xCtbC + x][yCtbC + y] is derived as follows:

$$\text{alfPicture[xCtbC + x][yCtbC + y] = Clip3(0, (1 << BitDepth)} - \text{1, sum)} \tag{1486}$$

Table 46 – Specification of y1, y2 and alfShiftC according to the vertical chroma sample position y and applyAlfLineBufBoundary

Condition	alfShiftC	y1	y2
(y = = ctbHeightC − 2 || y = = ctbHeightC − 3) && (applyAlfLineBufBoundary = = 1)	10	0	0
(y = = ctbHeightC − 1 || y = = ctbHeightC − 4) && (applyAlfLineBufBoundary = = 1)	7	1	1
Otherwise	7	1	2

8.8.5.5 ALF boundary position derivation process

Inputs of this process are:

– a luma location (xCtb, yCtb) specifying the top-left sample of the current luma coding tree block relative to the top-left sample of the current picture,

– a luma location (x, y) specifying the current sample relative to the top-left sample of the current luma coding tree block,

– a variable vbOffset specifying the offset for ALF virtual boundary.

Output of this process are:

– the left vertical boundary position clipLeftPos,

– the right vertical boundary position clipRightPos,

– the above horizontal boundary position clipTopPos,

– the below horizontal boundary position clipBottomPos,

– the top-left boundary flag clipTopLeftFlag,

– the bottom-right boundary flag clipBotRightFlag.

The variables clipLeftPos, clipRightPos, clipTopPos and clipBottomPos are set equal to −128.

The variables clipTopLeftFlag and clipBotRightFlag are both set equal to 0.

The variable clipTopPos is modified as follows:

– If y − (CtbSizeY − vbOffset) is greater than or equal to 0, the variable clipTopPos is set equal to yCtb + CtbSizeY − vbOffset.

– Otherwise, if VirtualBoundariesPresentFlag is equal to 1, and yCtb + y − VirtualBoundaryPosY[n] is greater than or equal to 0 and less than 3 for any n = 0..NumHorVirtualBoundaries − 1, the following applies:

$$\text{clipTopPos} = \text{VirtualBoundaryPosY}[\ n\ ] \tag{1487}$$

– Otherwise, if y is less than 3 and one or more of the following conditions are true, the variable clipTopPos is set equal to yCtb:

  – The top boundary of the current coding tree block is the top boundary of the tile, and pps_loop_filter_across_tiles_enabled_flag is equal to 0.

  – The top boundary of the current coding tree block is the top boundary of the slice, and pps_loop_filter_across_slices_enabled_flag is equal to 0.

  – The top boundary of the current coding tree block is the top boundary of the subpicture, and sps_loop_filter_across_subpic_enabled_flag[CurrSubpicIdx] is equal to 0.

The variable clipBottomPos is modified as follows:

– If VirtualBoundariesPresentFlag is equal to 1, VirtualBoundaryPosY[n] is not equal to any of pps_pic_height_in_luma_samples − 1 and 0, and VirtualBoundaryPosY[n] − yCtb − y is greater than 0 and less than 5 for any n = 0..NumHorVirtualBoundaries − 1, the following applies:

$$\text{clipBottomPos} = \text{VirtualBoundaryPosY}[\ n\ ] \tag{1488}$$

– Otherwise, if CtbSizeY − vbOffset − y is greater than 0 and is less than 5, the variable clipBottomPos is set equal to yCtb + CtbSizeY − vbOffset.

– Otherwise, if CtbSizeY − y is less than 5, and one or more of the following conditions are true, the variable clipBottomPos is set equal to yCtb + CtbSizeY:

  – The bottom boundary of the current coding tree block is the bottom boundary of the tile, and pps_loop_filter_across_tiles_enabled_flag is equal to 0.

  – The bottom boundary of the current coding tree block is the bottom boundary of the slice, and pps_loop_filter_across_slices_enabled_flag is equal to 0.

  – The bottom boundary of the current coding tree block is the bottom boundary of the subpicture, and sps_loop_filter_across_subpic_enabled_flag[CurrSubpicIdx] is equal to 0.

The variable clipLeftPos is modified as follows:

– If VirtualBoundariesPresentFlag is equal to 1, and xCtb + x − VirtualBoundaryPosX[n] is greater than or equal to 0 and less than 3 for any n = 0..NumVerVirtualBoundaries − 1, the following applies:

$$\text{clipLeftPos} = \text{VirtualBoundaryPosX}[\ n\ ] \tag{1489}$$

– Otherwise, if x is less than 3, and one or more of the following conditions are true, the variable clipLeftPos is set equal to xCtb:

  – The left boundary of the current coding tree block is the left boundary of the tile, and pps_loop_filter_across_tiles_enabled_flag is equal to 0.

  – The left boundary of the current coding tree block is the left boundary of the slice, and pps_loop_filter_across_slices_enabled_flag is equal to 0.

  – The left boundary of the current coding tree block is the left boundary of the subpicture, and sps_loop_filter_across_subpic_enabled_flag[CurrSubpicIdx] is equal to 0.

The variable clipRightPos is modified as follows:

– If VirtualBoundariesPresentFlag is equal to 1, and VirtualBoundaryPosX[n]− xCtb − x is greater than 0 and less than 5 for any n = 0..NumVerVirtualBoundaries − 1, the following applies:

$$\text{clipRightPos} = \text{VirtualBoundaryPosX}[\ n\ ] \tag{1490}$$

– Otherwise, if CtbSizeY − x is less than 5, and one or more of the following conditions are true, the variable clipRightPos is set equal to xCtb + CtbSizeY:

– The right boundary of the current coding tree block is the right boundary of the tile, and loop_filter_across_tiless_enabled_flag is equal to 0.

– The right boundary of the current coding tree block is the right boundary of the slice, and pps_loop_filter_across_slices_enabled_flag is equal to 0.

– The right boundary of the current coding tree block is the right boundary of the subpicture, and sps_loop_filter_across_subpic_enabled_flag[CurrSubpicIdx] is equal to 0.

The variable clipTopLeftFlag and clipBotRightFlag are modified as follows:

– If the coding tree block covering the luma position (xCtb, yCtb) and the coding tree block covering the luma position (xCtb − CtbSizeY, yCtb − CtbSizeY) belong to different slices, and pps_loop_filter_across_slices_enabled_flag is equal to 0, clipTopLeftFlag is set equal to 1.

– If the coding tree block covering the luma position (xCtb, yCtb) and the coding tree block covering the luma position (xCtb + CtbSizeY, yCtb + CtbSizeY) belong to different slices, and pps_loop_filter_across_slices_enabled_flag is equal to 0, clipBotRightFlag is set equal to 1.

8.8.5.6 ALF sample padding process

Inputs of this process are:

– a luma location (xCtb, yCtb) specifying the top-left sample of the current luma coding tree block relative to the top-left sample of the current picture,

– a location (x, y) specifying the neighbouring sample relative to the top-left sample of the current picture,

– a flag isChroma specifiying whether the colour componenet is chroma component or not,

– the left vertical boundary position clipLeftPos,

– the right vertical boundary position clipRightPos,

– the above horizontal boundary position clipTopPos,

– the below horizontal boundary position clipBottomPos,

– the top-left boundary flag clipTopLeftFlag,

– the bottom-right boundary flag clipBotRightFlag.

Outputs of this process are:

– modified location (x, y) specifying the neighbouring sample relative to the top-left sample of the current picture.

The variables picWidth, picHeight, xCtbCur, yCtbCur, CtbSizeHor, CtbSizeVer, topBry, botBry, leftBry and rightBry are derived as follows:

picWidth = isChroma ? pps_pic_width_in_luma_samples / SubWidthC :
 pps_pic_width_in_luma_samples (1491)

picHeight = isChroma ? pps_pic_height_in_luma_samples / SubHeightC :
 pps_pic_height_in_luma_samples (1492)

xCtbCur = isChroma ? xCtb / SubWidthC : xCtb (1493)

yCtbCur = isChroma ? yCtb / SubHeightC : yCtb (1494)

CtbSizeHor = isChroma ? CtbSizeY / SubWidthC : CtbSizeY (1495)

CtbSizeVer = isChroma ? CtbSizeY / SubHeightC : CtbSizeY (1496)

topBryPos = isChroma ? clipTopPos / SubHeightC : clipTopPos (1497)

botBryPos = isChroma ? clipBottomPos / SubHeightC : clipBottomPos (1498)

leftBryPos = isChroma ? clipLeftPos / SubWidthC : clipLeftPos (1499)

rightBryPos = isChroma ? clipRightPos / SubWidthC : clipRightPos (1500)

The variables x and y are modified as follows:

– When topBryPos is not less than 0, the following applies:

$$y = \text{Clip3}(\ \text{topBryPos},\ \text{picHeight} - 1,\ y\ ) \quad (1501)$$

– When botBryPos is not less than 0, the following applies:

$$y = \text{Clip3}(\ 0,\ \text{botBryPos} - 1,\ y\ ) \quad (1502)$$

– When leftBryPos is not less than 0, the following applies:

$$x = \text{Clip3}(\ \text{leftBryPos},\ \text{picWidth} - 1,\ x\ ) \quad (1503)$$

– When rightBryPos is not less than 0, the following applies:

$$x = \text{Clip3}(\ 0,\ \text{rightBryPos} - 1,\ x\ ) \quad (1504)$$

– When all of the following conditions are true, (x, y) is set equal to (xCtbCur, y):
  – clipTopLeftFlag is equal to true;
  – topBryPos is less than 0 and leftBryPos is less than 0;
  – x is less than xCtbCur and y is less than yCtbCur.
– When all of the following conditions are true, (x, y) is set equal to (xCtbCur + CtbSizeHor − 1, y):
  – clipBotRightFlag is equal to true;
  – botBryPos is less than 0 and rightBryPos is less than 0;
  – x is greater than xCtbCur + CtbSizeHor − 1 and y is greater than yCtbCur + CtbSizeVer − 1.

8.8.5.7 Cross-component filtering process

Inputs of this process are:

– a reconstructed luma picture sample array recPicture_L prior to the luma adaptive loop filtering process,
– a filtered reconstructed chroma picture sample array alfPicture_C,
– a chroma location (xCtbC, yCtbC) specifying the top-left sample of the current chroma coding tree block relative to the top-left sample of the current picture,
– a CTB width ccAlfWidth in chroma samples,
– a CTB height ccAlfHeight in chroma samples,
– cross-component filter coefficients CcAlfCoeff[j], with j = 0..6.

Output of this process is the modified filtered reconstructed chroma picture sample array ccAlfPicture.

For the derivation of the filtered reconstructed chroma samples ccAlfPicture[xCtbC + x][yCtbC + y], each reconstructed chroma sample inside the current chroma block of samples alfPicture_C[xCtbC + x][yCtbC + y] with x = 0..ccAlfWidth − 1, y = 0..ccAlfHeight − 1, is filtered as follows:

– The luma location (xL, yL) corresponding to the current chroma sample at chroma location (xCtbC + x, yCtbC + y) is set equal to ((xCtbC + x) * SubWidthC, (yCtbC + y) * SubHeightC).
– The variables clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag are derived by invoking the ALF boundary position derivation process as specified in clause 8.8.5.5 with (xCtbC * SubWidthC, yCtbC * SubHeightC), (x * SubWidthC, y *SubHeightC) and the variable vbOffset set equal to 4 as inputs.
– The luma locations ($h_{x+i,\,y+j}$, v_{y+j}) with i = −1..1, j = −1..2 inside the array recPicture_L are derived as follows:

$$h_{x+i,\,y+j} = \text{Clip3}(\ 0,\ \text{pps\_pic\_width\_in\_luma\_samples} - 1,\ xL + i\ ) \quad (1505)$$

$$v_{y+j} = \text{Clip3}(\ 0,\ \text{pps\_pic\_height\_in\_luma\_samples} - 1,\ yL + j\ ) \quad (1506)$$

– The location ($h_{x+i, y+j}$, v_{y+j}) is modified by invoking the ALF sample padding process as specified in clause 8.8.5.6 with (xCtbC * SubWidthC, yCtbC * SubHeightC), ($h_{x+i, y+j}$, v_{y+j}), the variable isChroma set equal to 0, clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag as input.

– The variable applyAlfLineBufBoundary is derived as follows:

  – If the bottom boundary of the current coding tree block is the bottom boundary of current picture and pps_pic_height_in_luma_samples − yCtbC * SubHeightC is less than or equal to CtbSizeY − 4, applyAlfLineBufBoundary is set equal to 0.

  – Otherwise, applyAlfLineBufBoundary is set equal to 1.

– The vertical sample position offsets yP1 and yP2 are specified in Table 47 according to the vertical luma sample position (y * SubHeightC) and applyAlfLineBufBoundary.

– The variable curr is derived as follows:

$$\text{curr} = \text{alfPicture}_C[\ \text{xCtbC} + \text{x}\ ][\ \text{yCtbC} + \text{y}\ ] \tag{1507}$$

– The array of cross-component filter coefficients f[j] is derived as follows with j = 0..6:

$$\text{f}[\ \text{j}\ ] = \text{CcAlfCoeff}[\ \text{j}\ ] \tag{1508}$$

– The variable sum is derived as follows:

$$\begin{aligned}\text{sum} = {} & \text{f}[\,0\,] * (\,\text{recPicture}_L[\,h_{x,\,y-yP1}\,][\,v_{y-yP1}\,] - \text{recPicture}_L[\,h_{x,\,y}\,][\,v_y\,]\,) + {} \\ & \text{f}[\,1\,] * (\,\text{recPicture}_L[\,h_{x-1,\,y}\,][\,v_y\,] - \text{recPicture}_L[\,h_{x,\,y}\,][\,v_y\,]\,) + {} \\ & \text{f}[\,2\,] * (\,\text{recPicture}_L[\,h_{x+1,\,y}\,][\,v_y\,] - \text{recPicture}_L[\,h_{x,\,y}\,][\,v_y\,]\,) + {} \\ & \text{f}[\,3\,] * (\,\text{recPicture}_L[\,h_{x-1,\,y+yP1}\,][\,v_{y+yP1}\,] - \text{recPicture}_L[\,h_{x,\,y}\,][\,v_y\,]\,) + {} \\ & \text{f}[\,4\,] * (\,\text{recPicture}_L[\,h_{x,\,y+yP1}\,][\,v_{y+yP1}\,] - \text{recPicture}_L[\,h_{x,\,y}\,][\,v_y\,]\,) + {} \\ & \text{f}[\,5\,] * (\,\text{recPicture}_L[\,h_{x+1,\,y+yP1}\,][\,v_{y+yP1}\,] - \text{recPicture}_L[\,h_{x,\,y}\,][\,v_y\,]\,) + {} \\ & \text{f}[\,6\,] * (\,\text{recPicture}_L[\,h_{x,\,y+yP2}\,][\,v_{y+yP2}\,] - \text{recPicture}_L[\,h_{x,\,y}\,][\,v_y\,]\,)\end{aligned} \tag{1509}$$

$$\text{scaledSum} = \text{Clip3}(\,-(\,1 \ll (\,\text{BitDepth} - 1\,)\,),\,(\,1 \ll (\,\text{BitDepth} - 1\,)\,) - 1,\,(\,\text{sum} + 64\,) \gg 7\,) \tag{1510}$$

$$\begin{aligned}\text{sum} = {} & (\,\text{SubHeightC} == 1\ \&\&\ (\,\text{y} == \text{CtbSizeY} - 3\ ||\ \text{y} == \text{CtbSizeY} - 4\,)\,)\ ? \\ & \text{curr} : \text{curr} + \text{scaledSum}\end{aligned} \tag{1511}$$

– The modified filtered reconstructed chroma picture sample ccAlfPicture[xCtbC + x][yCtbC + y] is derived as follows:

$$\text{ccAlfPicture}[\ \text{xCtbC} + \text{x}\ ][\ \text{yCtbC} + \text{y}\ ] = \text{Clip3}(\,0,\,(\,1 \ll \text{BitDepth}\,) - 1,\,\text{sum}\,) \tag{1512}$$

Table 47 – Specification of yP1 and yP2 according to the vertical luma sample position (y * SubHeightC) and applyAlfLineBufBoundary

Condition	yP1	yP2
(y * SubHeightC = = CtbSizeY − 5 || y * SubHeightC = = CtbSizeY − 4) && applyAlfLineBufBoundary = = 1	0	0
(y * SubHeightC = = CtbSizeY − 6 || y * SubHeightC = = CtbSizeY − 3) && applyAlfLineBufBoundary = = 1	1	1
Otherwise	1	2

9 Parsing process

9.1 General

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 is equal to ue(v), se(v), or ae(v) (see clause 9.3).

9.2 Parsing process for k-th order Exp-Golomb codes

9.2.1 General

This process is invoked when the descriptor of a syntax element in the syntax tables is equal to ue(v) or se(v).

Inputs to this process are bits from the RBSP.

Outputs of this process are syntax element values.

Syntax elements coded as ue(v) or se(v) are Exp-Golomb-coded with order k equal to 0. 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++ )                                    (1513)
    b = read_bits( 1 )
```

The variable codeNum is then assigned as follows:

$$\text{codeNum} = (2^{\text{leadingZeroBits}} - 1) * 2^{k} + \text{read\_bits}(\text{leadingZeroBits} + k) \tag{1514}$$

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 48 illustrates the structure of the 0-th order Exp-Golomb code by separating the bit string into "prefix" and "suffix" bits. The "prefix" bits are those bits that are parsed for the computation of leadingZeroBits, and are shown as either 0 or 1 in the bit string column of Table 48. The "suffix" bits are those bits that are parsed in the computation of codeNum and are shown as x_i in Table 48, with i in the range of 0 to leadingZeroBits − 1, inclusive. Each x_i is equal to either 0 or 1.

Table 48 – 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 49 illustrates explicitly the assignment of bit strings to codeNum values.

Table 49 – 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 (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 clause 9.2.2 with codeNum as input.

9.2.2 Mapping process for signed Exp-Golomb codes

Input to this process is codeNum as specified in clause 9.2.1.

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 50 provides the assignment rule.

Table 50 – 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)

9.3 CABAC parsing process for slice data

9.3.1 General

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.

The initialization process as specified in clause 9.3.2 is invoked when starting the parsing of the CTU syntax specified in clause 7.3.11.2 and one or more of the following conditions are true:

– The CTU is the first CTU in a slice.

– The CTU is the first CTU in a tile.

– The value of sps_entropy_coding_sync_enabled_flag is equal to 1 and the CTU is the first CTU in a CTU row of a tile.

The parsing of syntax elements proceeds as follows:

For each requested value of a syntax element a binarization is derived as specified in clause 9.3.3.

The binarization for the syntax element and the sequence of parsed bins determines the decoding process flow as described in clause 9.3.4.

The storage process for context variables is applied as follows:

– When ending the parsing of the CTU syntax in clause 7.3.11.2, sps_entropy_coding_sync_enabled_flag is equal to 1, and CtbAddrX is equal to CtbToTileColBd[CtbAddrX], the storage process for context variables as specified in clause 9.3.2.3 is invoked with TableStateIdx0Wpp and TableStateIdx1Wpp as outputs.

When sps_palette_enabled_flag is equal to 1, the storage process for palette predictor is applied as follows:

– When ending the parsing of the CTU syntax in clause 7.3.11.2 and the decoding process of the last CU in the CTU in clause 8.1.2, sps_entropy_coding_sync_enabled_flag is equal to 1 and CtbAddrX is equal to CtbToTileColBd[CtbAddrX], the storage process for palette predictor as speficied in clause 9.3.2.6 is invoked.

The whole CABAC parsing process for a syntax element synEl is illustrated in Figure 11.

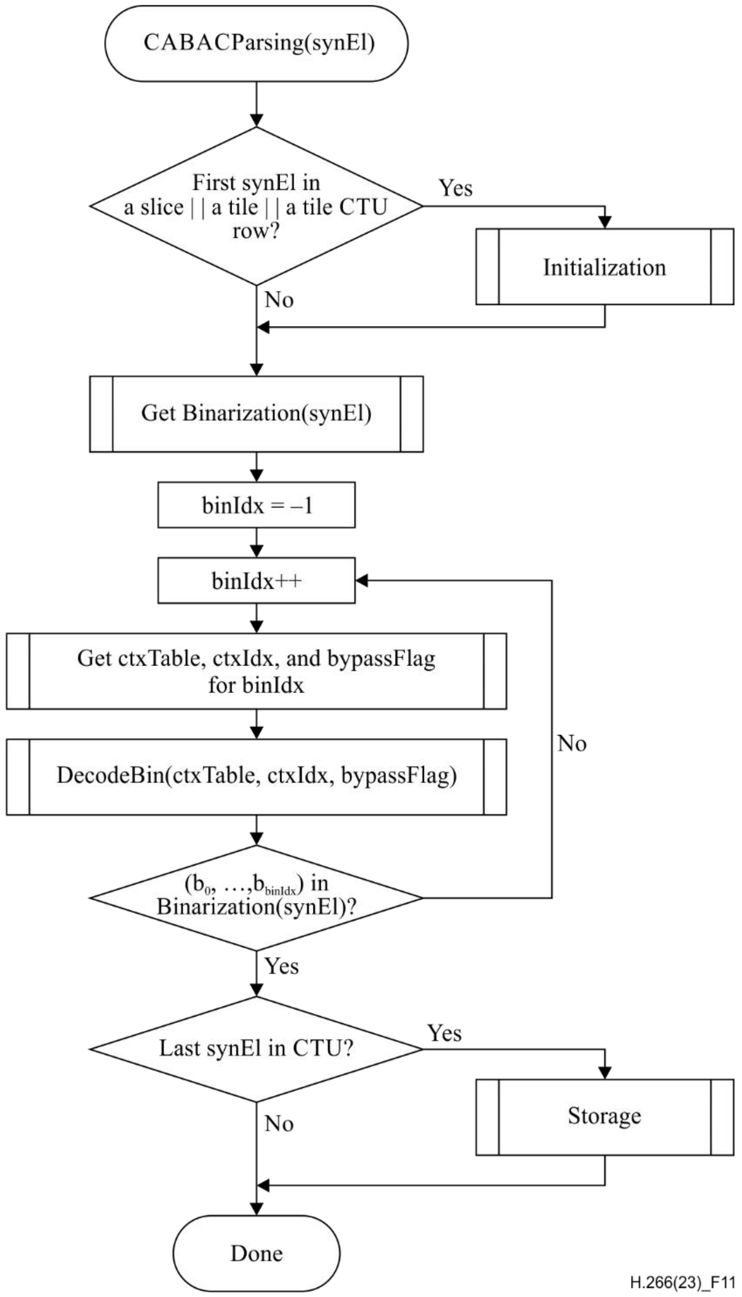

Figure 11 – Flowchart of CABAC parsing process for a syntax element synEl (informative)

9.3.2 Initialization process

9.3.2.1 General

Outputs of this process are initialized CABAC internal variables.

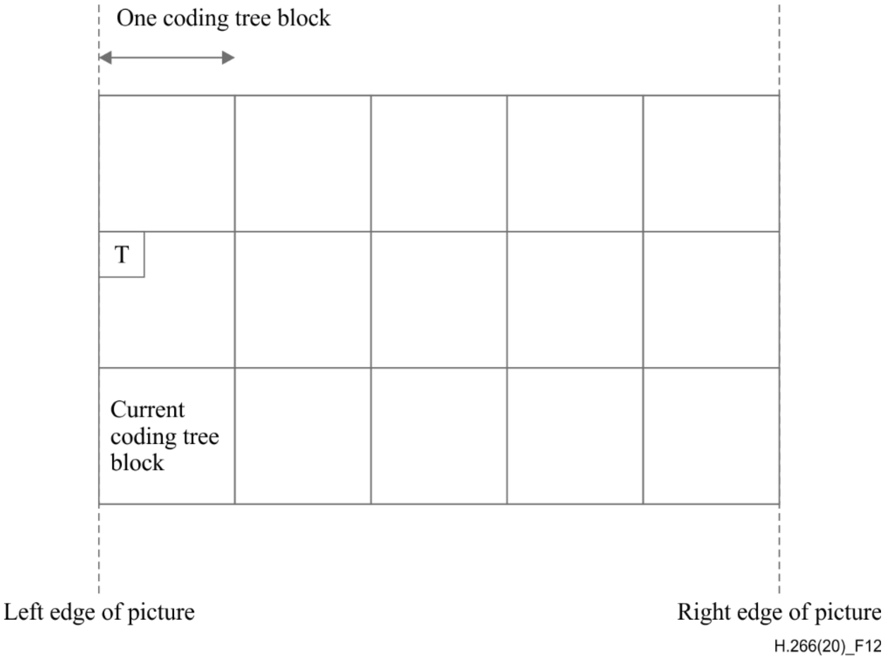

Figure 12 – Spatial neighbour T that is used to invoke the CTB availability derivation process relative to the current CTB (informative)

The context variables of the arithmetic decoding engine and the arrays PredictorPaletteSize and StatCoeff are initialized as follows:

– The array StatCoeff[i], with i = 0..2, is initialized as follows:

StatCoeff[i] = sps_persistent_rice_adaptation_enabled_flag ? 2 * Floor(Log2(BitDepth − 10) : 0 (1515)

– If the CTU is the first CTU in a slice or tile, the initialization process for context variables is invoked as specified in clause 9.3.2.2 and the array PredictorPaletteSize[chType], with chType = 0, 1, is initialized to 0.

– Otherwise, when sps_entropy_coding_sync_enabled_flag is equal to 1 and CtbAddrX is equal to CtbToTileColBd[CtbAddrX], the following applies:

  – The location (xNbT, yNbT) of the top-left luma sample of the spatial neighbouring block T (Figure 12) is derived using the location (x0, y0) of the top-left luma sample of the current CTB as follows:

  (xNbT, yNbT) = (x0, y0 − CtbSizeY) (1516)

  – The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location (xCurr, yCurr) set equal to (x0, y0), the neighbouring location (xNbY, yNbY) set equal to (xNbT, yNbT), checkPredModeY set equal to FALSE, and cIdx set equal to 0 as inputs, and the output is assigned to availableFlagT.

  – The synchronization processes for context variables and palette predictor are invoked as follows:

    – If availableFlagT is equal to 1, the following applies:

      – The synchronization process for context variables as specified in clause 9.3.2.4 is invoked with TableStateIdx0Wpp and TableStateIdx1Wpp as inputs.

      – When sps_palette_enabled_flag is equal to 1, the synchronization process for palette predictor as specified in clause 9.3.2.7 is invoked.

    – Otherwise, the initialization process for context variables is invoked as specified in clause 9.3.2.2 and the array PredictorPaletteSize[chType], with chType = 0, 1, is initialized to 0.

The decoding engine registers ivlCurrRange and ivlOffset both in 16 bit register precision are initialized by invoking the initialization process for the arithmetic decoding engine as specified in clause 9.3.2.5.

The whole initialization process for a syntax element synEl is illustrated in Figure 13.

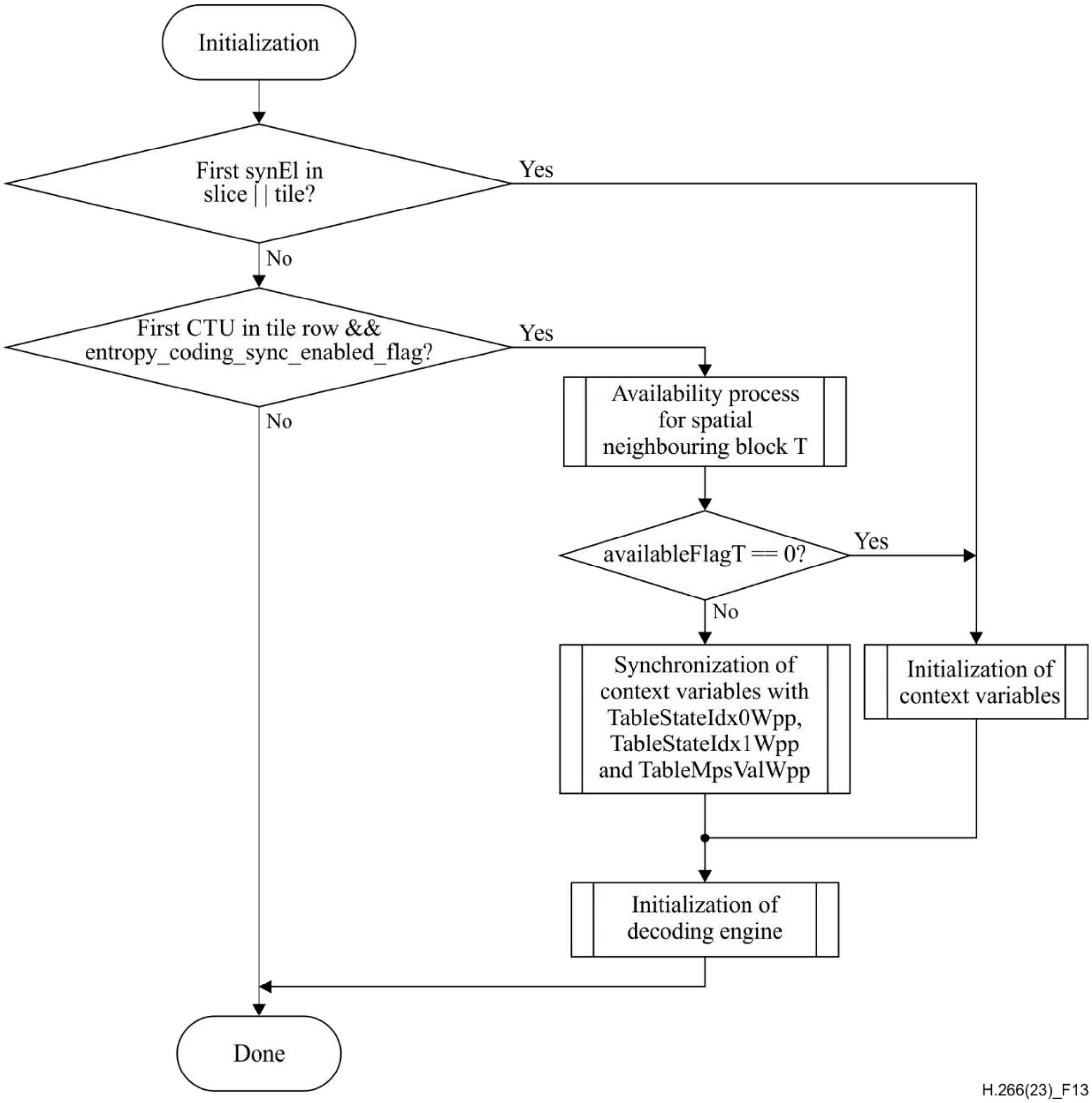

Figure 13 – Flowchart of CABAC initialization process (informative)

9.3.2.2 Initialization process for context variables

Outputs of this process are the initialized CABAC context variables indexed by ctxTable and ctxIdx.

For each context variable, the two variables pStateIdx0 and pStateIdx1 are initialized as follows:

– Table 52 to Table 126 contain the values of the 6 bit variable initValue used in the initialization of context variables that are assigned to all syntax elements in clauses 7.3.11.1 through 7.3.11.11, except end_of_slice_one_bit, end_of_tile_one_bit, and end_of_subset_one_bit.

– From the 6 bit table entry initValue, the two 3 bit variables slopeIdx and offsetIdx are derived as follows:

slopeIdx = initValue >> 3
offsetIdx = initValue & 7 (1517)

– The variables m and n, used in the initialization of context variables, are derived from slopeIdx and offsetIdx as follows:

m = slopeIdx − 4
n = (offsetIdx * 18) + 1 (1518)

– The two values assigned to pStateIdx0 and pStateIdx1 for the initialization are derived from $SliceQp_Y$, which is derived in Equation 140. Given the variables m and n, the initialization is specified as follows:

$$preCtxState = Clip3(1, 127, ((m * (Clip3(0, 63, SliceQp_Y) - 16)) >> 1) + n) \quad (1519)$$

– The two values assigned to pStateIdx0 and pStateIdx1 for the initialization are derived as follows:

$$\begin{array}{l} pStateIdx0 = preCtxState << 3 \\ pStateIdx1 = preCtxState << 7 \end{array} \quad (1520)$$

NOTE – The variables pStateIdx0 and pStateIdx1 correspond to the probability state indices as further described in clause 9.3.4.3.

Table 51 lists the range of ctxIdx values for which initialization is needed for each of the three initialization types, specified by the variable initType. It also lists the table number that includes the values of initValue needed for the initialization for each value of ctxIdx. For P and B slice types, the derivation of initType depends on the value of the sh_cabac_init_flag syntax element. The variable initType is derived as follows:

```
if( sh_slice_type  = =  I )
    initType = 0
else if( sh_slice_type  = =  P )
    initType = sh_cabac_init_flag ? 2 : 1                                    (1521)
else
    initType = sh_cabac_init_flag ? 1 : 2
```

Table 51 – Association of ctxIdx and syntax elements for each initializationType in the initialization process

Syntax structure	Syntax element	ctxTable	initType		
			0	1	2
coding_tree_unit( )	alf_ctb_flag[ ][ ][ ]	Table 52	0..8	9..17	18..26
	alf_use_aps_flag	Table 53	0	1	2
	alf_ctb_filter_alt_idx[ ][ ][ ]	Table 56	0..1	2..3	4..5
	alf_ctb_cc_cb_idc[ ][ ]	Table 54	0..2	3..5	6..8
	alf_ctb_cc_cr_idc[ ][ ]	Table 55	0..2	3..5	6..8
sao( )	sao_merge_left_flag sao_merge_up_flag	Table 57	0	1	2
	sao_type_idx_luma sao_type_idx_chroma	Table 58	0	1	2
coding_tree( )	split_cu_flag	Table 59	0..8	9..17	18..26
	split_qt_flag	Table 60	0..5	6..11	12..17
	mtt_split_cu_vertical_flag	Table 61	0..4	5..9	10..14
	mtt_split_cu_binary_flag	Table 62	0..3	4..7	8..11
	non_inter_flag	Table 63		0..1	2..3

Table 51 – Association of ctxIdx and syntax elements for each initializationType in the initialization process

Syntax structure	Syntax element	ctxTable	initType		
			0	**1**	**2**
coding_unit( )	cu_skip_flag[ ][ ]	Table 64	0..2	3..5	6..8
	pred_mode_ibc_flag	Table 65	0..2	3..5	6..8
	pred_mode_flag	Table 66		0..1	2..3
	pred_mode_plt_flag	Table 67	0	1	2
	cu_act_enabled_flag	Table 68	0	1	2
	intra_bdpcm_luma_flag	Table 69	0	1	2
	intra_bdpcm_luma_dir_flag	Table 70	0	1	2
	intra_mip_flag	Table 71	0..3	4..7	8..11
	intra_luma_ref_idx	Table 72	0..1	2..3	4..5
	intra_subpartitions_mode_flag	Table 73	0	1	2
	intra_subpartitions_split_flag	Table 74	0	1	2
	intra_luma_mpm_flag[ ][ ]	Table 75	0	1	2
	intra_luma_not_planar_flag[ ][ ]	Table 76	0..1	2..3	4..5
	intra_bdpcm_chroma_flag	Table 77	0	1	2
	intra_bdpcm_chroma_dir_flag	Table 78	0	1	2
	cclm_mode_flag	Table 79	0	1	2
	cclm_mode_idx	Table 80	0	1	2
	intra_chroma_pred_mode	Table 81	0	1	2
	general_merge_flag[ ][ ]	Table 82	0	1	2
	inter_pred_idc[ ][ ]	Table 83		0..5	6..11
	inter_affine_flag[ ][ ]	Table 84		0..2	3..5
	cu_affine_type_flag[ ][ ]	Table 85		0	1
	sym_mvd_flag[ ][ ]	Table 86		0	1
	ref_idx_l0[ ][ ], ref_idx_l1[ ][ ]	Table 87		0..1	2..3
	mvp_l0_flag[ ][ ], mvp_l1_flag[ ][ ]	Table 88	0	1	2
	amvr_flag[ ][ ]	Table 89		0..1	2..3
	amvr_precision_idx[ ][ ]	Table 90	0..2	3..5	6..8
	bcw_idx[ ][ ]	Table 91		0	1
	cu_coded_flag	Table 92	0	1	2
	cu_sbt_flag	Table 93		0..1	2..3
	cu_sbt_quad_flag	Table 94		0	1
	cu_sbt_horizontal_flag	Table 95		0..2	3..5

Table 51 – Association of ctxIdx and syntax elements for each initializationType in the initialization process

Syntax structure	Syntax element	ctxTable	initType		
			0	1	2
	cu_sbt_pos_flag	Table 96		0	1
	lfnst_idx	Table 97	0..2	3..5	6..8
	mts_idx	Table 98	0..3	4..7	8..11
palette_coding( )	copy_above_palette_indices_flag	Table 99	0	1	2
	palette_transpose_flag	Table 100	0	1	2
	run_copy_flag	Table 101	0..7	8..15	16..23
merge_data( )	regular_merge_flag[ ][ ]	Table 102		0..1	2..3
	mmvd_merge_flag[ ][ ]	Table 103		0	1
	mmvd_cand_flag[ ][ ]	Table 104		0	1
	mmvd_distance_idx[ ][ ]	Table 105		0	1
	ciip_flag[ ][ ]	Table 106		0	1
	merge_subblock_flag[ ][ ]	Table 107		0..2	3..5
	merge_subblock_idx[ ][ ]	Table 108		0	1
	merge_idx[ ][ ] merge_gpm_idx0[ ][ ] merge_gpm_idx1[ ][ ]	Table 109	0	1	2
mvd_coding( )	abs_mvd_greater0_flag[ ]	Table 110	0	1	2
	abs_mvd_greater1_flag[ ]	Table 111	0	1	2

Table 51 – Association of ctxIdx and syntax elements for each initializationType in the initialization process

Syntax structure	Syntax element	ctxTable	initType		
			0	1	2
transform_unit( )	tu_y_coded_flag[ ][ ]	Table 112	0..3	4..7	8..11
	tu_cb_coded_flag[ ][ ]	Table 113	0..1	2..3	4..5
	tu_cr_coded_flag[ ][ ]	Table 114	0..2	3..5	6..8
	cu_qp_delta_abs	Table 115	0..1	2..3	4..5
	cu_chroma_qp_offset_flag	Table 116	0	1	2
	cu_chroma_qp_offset_idx	Table 117	0	1	2
	transform_skip_flag[ ][ ][ ]	Table 118	0..1	2..3	4..5
	tu_joint_cbcr_residual_flag[ ][ ]	Table 119	0..2	3..5	6..8
residual_coding( )	last_sig_coeff_x_prefix	Table 120	0..22	23..45	46..68
	last_sig_coeff_y_prefix	Table 121	0..22	23..45	46..68
	sb_coded_flag[ ][ ]	Table 122	0..6	7..13	14..20
	sig_coeff_flag[ ][ ]	Table 123	0..62	63..125	126..188
	par_level_flag[ ]	Table 124	0..32	33..65	66..98
	abs_level_gtx_flag[ ][ ]	Table 125	0..71	72..143	144..215
	coeff_sign_flag[ ]	Table 126	0..5	6..11	12..17

Table 52 – Specification of initValue and shiftIdx for ctxIdx of alf_ctb_flag

Initialization variable	ctxIdx of alf_ctb_flag															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
initValue	62	39	39	54	39	39	31	39	39	13	23	46	4	61	54	19
shiftIdx	0	0	0	4	0	0	1	0	0	0	0	0	4	0	0	1
	16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**					
initValue	46	54	33	52	46	25	61	54	25	61	54					
shiftIdx	0	0	0	0	0	4	0	0	1	0	0					

Table 53 – Specification of initValue and shiftIdx for ctxIdx of alf_use_aps_flag

Initialization variable	ctxIdx of alf_use_aps_flag		
	0	**1**	**2**
initValue	46	46	46
shiftIdx	0	0	0

Table 54 – Specification of initValue and shiftIdx for ctxIdx of alf_ctb_cc_cb_idc

Initialization variable	ctxIdx of alf_ctb_cc_cb_idc								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
initValue	18	30	31	18	21	38	25	35	38
shiftIdx	4	1	4	4	1	4	4	1	4

Table 55 – Specification of initValue and shiftIdx for ctxIdx of alf_ctb_cc_cr_idc

Initialization variable	ctxIdx of alf_ctb_cc_cr_idc								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
initValue	18	30	31	18	21	38	25	28	38
shiftIdx	4	1	4	4	1	4	4	1	4

Table 56 – Specification of initValue and shiftIdx for ctxIdx of alf_ctb_filter_alt_idx

Initialization variable	ctxIdx of alf_ctb_filter_alt_idx					
	0	**1**	**2**	**3**	**4**	**5**
initValue	11	11	20	12	11	26
shiftIdx	0	0	0	0	0	0

Table 57 – Specification of initValue and shiftIdx for ctxIdx of sao_merge_left_flag and sao_merge_up_flag

Initialization variable	ctxIdx of sao_merge_left_flag and sao_merge_up_flag		
	0	**1**	**2**
initValue	60	60	2
shiftIdx	0	0	0

Table 58 – Specification of initValue and shiftIdx for ctxIdx of sao_type_idx_luma and sao_type_idx_chroma

Initialization variable	ctxIdx of sao_type_idx_luma and sao_type_idx_chroma		
	0	**1**	**2**
initValue	13	5	2
shiftIdx	4	4	4

Table 59 – Specification of initValue and shiftIdx for ctxIdx of split_cu_flag

Initialization variable	ctxIdx of split_cu_flag															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
initValue	19	28	38	27	29	38	20	30	31	11	35	53	12	6	30	13
shiftIdx	12	13	8	8	13	12	5	9	9	12	13	8	8	13	12	5
	16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**					
initValue	15	31	18	27	15	18	28	45	26	7	23					
shiftIdx	9	9	12	13	8	8	13	12	5	9	9					

Table 60 – Specification of initValue and shiftIdx for ctxIdx of split_qt_flag

Initialization variable	ctxIdx of split_qt_flag																	
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
initValue	27	6	15	25	19	37	20	14	23	18	19	6	26	36	38	18	34	21
shiftIdx	0	8	8	12	12	8	0	8	8	12	12	8	0	8	8	12	12	8

Table 61 – Specification of initValue and shiftIdx for ctxIdx of mtt_split_cu_vertical_flag

Initialization variable	ctxIdx of mtt_split_cu_vertical_flag														
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
initValue	43	42	29	27	44	43	35	37	34	52	43	42	37	42	44
shiftIdx	9	8	9	8	5	9	8	9	8	5	9	8	9	8	5

Table 62 – Specification of initValue and shiftIdx for ctxIdx of mtt_split_cu_binary_flag

Initialization variable	ctxIdx of mtt_split_cu_binary_flag											
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
initValue	36	45	36	45	43	37	21	22	28	29	28	29
shiftIdx	12	13	12	13	12	13	12	13	12	13	12	13

Table 63 – Specification of initValue and shiftIdx for ctxIdx of non_inter_flag

Initialization variable	ctxIdx of non_inter_flag			
	0	**1**	**2**	**3**
initValue	25	12	25	20
shiftIdx	1	0	1	0

Table 64 – Specification of initValue and shiftIdx for ctxIdx of cu_skip_flag

Initialization variable	ctxIdx of cu_skip_flag								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
initValue	0	26	28	57	59	45	57	60	46
shiftIdx	5	4	8	5	4	8	5	4	8

Table 65 – Specification of initValue and shiftIdx for ctxIdx of pred_mode_ibc_flag

Initialization variable	ctxIdx of pred_mode_ibc_flag								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
initValue	17	42	36	0	57	44	0	43	45
shiftIdx	1	5	8	1	5	8	1	5	8

Table 66 – Specification of initValue and shiftIdx for ctxIdx of pred_mode_flag

Initialization variable	ctxIdx of pred_mode_flag			
	0	**1**	**2**	**3**
initValue	40	35	40	35
shiftIdx	5	1	5	1

Table 67 – Specification of initValue and shiftIdx for ctxIdx of pred_mode_plt_flag

Initialization variable	ctxIdx of pred_mode_plt_flag		
	0	**1**	**2**
initValue	25	0	17
shiftIdx	1	1	1

Table 68 – Specification of initValue and shiftIdx for ctxIdx of cu_act_enabled_flag

Initialization variable	ctxIdx of cu_act_enabled_flag		
	0	**1**	**2**
initValue	52	46	46
shiftIdx	1	1	1

Table 69 – Specification of initValue and shiftIdx for ctxIdx of intra_bdpcm_luma_flag

Initialization variable	ctxIdx of intra_bdpcm_luma_flag		
	0	1	2
initValue	19	40	19
shiftIdx	1	1	1

Table 70 – Specification of initValue and shiftIdx for ctxIdx of intra_bdpcm_luma_dir_flag

Initialization variable	ctxIdx of intra_bdpcm_luma_dir_flag		
	0	1	2
initValue	35	36	21
shiftIdx	4	4	4

Table 71 – Specification of initValue and shiftIdx for ctxIdx of intra_mip_flag

Initialization variable	ctxIdx of intra_mip_flag											
	0	1	2	3	4	5	6	7	8	9	10	11
initValue	33	49	50	25	41	57	58	26	56	57	50	26
shiftIdx	9	10	9	6	9	10	9	6	9	10	9	6

Table 72 – Specification of initValue and shiftIdx for ctxIdx of intra_luma_ref_idx

Initialization variable	ctxIdx of intra_luma_ref_idx					
	0	1	2	3	4	5
initValue	25	60	25	58	25	59
shiftIdx	5	8	5	8	5	8

Table 73 – Specification of initValue and shiftIdx for ctxIdx of intra_subpartitions_mode_flag

Initialization variable	ctxIdx of intra_subpartitions_mode_flag		
	0	1	2
initValue	33	33	33
shiftIdx	9	9	9

Table 74 – Specification of initValue and shiftIdx for ctxIdx of intra_subpartitions_split_flag

Initialization variable	ctxIdx of intra_subpartitions_split_flag		
	0	1	2
initValue	43	36	43
shiftIdx	2	2	2

Table 75 – Specification of initValue and shiftIdx for ctxIdx of intra_luma_mpm_flag

Initialization variable	ctxIdx of intra_luma_mpm_flag		
	0	1	2
initValue	45	36	44
shiftIdx	6	6	6

Table 76 – Specification of initValue and shiftIdx for ctxIdx of intra_luma_not_planar_flag

Initialization variable	ctxIdx of intra_luma_not_planar_flag					
	0	1	2	3	4	5
initValue	13	28	12	20	13	6
shiftIdx	1	5	1	5	1	5

Table 77 – Specification of initValue and shiftIdx for ctxIdx of intra_bdpcm_chroma_flag

Initialization variable	ctxIdx of intra_bdpcm_chroma_flag		
	0	1	2
initValue	1	0	0
shiftIdx	1	1	1

Table 78 – Specification of initValue and shiftIdx for ctxIdx of intra_bdpcm_chroma_dir_flag

Initialization variable	ctxIdx of intra_bdpcm_chroma_dir_flag		
	0	1	2
initValue	27	13	28
shiftIdx	0	0	0

Table 79 – Specification of initValue and shiftIdx for ctxIdx of cclm_mode_flag

Initialization variable	ctxIdx of cclm_mode_flag		
	0	**1**	**2**
initValue	59	34	26
shiftIdx	4	4	4

Table 80 – Specification of initValue and shiftIdx for ctxIdx of cclm_mode_idx

Initialization variable	ctxIdx of cclm_mode_idx		
	0	**1**	**2**
initValue	27	27	27
shiftIdx	9	9	9

Table 81 – Specification of initValue and shiftIdx for ctxIdx of intra_chroma_pred_mode

Initialization variable	ctxIdx of intra_chroma_pred_mode		
	0	**1**	**2**
initValue	34	25	25
shiftIdx	5	5	5

Table 82 – Specification of initValue and shiftIdx for ctxIdx of general_merge_flag

Initialization variable	ctxIdx of general_merge_flag		
	0	**1**	**2**
initValue	26	21	6
shiftIdx	4	4	4

Table 83 – Specification of initValue and shiftIdx for ctxIdx of inter_pred_idc

Initialization variable	ctxIdx of inter_pred_idc											
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
initValue	7	6	5	12	4	40	14	13	5	4	3	40
shiftIdx	0	0	1	4	4	0	0	0	1	4	4	0

Table 84 – Specification of initValue and shiftIdx for ctxIdx of inter_affine_flag

Initialization variable	ctxIdx of inter_affine_flag					
	0	**1**	**2**	**3**	**4**	**5**
initValue	12	13	14	19	13	6
shiftIdx	4	0	0	4	0	0

Table 85 – Specification of initValue and shiftIdx for ctxIdx of cu_affine_type_flag

Initialization variable	ctxIdx of cu_affine_type_flag	
	0	**1**
initValue	35	35
shiftIdx	4	4

Table 86 – Specification of initValue and shiftIdx for ctxIdx of sym_mvd_flag

Initialization variable	ctxIdx of sym_mvd_flag	
	0	**1**
initValue	28	28
shiftIdx	5	5

Table 87 – Specification of initValue and shiftIdx for ctxIdx of ref_idx_l0 and ref_idx_l1

Initialization variable	ctxIdx of ref_idx_l0, ref_idx_l1			
	0	**1**	**2**	**3**
initValue	20	35	5	35
shiftIdx	0	4	0	4

Table 88 – Specification of initValue and shiftIdx for ctxIdx of mvp_l0_flag, mvp_l1_flag

Initialization variable	ctxIdx of mvp_l0_flag, mvp_l1_flag		
	0	**1**	**2**
initValue	42	34	34
shiftIdx	12	12	12

Table 89 – Specification of initValue and shiftIdx for ctxIdx of amvr_flag

Initialization variable	ctxIdx of amvr_flag			
	0	**1**	**2**	**3**
initValue	59	58	59	50
shiftIdx	0	0	0	0

Table 90 – Specification of initValue and shiftIdx for ctxIdx of amvr_precision_idx

Initialization variable	ctxIdx of amvr_precision_idx								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
initValue	35	34	35	60	48	60	38	26	60
shiftIdx	4	5	0	4	5	0	4	5	0

Table 91 – Specification of initValue and shiftIdx for ctxIdx of bcw_idx

Initialization variable	ctxIdx of bcw_idx	
	0	**1**
initValue	4	5
shiftIdx	1	1

Table 92 – Specification of initValue and shiftIdx for ctxIdx of cu_coded_flag

Initialization variable	ctxIdx of cu_coded_flag		
	0	**1**	**2**
initValue	6	5	12
shiftIdx	4	4	4

Table 93 – Specification of initValue and shiftIdx for ctxIdx of cu_sbt_flag

Initialization variable	ctxIdx of cu_sbt_flag			
	0	**1**	**2**	**3**
initValue	56	57	41	57
shiftIdx	1	5	1	5

Table 94 – Specification of initValue and shiftIdx for ctxIdx of cu_sbt_quad_flag

Initialization variable	ctxIdx of cu_sbt_quad_flag	
	0	**1**
initValue	42	42
shiftIdx	10	10

Table 95 – Specification of initValue and shiftIdx for ctxIdx of cu_sbt_horizontal_flag

Initialization variable	ctxIdx of cu_sbt_horizontal_flag					
	0	**1**	**2**	**3**	**4**	**5**
initValue	20	43	12	35	51	27
shiftIdx	8	4	1	8	4	1

Table 96 – Specification of initValue and shiftIdx for ctxIdx of cu_sbt_pos_flag

Initialization variable	ctxIdx of cu_sbt_pos_flag	
	0	**1**
initValue	28	28
shiftIdx	13	13

Table 97 – Specification of initValue and shiftIdx for ctxIdx of lfnst_idx

Initialization variable	ctxIdx of lfnst_idx								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
initValue	28	52	42	37	45	27	52	37	27
shiftIdx	9	9	10	9	9	10	9	9	10

Table 98 – Specification of initValue and shiftIdx for ctxIdx of mts_idx

Initialization variable	ctxIdx of mts_idx											
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
initValue	29	0	28	0	45	40	27	0	45	25	27	0
shiftIdx	8	0	9	0	8	0	9	0	8	0	9	0

Table 99 – Specification of initValue and shiftIdx for ctxIdx of copy_above_palette_indices_flag

Initialization variable	ctxIdx of copy_above_palette_indices_flag		
	0	**1**	**2**
initValue	42	59	50
shiftIdx	9	9	9

Table 100 – Specification of initValue and shiftIdx for ctxIdx of palette_transpose_flag

Initialization variable	ctxIdx of palette_transpose_flag		
	0	**1**	**2**
initValue	42	42	35
shiftIdx	5	5	5

Table 101 – Specification of initValue and shiftIdx for ctxIdx of run_copy_flag

Initialization variable	ctxIdx of run_copy_flag															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
initValue	50	37	45	30	46	45	38	46	51	30	30	38	23	38	53	46
shiftIdx	9	6	9	10	5	0	9	5	9	6	9	10	5	0	9	5
	16	**17**	**18**	**19**	**20**	**21**	**22**	**23**								
initValue	58	45	45	30	38	45	38	46								
shiftIdx	9	6	9	10	5	0	9	5								

Table 102 – Specification of initValue and shiftIdx for ctxIdx of regular_merge_flag

Initialization variable	ctxIdx of regular_merge_flag			
	0	**1**	**2**	**3**
initValue	38	7	46	15
shiftIdx	5	5	5	5

Table 103 – Specification of initValue and shiftIdx for ctxIdx of mmvd_merge_flag

Initialization variable	ctxIdx of mmvd_merge_flag	
	0	**1**
initValue	26	25
shiftIdx	4	4

Table 104 – Specification of initValue and shiftIdx for ctxIdx of mmvd_cand_flag

Initialization variable	ctxIdx of mmvd_cand_flag	
	0	**1**
initValue	43	43
shiftIdx	10	10

Table 105 – Specification of initValue and shiftIdx for ctxIdx of mmvd_distance_idx

Initialization variable	ctxIdx of mmvd_distance_idx	
	0	**1**
initValue	60	59
shiftIdx	0	0

Table 106 – Specification of initValue and shiftIdx for ctxIdx of ciip_flag

Initialization variable	ctxIdx of ciip_flag	
	0	**1**
initValue	57	57
shiftIdx	1	1

Table 107 – Specification of initValue and shiftIdx for ctxIdx of merge_subblock_flag

Initialization variable	ctxIdx of merge_subblock_flag					
	0	**1**	**2**	**3**	**4**	**5**
initValue	48	57	44	25	58	45
shiftIdx	4	4	4	4	4	4

Table 108 – Specification of initValue and shiftIdx for ctxIdx of merge_subblock_idx

Initialization variable	ctxIdx of merge_subblock_idx	
	0	**1**
initValue	5	4
shiftIdx	0	0

Table 109 – Specification of initValue and shiftIdx for ctxIdx of merge_idx, merge_gpm_idx0, and merge_gpm_idx1

Initialization variable	ctxIdx of merge_idx, merge_gpm_idx0, merge_gpm_idx1		
	0	1	2
initValue	34	20	18
shiftIdx	4	4	4

Table 110 – Specification of initValue and shiftIdx for ctxIdx of abs_mvd_greater0_flag

Initialization variable	ctxIdx of abs_mvd_greater0_flag		
	0	1	2
initValue	14	44	51
shiftIdx	9	9	9

Table 111 – Specification of initValue and shiftIdx for ctxIdx of abs_mvd_greater1_flag

Initialization variable	ctxIdx of abs_mvd_greater1_flag		
	0	1	2
initValue	45	43	36
shiftIdx	5	5	5

Table 112 – Specification of initValue and shiftIdx for ctxIdx of tu_y_coded_flag

Initialization variable	ctxIdx of tu_y_coded_flag											
	0	1	2	3	4	5	6	7	8	9	10	11
initValue	15	12	5	7	23	5	20	7	15	6	5	14
shiftIdx	5	1	8	9	5	1	8	9	5	1	8	9

Table 113 – Specification of initValue and shiftIdx for ctxIdx of tu_cb_coded_flag

Initialization variable	ctxIdx of tu_cb_coded_flag					
	0	1	2	3	4	5
initValue	12	21	25	28	25	37
shiftIdx	5	0	5	0	5	0

Table 114 – Specification of initValue and shiftIdx for ctxIdx of tu_cr_coded_flag

Initialization variable	ctxIdx of tu_cr_coded_flag								
	0	1	2	3	4	5	6	7	8
initValue	33	28	36	25	29	45	9	36	45
shiftIdx	2	1	0	2	1	0	2	1	0

Table 115 – Specification of initValue and shiftIdx for ctxIdx of cu_qp_delta_abs

Initialization variable	ctxIdx of cu_qp_delta_abs					
	0	1	2	3	4	5
initValue	35	35	35	35	35	35
shiftIdx	8	8	8	8	8	8

Table 116 – Specification of initValue and shiftIdx for ctxIdx of cu_chroma_qp_offset_flag

Initialization variable	ctxIdx of cu_chroma_qp_offset_flag		
	0	1	2
initValue	35	35	35
shiftIdx	8	8	8

Table 117 – Specification of initValue and shiftIdx for ctxIdx of cu_chroma_qp_offset_idx

Initialization variable	ctxIdx of cu_chroma_qp_offset_idx		
	0	1	2
initValue	35	35	35
shiftIdx	8	8	8

Table 118 – Specification of initValue and shiftIdx for ctxIdx of transform_skip_flag

Initialization variable	ctxIdx of transform_skip_flag					
	0	1	2	3	4	5
initValue	25	9	25	9	25	17
shiftIdx	1	1	1	1	1	1

Table 119 – Specification of initValue and shiftIdx for ctxIdx of tu_joint_cbcr_residual_flag

Initialization variable	ctxIdx of tu_joint_cbcr_residual_flag								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
initValue	12	21	35	27	36	45	42	43	52
shiftIdx	1	1	0	1	1	0	1	1	0

Table 120 – Specification of initValue and shiftIdx for ctxIdx of last_sig_coeff_x_prefix

Initialization variable	ctxIdx of last_sig_coeff_x_prefix															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
initValue	13	5	4	21	14	4	6	14	21	11	14	7	14	5	11	21
shiftIdx	8	5	4	5	4	4	5	4	1	0	4	1	0	0	0	0
	16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
initValue	30	22	13	42	12	4	3	6	13	12	6	6	12	14	14	13
shiftIdx	1	0	0	0	5	4	4	8	5	4	5	4	4	5	4	1
	32	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
initValue	12	29	7	6	13	36	28	14	13	5	26	12	4	18	6	6
shiftIdx	0	4	1	0	0	0	0	1	0	0	0	5	4	4	8	5
	48	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
initValue	12	14	6	4	14	7	6	4	29	7	6	6	12	28	7	13
shiftIdx	4	5	4	4	5	4	1	0	4	1	0	0	0	0	1	0
	64	**65**	**66**	**67**	**68**											
initValue	13	35	19	5	4											
shiftIdx	0	0	5	4	4											

Table 121 – Specification of initValue and shiftIdx for ctxIdx of last_sig_coeff_y_prefix

Initialization variable	ctxIdx of last_sig_coeff_y_prefix															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
initValue	13	5	4	6	13	11	14	6	5	3	14	22	6	4	3	6
shiftIdx	8	5	8	5	5	4	5	5	4	0	5	4	1	0	0	1
	16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
initValue	22	29	20	34	12	4	3	5	5	12	6	6	4	6	14	5
shiftIdx	4	0	0	0	6	5	5	8	5	8	5	5	4	5	5	4
	32	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
initValue	12	14	7	13	5	13	21	14	20	12	34	11	4	18	5	5
shiftIdx	0	5	4	1	0	0	1	4	0	0	0	6	5	5	8	5
	48	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
initValue	20	13	13	19	21	6	12	12	14	14	5	4	12	13	7	13
shiftIdx	8	5	5	4	5	5	4	0	5	4	1	0	0	1	4	0
	64	**65**	**66**	**67**	**68**											
initValue	12	41	11	5	27											
shiftIdx	0	0	6	5	5											

Table 122 – Specification of initValue and shiftIdx for ctxIdx of sb_coded_flag

Initialization variable	ctxIdx of sb_coded_flag											
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
initValue	18	31	25	15	18	20	38	25	30	25	45	18
shiftIdx	8	5	5	8	5	8	8	8	5	5	8	5
	12	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**			
initValue	12	29	25	45	25	14	18	35	45			
shiftIdx	8	8	8	5	5	8	5	8	8			

Table 123 – Specification of initValue and shiftIdx for ctxIdx of sig_coeff_flag

Initialization variable	ctxIdx of sig_coeff_flag															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
initValue	25	19	28	14	25	20	29	30	19	37	30	38	11	38	46	54
shiftIdx	12	9	9	10	9	9	9	10	8	8	8	10	9	13	8	8
	16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
initValue	27	39	39	39	44	39	39	39	18	39	39	39	27	39	39	39
shiftIdx	8	8	8	5	8	0	0	0	8	8	8	8	8	0	4	4
	32	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
initValue	0	39	39	39	25	27	28	37	34	53	53	46	19	46	38	39
shiftIdx	0	0	0	0	12	12	9	13	4	5	8	9	8	12	12	8
	48	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
initValue	52	39	39	39	11	39	39	39	19	39	39	39	25	28	38	17
shiftIdx	4	0	0	0	8	8	8	8	4	0	0	0	13	13	8	12
	64	**65**	**66**	**67**	**68**	**69**	**70**	**71**	**72**	**73**	**74**	**75**	**76**	**77**	**78**	**79**
initValue	41	42	29	25	49	43	37	33	58	51	30	19	38	38	46	34
shiftIdx	9	9	10	9	9	9	10	8	8	8	10	9	13	8	8	8
	80	**81**	**82**	**83**	**84**	**85**	**86**	**87**	**88**	**89**	**90**	**91**	**92**	**93**	**94**	**95**
initValue	54	54	39	6	39	39	39	19	39	54	39	19	39	39	39	56
shiftIdx	8	8	5	8	0	0	0	8	8	8	8	8	0	4	4	0
	96	**97**	**98**	**99**	**100**	**101**	**102**	**103**	**104**	**105**	**106**	**107**	**108**	**109**	**110**	**111**
initValue	39	39	39	17	34	35	21	41	59	60	38	35	45	53	54	44
shiftIdx	0	0	0	12	12	9	13	4	5	8	9	8	12	12	8	4
	112	**113**	**114**	**115**	**116**	**117**	**118**	**119**	**120**	**121**	**122**	**123**	**124**	**125**	**126**	**127**
initValue	39	39	39	34	38	62	39	26	39	39	39	40	35	44	17	41
shiftIdx	0	0	0	8	8	8	8	4	0	0	0	13	13	8	12	9
	128	**129**	**130**	**131**	**132**	**133**	**134**	**135**	**136**	**137**	**138**	**139**	**140**	**141**	**142**	**143**
initValue	49	36	1	49	50	37	48	51	58	45	26	45	53	46	49	54
shiftIdx	9	10	9	9	9	10	8	8	8	10	9	13	8	8	8	8
	144	**145**	**146**	**147**	**148**	**149**	**150**	**151**	**152**	**153**	**154**	**155**	**156**	**157**	**158**	**159**
initValue	61	39	35	39	39	39	19	54	39	39	50	39	39	39	0	39
shiftIdx	8	5	8	0	0	0	8	8	8	8	8	0	4	4	0	0
	160	**161**	**162**	**163**	**164**	**165**	**166**	**167**	**168**	**169**	**170**	**171**	**172**	**173**	**174**	**175**
initValue	39	39	9	49	50	36	48	59	59	38	34	45	38	31	58	39
shiftIdx	0	0	12	12	9	13	4	5	8	9	8	12	12	8	4	0
	176	**177**	**178**	**179**	**180**	**181**	**182**	**183**	**184**	**185**	**186**	**187**	**188**			
initValue	39	39	34	38	54	39	41	39	39	39	25	50	37			
shiftIdx	0	0	8	8	8	8	4	0	0	0	13	13	8			

Table 124 – Specification of initValue and shiftIdx for ctxIdx of par_level_flag

Initialization variable	ctxIdx of par_level_flag															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
initValue	33	25	18	26	34	27	25	26	19	42	35	33	19	27	35	35
shiftIdx	8	9	12	13	13	13	10	13	13	13	13	13	13	13	13	13
	16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
initValue	34	42	20	43	20	33	25	26	42	19	27	26	50	35	20	43
shiftIdx	10	13	13	13	13	8	12	12	12	13	13	13	13	13	13	13
	32	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
initValue	11	18	17	33	18	26	42	25	33	26	42	27	25	34	42	42
shiftIdx	6	8	9	12	13	13	13	10	13	13	13	13	13	13	13	13
	48	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
initValue	35	26	27	42	20	20	25	25	26	11	19	27	33	42	35	35
shiftIdx	13	10	13	13	13	13	8	12	12	12	13	13	13	13	13	13
	64	**65**	**66**	**67**	**68**	**69**	**70**	**71**	**72**	**73**	**74**	**75**	**76**	**77**	**78**	**79**
initValue	43	3	33	40	25	41	26	42	25	33	26	34	27	25	41	42
shiftIdx	13	6	8	9	12	13	13	13	10	13	13	13	13	13	13	13
	80	**81**	**82**	**83**	**84**	**85**	**86**	**87**	**88**	**89**	**90**	**91**	**92**	**93**	**94**	**95**
initValue	42	35	33	27	35	42	43	33	25	26	34	19	27	33	42	43
shiftIdx	13	13	10	13	13	13	13	8	12	12	12	13	13	13	13	13
	96	**97**	**98**													
initValue	35	43	11													
shiftIdx	13	13	6													

Table 125 – Specification of initValue and shiftIdx for ctxIdx of abs_level_gtx_flag

Initialization variable	ctxIdx of abs_level_gtx_flag															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
initValue	25	25	11	27	20	21	33	12	28	21	22	34	28	29	29	30
shiftIdx	9	5	10	13	13	10	9	10	13	13	13	9	10	10	10	13
	16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
initValue	36	29	45	30	23	40	33	27	28	21	37	36	37	45	38	46
shiftIdx	8	9	10	10	13	8	8	9	12	12	10	5	9	9	9	13
	32	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
initValue	25	1	40	25	33	11	17	25	25	18	4	17	33	26	19	13
shiftIdx	1	5	9	9	9	6	5	9	10	10	9	9	9	9	9	9
	48	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
initValue	33	19	20	28	22	40	9	25	18	26	35	25	26	35	28	37
shiftIdx	6	8	9	9	10	1	5	8	8	9	6	6	9	8	8	9
	64	**65**	**66**	**67**	**68**	**69**	**70**	**71**	**72**	**73**	**74**	**75**	**76**	**77**	**78**	**79**
initValue	11	5	5	14	10	3	3	3	0	17	26	19	35	21	25	34
shiftIdx	4	2	1	6	1	1	1	1	9	5	10	13	13	10	9	10
	80	**81**	**82**	**83**	**84**	**85**	**86**	**87**	**88**	**89**	**90**	**91**	**92**	**93**	**94**	**95**
initValue	20	28	29	33	27	28	29	22	34	28	44	37	38	0	25	19
shiftIdx	13	13	13	9	10	10	10	13	8	9	10	10	13	8	8	9
	96	**97**	**98**	**99**	**100**	**101**	**102**	**103**	**104**	**105**	**106**	**107**	**108**	**109**	**110**	**111**
initValue	20	13	14	57	44	30	30	23	17	0	1	17	25	18	0	9
shiftIdx	12	12	10	5	9	9	9	13	1	5	9	9	9	6	5	9
	112	**113**	**114**	**115**	**116**	**117**	**118**	**119**	**120**	**121**	**122**	**123**	**124**	**125**	**126**	**127**

Table 125 – Specification of initValue and shiftIdx for ctxIdx of abs_level_gtx_flag

Initialization variable	ctxIdx of abs_level_gtx_flag															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
initValue	25	33	34	9	25	18	26	20	25	18	19	27	29	17	9	25
shiftIdx	10	10	9	9	9	9	9	9	6	8	9	9	10	1	5	8
	128	**129**	**130**	**131**	**132**	**133**	**134**	**135**	**136**	**137**	**138**	**139**	**140**	**141**	**142**	**143**
initValue	10	18	4	17	33	19	20	29	18	11	4	28	2	10	3	3
shiftIdx	8	9	6	6	9	8	8	9	4	2	1	6	1	1	1	1
	144	**145**	**146**	**147**	**148**	**149**	**150**	**151**	**152**	**153**	**154**	**155**	**156**	**157**	**158**	**159**
initValue	0	0	33	34	35	21	25	34	35	28	29	40	42	43	29	30
shiftIdx	9	5	10	13	13	10	9	10	13	13	13	9	10	10	10	13
	160	**161**	**162**	**163**	**164**	**165**	**166**	**167**	**168**	**169**	**170**	**171**	**172**	**173**	**174**	**175**
initValue	49	36	37	45	38	0	40	34	43	36	37	57	52	45	38	46
shiftIdx	8	9	10	10	13	8	8	9	12	12	10	5	9	9	9	13
	176	**177**	**178**	**179**	**180**	**181**	**182**	**183**	**184**	**185**	**186**	**187**	**188**	**189**	**190**	**191**
initValue	25	0	0	17	25	26	0	9	25	33	19	0	25	33	26	20
shiftIdx	1	5	9	9	9	6	5	9	10	10	9	9	9	9	9	9
	192	**193**	**194**	**195**	**196**	**197**	**198**	**199**	**200**	**201**	**202**	**203**	**204**	**205**	**206**	**207**
initValue	25	33	27	35	22	25	1	25	33	26	12	25	33	27	28	37
shiftIdx	6	8	9	9	10	1	5	8	8	9	6	6	9	8	8	9
	208	**209**	**210**	**211**	**212**	**213**	**214**	**215**								
initValue	19	11	4	6	3	4	4	5								
shiftIdx	4	2	1	6	1	1	1	1								

Table 126 – Specification of initValue and shiftIdx for ctxIdx of coeff_sign_flag

Initialization variable	ctxIdx of coeff_sign_flag																	
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
initValue	12	17	46	28	25	46	5	10	53	43	25	46	35	25	46	28	33	38
shiftIdx	1	4	4	5	8	8	1	4	4	5	8	8	1	4	4	5	8	8

9.3.2.3 Storage process for context variables

Inputs to this process are:

– The CABAC context variables indexed by ctxTable and ctxIdx.

Outputs of this process are:

– The arrays TableStateIdx0Wpp and TableStateIdx1Wpp containing the values of the variables pStateIdx0 and pStateIdx1 used in the initialization process of context variables that are assigned to all syntax elements in clauses 7.3.11.1 through 7.3.11.11, except end_of_slice_one_bit, end_of_tile_one_bit, and end_of_subset_one_bit.

For each context variable, the values of pStateIdx0 and pStateIdx1 are stored into the corresponding entries in the arrays TableStateIdx0Wpp and TableStateIdx1Wpp, respectively.

The storage process for context variables is illustrated in the flowchart of Figure 14.

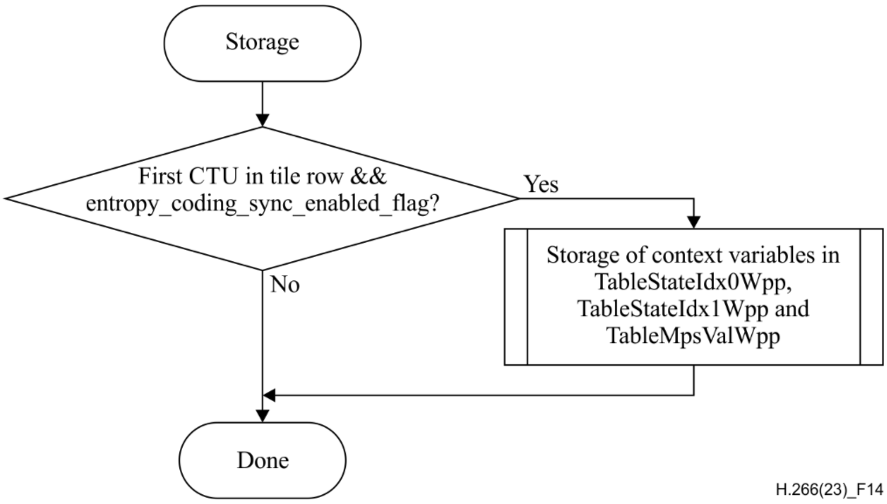

Figure 14 – Flowchart of CABAC storage process (informative)

9.3.2.4 Synchronization process for context variables

Inputs to this process are:

– The arrays TableStateIdx0Wpp and TableStateIdx1Wpp containing the values of the variables pStateIdx0 and pStateIdx1 used in the storage process of context variables that are assigned to all syntax elements in clauses 7.3.11.1 through 7.3.11.11, except end_of_slice_one_bit, end_of_tile_one_bit, and end_of_subset_one_bit.

Outputs of this process are:

– The initialized CABAC context variables indexed by ctxTable and ctxIdx.

For each context variable, the variables pStateIdx0 and pStateIdx1 are set equal to the corresponding entries in the arrays TableStateIdx0Wpp and TableStateIdx1Wpp, respectively.

9.3.2.5 Initialization process for the arithmetic decoding engine

Outputs of this process are the initialized decoding engine registers ivlCurrRange and ivlOffset both in 16 bit register precision.

The status of the arithmetic decoding engine is represented by the variables ivlCurrRange and ivlOffset. In the initialization procedure of the arithmetic decoding process, ivlCurrRange is set equal to 510 and ivlOffset is set equal to the value returned from read_bits(9) interpreted as a 9 bit binary representation of an unsigned integer with the most significant bit written first.

The bitstream shall not contain data that result in a value of ivlOffset being equal to 510 or 511.

NOTE – The description of the arithmetic decoding engine in this Specification utilizes the 16-bit register precision. However, a minimum register precision of 9 bits is required for storing the values of the variables ivlCurrRange and ivlOffset after invocation of the arithmetic decoding process (DecodeBin) as specified in clause 9.3.4.3. The arithmetic decoding process for a binary decision (DecodeDecision) as specified in clause 9.3.4.3.2 and the decoding process for a binary decision before termination (DecodeTerminate) as specified in clause 9.3.4.3.5 require a minimum register precision of 9 bits for the variables ivlCurrRange and ivlOffset. The bypass decoding process for binary decisions (DecodeBypass) as specified in clause 9.3.4.3.4 requires a minimum register precision of 10 bits for the variable ivlOffset and a minimum register precision of 9 bits for the variable ivlCurrRange.

9.3.2.6 Storage process for palette predictor

This process stores the values of the arrays PredictorPaletteSize and PredictorPaletteEntries in the arrays TablePaletteSizeWpp and TablePaletteEntriesWpp as follows:

```
for( cIdx = 0; cIdx < 3; cIdx++ ) {
    chType = cIdx  = =  0 ? 0 : 1
    TablePaletteSizeWpp[ chType ] = PredictorPaletteSize[ chType ]                    (1522)
    for( i = 0; i < PredictorPaletteSize[ chType ]; i++ )
        TablePaletteEntriesWpp[ cIdx ][ i ] = PredictorPaletteEntries[ cIdx ][ i ]
}
```

9.3.2.7 Synchronization process for palette predictor

This process synchronizes the values of the arrays PredictorPaletteSize and PredictorPaletteEntries in the arrays TablePaletteSizeWpp and TablePaletteEntriesWpp as follows:

```
for( cIdx = 0; cIdx < 3; cIdx++ ) {
    chType = cIdx  = =  0 ? 0 : 1
    PredictorPaletteSize[ chType ] = TablePaletteSizeWpp[ chType ]                    (1523)
    for( i = 0; i < PredictorPaletteSize[ chType ]; i++ )
        PredictorPaletteEntries[ cIdx ][ i ] = TablePaletteEntriesWpp[ cIdx ][ i ]
}
```

9.3.3 Binarization process

9.3.3.1 General

Input to this process is a request for a syntax element.

Output of this process is the binarization of the syntax element.

Table 127 specifies the type of binarization process associated with each syntax element and corresponding inputs.

The specification of the truncated Rice (TR) binarization process, the truncated binary (TB) binarization process, the k-th order Exp-Golomb (EGk) binarization process and the fixed-length (FL) binarization process are given in clauses 9.3.3.3 through 9.3.3.7, respectively.

Table 127 – Syntax elements and associated binarizations

Syntax structure	Syntax element	Binarization	
		Process	Input parameters
slice_data( )	end_of_slice_one_bit	FL	cMax = 1
	end_of_tile_one_bit	FL	cMax = 1
	end_of_subset_one_bit	FL	cMax = 1
coding_tree_unit( )	alf_ctb_flag[ ][ ][ ]	FL	cMax = 1
	alf_use_aps_flag	FL	cMax = 1
	alf_luma_fixed_filter_idx	TB	cMax = 15
	alf_luma_prev_filter_idx	TB	cMax = sh_num_alf_aps_ids_luma − 1
	alf_ctb_filter_alt_idx[ ][ ][ ]	TR	cMax = alf_chroma_num_alt_filters_minus1, cRiceParam = 0
	alf_ctb_cc_cb_idc[ ][ ]	TR	cMax = (alf_cc_cb_filters_signalled_minus1 + 1), cRiceParam = 0
	alf_ctb_cc_cr_idc[ ][ ]	TR	cMax = (alf_cc_cr_filters_signalled_minus1 + 1), cRiceParam = 0
sao( )	sao_merge_left_flag	FL	cMax = 1
	sao_merge_up_flag	FL	cMax = 1
	sao_type_idx_luma	TR	cMax = 2, cRiceParam = 0
	sao_type_idx_chroma	TR	cMax = 2, cRiceParam = 0
	sao_offset_abs[ ][ ][ ][ ]	TR	cMax = (1 << (Min(BitDepth, 10) − 5)) − 1, cRiceParam = 0
	sao_offset_sign_flag[ ][ ][ ][ ]	FL	cMax = 1
	sao_band_position[ ][ ][ ]	FL	cMax = 31
	sao_eo_class_luma	FL	cMax = 3
	sao_eo_class_chroma	FL	cMax = 3

Table 127 – Syntax elements and associated binarizations

Syntax structure	Syntax element	Binarization	
		Process	**Input parameters**
coding_tree( )	split_cu_flag	FL	cMax = 1
	split_qt_flag	FL	cMax = 1
	mtt_split_cu_vertical_flag	FL	cMax = 1
	mtt_split_cu_binary_flag	FL	cMax = 1
	non_inter_flag	FL	cMax = 1
coding_unit( )	cu_skip_flag[ ][ ]	FL	cMax = 1
	pred_mode_ibc_flag	FL	cMax = 1
	pred_mode_plt_flag	FL	cMax = 1
	cu_act_enabled_flag	FL	cMax = 1
	pred_mode_flag	FL	cMax = 1
	intra_bdpcm_luma_flag	FL	cMax = 1
	intra_bdpcm_luma_dir_flag	FL	cMax = 1
	intra_mip_flag	FL	cMax = 1
	intra_mip_transposed_flag[ ][ ]	FL	cMax = 1
	intra_mip_mode[ ][ ]	TB	cMax = (cbWidth = = 4 && cbHeight = = 4) ? 15 : (((cbWidth = = 4 || cbHeight = = 4) || (cbWidth = = 8 && cbHeight = = 8)) ? 7 : 5)
	intra_luma_ref_idx	TR	cMax = 2, cRiceParam = 0
	intra_subpartitions_mode_flag	FL	cMax = 1
	intra_subpartitions_split_flag	FL	cMax = 1
	intra_luma_mpm_flag[ ][ ]	FL	cMax = 1
	intra_luma_not_planar_flag[ ][ ]	FL	cMax = 1
	intra_luma_mpm_idx[ ][ ]	TR	cMax = 4, cRiceParam = 0
	intra_luma_mpm_remainder[ ][ ]	TB	cMax = 60
	intra_bdpcm_chroma_flag	FL	cMax = 1
	intra_bdpcm_chroma_dir_flag	FL	cMax = 1
	cclm_mode_flag	FL	cMax = 1
	cclm_mode_idx	TR	cMax = 2, cRiceParam = 0
	intra_chroma_pred_mode	9.3.3.8	-
	general_merge_flag[ ][ ]	FL	cMax = 1
	inter_pred_idc[ ][ ]	9.3.3.9	cbWidth, cbHeight
	inter_affine_flag[ ][ ]	FL	cMax = 1
	cu_affine_type_flag[ ][ ]	FL	cMax = 1

Table 127 – Syntax elements and associated binarizations

Syntax structure	Syntax element	Binarization	
		Process	**Input parameters**
	sym_mvd_flag[ ][ ]	FL	cMax = 1
	ref_idx_l0[ ][ ]	TR	cMax = NumRefIdxActive[0] − 1, cRiceParam = 0
	mvp_l0_flag[ ][ ]	FL	cMax = 1
	ref_idx_l1[ ][ ]	TR	cMax = NumRefIdxActive[1] − 1, cRiceParam = 0
	mvp_l1_flag[ ][ ]	FL	cMax = 1
	amvr_flag[ ][ ]	FL	cMax = 1
	amvr_precision_idx[ ][ ]	TR	cMax = (inter_affine_flag = = 0 && CuPredMode[0][x0][y0] != MODE_IBC) ? 2 : 1, cRiceParam = 0
	bcw_idx[ ][ ]	TR	cMax = NoBackwardPredFlag ? 4: 2, cRiceParam = 0
	cu_coded_flag	FL	cMax = 1
	cu_sbt_flag	FL	cMax = 1
	cu_sbt_quad_flag	FL	cMax = 1
	cu_sbt_horizontal_flag	FL	cMax = 1
	cu_sbt_pos_flag	FL	cMax = 1
	lfnst_idx	TR	cMax = 2, cRiceParam = 0
	mts_idx	TR	cMax = 4, cRiceParam = 0
palette_coding( )	palette_predictor_run	EG0	-
	num_signalled_palette_entries	EG0	-
	new_palette_entries	FL	cMax = (1 << BitDepth) − 1
	palette_escape_val_present_flag	FL	cMax = 1
	palette_idx_idc	9.3.3.13	MaxPaletteIndex
	palette_transpose_flag	FL	cMax = 1
	copy_above_palette_indices_flag	FL	cMax = 1
	run_copy_flag	FL	cMax = 1
	palette_escape_val	EG5	

Table 127 – Syntax elements and associated binarizations

Syntax structure	Syntax element	Binarization	
		Process	Input parameters
merge_data( )	regular_merge_flag[ ][ ]	FL	cMax = 1
	mmvd_merge_flag[ ][ ]	FL	cMax = 1
	mmvd_cand_flag[ ][ ]	FL	cMax = 1
	mmvd_distance_idx[ ][ ]	TR	cMax = 7, cRiceParam = 0
	mmvd_direction_idx[ ][ ]	FL	cMax = 3
	ciip_flag[ ][ ]	FL	cMax = 1
	merge_subblock_flag[ ][ ]	FL	cMax = 1
	merge_subblock_idx[ ][ ]	TR	cMax = MaxNumSubblockMergeCand − 1, cRiceParam = 0
	merge_gpm_partition_idx[ ][ ]	FL	cMax = 63
	merge_gpm_idx0[ ][ ]	TR	cMax = MaxNumGpmMergeCand − 1, cRiceParam = 0
	merge_gpm_idx1[ ][ ]	TR	cMax = MaxNumGpmMergeCand − 2, cRiceParam = 0
	merge_idx[ ][ ]	TR	cMax = (CuPredMode[0][x0][y0] != MODE_IBC ? MaxNumMergeCand : MaxNumIbcMergeCand) − 1, cRiceParam = 0
mvd_coding( )	abs_mvd_greater0_flag[ ]	FL	cMax = 1
	abs_mvd_greater1_flag[ ]	FL	cMax = 1
	abs_mvd_minus2[ ]	9.3.3.14	-
	mvd_sign_flag[ ]	FL	cMax = 1
transform_unit( )	tu_y_coded_flag[ ][ ]	FL	cMax = 1
	tu_cb_coded_flag[ ][ ]	FL	cMax = 1
	tu_cr_coded_flag[ ][ ]	FL	cMax = 1
	cu_qp_delta_abs	9.3.3.10	-
	cu_qp_delta_sign_flag	FL	cMax = 1
	cu_chroma_qp_offset_flag	FL	cMax = 1
	cu_chroma_qp_offset_idx	TR	cMax = pps_chroma_qp_offset_list_len_minus1, cRiceParam = 0
	transform_skip_flag[ ][ ][ ]	FL	cMax = 1
	tu_joint_cbcr_residual_flag[ ][ ]	FL	cMax = 1

Table 127 – Syntax elements and associated binarizations

Syntax structure	Syntax element	Binarization	
		Process	Input parameters
residual_coding( )	last_sig_coeff_x_prefix	TR	cMax = (Log2ZoTbWidth << 1) − 1, cRiceParam = 0
	last_sig_coeff_y_prefix	TR	cMax = (Log2ZoTbHeight << 1) − 1, cRiceParam = 0
	last_sig_coeff_x_suffix	FL	cMax = (1 << ((last_sig_coeff_x_prefix >> 1) − 1)) − 1
	last_sig_coeff_y_suffix	FL	cMax = (1 << ((last_sig_coeff_y_prefix >> 1) − 1)) − 1
	sb_coded_flag[ ][ ]	FL	cMax = 1
	sig_coeff_flag[ ][ ]	FL	cMax = 1
	par_level_flag[ ]	FL	cMax = 1
	abs_level_gtx_flag[ ][ ]	FL	cMax = 1
	abs_remainder[ ]	9.3.3.11	cIdx, current sub-block index i, x0, y0, xC, yC
	dec_abs_level[ ]	9.3.3.12	cIdx, x0, y0, xC, yC
	coeff_sign_flag[ ]	FL	cMax = 1

9.3.3.2 Rice parameter derivation process for abs_remainder[ ] and dec_abs_level[ ]

Inputs to this process are the base level baseLevel, the colour component index cIdx, the luma location (x0, y0) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture, and the current coefficient scan location (xC, yC).

Output of this process is the Rice parameter cRiceParam.

Given the array AbsLevel[x][y] for the transform block with component index cIdx and the top-left luma location (x0, y0), the variable locSumAbs is derived as specified by the following pseudo-code process:

```
locSumAbs = 0
if( xC < ( 1 << Log2FullTbWidth ) − 1 ) {
    locSumAbs += AbsLevel[ xC + 1 ][ yC ]
    if( xC < ( 1 << Log2FullTbWidth ) − 2 )
        locSumAbs += AbsLevel[ xC + 2 ][ yC ]
    else
        locSumAbs += HistValue
    if( yC < ( 1 << Log2FullTbHeight ) − 1 )
        locSumAbs += AbsLevel[ xC + 1 ][ yC + 1 ]                    (1524)
    else
        locSumAbs += HistValue
} else
    locSumAbs += 2 * HistValue
if( yC < ( 1 << Log2FullTbHeight ) − 1 ) {
    locSumAbs += AbsLevel[ xC ][ yC + 1 ]
    if( yC < ( 1 << Log2FullTbHeight ) − 2 )
        locSumAbs += AbsLevel[ xC ][ yC + 2 ]
    else
        locSumAbs += HistValue
} else
    locSumAbs += HistValue
```

The lists Tx[ ] and Rx[ ] are specified as follows:

Tx[ ] = { 32, 128, 512, 2048 } (1525)

$$\text{Rx}[\,] = \{\,0, 2, 4, 6, 8\,\} \tag{1526}$$

The value of the variable shiftVal is derived as follows:

$$\begin{array}{l}
\text{if(!sps_rrc_rice_extension_flag)} \\
\quad \text{shiftVal} = 0 \\
\text{else} \\
\quad \text{shiftVal} = (\,\text{locSumAbs} < \text{Tx}[\,0\,]\,)\ ?\ \text{Rx}[\,0\,] : (\,(\,\text{locSumAbs} < \text{Tx}[\,1\,]\,)\ ?\ \text{Rx}[\,1\,] : \\
\qquad (\,(\,\text{locSumAbs} < \text{Tx}[\,2\,]\,)\ ?\ \text{Rx}[\,2\,] : (\,(\,\text{locSumAbs} < \text{Tx}[\,3\,]\,)\ ?\ \text{Rx}[\,3\,] : \text{Rx}[4]\,)\,)\,)
\end{array} \tag{1527}$$

The value of locSumAbs is updated as follows:

$$\text{locSumAbs} = \text{Clip3}(\,0, 31, (\,\text{locSumAbs} \gg \text{shiftVal}\,) - \text{baseLevel} * 5\,) \tag{1528}$$

Given the variable locSumAbs, the Rice parameter cRiceParam is first derived as specified in Table 128, and then updated as follows:

$$\text{cRiceParam} = \text{cRiceParam} + \text{shiftVal} \tag{1529}$$

When baseLevel is equal to 0, the variable ZeroPos[n] is derived as follows:

$$\text{ZeroPos}[\,n\,] = (\,\text{QState} < 2\ ?\ 1 : 2\,) \ll \text{cRiceParam} \tag{1530}$$

Table 128 – Specification of cRiceParam based on locSumAbs

locSumAbs	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
cRiceParam	0	0	0	0	0	0	0	1	1	1	1	1	1	1	2	2
locSumAbs	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
cRiceParam	2	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3

9.3.3.3 Truncated Rice binarization process

Input to this process is a request for a truncated Rice (TR) binarization, cMax and cRiceParam.

Output of this process is the TR binarization associating each value symbolVal with a corresponding bin string.

A TR bin string is a concatenation of a prefix bin string and, when present, a suffix bin string.

For the derivation of the prefix bin string, the following applies:

– The prefix value of symbolVal, prefixVal, is derived as follows:

$$\text{prefixVal} = \text{symbolVal} \gg \text{cRiceParam} \tag{1531}$$

– The prefix of the TR bin string is specified as follows:

  – If prefixVal is less than cMax >> cRiceParam, the prefix bin string is a bit string of length prefixVal + 1 indexed by binIdx. The bins for binIdx less than prefixVal are equal to 1. The bin with binIdx equal to prefixVal is equal to 0. Table 129 illustrates the bin strings of this unary binarization for prefixVal.

  – Otherwise, the bin string is a bit string of length cMax >> cRiceParam with all bins being equal to 1.

Table 129 – Bin string of the unary binarization (informative)

prefixVal	Bin string					
0	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

When cMax is greater than symbolVal and cRiceParam is greater than 0, the suffix of the TR bin string is present and it is derived as follows:

– The suffix value suffixVal is derived as follows:

$$\text{suffixVal} = \text{symbolVal} - (\ \text{prefixVal} \ll \text{cRiceParam}\ ) \quad (1532)$$

– The suffix of the TR bin string is specified by invoking the fixed-length (FL) binarization process as specified in clause 9.3.3.7 for suffixVal with a cMax value equal to (1 << cRiceParam) − 1.

NOTE – For the input parameter cRiceParam = 0, the TR binarization is exactly a truncated unary binarization and it is always invoked with a cMax value equal to the largest possible value of the syntax element being decoded.

9.3.3.4 Truncated binary (TB) binarization process

Input to this process is a request for a TB binarization for a syntax element with value synVal and cMax. Output of this process is the TB binarization of the syntax element.The bin string of the TB binarization process of a syntax element synVal is specified as follows:

$$\begin{aligned} &n = \text{cMax} + 1 \\ &k = \text{Floor}(\ \text{Log2}(\ n\ )\ ) \\ &u = (\ 1 \ll (\ k + 1\ )\ ) - n \end{aligned} \quad (1533)$$

– If synVal is less than u, the TB bin string is derived by invoking the FL binarization process specified in clause 9.3.3.7 for synVal with a cMax value equal to (1 << k) − 1.

– Otherwise (synVal is greater than or equal to u), the TB bin string is derived by invoking the FL binarization process specified in clause 9.3.3.7 for (synVal + u) with a cMax value equal to (1 << (k + 1)) − 1.

9.3.3.5 k-th order Exp-Golomb binarization process

Inputs to this process is a request for a k-th order Exp-Golomb (EGk) binarization.

Output of this process is the EGk binarization associating each value symbolVal with a corresponding bin string.

The bin string of the EGk binarization process for each value symbolVal is specified as follows, where 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:

```
absV = Abs( symbolVal )
stopLoop = 0
do
    if( absV >= ( 1 << k ) ) {
        put( 1 )
        absV = absV − ( 1 << k )
        k++
    } else {
        put( 0 )                                        (1534)
        while( k− − )
```

```
            put( ( absV  >>  k ) & 1 )
        stopLoop = 1
    }
while( !stopLoop )
```

NOTE – The specification for the k-th order Exp-Golomb (EGk) code uses 1s and 0s in reverse meaning for the unary part of the Exp-Golomb code of k-th order as specified in clause 9.2.

9.3.3.6 Limited k-th order Exp-Golomb binarization process

Inputs to this process is a request for a limited k-th order Exp-Golomb (EGk) binarization, the order k, the variables maxPreExtLen and truncSuffixLen.

Output of this process is the limited EGk binarization associating each value symbolVal with a corresponding bin string.

The bin string of the limited EGk binarization process for each value symbolVal is specified as follows, where 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:

```
codeValue = symbolVal  >>  k
preExtLen = 0
while( ( preExtLen < maxPreExtLen )  &&  ( codeValue > ( ( 2  <<  preExtLen ) − 2 ) ) ) {
    preExtLen++
    put( 1 )
}
if( preExtLen  = =  maxPreExtLen )                                              (1535)
    escapeLength = truncSuffixLen
else {
    escapeLength = preExtLen + k
    put( 0 )
}
symbolVal = symbolVal − ( ( ( 1  <<  preExtLen ) − 1 )  <<  k )
while( ( escapeLength− − ) > 0 )
    put( ( symbolVal  >>  escapeLength ) & 1 )
```

9.3.3.7 Fixed-length binarization process

Inputs to this process are a request for a fixed-length (FL) binarization and cMax.

Output of this process is the FL binarization associating each value symbolVal with a corresponding bin string.

FL binarization is constructed by using the fixedLength-bit unsigned integer bin string of the symbol value symbolVal, where fixedLength = Ceil(Log2(cMax + 1)). The indexing of bins for the FL binarization is such that the binIdx = 0 relates to the most significant bit with increasing values of binIdx towards the least significant bit.

9.3.3.8 Binarization process for intra_chroma_pred_mode

Input to this process is a request for a binarization for the syntax element intra_chroma_pred_mode.

Output of this process is the binarization of the syntax element.

The binarization for the syntax element intra_chroma_pred_mode is specified in Table 130.

Table 130 – Binarization for intra_chroma_pred_mode

Value of intra_chroma_pred_mode	Bin string
0	100
1	101
2	110
3	111
4	0

9.3.3.9 Binarization process for inter_pred_idc

Input to this process is a request for a binarization for the syntax element inter_pred_idc, the current luma coding block width cbWidth and the current luma coding block height cbHeight.

Output of this process is the binarization of the syntax element.

The binarization for the syntax element inter_pred_idc is specified in Table 131.

Table 131 – Binarization for inter_pred_idc

Value of inter_pred_idc	Name of inter_pred_idc	Bin string	
		(cbWidth + cbHeight) > 12	(cbWidth + cbHeight) = = 12
0	PRED_L0	00	0
1	PRED_L1	01	1
2	PRED_BI	1	-

9.3.3.10 Binarization process for cu_qp_delta_abs

Input to this process is a request for a binarization for the syntax element cu_qp_delta_abs.

Output of this process is the binarization of the syntax element.

The binarization of the syntax element cu_qp_delta_abs is a concatenation of a prefix bin string and (when present) a suffix bin string.

For the derivation of the prefix bin string, the following applies:

– The prefix value of cu_qp_delta_abs, prefixVal, is derived as follows:

$$\text{prefixVal} = \text{Min}(\text{ cu\_qp\_delta\_abs, 5 })\tag{1536}$$

– The prefix bin string is specified by invoking the TR binarization process as specified in clause 9.3.3.3 for prefixVal with cMax = 5 and cRiceParam = 0.

When prefixVal is greater than 4, the suffix bin string is present and it is derived as follows:

– The suffix value of cu_qp_delta_abs, suffixVal, is derived as follows:

$$\text{suffixVal} = \text{cu\_qp\_delta\_abs} - 5\tag{1537}$$

– The suffix bin string is specified by invoking the k-th order EGk binarization process as specified in clause 9.3.3.5 for suffixVal with the Exp-Golomb order k set equal to 0.

9.3.3.11 Binarization process for abs_remainder[ ]

Input to this process is a request for a binarization for the syntax element abs_remainder[n], the colour component cIdx, the current sub-block index i, and the luma location (x0, y0) specifying the top-left sample of the current luma transform block relative to the top-left luma sample of the picture, and the current coefficient scan location (xC, yC).

Output of this process is the binarization of the syntax element.

The variables lastAbsRemainder and lastRiceParam are derived as follows:

– If this process is invoked for the first time for the current sub-block index i, lastAbsRemainder and lastRiceParam are both set equal to 0.

– Otherwise (this process is not invoked for the first time for the current sub-block index i), lastAbsRemainder and lastRiceParam are set equal to the values of abs_remainder[n] and cRiceParam, respectively, that have been derived during the last invocation of the binarization process for the syntax element abs_remainder[n] as specified in this clause.

The variable baseLevel is derived as follows:

– If sps_rrc_rice_extension_flag is equal to 0, baseLevel is set equal to 4.

– Otherwise (sps_rrc_rice_extension_flag is equal to 1), baseLevel is set as follows:

$$\text{baseLevel} = (\text{BitDepth} > 12) ? ((\text{sh\_slice\_type} == \text{I}) ? 1 : 2) : ((\text{sh\_slice\_type} == \text{I}) ? 2 : 3) \quad (1538)$$

The Rice parameter cRiceParam is derived as follows:

– If transform_skip_flag[x0][y0][cIdx] is equal to 1 and sh_ts_residual_coding_disabled_flag is equal to 0, the Rice parameter cRiceParam is set equal to sh_ts_residual_coding_rice_idx_minus1 + 1.

– Otherwise, the Rice parameter cRiceParam is derived by invoking the Rice parameter derivation process for abs_remainder[ ] as specified in clause 9.3.3.2 with baseLevel, the colour component index cIdx, the luma location (x0, y0), and the current coefficient scan location (xC, yC) as inputs.

The variable cMax is derived from cRiceParam as:

$$\text{cMax} = 6 \ll \text{cRiceParam} \quad (1539)$$

The binarization of the syntax element abs_remainder[n] is a concatenation of a prefix bin string and (when present) a suffix bin string.

For the derivation of the prefix bin string, the following applies:

– The prefix value of abs_remainder[n], prefixVal, is derived as follows:

$$\text{prefixVal} = \text{Min}(\text{cMax}, \text{abs\_remainder}[n]) \quad (1540)$$

– The prefix bin string is specified by invoking the TR binarization process as specified in clause 9.3.3.3 for prefixVal with the variables cMax and cRiceParam as inputs.

When the prefix bin string is equal to the bit string of length 6 with all bits equal to 1, the suffix bin string is present and it is derived as follows:

– The suffix value of abs_remainder[n], suffixVal, is derived as follows:

$$\text{suffixVal} = \text{abs\_remainder}[n] - \text{cMax} \quad (1541)$$

– The suffix bin string is specified by invoking the limited k-th order EGk binarization process as specified in clause 9.3.3.6 for the binarization of suffixVal with the Exp-Golomb order k set equal to cRiceParam + 1, the variable maxPreExtLen set equal to 26 − Log2TransformRange and the variable truncSuffixLen set equal to Log2TransformRange as inputs.

9.3.3.12 Binarization process for dec_abs_level[ ]

Input to this process is a request for a binarization of the syntax element dec_abs_level[n], the colour component cIdx, the luma location (x0, y0) specifying the top-left sample of the current transform block relative to the top-left luma sample of the picture, and the current coefficient scan location (xC, yC).

Output of this process is the binarization of the syntax element.

The Rice parameter cRiceParam is derived by invoking the Rice parameter derivation process for dec_abs_level[ ] as specified in clause 9.3.3.2 with the variable baseLevel set equal to 0, the colour component index cIdx, the luma location (x0, y0), and the current coefficient scan location (xC, yC) as inputs.

The variable cMax is derived from cRiceParam as:

$$\text{cMax} = 6 \ll \text{cRiceParam} \quad (1542)$$

The binarization of dec_abs_level[n] is a concatenation of a prefix bin string and (when present) a suffix bin string.

For the derivation of the prefix bin string, the following applies:

– The prefix value of dec_abs_level[n], prefixVal, is derived as follows:

$$\text{prefixVal} = \text{Min}(\text{cMax}, \text{dec\_abs\_level}[n]) \quad (1543)$$

– The prefix bin string is specified by invoking the TR binarization process as specified in clause 9.3.3.3 for prefixVal with the variables cMax and cRiceParam as inputs.

When the prefix bin string is equal to the bit string of length 6 with all bits equal to 1, the suffix bin string is present and it is derived as follows:

– The suffix value of dec_abs_level[n], suffixVal, is derived as follows:

$$\text{suffixVal} = \text{dec\_abs\_level}[\ n\ ] - \text{cMax} \quad (1544)$$

– The suffix bin string is specified by invoking the limited k-th order EGk binarization process as specified in clause 9.3.3.6 for the binarization of suffixVal with the Exp-Golomb order k set equal to cRiceParam + 1, the variable maxPreExtLen set equal to 26 − Log2TransformRange and the variable truncSuffixLen set equal to Log2TransformRange as inputs.

9.3.3.13 Binarization process for palette_idx_idc

Input to this process is a request for a binarization for the syntax element palette_idx_idc and the variable MaxPaletteIndex.

Output of this process is the binarization of the syntax element.

The variable cMax is derived as follows:

– If this process is invoked for the first time for the current block, cMax is set equal to MaxPaletteIndex.

– Otherwise (this process is not invoked for the first time for the current block), cMax is set equal to MaxPaletteIndex minus 1.

The binarization for the palette_idx_idc is derived by invoking the TB binarization process specified in clause 9.3.3.4 with cMax.

9.3.3.14 Binarization process for abs_mvd_minus2

Input to this process is a request for a binarization for the syntax element abs_mvd_minus2.

Output of this process is the binarization of the syntax element.

The abs_mvd_minus2 bin string is specified by invoking the limited k-th order EGk binarization process as specified in clause 9.3.3.6 with Exp-Golomb order k set equal to 1, the variable maxPreExtLen set equal to 15 and the variable truncSuffixLen set equal to 17 as inputs.

NOTE – The binarization is equivalent to an EG1 binarization for values of abs_mvd_minus2 less than or equal to $2^{17} - 3$. The bin string corresponding to the greatest value $2^{17} - 2$ for abs_mvd_minus2 is "1111 1111 1111 1111 0000 0000 0000 0000".

9.3.4 Decoding process flow

9.3.4.1 General

Inputs to this process are all bin strings of the binarization of the requested syntax element as specified in clause 9.3.3.

Output of this process is the value of the syntax element.

This process specifies how each bin of a bin string is parsed for each syntax element. After parsing each bin, the resulting bin string is compared to all bin strings of the binarization of the syntax element and the following applies:

– If the bin string is equal to one of the bin strings, the corresponding value of the syntax element is the output.

– Otherwise (the bin 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. The derivation process for ctxTable, ctxIdx, and bypassFlag as specified in clause 9.3.4.2 is invoked with binIdx as input and ctxTable, ctxIdx and bypassFlag as outputs.

2. The arithmetic decoding process as specified in clause 9.3.4.3 is invoked with ctxTable, ctxIdx and bypassFlag as inputs and the value of the bin as output.

9.3.4.2 Derivation process for ctxTable, ctxIdx and bypassFlag

9.3.4.2.1 General

Input to this process is the position of the current bin within the bin string, binIdx.

Outputs of this process are ctxTable, ctxIdx and bypassFlag.

The values of ctxTable, ctxIdx and bypassFlag are derived as follows based on the entries for binIdx of the corresponding syntax element in Table 132:

– If the entry in Table 132 is not equal to any of "bypass", "terminate" and "na", the values of binIdx are decoded by invoking the DecodeDecision process as specified in clause 9.3.4.3.2 and the following applies:
  – ctxTable is specified in Table 51
  – The variable ctxInc is specified by the corresponding entry in Table 132 and when more than one value is listed in Table 132 for a binIdx, the assignment process for ctxInc for that binIdx is further specified in the clauses given in parenthesis.
  – The variable ctxIdxOffset set equal to the smallest value of ctxIdx is specified in Table 51 for the current value of initType and the current syntax element.
  – ctxIdx is set equal to the sum of ctxInc and ctxIdxOffset.
  – bypassFlag is set equal to 0.
– Otherwise, if the entry in Table 132 is equal to "bypass", the values of binIdx are decoded by invoking the DecodeBypass process as specified in clause 9.3.4.3.4 and the following applies:
  – ctxTable is set equal to 0.
  – ctxIdx is set equal to 0.
  – bypassFlag is set equal to 1.
– Otherwise, if the entry in Table 132 is equal to "terminate", the values of binIdx are decoded by invoking the DecodeTerminate process as specified in clause 9.3.4.3.5 and the following applies:
  – ctxTable is set equal to 0.
  – ctxIdx is set equal to 0.
  – bypassFlag is set equal to 0.
– Otherwise (the entry in Table 132 is equal to "na"), the values of binIdx do not occur for the corresponding syntax element.

Table 132 – Assignment of ctxInc to syntax elements with context coded bins

Syntax element	binIdx					
	0	1	2	3	4	>= 5
end_of_slice_one_bit	terminate	na	na	na	na	na
end_of_tile_one_bit	terminate	na	na	na	na	na
end_of_subset_one_bit	terminate	na	na	na	na	na
alf_ctb_flag[ ][ ][ ]	0..8 (clause 9.3.4.2.2)	na	na	na	na	na
alf_use_aps_flag	0	na	na	na	na	na
alf_luma_fixed_filter_idx	bypass	bypass	bypass	bypass	bypass	bypass
alf_luma_prev_filter_idx	bypass	bypass	bypass	bypass	bypass	bypass
alf_ctb_filter_alt_idx[0][ ][ ]	0	0	0	0	0	0
alf_ctb_filter_alt_idx[1][ ][ ]	1	1	1	1	1	1
alf_ctb_cc_cb_idc[ ][ ]	0..2 (clause 9.3.4.2.2)	bypass	bypass	bypass	bypass	bypass
alf_ctb_cc_cr_idc[ ][ ]	0..2 (clause 9.3.4.2.2)	bypass	bypass	bypass	bypass	bypass
sao_merge_left_flag	0	na	na	na	na	na
sao_merge_up_flag	0	na	na	na	na	na
sao_type_idx_luma	0	bypass	na	na	na	na

Table 132 – Assignment of ctxInc to syntax elements with context coded bins

Syntax element	binIdx					
	0	1	2	3	4	>= 5
sao_type_idx_chroma	0	bypass	na	na	na	na
sao_offset_abs[ ][ ][ ][ ]	bypass	bypass	bypass	bypass	bypass	na
sao_offset_sign_flag[ ][ ][ ][ ]	bypass	na	na	na	na	na
sao_band_position[ ][ ][ ]	bypass	bypass	bypass	bypass	bypass	bypass
sao_eo_class_luma	bypass	bypass	na	na	na	na
sao_eo_class_chroma	bypass	bypass	na	na	na	na
split_cu_flag	0..8 (clause 9.3.4.2.2)	na	na	na	na	na
split_qt_flag	0..5 (clause 9.3.4.2.2)	na	na	na	na	na
mtt_split_cu_vertical_flag	0..4 (clause 9.3.4.2.3)	na	na	na	na	na
mtt_split_cu_binary_flag	(2 * mtt_split_cu_vertical_flag) + (mttDepth <= 1 ? 1 : 0)	na	na	na	na	na
non_inter_flag	0,1 (clause 9.3.4.2.2)	na	na	na	na	na
cu_skip_flag[ ][ ]	0,1,2 (clause 9.3.4.2.2)	na	na	na	na	na
pred_mode_flag	0,1 (clause 9.3.4.2.2)	na	na	na	na	na
pred_mode_ibc_flag	0,1,2 (clause 9.3.4.2.2)	na	na	na	na	na
pred_mode_plt_flag	0	na	na	na	na	na
cu_act_enabled_flag	0	na	na	na	na	na
intra_bdpcm_luma_flag	0	na	na	na	na	na
intra_bdpcm_luma_dir_flag	0	na	na	na	na	na
intra_mip_flag	(Abs(Log2(cbWidth) − Log2(cbHeight)) > 1) ? 3 : (0,1,2 (clause 9.3.4.2.2))	na	na	na	na	na
intra_mip_transposed_flag[ ][ ]	bypass	na	na	na	na	na
intra_mip_mode[ ][ ]	bypass	bypass	bypass	bypass	bypass	na
intra_luma_ref_idx	0	1	na	na	na	na
intra_subpartitions_mode_flag	0	na	na	na	na	na
intra_subpartitions_split_flag	0	na	na	na	na	na
intra_luma_mpm_flag[ ][ ]	0	na	na	na	na	na
intra_luma_not_planar_flag[ ][ ]	!intra_subpartitions_mode_flag	na	na	na	na	na
intra_luma_mpm_idx[ ][ ]	bypass	bypass	bypass	bypass	na	na
intra_luma_mpm_remainder[ ][ ]	bypass	bypass	bypass	bypass	bypass	bypass
intra_bdpcm_chroma_flag	0	na	na	na	na	na
intra_bdpcm_chroma_dir_flag	0	na	na	na	na	na
cclm_mode_flag	0	na	na	na	na	na
cclm_mode_idx	0	bypass	na	na	na	na
intra_chroma_pred_mode	0	bypass	bypass	na	na	na

Table 132 – Assignment of ctxInc to syntax elements with context coded bins

Syntax element	binIdx					
	0	**1**	**2**	**3**	**4**	**>= 5**
palette_predictor_run	bypass	bypass	bypass	bypass	bypass	bypass
num_signalled_palette_entries	bypass	bypass	bypass	bypass	bypass	bypass
new_palette_entries	bypass	bypass	bypass	bypass	bypass	bypass
palette_escape_val_present_flag	bypass	na	na	na	na	na
palette_transpose_flag	0	na	na	na	na	na
palette_idx_idc	bypass	bypass	bypass	bypass	bypass	bypass
copy_above_palette_indices_flag	0	na	na	na	na	na
run_copy_flag	0..7 (clause 9.3.4.2.11)	na	na	na	na	na
palette_escape_val	bypass	bypass	bypass	bypass	bypass	bypass
general_merge_flag[ ][ ]	0	na	na	na	na	na
regular_merge_flag[ ][ ]	cu_skip_flag[ ][ ] ? 0 : 1	na	na	na	na	na
mmvd_merge_flag[ ][ ]	0	na	na	na	na	na
mmvd_cand_flag[ ][ ]	0	na	na	na	na	na
mmvd_distance_idx[ ][ ]	0	bypass	bypass	bypass	bypass	bypass
mmvd_direction_idx[ ][ ]	bypass	bypass	na	na	na	na
merge_subblock_flag[ ][ ]	0,1,2 (clause 9.3.4.2.2)	na	na	na	na	na
merge_subblock_idx[ ][ ]	0	bypass	bypass	bypass	bypass	na
ciip_flag[ ][ ]	0	na	na	na	na	na
merge_idx[ ][ ]	0	bypass	bypass	bypass	bypass	na
merge_gpm_partition_idx[ ][ ]	bypass	bypass	bypass	bypass	bypass	bypass
merge_gpm_idx0[ ][ ]	0	bypass	bypass	bypass	bypass	na
merge_gpm_idx1[ ][ ]	0	bypass	bypass	bypass	na	na
inter_pred_idc[ ][ ]	(cbWidth + cbHeight) > 12 ? 7 − ((1 + Log2(cbWidth) + Log2(cbHeight)) >> 1) : 5	5	na	na	na	na
inter_affine_flag[ ][ ]	0,1,2 (clause 9.3.4.2.2)	na	na	na	na	na
cu_affine_type_flag[ ][ ]	0	na	na	na	na	na
sym_mvd_flag[ ][ ]	0	na	na	na	na	na
ref_idx_l0[ ][ ]	0	1	bypass	bypass	bypass	bypass
ref_idx_l1[ ][ ]	0	1	bypass	bypass	bypass	bypass
mvp_l0_flag[ ][ ]	0	na	na	na	na	na
mvp_l1_flag[ ][ ]	0	na	na	na	na	na
amvr_flag[ ][ ]	inter_affine_flag[ ][ ] ? 1 : 0	na	na	na	na	na
amvr_precision_idx[ ][ ]	(CuPredMode[0][x0][y0] = = MODE_IBC) ? 1 : (inter_affine_flag = = 0 ? 0 : 2)	1	na	na	na	na
bcw_idx[ ][ ] NoBackwardPredFlag = = 0	0	bypass	na	na	na	na
bcw_idx[ ][ ] NoBackwardPredFlag = = 1	0	bypass	bypass	bypass	na	na

Table 132 – Assignment of ctxInc to syntax elements with context coded bins

Syntax element	binIdx					
	0	**1**	**2**	**3**	**4**	**>= 5**
cu_coded_flag	0	na	na	na	na	na
cu_sbt_flag	(cbWidth * cbHeight <= 256) ? 1 : 0	na	na	na	na	na
cu_sbt_quad_flag	0	na	na	na	na	na
cu_sbt_horizontal_flag	(cbWidth = = cbHeight) ? 0 : (cbWidth < cbHeight) ? 1 : 2	na	na	na	na	na
cu_sbt_pos_flag	0	na	na	na	na	na
lfnst_idx	(treeType != SINGLE_TREE) ? 1 : 0	2	na	na	na	na
mts_idx	0	1	2	3	na	na
abs_mvd_greater0_flag[ ]	0	na	na	na	na	na
abs_mvd_greater1_flag[ ]	0	na	na	na	na	na
abs_mvd_minus2[ ]	bypass	bypass	bypass	bypass	bypass	bypass
mvd_sign_flag[ ]	bypass	na	na	na	na	na
tu_y_coded_flag[ ][ ]	0,1,2,3 (clause 9.3.4.2.5)	na	na	na	na	na
tu_cb_coded_flag[ ][ ]	intra_bdpcm_chroma_flag ? 1 : 0	na	na	na	na	na
tu_cr_coded_flag[ ][ ]	intra_bdpcm_chroma_flag ? 2 : tu_cb_coded_flag[ ][ ]	na	na	na	na	na
cu_qp_delta_abs	0	1	1	1	1	bypass
cu_qp_delta_sign_flag	bypass	na	na	na	na	na
cu_chroma_qp_offset_flag	0	na	na	na	na	na
cu_chroma_qp_offset_idx	0	0	0	0	0	na
transform_skip_flag[ ][ ][cIdx]	cIdx = = 0 ? 0 : 1	na	na	na	na	na
tu_joint_cbcr_residual_flag[ ][ ]	2 * tu_cb_coded_flag[ ][ ] + tu_cr_coded_flag[ ][ ] − 1	na	na	na	na	na
last_sig_coeff_x_prefix	0..22 (clause 9.3.4.2.4)					
last_sig_coeff_y_prefix	0..22 (clause 9.3.4.2.4)					
last_sig_coeff_x_suffix	bypass	bypass	bypass	bypass	bypass	bypass
last_sig_coeff_y_suffix	bypass	bypass	bypass	bypass	bypass	bypass
sb_coded_flag[ ][ ]	0..6 (clause 9.3.4.2.6)	na	na	na	na	na
sig_coeff_flag[ ][ ]	0..62 (clause 9.3.4.2.8)	na	na	na	na	na
par_level_flag[ ]	0..32 (clause 9.3.4.2.9)	na	na	na	na	na
abs_level_gtx_flag[ ][ ]	0..71 (clause 9.3.4.2.9)	na	na	na	na	na
abs_remainder[ ]	bypass	bypass	bypass	bypass	bypass	bypass
dec_abs_level[ ]	bypass	bypass	bypass	bypass	bypass	bypass
coeff_sign_flag[ ] transform_skip_flag[x0][y0][cIdx] = = 0 || n > lastScanPosPass1 || sh_ts_residual_coding_disabled_flag	bypass	na	na	na	na	na

Table 132 – Assignment of ctxInc to syntax elements with context coded bins

Syntax element	binIdx					
	0	1	2	3	4	>= 5
coeff_sign_flag[ ] transform_skip_flag[x0][y0][cIdx] = = 1 && n <= lastScanPosPass1 && !sh_ts_residual_coding_disabled_flag	0..5 (clause 9.3.4.2.10)	na	na	na	na	na

9.3.4.2.2 Derivation process of ctxInc using left and above syntax elements

For the syntax elements alf_ctb_flag[cIdx][ctbX][ctbY], alf_ctb_cc_cb_idc[ctbX][ctbY], and alf_ctb_cc_cr_idc[ctbX][ctbY], input to this process is the luma location (x0, y0) set equal to (xCtb, yCtb) specifying the top-left luma sample of the current coding tree unit relative to the top-left sample of the current picture, the colour component cIdx for alf_ctb_flag[cIdx][ctbX][ctbY], cIdx equal to 1 for alf_ctb_cc_cr_idc[ctbX][ctbY] and cIdx equal to 2 for alf_ctb_cc_cr_idc[ctbX][ctbY]. The location (ctbX, ctbY) is set equal to (x0 >> CtbLog2SizeY, y0 >> CtbLog2SizeY), the location (ctbAx, ctbAy) is set equal to (x0 >> CtbLog2SizeY, (y0 − 1) >> CtbLog2SizeY), and the location (ctbLx, ctbLy) is set equal to ((x0 − 1) >> CtbLog2SizeY, y0 >> CtbLog2SizeY).

For the syntax elements split_qt_flag, split_cu_flag, and non_inter_flag, input to this process is the luma location (x0, y0) specifying the top-left luma sample of the current luma block relative to the top-left sample of the current picture, the current coding quadtree depth cqtDepth for split_qt_flag, the width and the height of the current coding block in luma samples cbWidth and cbHeight, and the variables allowSplitBtVer, allowSplitBtHor, allowSplitTtVer, allowSplitTtHor, and allowSplitQt as derived in the coding tree semantics in clause 7.4.12.4 for split_cu_flag. The dual tree channel type chType is set equal to 1 if the variable treeType in the associated coding tree syntax is equal to DUAL_TREE_CHROMA, and set equal to 0 otherwise. The colour component cIdx is set equal to chType.

For the syntax elements cu_skip_flag[x0][y0], pred_mode_flag[x0][y0], pred_mode_ibc_flag[x0][y0], intra_mip_flag, inter_affine_flag[x0][y0] and merge_subblock_flag[x0][y0], input to this process is the luma location (x0, y0) specifying the top-left luma sample of the current luma block relative to the top-left sample of the current picture, and the dual tree channel type chType for pred_mode_flag[x0][y0] and pred_mode_ibc_flag[x0][y0]. The colour component cIdx is set equal to chType.

Output of this process is ctxInc.

The location (xNbL, yNbL) is set equal to (x0 − 1, y0) and the derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location (xCurr, yCurr) set equal to (x0, y0), the neighbouring location (xNbY, yNbY) set equal to (xNbL, yNbL), checkPredModeY set equal to FALSE, and cIdx as inputs, and the output is assigned to availableL.

The location (xNbA, yNbA) is set equal to (x0, y0 − 1) and the derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location (xCurr, yCurr) set equal to (x0, y0), the neighbouring location (xNbY, yNbY) set equal to (xNbA, yNbA), checkPredModeY set equal to FALSE, and cIdx as inputs, and the output is assigned to availableA.

The assignment of ctxInc is specified as follows with condL and condA specified in Table 133:

– For the syntax elements alf_ctb_flag[cIdx][ctbX][ctbY], alf_ctb_cc_cb_idc[ctbX][ctbY], alf_ctb_cc_cr_idc[ctbX][ctbY], split_qt_flag, split_cu_flag, cu_skip_flag[x0][y0], pred_mode_ibc_flag[x0][y0], intra_mip_flag, inter_affine_flag[x0][y0] and merge_subblock_flag[x0][y0]:

ctxInc = (condL && availableL) + (condA && availableA) + ctxSetIdx * 3 (1545)

– For the syntax elements pred_mode_flag[x0][y0] and non_inter_flag:

ctxInc = (condL && availableL) | | (condA && availableA) (1546)

Table 133 – Specification of ctxInc using left and above syntax elements

Syntax element	condL	condA	ctxSetIdx
alf_ctb_flag[cIdx][ctbX][ctbY]	alf_ctb_flag[cIdx][ctbLx][ctbLy]	alf_ctb_flag[cIdx][ctbAx][ctbAy]	cIdx
alf_ctb_cc_cb_idc[ctbX][ctbY]	alf_ctb_cc_cb_idc[ctbLx][ctbLy]	alf_ctb_cc_cb_idc[ctbAx][ctbAy]	0
alf_ctb_cc_cr_idc[ctbX][ctbY]	alf_ctb_cc_cr_idc[ctbLx][ctbLy]	alf_ctb_cc_cr_idc[ctbAx][ctbAy]	0
split_qt_flag	CqtDepth[chType][xNbL][yNbL] > cqtDepth	CqtDepth[chType][xNbA][yNbA] > cqtDepth	cqtDepth >= 2
split_cu_flag	CbHeight[chType][xNbL][yNbL] < cbHeight	CbWidth[chType][xNbA][yNbA] < cbWidth	(allowSplitBtVer + allowSplitBtHor + allowSplitTtVer + allowSplitTtHor + 2 * allowSplitQt − 1) / 2
non_inter_flag	CuPredMode[chType][xNbL][yNbL] = = MODE_INTRA	CuPredMode[chType][xNbA][yNbA] = = MODE_INTRA	0
cu_skip_flag[x0][y0]	CuSkipFlag[xNbL][yNbL]	CuSkipFlag[xNbA][yNbA]	0
pred_mode_flag[x0][y0]	CuPredMode[chType][xNbL][yNbL] = = MODE_INTRA	CuPredMode[chType][xNbA][yNbA] = = MODE_INTRA	0
pred_mode_ibc_flag[x0][y0]	CuPredMode[chType][xNbL][yNbL] = = MODE_IBC	CuPredMode[chType][xNbA][yNbA] = = MODE_IBC	0
intra_mip_flag	IntraMipFlag[xNbL][yNbL]	IntraMipFlag[xNbA][yNbA]	0
merge_subblock_flag[x0][y0]	MergeSubblockFlag[xNbL][yNbL] || InterAffineFlag[xNbL][yNbL]	MergeSubblockFlag[xNbA][yNbA] || InterAffineFlag[xNbA][yNbA]	0
inter_affine_flag [x0][y0]	MergeSubblockFlag[xNbL][yNbL] || InterAffineFlag[xNbL][yNbL]	MergeSubblockFlag[xNbA][yNbA] || InterAffineFlag[xNbA][yNbA]	0

9.3.4.2.3 Derivation process of ctxInc for the syntax element mtt_split_cu_vertical_flag

Inputs to this process are the luma location (x0, y0) specifying the top-left luma sample of the current luma block relative to the top-left sample of the current picture, the dual tree channel type chType, the colour component index cIdx, the width and the height of the current coding block in luma samples cbWidth and cbHeight, and the variables allowSplitBtVer, allowSplitBtHor, allowSplitTtVer, and allowSplitTtHor as derived in the coding tree semantics in clause 7.4.12.4.

Output of this process is ctxInc.

The location (xNbL, yNbL) is set equal to (x0 − 1, y0) and the derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location (xCurr, yCurr) set equal to (x0, y0) and the neighbouring location (xNbY, yNbY) set equal to (xNbL, yNbL), checkPredModeY set equal to FALSE, and cIdx as inputs, and the output is assigned to availableL.

The location (xNbA, yNbA) is set equal to (x0, y0 − 1) and the derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location (xCurr, yCurr) set equal to (x0, y0), the neighbouring location (xNbY, yNbY) set equal to (xNbA, yNbA), checkPredModeY set equal to FALSE, and cIdx as inputs, and the output is assigned to availableA.

The assignment of ctxInc is specified as follows:

– If allowSplitBtVer + allowSplitTtVer is greater than allowSplitBtHor + allowSplitTtHor, ctxInc is set equal to 4.

– Otherwise, if allowSplitBtVer + allowSplitTtVer is less than allowSplitBtHor + allowSplitTtHor, ctxInc is set equal to 3.

– Otherwise, the following applies:

  – The variables dA and dL are derived as follows

$$dA = cbWidth / (availableA ? CbWidth[chType][xNbA][yNbA] : 1) \quad (1547)$$

$$dL = cbHeight / (availableL ? CbHeight[chType][xNbL][yNbL] : 1) \quad (1548)$$

– If one or more of the following conditions are true, ctxInc is set equal to 0:
  – dA is equal to dL,
  – availableA is equal to FALSE,
  – availableL is equal to FALSE.
– Otherwise, if dA is less than dL, ctxInc is set equal to 1.
– Otherwise, ctxInc is set equal to 2.

9.3.4.2.4 Derivation process of ctxInc for the syntax elements last_sig_coeff_x_prefix and last_sig_coeff_y_prefix

Inputs to this process are the variable binIdx and the colour component index cIdx.

Output of this process is the variable ctxInc.

The variable log2TbSize is derived as follows:

– If the syntax element to be parsed is last_sig_coeff_x_prefix, log2TbSize is set equal to Log2FullTbWidth.
– Otherwise (the syntax element to be parsed is last_sig_coeff_y_prefix), log2TbSize is set equal to Log2FullTbHeight.

The variables ctxOffset and ctxShift are derived as follows:

– If cIdx is equal to 0, ctxOffset is set equal to offsetY[log2TbSize − 1] and ctxShift is set equal to (log2TbSize + 1) >> 2 with the list offsetY specified as follows:

$$\text{offsetY[ ]} = \{0, 0, 3, 6, 10, 15\} \tag{1549}$$

– Otherwise (cIdx is greater than 0), ctxOffset is set equal to 20 and ctxShift is set equal to Clip3(0, 2, $2^{\text{log2TbSize}}$ >> 3).

The variable ctxInc is derived as follows:

$$\text{ctxInc} = (\text{binIdx} >> \text{ctxShift}) + \text{ctxOffset} \tag{1550}$$

9.3.4.2.5 Derivation process of ctxInc for the syntax element tu_y_coded_flag

Input to this process is the luma location (x0, y0) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture.

Output of this process is the variable ctxInc.

The variable ctxInc is derived as follows:

– If BdpcmFlag[x0][y0][0] is equal to 1, ctxInc is set equal to 1.
– Otherwise, if IntraSubPartitionsSplitType is equal to ISP_NO_SPLIT, ctxInc is set equal to 0.
– Otherwise (BdpcmFlag[x0][y0][0] is equal to 0 and IntraSubPartitionsSplitType is not equal to ISP_NO_SPLIT), the following applies:
  – The variable prevTuCbfY is derived as follows:
    – If the current transform unit is the first one to be parsed in a coding unit, prevTuCbfY is set equal to 0.
    – Otherwise, prevTuCbfY is set equal to the value of tu_y_coded_flag of the previous luma transform unit in the current coding unit.
  – The variable ctxInc is derived as follows:

$$\text{ctxInc} = 2 + \text{prevTuCbfY} \tag{1551}$$

9.3.4.2.6 Derivation process of ctxInc for the syntax element sb_coded_flag

Inputs to this process are the colour component index cIdx, the luma location (x0, y0) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture, the current sub-block scan location (xS, yS), and the previously decoded bins of the syntax element sb_coded_flag.

Output of this process is the variable ctxInc.

The variable csbfCtx is derived using the current location (xS, yS), two previously decoded bins of the syntax element sb_coded_flag in scan order, Log2ZoTbWidth and Log2ZoTbHeight, as follows:

– The variables log2SbWidth and log2SbHeight are derived as follows:

$$\text{log2SbWidth} = (\ \text{Min}(\ \text{Log2ZoTbWidth},\ \text{Log2ZoTbHeight}\ ) < 2\ ?\ 1 : 2\ ) \quad (1552)$$

$$\text{log2SbHeight} = \text{log2SbWidth} \quad (1553)$$

– The variables log2SbWidth and log2SbHeight are modified as follows:

  – If Log2ZoTbWidth is less than 2 and cIdx is equal to 0, the following applies:

$$\text{log2SbWidth} = \text{Log2ZoTbWidth} \quad (1554)$$

$$\text{log2SbHeight} = 4 - \text{log2SbWidth} \quad (1555)$$

  – Otherwise, if Log2ZoTbHeight is less than 2 and cIdx is equal to 0, the following applies:

$$\text{log2SbHeight} = \text{Log2ZoTbHeight} \quad (1556)$$

$$\text{log2SbWidth} = 4 - \text{log2SbHeight} \quad (1557)$$

– The variable csbfCtx is initialized with 0 and modified as follows:

  – If transform_skip_flag[x0][y0][cIdx] is equal to 1 and sh_ts_residual_coding_disabled_flag is equal to 0, the following applies:

    – When xS is greater than 0, csbfCtx is modified as follows:

$$\text{csbfCtx}\ \mathrel{+}=\ \text{sb\_coded\_flag}[\ \text{xS} - 1\ ][\ \text{yS}\ ] \quad (1558)$$

    – When yS is greater than 0, csbfCtx is modified as follows:

$$\text{csbfCtx}\ \mathrel{+}=\ \text{sb\_coded\_flag}[\ \text{xS}\ ][\ \text{yS} - 1\ ] \quad (1559)$$

  – Otherwise (transform_skip_flag[x0][y0][cIdx] is equal to 0 or sh_ts_residual_coding_disabled_flag is equal to 1), the following applies:

    – When xS is less than (1 << (Log2ZoTbWidth − log2SbWidth)) − 1, csbfCtx is modified as follows:

$$\text{csbfCtx}\ \mathrel{+}=\ \text{sb\_coded\_flag}[\ \text{xS} + 1\ ][\ \text{yS}\ ] \quad (1560)$$

    – When yS is less than (1 << (Log2ZoTbHeight − log2SbHeight)) − 1, csbfCtx is modified as follows:

$$\text{csbfCtx}\ \mathrel{+}=\ \text{sb\_coded\_flag}[\ \text{xS}\ ][\ \text{yS} + 1\ ] \quad (1561)$$

The context index increment ctxInc is derived using the colour component index cIdx and csbfCtx as follows:

– If transform_skip_flag[x0][y0][cIdx] is equal to 1 and sh_ts_residual_coding_disabled_flag is equal to 0, ctxInc is derived as follows:

$$\text{ctxInc} = 4 + \text{csbfCtx} \quad (1562)$$

– Otherwise (transform_skip_flag[x0][y0][cIdx] is equal to 0 or sh_ts_residual_coding_disabled_flag is equal to 1), ctxInc is derived as follows:

  – If cIdx is equal to 0, the following applies:

$$\text{ctxInc} = \text{Min}(\ \text{csbfCtx},\ 1\ ) \quad (1563)$$

  – Otherwise (cIdx is greater than 0), ctxInc is derived as follows:

$$\text{ctxInc} = 2 + \text{Min}(\ \text{csbfCtx},\ 1\ ) \quad (1564)$$

9.3.4.2.7 Derivation process for the variables locNumSig and locSumAbsPass1

Inputs to this process are the colour component index cIdx, the luma location (x0, y0) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture, and the current coefficient scan location (xC, yC).

Outputs of this process are the variables locNumSig and locSumAbsPass1.

Given the syntax elements sig_coeff_flag[x][y] and the array AbsLevelPass1[x][C] for the transform block with component index cIdx and the top-left luma location (x0, y0), the variables locNumSig and locSumAbsPass1 are derived as specified by the following pseudo-code process:

```
locNumSig       = 0
locSumAbsPass1 = 0
if( transform_skip_flag[ x0 ][ y0 ][ cIdx ]  &&  !sh_ts_residual_coding_disabled_flag ) {
    if( xC > 0 ) {
        locNumSig        +=  sig_coeff_flag[ xC − 1 ][ yC ]
        locSumAbsPass1  +=  AbsLevelPass1[ xC − 1 ][ yC ]
    }
    if( yC > 0 ) {
        locNumSig        +=  sig_coeff_flag[ xC ][ yC − 1 ]
        locSumAbsPass1  +=  AbsLevelPass1[ xC ][ yC − 1 ]
    }
} else {
    if( xC < ( 1  <<  Log2ZoTbWidth ) − 1 ) {
        locNumSig        +=  sig_coeff_flag[ xC + 1 ][ yC ]
        locSumAbsPass1  +=  AbsLevelPass1[ xC + 1 ][ yC ]
        if( xC < ( 1  <<  Log2ZoTbWidth ) − 2 ) {
            locNumSig        +=  sig_coeff_flag[ xC + 2 ][ yC ]
            locSumAbsPass1  +=  AbsLevelPass1[ xC + 2 ][ yC ]
        }
        if( yC < ( 1  <<  Log2ZoTbHeight ) − 1 ) {
            locNumSig        +=  sig_coeff_flag[ xC + 1 ][ yC + 1 ]                    (1565)
            locSumAbsPass1  +=  AbsLevelPass1[ xC + 1 ][ yC + 1 ]
        }
    }
    if( yC < ( 1  <<  Log2ZoTbHeight ) − 1 ) {
        locNumSig        +=  sig_coeff_flag[ xC ][ yC + 1 ]
        locSumAbsPass1  +=  AbsLevelPass1[ xC ][ yC + 1 ]
        if( yC < ( 1  <<  Log2ZoTbHeight ) − 2 ) {
            locNumSig        +=  sig_coeff_flag[ xC ][ yC + 2 ]
            locSumAbsPass1  +=  AbsLevelPass1[ xC ][ yC + 2 ]
        }
    }
}
```

9.3.4.2.8 Derivation process of ctxInc for the syntax element sig_coeff_flag

Inputs to this process are the colour component index cIdx, the luma location (x0, y0) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture, and the current coefficient scan location (xC, yC).

Output of this process is the variable ctxInc.

The variable locSumAbsPass1 is derived by invoking the derivation process for the variables locNumSig and locSumAbsPass1 specifies in clause 9.3.4.2.7 with colour component index cIdx, the luma location (x0, y0), and the current coefficient scan location (xC, yC) as inputs.

The variable d is set equal to xC + yC.

The variable ctxInc is derived as follows:

– If transform_skip_flag[x0][y0][cIdx] is equal to 1 and sh_ts_residual_coding_disabled_flag is equal to 0, the following applies:

ctxInc = 60 + locNumSig (1566)

– Otherwise (transform_skip_flag[x0][y0][cIdx] is equal to 0 or sh_ts_residual_coding_disabled_flag is equal to 1), the following applies:

– If cIdx is equal to 0, ctxInc is derived as follows:

$$\text{ctxInc} = 12 * \text{Max}(\ 0, \text{QState} - 1\ ) + \text{Min}(\ (\ \text{locSumAbsPass1} + 1\ ) \gg 1, 3\ ) + (\ d < 2\ ?\ 8 : (\ d < 5\ ?\ 4 : 0\ )\ ) \tag{1567}$$

– Otherwise (cIdx is greater than 0), ctxInc is derived as follows:

$$\text{ctxInc} = 36 + 8 * \text{Max}(\ 0, \text{QState} - 1\ ) + \text{Min}(\ (\ \text{locSumAbsPass1} + 1\ ) \gg 1, 3\ ) + (\ d < 2\ ?\ 4 : 0\ ) \tag{1568}$$

9.3.4.2.9 Derivation process of ctxInc for the syntax elements par_level_flag and abs_level_gtx_flag

Inputs to this process are the colour component index cIdx, the luma location (x0, y0) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture, and the current coefficient scan location (xC, yC).

Output of this process is the variable ctxInc.

The variable ctxInc is derived as follows:

– If transform_skip_flag[x0][y0][cIdx] is equal to 1 and sh_ts_residual_coding_disabled_flag is equal to 0, the following applies:

– If the syntax element is par_level_flag[n], the following applies:

$$\text{ctxInc} = 32 \tag{1569}$$

– Otherwise, if the syntax element is abs_level_gtx_flag[n][0], the following applies:

– If BdpcmFlag[x0][y0][cIdx] is equal to 1, ctxInc is derived as follows:

$$\text{ctxInc} = 67 \tag{1570}$$

– Otherwise, if xC is greater than 0 and yC is greater than 0, ctxInc is derived as follows:

$$\text{ctxInc} = 64 + \text{sig\_coeff\_flag}[\ xC - 1\ ][\ yC\ ] + \text{sig\_coeff\_flag}[\ xC\ ][\ yC - 1\ ] \tag{1571}$$

– Otherwise, if xC is greater than 0, ctxInc is derived as follows:

$$\text{ctxInc} = 64 + \text{sig\_coeff\_flag}[\ xC - 1\ ][\ yC\ ] \tag{1572}$$

– Otherwise, if yC is greater than 0, ctxInc is derived as follows:

$$\text{ctxInc} = 64 + \text{sig\_coeff\_flag}[\ xC\ ][\ yC - 1\ ] \tag{1573}$$

– Otherwise, ctxInc is derived as follows:

$$\text{ctxInc} = 64 \tag{1574}$$

– Otherwise, if the syntax element is abs_level_gtx_flag[n][j] with j > 0, the following applies:

$$\text{ctxInc} = 67 + j \tag{1575}$$

– Otherwise (transform_skip_flag[x0][y0][cIdx] is equal to 0 or sh_ts_residual_coding_disabled_flag is equal to 1), the following applies:

– The variables locNumSig and locSumAbsPass1 are derived by invoking the derivation process for the variables locNumSig and locSumAbsPass1 specified in clause 9.3.4.2.7 with colour component index cIdx, the luma location (x0, y0), and the current coefficient scan location (xC, yC) as inputs.

– The variable ctxOffset is set equal to Min(locSumAbsPass1 − locNumSig, 4).

– The variable d is set equal to xC + yC.

– If xC is equal to LastSignificantCoeffX and yC is equal to LastSignificantCoeffY, ctxInc is derived as follows:

$$\text{ctxInc} = (\text{ cIdx } == 0 \text{ ? } 0 : 21) \tag{1576}$$

– Otherwise, if cIdx is equal to 0, ctxInc is derived as follows:

$$\text{ctxInc} = 1 + \text{ctxOffset} + (\text{d} == 0 \text{ ? } 15 : (\text{d} < 3 \text{ ? } 10 : (\text{d} < 10 \text{ ? } 5 : 0))) \tag{1577}$$

– Otherwise (cIdx is greater than 0), ctxInc is derived as follows:

$$\text{ctxInc} = 22 + \text{ctxOffset} + (\text{d} == 0 \text{ ? } 5 : 0) \tag{1578}$$

– When the syntax element is abs_level_gtx_flag[n][1], the following applies:

$$\text{ctxInc} \mathrel{+}= 32 \tag{1579}$$

9.3.4.2.10 Derivation process of ctxInc for the syntax element coeff_sign_flag for transform skip mode

Inputs to this process are the colour component index cIdx, the luma location (x0, y0) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture, the current coefficient scan location (xC, yC)

Output of this process is the variable ctxInc.

The variables leftSign and aboveSign are derived as follows:

$$\text{leftSign} = (\text{xC} == 0) \text{ ? } 0 : \text{CoeffSignLevel[xC} - 1 \text{][yC]} \tag{1580}$$

$$\text{aboveSign} = (\text{yC} == 0) \text{ ? } 0 : \text{CoeffSignLevel[xC][yC} - 1 \text{]} \tag{1581}$$

The variable ctxInc is derived as follows:

– If leftSign is equal to 0 and aboveSign is equal to 0, or if leftSign is equal to −aboveSign, the following applies:

$$\text{ctxInc} = (\text{BdpcmFlag[x0][y0][cIdx]} == 0 \text{ ? } 0 : 3) \tag{1582}$$

– Otherwise, if leftSign is greater than or equal to 0 and aboveSign is greater than or equal to 0, the following applies:

$$\text{ctxInc} = (\text{BdpcmFlag[x0][y0][cIdx]} == 0 \text{ ? } 1 : 4) \tag{1583}$$

– Otherwise, the following applies:

$$\text{ctxInc} = (\text{BdpcmFlag[x0][y0][cIdx]} == 0 \text{ ? } 2 : 5) \tag{1584}$$

9.3.4.2.11 Derivation process of ctxInc for the syntax element run_copy_flag

Inputs to this process are the variables PreviousRunType, PreviousRunPosition, and the current scan position curPos.

Output of this process is the variable ctxInc.

The variable binDist is set equal to curPos − PreviousRunPosition − 1.

The variable ctxInc is provided by Table 134 depending on binDist and PreviousRunType.

Table 134 – Specification of ctxInc depending on binDist and PreviousRunType

binDist	**0**	**1**	**2**	**3**	**>=4**
PreviousRunType == 1	5	6	6	7	7
PreviousRunType == 0	0	1	2	3	4

9.3.4.3 Arithmetic decoding process

9.3.4.3.1 General

Inputs to this process are ctxTable, ctxIdx, and bypassFlag, as derived in clause 9.3.4.2, and the state variables ivlCurrRange and ivlOffset of the arithmetic decoding engine.

Output of this process is the value of the bin.

Figure 15 illustrates the whole arithmetic decoding process for a single bin. For decoding the value of a bin, the context index table ctxTable, the ctxIdx and the bypassFlag are passed to the arithmetic decoding process DecodeBin(ctxTable, ctxIdx, bypassFlag), which is specified as follows:

– If bypassFlag is equal to 1, DecodeBypass( ) as specified in clause 9.3.4.3.4 is invoked.

– Otherwise, if bypassFlag is equal to 0, ctxTable is equal to 0, and ctxIdx is equal to 0, DecodeTerminate( ) as specified in clause 9.3.4.3.5 is invoked.

– Otherwise (bypassFlag is equal to 0 and ctxTable is not equal to 0), DecodeDecision(ctxTable, ctxIdx) as specified in clause 9.3.4.3.2 is invoked.

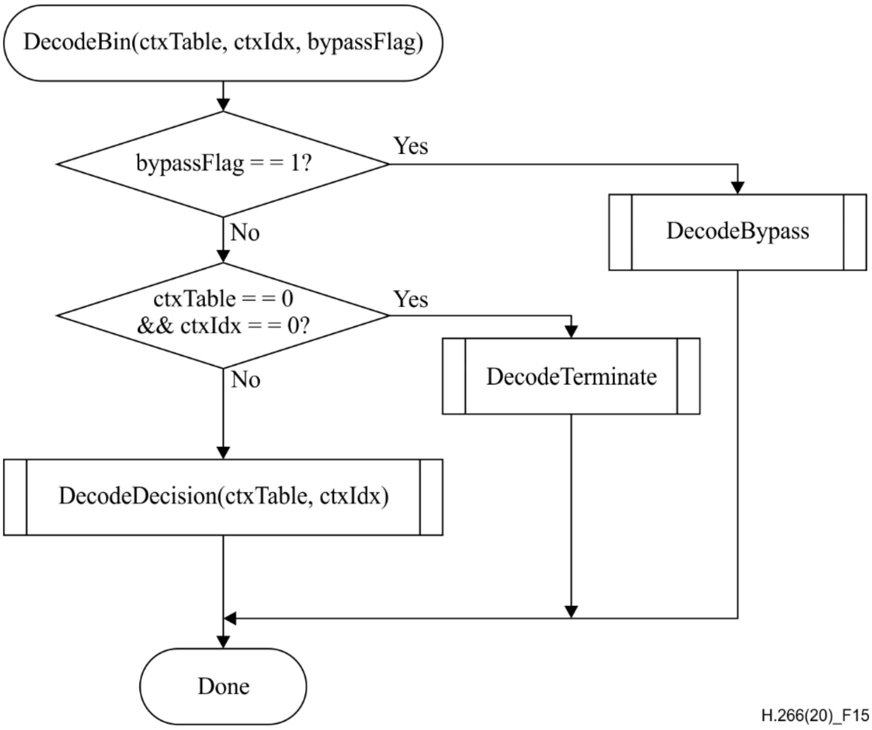

Figure 15 – Flowchart of the arithmetic decoding process for a single bin (informative)

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 ivlCurrRange would be subdivided into two sub-intervals having range p(0) * ivlCurrRange and ivlCurrRange − p(0) * ivlCurrRange, respectively. Depending on the decision, which has been observed, the corresponding sub-interval would be chosen as the new code interval, and a binary code string pointing into that interval would 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 the 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 Specification has the following three distinct properties:

– The probability estimation is performed by means of a two exponential decay estimators, where the average of the probability estimates is used for determining sub-intervals.

– The range ivlCurrRange representing the state of the coding engine and the probability estimate are quantized to reduced-precision values to allow for a reduced-precision multiplication to determine the product ivlCurrRange and the probability estimate.

– 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.

9.3.4.3.2 Arithmetic decoding process for a binary decision

9.3.4.3.2.1 General

Inputs to this process are the variables ctxTable, ctxIdx, ivlCurrRange, and ivlOffset.

Outputs of this process are the decoded value binVal, and the updated variables ivlCurrRange and ivlOffset.

Figure 16 shows the flowchart for decoding a single decision (DecodeDecision):

1. The value of the variable ivlLpsRange is derived as follows:

– Given the current value of ivlCurrRange, the variable qRangeIdx is derived as follows:

qRangeIdx = ivlCurrRange >> 5 (1585)

– Given qRangeIdx, pStateIdx0 and pStateIdx1 associated with ctxTable and ctxIdx, valMps and ivlLpsRange are derived as follows:

pState = pStateIdx1 + 16 * pStateIdx0
valMps = pState >> 14
ivlLpsRange = (qRangeIdx * ((valMps ? 32767 − pState : pState) >> 9) >> 1) + 4 (1586)

2. The variable ivlCurrRange is set equal to ivlCurrRange − ivlLpsRange and the following applies:

– If ivlOffset is greater than or equal to ivlCurrRange, the variable binVal is set equal to 1 − valMps, ivlOffset is decremented by ivlCurrRange, and ivlCurrRange is set equal to ivlLpsRange.

– Otherwise, the variable binVal is set equal to valMps.

Given the value of binVal, the state transition is performed as specified in clause 9.3.4.3.2.2. Depending on the current value of ivlCurrRange, renormalization is performed as specified in clause 9.3.4.3.3.

9.3.4.3.2.2 State transition process

Inputs to this process are the current pStateIdx0 and pStateIdx1, and the decoded value binVal.

Outputs of this process are the updated pStateIdx0 and pStateIdx1 of the context variable associated with ctxTable and ctxIdx.

The variables shift0 and shift1 are derived from the shiftIdx value associated with ctxTable and ctxIdx in clause 9.3.2.2.

shift0 = (shiftIdx >> 2) + 2
shift1 = (shiftIdx & 3) + 3 + shift0 (1587)

Depending on the decoded value binVal, the update of the two variables pStateIdx0 and pStateIdx1 associated with ctxTable and ctxIdx is derived as follows:

pStateIdx0 = pStateIdx0 − (pStateIdx0 >> shift0) + (1023 * binVal >> shift0)
pStateIdx1 = pStateIdx1 − (pStateIdx1 >> shift1) + (16383 * binVal >> shift1) (1588)

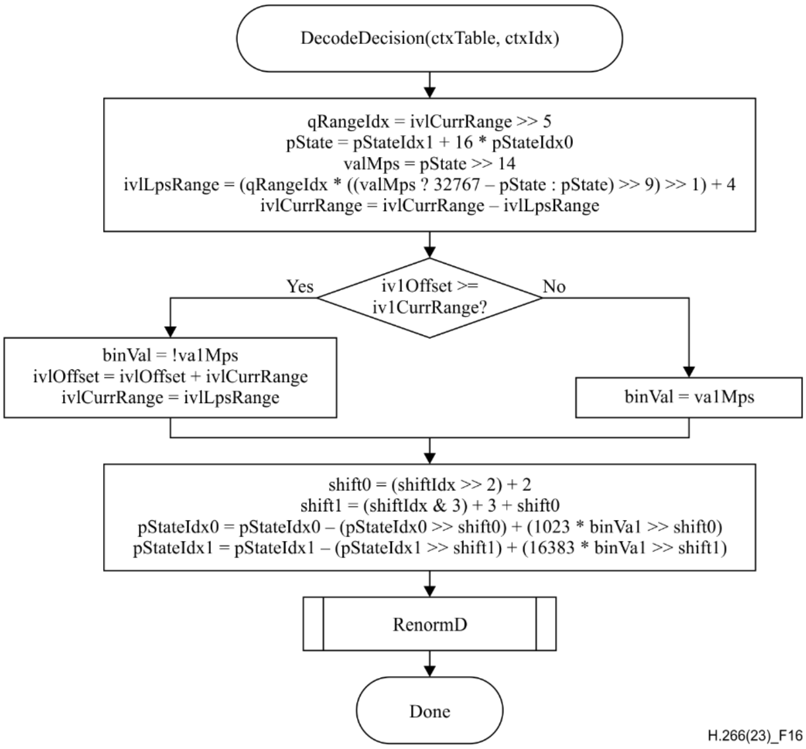

Figure 16 – Flowchart for decoding a decision

9.3.4.3.3 Renormalization process in the arithmetic decoding engine

Inputs to this process are bits from slice data and the variables ivlCurrRange and ivlOffset.

Outputs of this process are the updated variables ivlCurrRange and ivlOffset.

A flowchart of the renormalization is shown in Figure 17. The current value of ivlCurrRange is first compared to 256 and further steps are specified as follows:

– If ivlCurrRange is greater than or equal to 256, no renormalization is needed and the RenormD process is finished;

– Otherwise (ivlCurrRange is less than 256), the renormalization loop is entered. Within this loop, the value of ivlCurrRange is doubled, i.e., left-shifted by 1 and a single bit is shifted into ivlOffset by using read_bits(1).

The bitstream shall not contain data that result in a value of ivlOffset being greater than or equal to ivlCurrRange upon completion of this process.

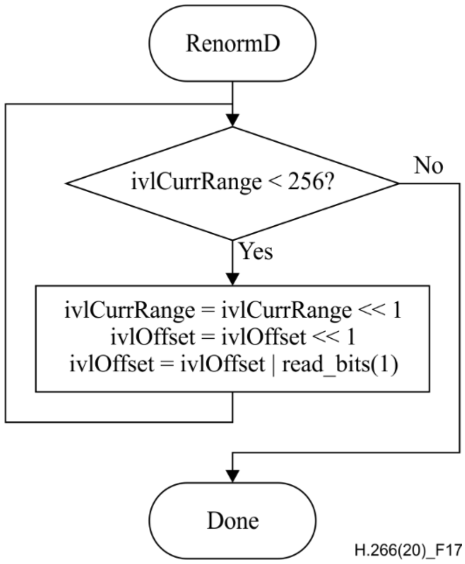

Figure 17 – Flowchart of renormalization

9.3.4.3.4 Bypass decoding process for binary decisions

Inputs to this process are bits from slice data and the variables ivlCurrRange and ivlOffset.

Outputs of this process are the updated variable ivlOffset and the decoded value binVal.

The bypass decoding process is invoked when bypassFlag is equal to 1. Figure 18 shows a flowchart of the corresponding process.

First, the value of ivlOffset is doubled, i.e., left-shifted by 1 and a single bit is shifted into ivlOffset by using read_bits(1). Then, the value of ivlOffset is compared to the value of ivlCurrRange and further steps are specified as follows:

– If ivlOffset is greater than or equal to ivlCurrRange, the variable binVal is set equal to 1 and ivlOffset is decremented by ivlCurrRange.

– Otherwise (ivlOffset is less than ivlCurrRange), the variable binVal is set equal to 0.

The bitstream shall not contain data that result in a value of ivlOffset being greater than or equal to ivlCurrRange upon completion of this process.

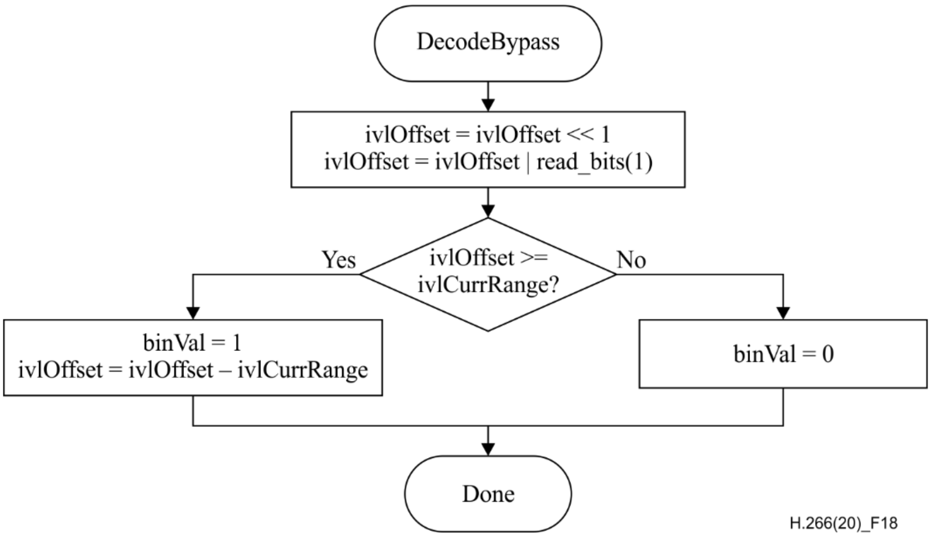

Figure 18 – Flowchart of bypass decoding process

9.3.4.3.5 Decoding process for binary decisions before termination

Inputs to this process are bits from slice data and the variables ivlCurrRange and ivlOffset.

Outputs of this process are the updated variables ivlCurrRange and ivlOffset, and the decoded value binVal.

This decoding process applies to decoding of end_of_slice_one_bit, end_of_tile_one_bit, and end_of_subset_one_bit corresponding to ctxTable equal to 0 and ctxIdx equal to 0. Figure 19 shows the flowchart of the corresponding decoding process, which is specified as follows:

First, the value of ivlCurrRange is decremented by 2. Then, the value of ivlOffset is compared to the value of ivlCurrRange and further steps are specified as follows:

– If ivlOffset is greater than or equal to ivlCurrRange, the variable binVal is set equal to 1, no renormalization is carried out, and CABAC decoding is terminated. The last bit inserted in register ivlOffset is equal to 1. When decoding end_of_slice_one_bit, this last bit inserted in register ivlOffset is interpreted as rbsp_stop_one_bit. When decoding end_of_tile_one_bit or end_of_subset_one_bit, this last bit inserted in register ivlOffset is interpreted as byte_alignment_bit_equal_to_one.

– Otherwise (ivlOffset is less than ivlCurrRange), the variable binVal is set equal to 0 and renormalization is performed as specified in clause 9.3.4.3.3.

NOTE – This procedure could also be implemented using DecodeDecision(ctxTable, ctxIdx, bypassFlag) with ctxTable = 0, ctxIdx = 0 and bypassFlag = 0. In the case where the decoded value is equal to 1, 7 more bits would be read by DecodeDecision(ctxTable, ctxIdx, bypassFlag) and a decoding process would have to adjust its bitstream pointer accordingly to properly decode following syntax elements.

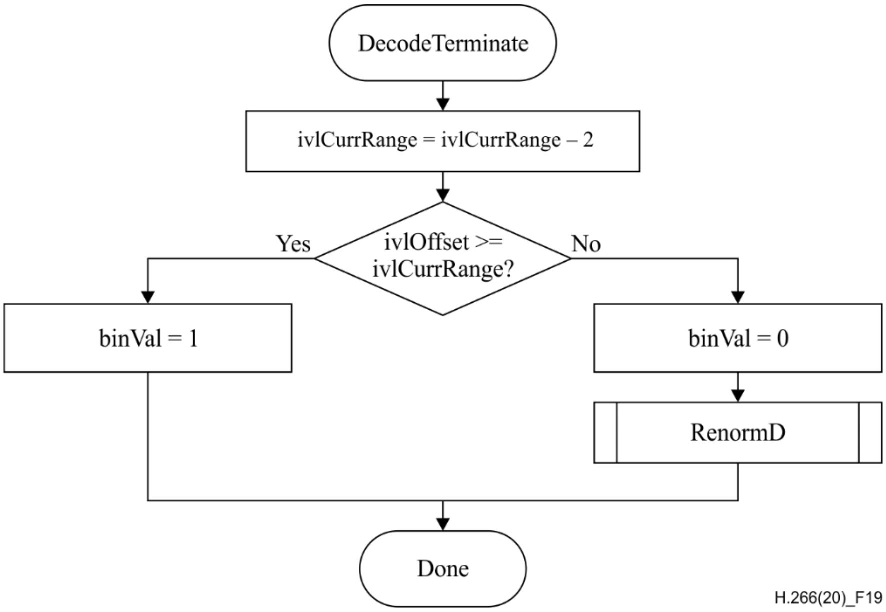

Figure 19 – Flowchart of decoding a decision before termination

Annex A

Profiles, tiers and levels

(This annex forms an integral part of this Recommendation | International Standard.)

A.1 Overview of profiles, tiers and levels

Profiles, tiers and levels specify restrictions on bitstreams and hence limits on the capabilities needed to decode the bitstreams. Profiles, tiers and levels are also used to indicate the capability of individual decoder implementations and interoperability points between encoders and decoders.

Each profile specifies a subset of algorithmic features and limits that shall be supported by all decoders conforming to that profile.

NOTE 1 – Encoders are not required to make use of any particular subset of features supported in a profile.

Each level of a tier specifies a set of limits on the values that may be taken by the syntax elements of this Specification. The same set of tier and level definitions is usually used with all profiles, but individual implementations may support a different tier, and within a tier, a different level for each supported profile. For any given profile, a level of a tier generally corresponds to a particular decoder processing load and memory capability.

In this annex, phrases like "the bitstream" are to be intepreted as "the bitstream of the operation point", and phrases like "AU n" and "a layer" are to be interpreted as "AU n in the bitstream of the operation point" and "a layer in the bitstream of the operation point", respectively, where "the operation point" is the operation point with which the profile, tier, or level is associated with.

For each operation point identified by TargetOlsIdx and Htid, the profile, tier, and level information is indicated through general_profile_idc, general_tier_flag, and sublayer_level_idc[Htid], all found in or derived from the profile_tier_level( ) syntax structure in the VPS that applies to the OLS identified by TargetOlsIdx.

When no VPS is available, the profile and tier information is indicated through general_profile_idc and general_tier_flag in the SPS, and the level information is indicated as follows:

– If Htid is provided by external means indicating the highest TemporalId of any NAL unit in the bitstream, the level information is indicated through sublayer_level_idc[Htid] found in or derived from the SPS.

– Otherwise (Htid is not provided by external means), the level information is indicated through general_level_idc in the SPS.

NOTE 2 – Decoders are not required to extract a subset of the bitstream; any such extraction process that might be a part of the system is considered outside of the scope of the decoding process specified by this Specification. The values TargetOlsIdx and Htid are not necessary for the operation of the decoding process, could be provided by external means, and can be used to check the conformance of the bitstream.

A.2 Requirements on video decoder capability

Capabilities of video decoders conforming to this Specification are specified in terms of the ability to decode video streams conforming to the constraints of profiles, tiers and levels specified in this annex and other annexes. When expressing the capabilities of a decoder for a specified profile, the tier and level supported for that profile should also be expressed.

Specific values are specified in this annex for the syntax elements general_profile_idc, general_tier_flag, general_level_idc, and sublayer_level_idc[i]. All other values of general_profile_idc, general_level_idc, and sublayer_level_idc[i] are reserved for future use by ITU-T | ISO/IEC.

Decoders should not infer that a reserved value of general_profile_idc between the values specified in this Specification indicates intermediate capabilities between the specified profiles, as there are no restrictions on the method to be chosen by ITU-T | ISO/IEC for the use of such future reserved values. However, decoders shall infer that a reserved value of general_level_idc or sublayer_level_idc[i] associated with a particular value of general_tier_flag between the values specified in this Specification indicates intermediate capabilities between the specified levels of the tier.

A.3 Profiles

A.3.1 Main 10 and Main 10 Still Picture profiles

Bitstreams conforming to the Main 10 or Main 10 Still Picture profile shall obey the following constraints:

– Referenced SPSs shall have ptl_multilayer_enabled_flag equal to 0.

- In a bitstream conforming to the Main 10 Still Picture profile, the bitstream shall contain only one picture.
- Referenced SPSs shall have sps_chroma_format_idc equal to 0 or 1.
- Referenced SPSs shall have sps_bitdepth_minus8 in the range of 0 to 2, inclusive.
- Referenced SPSs shall have sps_palette_enabled_flag equal to 0.
- The tier and level constraints specified for the Main 10 or Main 10 Still Picture profile in clause A.4, as applicable, shall be fulfilled.

Conformance of a bitstream to the Main 10 profile is indicated by general_profile_idc being equal to 1.

Conformance of a bitstream to the Main 10 Still Picture profile is indicated by general_profile_idc being equal to 65.

NOTE – When the conformance of a bitstream to the Main 10 Still Picture profile is indicated by general_profile_idc being equal to 65, the conditions for indication of the conformance of the bitstream to the Main 10 profile are also fulfilled.

Decoders conforming to the Main 10 profile at a specific level of a specific tier shall be capable of decoding all bitstreams for which all of the following conditions apply:

- The bitstream is indicated to conform to the Main 10 or Main 10 Still Picture profile.
- The bitstream is indicated to conform to a tier that is lower than or equal to the specified tier.
- The bitstream is indicated to conform to a level that is not level 15.5 and is lower than or equal to the specified level.

Decoders conforming to the Main 10 Still Picture profile at a specific level of a specific tier shall be capable of decoding all bitstreams for which all of the following conditions apply:

- The bitstream is indicated to conform to the Main 10 Still Picture profile.
- The bitstream is indicated to conform to a tier that is lower than or equal to the specified tier.
- The bitstream is indicated to conform to a level that is not level 15.5 and is lower than or equal to the specified level.

Decoders conforming to the Main 10 Still Picture profile at a specific level of a specific tier shall also be capable of decoding of the first picture of a bitstream when both of the following conditions apply:

- The bitstream is indicated to conform to the Main 10 profile, to conform to a tier that is lower than or equal to the specified tier, and to conform to a level that is not level 15.5 and is lower than or equal to the specified level.
- The first picture of the bitstream is an IRAP picture or is a GDR picture with ph_recovery_poc_cnt equal to 0, is in an output layer, and has ph_pic_output_flag equal to 1.

A.3.2 Main 10 4:4:4 and Main 10 4:4:4 Still Picture profiles

Bitstreams conforming to the Main 10 4:4:4 or Main 10 4:4:4 Still Picture profile shall obey the following constraints:

- Referenced SPSs shall have ptl_multilayer_enabled_flag equal to 0.
- In a bitstream conforming to the Main 10 4:4:4 Still Picture profile, the bitstream shall contain only one picture.
- Referenced SPSs shall have sps_chroma_format_idc in the range of 0 to 3, inclusive.
- Referenced SPSs shall have sps_bitdepth_minus8 in the range of 0 to 2, inclusive.
- The tier and level constraints specified for the Main 10 4:4:4 or Main 10 4:4:4 Still Picture profile in clause A.4, as applicable, shall be fulfilled.

Conformance of a bitstream to the Main 10 4:4:4 profile is indicated by general_profile_idc being equal to 33.

Conformance of a bitstream to the Main 10 4:4:4 Still Picture profile is indicated by general_profile_idc being equal to 97.

NOTE – When the conformance of a bitstream to the Main 10 4:4:4 Still Picture profile is indicated by general_profile_idc being equal to 97, the conditions for indication of the conformance of the bitstream to the Main 10 4:4:4 profile are also fulfilled.

Decoders conforming to the Main 10 4:4:4 profile at a specific level of a specific tier shall be capable of decoding all bitstreams for which all of the following conditions apply:

- The bitstream is indicated to conform to the Main 10 4:4:4, Main 10, Main 10 4:4:4 Still Picture, or Main 10 Still Picture profile.
- The bitstream is indicated to conform to a tier that is lower than or equal to the specified tier.
- The bitstream is indicated to conform to a level that is not level 15.5 and is lower than or equal to the specified level.

Decoders conforming to the Main 10 4:4:4 Still Picture profile at a specific level of a specific tier shall be capable of decoding all bitstreams for which all of the following conditions apply:

– The bitstream is indicated to conform to the Main 10 4:4:4 Still Picture or Main 10 Still Picture profile.

– The bitstream is indicated to conform to a tier that is lower than or equal to the specified tier.

– The bitstream is indicated to conform to a level that is not level 15.5 and is lower than or equal to the specified level.

Decoders conforming to the Main 10 4:4:4 Still Picture profile at a specific level of a specific tier shall also be capable of decoding of the first picture of a bitstream when both of the following conditions apply:

– The bitstream is indicated to conform to the Main 10 or Main 10 4:4:4 profile, to conform to a tier that is lower than or equal to the specified tier, to conform to a level that is not level 15.5 and is lower than or equal to the specified level.

– The first picture of the bitstream is an IRAP picture or is a GDR picture with ph_recovery_poc_cnt equal to 0, is in an output layer, and has ph_pic_output_flag equal to 1.

A.3.3 Multilayer Main 10 profile

Bitstreams conforming to the Multilayer Main 10 shall obey the following constraints:

– Referenced SPSs shall have sps_chroma_format_idc equal to 0 or 1.

– Referenced SPSs shall have sps_bitdepth_minus8 in the range of 0 to 2, inclusive.

– Referenced SPSs shall have sps_palette_enabled_flag equal to 0.

– The tier and level constraints specified for the Multilayer Main 10 profile in clause A.4, as applicable, shall be fulfilled.

Conformance of a bitstream to the Multilayer Main 10 profile is indicated by general_profile_idc being equal to 17.

Decoders conforming to the Multilayer Main 10 profile at a specific level of a specific tier shall be capable of decoding all bitstreams for which all of the following conditions apply:

– The bitstream is indicated to conform to the Multilayer Main 10, Main 10, or Main 10 Still Picture profile.

– The bitstream is indicated to conform to a tier that is lower than or equal to the specified tier.

– The bitstream is indicated to conform to a level that is not level 15.5 and is lower than or equal to the specified level.

A.3.4 Multilayer Main 10 4:4:4 profile

Bitstreams conforming to the Multilayer Main 10 4:4:4 profile shall obey the following constraints:

– Referenced SPSs shall have sps_chroma_format_idc in the range of 0 to 3, inclusive.

– Referenced SPSs shall have sps_bitdepth_minus8 in the range of 0 to 2, inclusive.

– The tier and level constraints specified for the Multilayer Main 10 4:4:4 profile in clause A.4, as applicable, shall be fulfilled.

Conformance of a bitstream to the Multilayer Main 10 4:4:4 profile is indicated by general_profile_idc being equal to 49.

Decoders conforming to the Multilayer Main 10 4:4:4 profile at a specific level of a specific tier shall be capable of decoding all bitstreams for which all of the following conditions apply:

– The bitstream is indicated to conform to the Multilayer Main 10 4:4:4, Multilayer Main 10, Main 10 4:4:4, Main 10, Main 10 4:4:4 Still Picture, or Main 10 Still Picture profile.

– The bitstream is indicated to conform to a tier that is lower than or equal to the specified tier.

– The bitstream is indicated to conform to a level that is not level 15.5 and is lower than or equal to the specified level.

A.3.5 Operation range extensions profiles

The following profiles, collectively referred to as the operation range extensions profiles, are specified in this clause:

– The Main 12, Main 12 4:4:4 and Main 16 4:4:4 profiles

– The Main 12 Intra, Main 12 4:4:4 Intra and Main 16 4:4:4 Intra profiles

– The Main 12 Still Picture, Main 12 4:4:4 Still Picture and Main 16 4:4:4 Still Picture profiles

Bitstreams conforming to the operation range extensions profiles shall obey the following constraints:

– Referenced SPSs shall have ptl_multilayer_enabled_flag equal to 0.
– In bitstreams conforming to the Main 12 Still Picture, Main 12 4:4:4 Still Picture or Main 16 4:4:4 Still Picture profile, the bitstream shall contain only one picture.
– The allowed values for syntax elements as specified in Table A.1 shall be fulfilled.
– The tier and level constraints specified for the Main 12, Main 12 4:4:4, Main 16 4:4:4, Main 12 Intra, Main 12 4:4:4 Intra or Main 16 4:4:4 Intra profile in clause A.4, as applicable, shall be fulfilled.
– In bitstreams conforming to the Main 12 Intra, Main 12 4:4:4 Intra or Main 16 4:4:4 Intra profile, all pictures shall be GDR pictures with ph_recovery_poc_cnt equal to 0 or IRAP pictures.

Table A.1 – Allowed values for syntax elements in the operation range extensions profiles

Profile for which constraint is specified	sps_chroma_format_idc	sps_bitdepth_minus8	sps_palette_enabled_flag	sps_extended_precision_flag sps_ts_residual_coding_rice_present_in_sh_flag, sps_rrc_rice_extension_flag, sps_persistent_rice_adaptation_enabled_flag sps_reverse_last_sig_coeff_enabled_flag
Main 12	0..1	0..4	0	0
Main 12 4:4:4	0..3	0..4	0..1	0..1
Main 16 4:4:4	0..3	0..8	0..1	0..1
Main 12 Intra	0..1	0..4	0	0
Main 12 4:4:4 Intra	0..3	0..4	0..1	0..1
Main 16 4:4:4 Intra	0..3	0..8	0..1	0..1
Main 12 Still Picture	0..1	0..4	0	0
Main 12 4:4:4 Still Picture	0..3	0..4	0..1	0..1
Main 16 4:4:4 Still Picture	0..3	0..8	0..1	0..1

Conformance of a bitstream to the Main 12 profile is indicated by general_profile_idc being equal to 2.

Conformance of a bitstream to the Main 12 Intra profile is indicated by general_profile_idc being equal to 10.

Conformance of a bitstream to the Main 12 Still Picture profile is indicated by general_profile_idc being equal to 66.

Conformance of a bitstream to the Main 12 4:4:4 profile is indicated by general_profile_idc being equal to 34.

Conformance of a bitstream to the Main 12 4:4:4 Intra profile is indicated by general_profile_idc being equal to 42.

Conformance of a bitstream to the Main 12 4:4:4 Still Picture profile is indicated by general_profile_idc being equal to 98.

Conformance of a bitstream to the Main 16 4:4:4 profile is indicated by general_profile_idc being equal to 35.

Conformance of a bitstream to the Main 16 4:4:4 Intra profile is indicated by general_profile_idc being equal to 43.

Conformance of a bitstream to the Main 16 4:4:4 Still Picture profile is indicated by general_profile_idc being equal to 99.

Decoders conforming to an operation range extensions profile at a specific level (identified by a specific value of general_level_idc) of a specific tier (identified by a specific value of general_tier_flag) shall be capable of decoding all bitstreams and sub-layer representations for which all of the following conditions apply:

– Any of the following conditions apply:

  - o The decoder conforms to the Main 12 profile, and the bitstream is indicated to conform to the Main 10, Main 10 Still Picture, Main 12, Main 12 Intra, or Main 12 Still Picture profile.

  - o The decoder conforms to the Main 12 4:4:4 profile, and the bitstream is indicated to conform to the Main 10, Main 10 Still Picture, Main 10 4:4:4, Main 10 4:4:4 Still Picture, Main 12, Main 12 Intra, Main 12 Still Picture, Main 12 4:4:4, Main 12 4:4:4 Intra, or Main 12 4:4:4 Still Picture profile.

  - o The decoder conforms to the Main 16 4:4:4 profile, and the bitstream is indicated to conform to Main 10, Main 10 Still Picture, Main 10 4:4:4, Main 10 4:4:4 Still Picture, or any of the operation range extensions profile.

  - o The decoder conforms to the Main 12 Intra profile, and either 1) the bitstream is indicated to conform to the Main 10 Still Picture, Main 12 Intra, or Main 12 Still Picture profile, or 2) the gci_all_rap_pictures_constraint_flag is equal to 1 and the bitstream is indicated to conform to the Main 10 or Main 12 profile.

  - o The decoder conforms to the Main 12 4:4:4 Intra profile, and either 1) the bitstream is indicated to conform to the Main 10 Still Picture, Main 10 4:4:4 Still Picture, Main 12 Intra, Main 12 4:4:4 Intra, Main 12 Still Picture, or Main 12 4:4:4 Still Picture profile, or 2) the gci_all_rap_pictures_constraint_flag is equal to 1 and the bitstream is indicated to conform to the Main 10, Main 10 4:4:4, Main 12, or Main 12 4:4:4 profile.

  - o The decoder conforms to the Main 16 4:4:4 Intra profile, and either 1) the bitstream is indicated to conform to the Main 10 Still Picture, Main 10 4:4:4 Still Picture, Main 12 Intra, Main 12 4:4:4 Intra, Main 16 4:4:4 Intra, Main 12 Still Picture, Main 12 4:4:4 Still Picture, or Main 16 4:4:4 Still Picture profile, or 2) the gci_all_rap_pictures_constraint_flag is equal to 1 and the bitstream is indicated to conform to the Main 10, Main 10 4:4:4, Main 12, Main 12 4:4:4, or Main 16 4:4:4 profile.

  - o The decoder conforms to the Main 12 Still Picture profile, and the bitstream is indicated to conform to the Main 10 Still Picture or Main 12 Still Picture profile.

  - o The decoder conforms to the Main 12 4:4:4 Still Picture profile, and the bitstream is indicated to conform to the Main 10 Still Picture, Main 10 4:4:4 Still Picture, Main 12 Still Picture or Main 12 4:4:4 Still Picture profile.

  - o The decoder conforms to the Main 16 4:4:4 Still Picture profile, and the bitstream is indicated to conform to the Main 10 Still Picture, Main 10 4:4:4 Still Picture, Main 12 Still Picture, Main 12 4:4:4 Still Picture, or Main 16 4:4:4 Still Picture profile.

– The bitstream is indicated to conform to a tier that is lower than or equal to the specified tier.

– The bitstream is indicated to conform to a level that is not level 15.5 and is lower than or equal to the specified level.

Decoders conforming to the Main 12 Still Picture profile at a specific level of a specific tier shall also be capable of decoding of the first picture of a bitstream when both of the following conditions apply:

– That bitstream is indicated to conform to the Main 10, Main 12, or Main 12 Intra profile, to conform to a tier that is lower than or equal to the specified tier, and to conform to a level that is not level 15.5 and is lower than or equal to the specified level.

– That picture is a GDR picture with ph_recovery_poc_cnt equal to 0 or an IRAP picture, is in an output layer, and has ph_pic_output_flag equal to 1.

Decoders conforming to the Main 12 4:4:4 Still Picture profile at a specific level of a specific tier shall also be capable of decoding of the first picture of a bitstream when both of the following conditions apply:

– That bitstream is indicated to conform to the Main 10, Main 10 4:4:4, Main 12, Main 12 Intra, Main 12 4:4:4, or Main 12 4:4:4 Intra profile, to conform to a tier that is lower than or equal to the specified tier, and to conform to a level that is not level 15.5 and is lower than or equal to the specified level.

– That picture is a GDR picture with ph_recovery_poc_cnt equal to 0 or an IRAP picture, is in an output layer, and has ph_pic_output_flag equal to 1.

Decoders conforming to the Main 16 4:4:4 Still Picture profile at a specific level of a specific tier shall also be capable of decoding of the first picture of a bitstream when both of the following conditions apply:

– That bitstream is indicated to conform to the Main 10, Main 10 4:4:4, Main 12, Main 12 Intra, Main 12 4:4:4, Main 12 4:4:4 Intra, Main 16 4:4:4, or Main 16 4:4:4 Intra profile, to conform to a tier that is lower than or equal to the specified tier, and to conform to a level that is not level 15.5 and is lower than or equal to the specified level.

– That picture is a GDR picture with ph_recovery_poc_cnt equal to 0 or an IRAP picture, is in an output layer, and has ph_pic_output_flag equal to 1.

A.4 Tiers and levels

A.4.1 General tier and level limits

For purposes of comparison of tier capabilities, the tier with general_tier_flag equal to 0 (i.e., the Main tier) is considered to be a lower tier than the tier with general_tier_flag equal to 1 (i.e., the High tier).

For purposes of comparison of level capabilities, a particular level of a specific tier is considered to be a lower level than some other level of the same tier when the value of the general_level_idc or sublayer_level_idc[i] of the particular level is less than that of the other level.

The following is specified for expressing the constraints in this annex:

– Let AU n be the n-th AU in decoding order, with the first AU being AU 0 (i.e., the 0-th AU).

– For an OLS with OLS index TargetOlsIdx, the variables PicWidthMaxInSamplesY, PicHeightMaxInSamplesY, and PicSizeMaxInSamplesY, and the applicable dpb_parameters( ) syntax structure are derived as follows:

  o If NumLayersInOls[TargetOlsIdx] is equal to 1, PicWidthMaxInSamplesY is set equal to sps_pic_width_max_in_luma_samples, PicHeightMaxInSamplesY is set equal to sps_pic_height_max_in_luma_samples, and PicSizeMaxInSamplesY is set equal to PicWidthMaxInSamplesY * PicHeightMaxInSamplesY, where sps_pic_width_max_in_luma_samples and sps_pic_height_max_in_luma_samples are found in the SPS referred to by the layer in the OLS, and the applicable dpb_parameters( ) syntax structure is also found in that SPS.

  o Otherwise (NumLayersInOls[TargetOlsIdx] is greater than 1), PicWidthMaxInSamplesY is set equal to vps_ols_dpb_pic_width[MultiLayerOlsIdx[TargetOlsIdx]], PicHeightMaxInSamplesY is set equal to vps_ols_dpb_pic_height[MultiLayerOlsIdx[TargetOlsIdx]], PicSizeMaxInSamplesY is set equal to PicWidthMaxInSamplesY * PicHeightMaxInSamplesY, and the applicable dpb_parameters( ) syntax structure is identified by vps_ols_dpb_params_idx[MultiLayerOlsIdx[TargetOlsIdx]] found in the VPS.

Table A.2 specifies the limits for each level of each tier for levels other than level 15.5.

NOTE 1 – Since there are no limits specified by Table A.2 for level 15.5, it is not possible in general for a practical decoder to be assured of being able to decode all bitstreams that conform to this level. The purpose of the definition of level 15.5 is to provide a suitable label for bitstreams that can exceed the limits of all other specified levels. When the bitstream is indicated to conform to level 15.5, a decoder is expected to examine the characteristics of the bitstream during its operation in order to determine whether it is capable of decoding the bitstream.

When the specified level is not level 15.5, bitstreams conforming to a profile at a specified tier and level shall obey the following constraints for each bitstream conformance test as specified in Annex C:

a) PicSizeMaxInSamplesY shall be less than or equal to MaxLumaPs, where MaxLumaPs is specified in Table A.2.

b) The value of PicWidthMaxInSamplesY shall be less than or equal to Sqrt(MaxLumaPs * 8).

c) The value of PicHeightMaxInSamplesY shall be less than or equal to Sqrt(MaxLumaPs * 8).

d) For each referenced PPS, the value of NumTileColumns shall be less than or equal to MaxTileCols and the value of NumTilesInPic shall be less than or equal to MaxTilesPerAu, where MaxTileCols and MaxTilesPerAu are specified in Table A.2.

e) For each referenced PPS, the value of ColWidthVal[i] * CtbSizeY, for each i in the range of 0 to NumTileColumns − 1, inclusive, shall be less than or equal to 0.5 * Sqrt(Max(level4Val, MaxLumaPs) * 8), where MaxLumaPs is specified in Table A.2 and level4Val is equal to 2 228 224.

   NOTE 2 – The maximum tile width in luma samples is also less than or equal to the picture width, which is less than or equal to Sqrt(MaxLumaPs * 8).

f) For the VCL HRD parameters, CpbSize[Htid][i] shall be less than or equal to CpbVclFactor * MaxCPB for at least one value of i in the range of 0 to hrd_cpb_cnt_minus1, inclusive, where CpbSize[Htid][i] is specified in clause 7.4.6.3 based on parameters selected as specified in clause C.1, CpbVclFactor is specified in Table A.4 and MaxCPB is specified in Table A.2 in units of CpbVclFactor bits.

g) For the NAL HRD parameters, CpbSize[Htid][i] shall be less than or equal to CpbNalFactor * MaxCPB for at least one value of i in the range of 0 to hrd_cpb_cnt_minus1, inclusive, where CpbSize[Htid][i] is specified in clause 7.4.6.3 based on parameters selected as specified in clause C.1, CpbNalFactor is specified in Table A.4, and MaxCPB is specified in Table A.2 in units of CpbNalFactor bits.

h) The value of BinCountsInPicNalUnits shall be less than or equal to vclByteScaleFactor * NumBytesInPicVclNalUnits + (RawMinCuBits * PicSizeInMinCbsY) ÷ 32.

i) For each subpicture with subpicture index subpicIdxA for which sps_subpic_treated_as_pic_flag[subpicIdxA] is equal to 1, the value of BinCountsInSubpicNalUnits shall be less than or equal to vclByteScaleFactor * NumBytesInSubpicVclNalUnits + (RawMinCuBits * subpicSizeInMinCbsY) ÷ 32.

   NOTE 3 – The constraints on the maximum number of bins resulting from decoding the contents of the coded slice NAL units of a coded picture or subpicture can be met by inserting a number of rbsp_cabac_zero_word syntax elements to increase the value of NumBytesInPicVclNalUnits. Each rbsp_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 rbsp_cabac_zero_word).

A tier and level to which a bitstream conforms are indicated by the syntax elements general_tier_flag and general_level_idc, and a level to which a sublayer representation conforms are indicated by the syntax element sublayer_level_idc[i], as follows:

– If the specified level is not level 15.5, general_tier_flag equal to 0 indicates conformance to the Main tier, general_tier_flag equal to 1 indicates conformance to the High tier, according to the tier constraints specified in Table A.2 and general_tier_flag shall be equal to 0 for levels below level 4 (corresponding to the entries in Table A.2 marked with "-"). Otherwise (the specified level is level 15.5), it is a requirement of bitstream conformance that general_tier_flag shall be equal to 1 and the value 0 for general_tier_flag is reserved for future use by ITU-T | ISO/IEC and decoders shall ignore the value of general_tier_flag.
– general_level_idc and sublayer_level_idc[i] shall be set equal to a value of general_level_idc for the level number specified in Table A.2.

Table A.2 – General tier and level limits

Level	general_level_idc value*	Max luma picture size MaxLumaPs (samples)	Max CPB size MaxCPB (CpbVclFactor or CpbNalFactor bits)		Max slices per AU MaxSlicesPerAu	Max # of tiles MaxTilesPerAu	Max # of tile columns MaxTileCols
			Main tier	High tier			
1.0	16	36 864	350	-	16	1	1
2.0	32	122 880	1 500	-	16	1	1
2.1	35	245 760	3 000	-	20	1	1
3.0	48	552 960	6 000	-	30	4	2
3.1	51	983 040	10 000	-	40	9	3
4.0	64	2 228 224	12 000	30 000	75	25	5
4.1	67	2 228 224	20 000	50 000	75	25	5
5.0	80	8 912 896	25 000	100 000	200	110	10
5.1	83	8 912 896	40 000	160 000	200	110	10
5.2	86	8 912 896	60 000	240 000	200	110	10
6.0	96	35 651 584	80 000	240 000	600	440	20
6.1	99	35 651 584	120 000	480 000	600	440	20
6.2	102	35 651 584	180 000	800 000	600	440	20
6.3	105	80 216 064	240 000	1 600 000	1 000	990	30
* The level numbers in this table are in the form of "majorNum.minorNum", and the value of general_level_idc for each of the levels is equal to majorNum * 16 + minorNum * 3.							

A.4.2 Profile-specific level limits

NOTE – The list of profiles for which constraints are specified in this clause does not include the Main 10 Still Picture, Main 10 4:4:4 Still Picture, Main 12 Still Picture, Main 12 4:4:4 Still Picture, and Main 16 4:4:4 Still Picture profiles. Thus, there are no constraints specified for these profiles that involve the variables derived in this clause, including FrVal, BrNalFactor, BrVclFactor, MinCr, MaxDpbSize, and AuSizeMaxInSamplesY[n].

The following is specified for expressing the constraints in this annex:

– Let the variable FrVal be set equal to 1 ÷ 300 if general_tier_flag is equal to 0 and set equal to 1 ÷ 960 otherwise.

The variable HbrFactor is defined as follows:

– If the bitstream is indicated to conform to the Main 10, Main 10 4:4:4, Multilayer Main 10, or Multilayer Main 10 4:4:4 profile, HbrFactor is set equal to 1.

– Otherwise, when the bitstream is indicated to conform to the Main 12, Main 12 Intra, Main 12 4:4:4, Main 12 4:4:4 Intra, Main 16 4:4:4, or Main 16 4:4:4 Intra profile, the following applies:

  o If the bitstream is indicated to conform to the Main tier, HbrFactor is set equal to 1.

  o Otherwise (the bitstream is indicated to conform to the High tier), HbrFactor is set equal to 2.

The variable BrVclFactor, which represents the VCL bit rate scale factor, is set equal to CpbVclFactor * HbrFactor.

The variable BrNalFactor, which represents the NAL bit rate scale factor, is set equal to CpbNalFactor * HbrFactor.

The variable MinCr is set equal to MinCrBase * MinCrScaleFactor ÷ HbrFactor.

When the specified level is not level 15.5, the value of dpb_max_dec_pic_buffering_minus1[Htid] + 1 shall be less than or equal to MaxDpbSize, which is derived as follows:

```
if( 2 * PicSizeMaxInSamplesY <= MaxLumaPs )
    MaxDpbSize = 2 * maxDpbPicBuf
else if( 3 * PicSizeMaxInSamplesY <= 2 * MaxLumaPs )
    MaxDpbSize = 3 * maxDpbPicBuf / 2                                   (1589)
else
    MaxDpbSize = maxDpbPicBuf
```

where MaxLumaPs is specified in Table A.2, maxDpbPicBuf is equal to 8, and dpb_max_dec_pic_buffering_minus1[Htid] is found in or derived from the applicable dpb_parameters( ) syntax structure.

Let numDecPics be the number of pictures in AU n. The variable AuSizeMaxInSamplesY[n] is set equal to PicSizeMaxInSamplesY * numDecPics.

When the specified level is not level 15.5, bitstreams conforming to the Main 10, Main 10 4:4:4, Multilayer Main 10, Multilayer Main 10 4:4:4, Main 12, Main 12 4:4:4, Main 16 4:4:4, Main 12 Intra, Main 12 4:4:4 Intra, or Main 16 4:4:4 Intra profile at a specified tier and level shall obey the following constraints for each bitstream conformance test as specified in Annex C:

a) The nominal removal time of AU n (with n greater than 0) from the CPB, as specified in clause C.2.3, shall satisfy the constraint that AuNominalRemovalTime[n] − AuCpbRemovalTime[n − 1] is greater than or equal to Max(AuSizeMaxInSamplesY[n − 1] ÷ MaxLumaSr, FrVal), where MaxLumaSr is the value specified in Table A.3 that applies to AU n − 1.

b) The difference between consecutive output times of pictures of different AUs from the DPB, as specified in clause C.3.3, shall satisfy the constraint that DpbOutputInterval[n] is greater than or equal to Max(AuSizeMaxInSamplesY[n] ÷ MaxLumaSr, FrVal), where MaxLumaSr is the value specified in Table A.3 for AU n, provided that AU n has a picture that is output and AU n is not the last AU of the bitstream that has a picture that is output.

c) The removal time of AU 0 shall satisfy the constraint that the number of slices in AU 0 is less than or equal to Min(Max(1, MaxSlicesPerAu * MaxLumaSr / MaxLumaPs * (AuCpbRemovalTime[0] − AuNominalRemovalTime[0]) + MaxSlicesPerAu * AuSizeMaxInSamplesY[0] / MaxLumaPs), MaxSlicesPerAu), where MaxSlicesPerAu, MaxLumaPs and MaxLumaSr are the values specified in Table A.2 and Table A.3, respectively, that apply to AU 0.

d) The difference between consecutive CPB removal times of AUs n and n − 1 (with n greater than 0) shall satisfy the constraint that the number of slices in AU n is less than or equal to Min((Max(1, MaxSlicesPerAu * MaxLumaSr / MaxLumaPs * (AuCpbRemovalTime[n] − AuCpbRemovalTime[n − 1]))), MaxSlicesPerAu), where MaxSlicesPerAu, MaxLumaPs, and MaxLumaSr are the values specified in Table A.2 and Table A.3 that apply to AU n.

e) For the VCL HRD parameters, BitRate[Htid][i] shall be less than or equal to BrVclFactor * MaxBR for at least one value of i in the range of 0 to hrd_cpb_cnt_minus1, inclusive, where BitRate[Htid][i] is specified in clause 7.4.6.3 based on parameters selected as specified in clause C.1 and MaxBR is specified in Table A.3 in units of BrVclFactor bits/s.

f) For the NAL HRD parameters, BitRate[Htid][i] shall be less than or equal to BrNalFactor * MaxBR for at least one value of i in the range of 0 to hrd_cpb_cnt_minus1, inclusive, where BitRate[Htid][i] is specified in clause 7.4.6.3 based on parameters selected as specified in clause C.1 and MaxBR is specified in Table A.3 in units of BrNalFactor bits/s.

g) The sum of the NumBytesInNalUnit variables for AU 0 shall be less than or equal to FormatCapabilityFactor * (Max(AuSizeMaxInSamplesY[0], FrVal * MaxLumaSr) + MaxLumaSr * (AuCpbRemovalTime[0] − AuNominalRemovalTime[0])) ÷ MinCr, where MaxLumaSr and FormatCapabilityFactor are the values specified in Table A.3 and Table A.4, respectively, that apply to AU 0.

h) The sum of the NumBytesInNalUnit variables for AU n (with n greater than 0) shall be less than or equal to FormatCapabilityFactor * MaxLumaSr * (AuCpbRemovalTime[n] − AuCpbRemovalTime[n − 1]) ÷ MinCr, where MaxLumaSr and FormatCapabilityFactor are the values specified in Table A.3 and Table A.4 respectively, that apply to AU n.

i) The removal time of AU 0 shall satisfy the constraint that the number of tiles in AU 0 is less than or equal to Min(Max(1, MaxTilesPerAu * 120 * (AuCpbRemovalTime[0] − AuNominalRemovalTime[0]) + MaxTilesPerAu * AuSizeMaxInSamplesY[0] / MaxLumaPs), MaxTilesPerAu), where MaxTilesPerAu is the value specified in Table A.2 that applies to AU 0.

j) The difference between consecutive CPB removal times of AUs n and n − 1 (with n greater than 0) shall satisfy the constraint that the number of tiles in AU n is less than or equal to Min(Max(1, MaxTilesPerAu * 120 *

(AuCpbRemovalTime[n] − AuCpbRemovalTime[n − 1])), MaxTilesPerAu), where MaxTilesPerAu is the value specified in Table A.2 that apply to AU n.

Table A.3 – Tier and level limits for the video profiles

Level	Max luma sample rate MaxLumaSr (samples/sec)	Max bit rate MaxBR (BrVclFactor or BrNalFactor bits/s)		Min compression ratio MinCrBase	
		Main tier	High tier	Main tier	High tier
1.0	552 960	128	-	2	-
2.0	3 686 400	1 500	-	2	-
2.1	7 372 800	3 000	-	2	-
3.0	16 588 800	6 000	-	2	-
3.1	33 177 600	10 000	-	2	-
4.0	66 846 720	12 000	30 000	4	4
4.1	133 693 440	20 000	50 000	4	4
5.0	267 386 880	25 000	100 000	6	4
5.1	534 773 760	40 000	160 000	8	4
5.2	1 069 547 520	60 000	240 000	8	4
6.0	1 069 547 520	60 000	240 000	8	4
6.1	2 139 095 040	120 000	480 000	8	4
6.2	4 278 190 080	240 000	800 000	8	4
6.3	4 812 963 840	320 000	1 600 000	8	4

Table A.4 – Specification of CpbVclFactor, CpbNalFactor, FormatCapabilityFactor and MinCrScaleFactor

Profiles	CpbVclFactor	CpbNalFactor	FormatCapabilityFactor	MinCrScaleFactor
Main 10, Multilayer Main 10	1 000	1 100	1.875	1.00
Main 10 Still Picture	1 000	1 100	n.a.	n.a.
Main 10 4:4:4, Multilayer Main 10 4:4:4	2 500	2 750	3.750	0.75
Main 10 4:4:4 Still Picture	2 500	2 750	n.a.	n.a.
Main 12	1 200	1 320	1.875	1.00
Main 12 Intra	2 400	2 640	1.875	1.00
Main 12 Still Picture	2 400	2 640	n.a.	n.a.
Main 12 4:4:4	3 000	3 300	3.750	0.75
Main 12 4:4:4 Intra	6 000	6 600	3.750	0.75
Main 12 4:4:4 Still Picture	6 000	6 600	n.a.	n.a.
Main 16 4:4:4	4 000	4 400	6.000	0.75
Main 16 4:4:4 Intra	8 000	8 800	6.000	0.75
Main 16 4:4:4 Still Picture	8 000	8 800	n.a.	n.a.

Informative clause A.4.3 shows the effect of these limits on picture rates for several example picture formats.

A.4.3 Effect of level limits on picture rate (informative)

This clause does not form an integral part of this Specification.

Informative Table A.5, Table A.6 and Table A.7 provide examples of maximum picture rates at the Main tier for the Main 10, Main 10 4:4:4, Multilayer Main 10, Multilayer Main 10 4:4:4, Main 12, Main 12 4:4:4, and Main 16 4:4:4 profiles.

Table A.5 – Maximum picture rates (pictures per second) at the Main tier, level 1.0 to 4.1 for some example picture sizes when MinCbSizeY is equal to 64

Level:				**1.0**	**2.0**	**2.1**	**3.0**	**3.1**	**4.0**	**4.1**
Max luma picture size (samples):				36 864	122 880	245 760	552 960	983 040	2 228 224	2 228 224
Max luma sample rate (samples/sec)				552 960	3 686 400	7 372 800	16 588 800	33 177 600	66 846 720	133 693 440
Format nickname	**Luma width**	**Luma height**	**Luma picture size**							
SQCIF	**128**	**96**	16 384	33.7	225.0	300.0	300.0	300.0	300.0	300.0
QCIF	**176**	**144**	36 864	15.0	100.0	200.0	300.0	300.0	300.0	300.0
QVGA	**320**	**240**	81 920	-	45.0	90.0	202.5	300.0	300.0	300.0
525 SIF	**352**	**240**	98 304	-	37.5	75.0	168.7	300.0	300.0	300.0
CIF	**352**	**288**	122 880	-	30.0	60.0	135.0	270.0	300.0	300.0
525 HHR	**352**	**480**	196 608	-	-	37.5	84.3	168.7	300.0	300.0
625 HHR	**352**	**576**	221 184	-	-	33.3	75.0	150.0	300.0	300.0
Q720p	**640**	**360**	245 760	-	-	30.0	67.5	135.0	272.0	300.0
VGA	**640**	**480**	327 680	-	-	-	50.6	101.2	204.0	300.0
525 4SIF	**704**	**480**	360 448	-	-	-	46.0	92.0	185.4	300.0
525 SD	**720**	**480**	393 216	-	-	-	42.1	84.3	170.0	300.0
4CIF	**704**	**576**	405 504	-	-	-	40.9	81.8	164.8	300.0
625 SD	**720**	**576**	442 368	-	-	-	37.5	75.0	151.1	300.0
480p (16:9)	**864**	**480**	458 752	-	-	-	36.1	72.3	145.7	291.4
SVGA	**800**	**600**	532 480	-	-	-	31.1	62.3	125.5	251.0
QHD	**960**	**540**	552 960	-	-	-	30.0	60.0	120.8	241.7
XGA	**1 024**	**768**	786 432	-	-	-	-	42.1	85.0	170.0
720p HD	**1 280**	**720**	983 040	-	-	-	-	33.7	68.0	136.0
4VGA	**1 280**	**960**	1 228 800	-	-	-	-	-	54.4	108.8
SXGA	**1 280**	**1 024**	1 310 720	-	-	-	-	-	51.0	102.0
525 16SIF	**1 408**	**960**	1 351 680	-	-	-	-	-	49.4	98.9
16CIF	**1 408**	**1 152**	1 622 016	-	-	-	-	-	41.2	82.4
4SVGA	**1 600**	**1 200**	1 945 600	-	-	-	-	-	34.3	68.7
1080 HD	**1 920**	**1 080**	2 088 960	-	-	-	-	-	32.0	64.0
2K×1K	**2 048**	**1 024**	2 097 152	-	-	-	-	-	31.8	63.7
2K×1080	**2 048**	**1 080**	2 228 224	-	-	-	-	-	30.0	60.0
4XGA	**2 048**	**1 536**	3 145 728	-	-	-	-	-	-	-
16VGA	**2 560**	**1 920**	4 915 200	-	-	-	-	-	-	-
3616×1536 (2.35:1)	**3 616**	**1 536**	5 603 328	-	-	-	-	-	-	-
3672×1536 (2.39:1)	**3 680**	**1 536**	5 701 632	-	-	-	-	-	-	-
3840×2160 (4*HD)	**3 840**	**2 160**	8 355 840	-	-	-	-	-	-	-
4K×2K	**4 096**	**2 048**	8 388 608	-	-	-	-	-	-	-
4096×2160	**4 096**	**2 160**	8 912 896	-	-	-	-	-	-	-
4096×2304 (16:9)	**4 096**	**2 304**	9 437 184	-	-	-	-	-	-	-
7680×4320	**7 680**	**4 320**	33 423 360	-	-	-	-	-	-	-
8192×4096	**8 192**	**4 096**	33 554 432	-	-	-	-	-	-	-
8192×4320	**8 192**	**4 320**	35 651 584	-	-	-	-	-	-	-

Table A.6 – Maximum picture rates (pictures per second) at the Main tier, level 5.0 to 5.2 for some example picture sizes when MinCbSizeY is equal to 64

Level:				**5.0**	**5.1**	**5.2**
Max luma picture size (samples):				8 912 896	8 912 896	8 912 896
Max luma sample rate (samples/sec)				267 386 880	534 773 760	1 069 547 520
Format nickname	**Luma width**	**Luma height**	**Luma picture size**			
SQCIF	**128**	**96**	16 384	300.0	300.0	300.0
QCIF	**176**	**144**	36 864	300.0	300.0	300.0
QVGA	**320**	**240**	81 920	300.0	300.0	300.0
525 SIF	**352**	**240**	98 304	300.0	300.0	300.0
CIF	**352**	**288**	122 880	300.0	300.0	300.0
525 HHR	**352**	**480**	196 608	300.0	300.0	300.0
625 HHR	**352**	**576**	221 184	300.0	300.0	300.0
Q720p	**640**	**360**	245 760	300.0	300.0	300.0
VGA	**640**	**480**	327 680	300.0	300.0	300.0
525 4SIF	**704**	**480**	360 448	300.0	300.0	300.0
525 SD	**720**	**480**	393 216	300.0	300.0	300.0
4CIF	**704**	**576**	405 504	300.0	300.0	300.0
625 SD	**720**	**576**	442 368	300.0	300.0	300.0
480p (16:9)	**864**	**480**	458 752	300.0	300.0	300.0
SVGA	**800**	**600**	532 480	300.0	300.0	300.0
QHD	**960**	**540**	552 960	300.0	300.0	300.0
XGA	**1 024**	**768**	786 432	300.0	300.0	300.0
720p HD	**1 280**	**720**	983 040	272.0	300.0	300.0
4VGA	**1 280**	**960**	1 228 800	217.6	300.0	300.0
SXGA	**1 280**	**1 024**	1 310 720	204.0	300.0	300.0
525 16SIF	**1 408**	**960**	1 351 680	197.8	300.0	300.0
16CIF	**1 408**	**1 152**	1 622 016	164.8	300.0	300.0
4SVGA	**1 600**	**1 200**	1 945 600	137.4	274.8	300.0
1080 HD	**1 920**	**1 080**	2 088 960	128.0	256.0	300.0
2K×1K	**2 048**	**1 024**	2 097 152	127.5	255.0	300.0
2K×1080	**2 048**	**1 080**	2 228 224	120.0	240.0	300.0
4XGA	**2 048**	**1 536**	3 145 728	85.0	170.0	300.0
16VGA	**2 560**	**1 920**	4 915 200	54.4	108.8	217.6
3616×1536 (2.35:1)	**3 616**	**1 536**	5 603 328	47.7	95.4	190.8
3672×1536 (2.39:1)	**3 680**	**1 536**	5 701 632	46.8	93.7	187.5
3840×2160 (4*HD)	**3 840**	**2 160**	8 355 840	32.0	64.0	128.0
4K×2K	**4 096**	**2 048**	8 388 608	31.8	63.7	127.5
4096×2160	**4 096**	**2 160**	8 912 896	30.0	60.0	120.0
4096×2304 (16:9)	**4 096**	**2 304**	9 437 184	-	-	-
4096×3072	**4 096**	**3 072**	12 582 912	-	-	-
7680×4320	**7 680**	**4 320**	33 423 360	-	-	-
8192×4096	**8 192**	**4 096**	33 554 432	-	-	-
8192×4320	**8 192**	**4 320**	35 651 584	-	-	-

Table A.7 – Maximum picture rates (pictures per second) at the Main tier, level 6.0 to 6.3 for some example picture sizes when MinCbSizeY is equal to 64

Level:				**6.0**	**6.1**	**6.2**	**6.3**
Max luma picture size (samples):				35 651 584	35 651 584	35 651 584	80 216 064
Max luma sample rate (samples/sec)				1 069 547 520	2 139 095 040	4 278 190 080	4 812 963 840
Format nickname	**Luma width**	**Luma height**	**Luma picture size**				
SQCIF	**128**	**96**	16 384	300.0	300.0	300.0	300.0
QCIF	**176**	**144**	36 864	300.0	300.0	300.0	300.0
QVGA	**320**	**240**	81 920	300.0	300.0	300.0	300.0
525 SIF	**352**	**240**	98 304	300.0	300.0	300.0	300.0
CIF	**352**	**288**	122 880	300.0	300.0	300.0	300.0
525 HHR	**352**	**480**	196 608	300.0	300.0	300.0	300.0
625 HHR	**352**	**576**	221 184	300.0	300.0	300.0	300.0
Q720p	**640**	**360**	245 760	300.0	300.0	300.0	300.0
VGA	**640**	**480**	327 680	300.0	300.0	300.0	300.0
525 4SIF	**704**	**480**	360 448	300.0	300.0	300.0	300.0
525 SD	**720**	**480**	393 216	300.0	300.0	300.0	300.0
4CIF	**704**	**576**	405 504	300.0	300.0	300.0	300.0
625 SD	**720**	**576**	442 368	300.0	300.0	300.0	300.0
480p (16:9)	**864**	**480**	458 752	300.0	300.0	300.0	300.0
SVGA	**800**	**600**	532 480	300.0	300.0	300.0	300.0
QHD	**960**	**540**	552 960	300.0	300.0	300.0	300.0
XGA	**1 024**	**768**	786 432	300.0	300.0	300.0	300.0
720p HD	**1 280**	**720**	983 040	300.0	300.0	300.0	300.0
4VGA	**1 280**	**960**	1 228 800	300.0	300.0	300.0	300.0
SXGA	**1 280**	**1 024**	1 310 720	300.0	300.0	300.0	300.0
525 16SIF	**1 408**	**960**	1 351 680	300.0	300.0	300.0	300.0
16CIF	**1 408**	**1 152**	1 622 016	300.0	300.0	300.0	300.0
4SVGA	**1 600**	**1 200**	1 945 600	300.0	300.0	300.0	300.0
1080 HD	**1 920**	**1 080**	2 088 960	300.0	300.0	300.0	300.0
2K×1K	**2 048**	**1 024**	2 097 152	300.0	300.0	300.0	300.0
2K×1080	**2 048**	**1 080**	2 228 224	300.0	300.0	300.0	300.0
4XGA	**2 048**	**1 536**	3 145 728	300.0	300.0	300.0	300.0
16VGA	**2 560**	**1 920**	4 915 200	217.6	300.0	300.0	300.0
3616×1536 (2.35:1)	**3 616**	**1 536**	5 603 328	190.8	300.0	300.0	300.0
3672×1536 (2.39:1)	**3 680**	**1 536**	5 701 632	187.5	300.0	300.0	300.0
3840×2160 (4*HD)	**3 840**	**2 160**	8 355 840	128.0	256.0	300.0	300.0
4K×2K	**4 096**	**2 048**	8 388 608	127.5	255.0	300.0	300.0
4096×2160	**4 096**	**2 160**	8 912 896	120.0	240.0	300.0	300.0
4096×2304 (16:9)	**4 096**	**2 304**	9 437 184	113.3	226.6	300.0	300.0
4096×3072	**4 096**	**3 072**	12 582 912	85.0	170.0	300.0	300.0
7680×4320	**7 680**	**4 320**	33 423 360	32.0	64.0	128.0	144.0
8192×4096	**8 192**	**4 096**	33 554 432	31.8	63.7	127.5	143.4
8192×4320	**8 192**	**4 320**	35 651 584	30.0	60.0	120.0	135.0
11520×6480	**11 520**	**6 480**	74 649 600	-	-	-	64.0
12288×6144	**12 288**	**6 144**	75 497 472	-	-	-	63.7
12288×6480	**12 288**	**6 480**	79 626 240	-	-	-	60.0

Informative Table A.8, Table A.9, and Table A.10 provide examples of maximum picture rates at the High tier for the Main 10, Main 10 4:4:4, Multilayer Main 10, Multilayer Main 10 4:4:4, Main 12, Main 12 4:4:4, and Main 16 4:4:4 profiles.

Table A.8 – Maximum picture rates (pictures per second) at the High tier, level 1.0 to 4.1 for some example picture sizes when MinCbSizeY is equal to 64

Level:				**1.0**	**2.0**	**2.1**	**3.0**	**3.1**	**4.0**	**4.1**
Max luma picture size (samples):				36 864	122 880	245 760	552 960	983 040	2 228 224	2 228 224
Max luma sample rate (samples/sec)				552 960	3 686 400	7 372 800	16 588 800	33 177 600	66 846 720	133 693 440
Format nickname	**Luma width**	**Luma height**	**Luma picture size**							
SQCIF	**128**	**96**	16 384	33.7	225.0	450.0	960.0	960.0	960.0	960.0
QCIF	**176**	**144**	36 864	15.0	100.0	200.0	450.0	900.0	960.0	960.0
QVGA	**320**	**240**	81 920	-	45.0	90.0	202.5	405.0	816.0	960.0
525 SIF	**352**	**240**	98 304	-	37.5	75.0	168.7	337.5	680.0	960.0
CIF	**352**	**288**	122 880	-	30.0	60.0	135.0	270.0	544.0	960.0
525 HHR	**352**	**480**	196 608	-	-	37.5	84.3	168.7	340.0	680.0
625 HHR	**352**	**576**	221 184	-	-	33.3	75.0	150.0	302.2	604.4
Q720p	**640**	**360**	245 760	-	-	30.0	67.5	135.0	272.0	544.0
VGA	**640**	**480**	327 680	-	-	-	50.6	101.2	204.0	408.0
525 4SIF	**704**	**480**	360 448	-	-	-	46.0	92.0	185.4	370.9
525 SD	**720**	**480**	393 216	-	-	-	42.1	84.3	170.0	340.0
4CIF	**704**	**576**	405 504	-	-	-	40.9	81.8	164.8	329.6
625 SD	**720**	**576**	442 368	-	-	-	37.5	75.0	151.1	302.2
480p (16:9)	**864**	**480**	458 752	-	-	-	36.1	72.3	145.7	291.4
SVGA	**800**	**600**	532 480	-	-	-	31.1	62.3	125.5	251.0
QHD	**960**	**540**	552 960	-	-	-	30.0	60.0	120.8	241.7
XGA	**1 024**	**768**	786 432	-	-	-	-	42.1	85.0	170.0
720p HD	**1 280**	**720**	983 040	-	-	-	-	33.7	68.0	136.0
4VGA	**1 280**	**960**	1 228 800	-	-	-	-	-	54.4	108.8
SXGA	**1 280**	**1 024**	1 310 720	-	-	-	-	-	51.0	102.0
525 16SIF	**1 408**	**960**	1 351 680	-	-	-	-	-	49.4	98.9
16CIF	**1 408**	**1 152**	1 622 016	-	-	-	-	-	41.2	82.4
4SVGA	**1 600**	**1 200**	1 945 600	-	-	-	-	-	34.3	68.7
1080 HD	**1 920**	**1 080**	2 088 960	-	-	-	-	-	32.0	64.0
2K×1K	**2 048**	**1 024**	2 097 152	-	-	-	-	-	31.8	63.7
2K×1080	**2 048**	**1 080**	2 228 224	-	-	-	-	-	30.0	60.0
4XGA	**2 048**	**1 536**	3 145 728	-	-	-	-	-	-	-
16VGA	**2 560**	**1 920**	4 915 200	-	-	-	-	-	-	-
3616×1536 (2.35:1)	**3 616**	**1 536**	5 603 328	-	-	-	-	-	-	-
3672×1536 (2.39:1)	**3 680**	**1 536**	5 701 632	-	-	-	-	-	-	-
3840×2160 (4*HD)	**3 840**	**2 160**	8 355 840	-	-	-	-	-	-	-
4K×2K	**4 096**	**2 048**	8 388 608	-	-	-	-	-	-	-
4096×2160	**4 096**	**2 160**	8 912 896	-	-	-	-	-	-	-
4096×2304 (16:9)	**4 096**	**2 304**	9 437 184	-	-	-	-	-	-	-
7680×4320	**7 680**	**4 320**	33 423 360	-	-	-	-	-	-	-
8192×4096	**8 192**	**4 096**	33 554 432	-	-	-	-	-	-	-
8192×4320	**8 192**	**4 320**	35 651 584	-	-	-	-	-	-	-

Table A.9 – Maximum picture rates (pictures per second) at the High tier, level 5.0 to 5.2 for some example picture sizes when MinCbSizeY is equal to 64

Level:				5.0	5.1	5.2
Max luma picture size (samples):				8 912 896	8 912 896	8 912 896
Max luma sample rate (samples/sec)				267 386 880	534 773 760	1 069 547 520
Format nickname	**Luma width**	**Luma height**	**Luma picture size**			
SQCIF	**128**	**96**	16 384	960.0	960.0	960.0
QCIF	**176**	**144**	36 864	960.0	960.0	960.0
QVGA	**320**	**240**	81 920	960.0	960.0	960.0
525 SIF	**352**	**240**	98 304	960.0	960.0	960.0
CIF	**352**	**288**	122 880	960.0	960.0	960.0
525 HHR	**352**	**480**	196 608	960.0	960.0	960.0
625 HHR	**352**	**576**	221 184	960.0	960.0	960.0
Q720p	**640**	**360**	245 760	960.0	960.0	960.0
VGA	**640**	**480**	327 680	816.0	960.0	960.0
525 4SIF	**704**	**480**	360 448	741.8	960.0	960.0
525 SD	**720**	**480**	393 216	680.0	960.0	960.0
4CIF	**704**	**576**	405 504	659.3	960.0	960.0
625 SD	**720**	**576**	442 368	604.4	960.0	960.0
480p (16:9)	**864**	**480**	458 752	582.8	960.0	960.0
SVGA	**800**	**600**	532 480	502.1	960.0	960.0
QHD	**960**	**540**	552 960	483.5	960.0	960.0
XGA	**1 024**	**768**	786 432	340.0	680.0	960.0
720p HD	**1 280**	**720**	983 040	272.0	544.0	960.0
4VGA	**1 280**	**960**	1 228 800	217.6	435.2	870.4
SXGA	**1 280**	**1 024**	1 310 720	204.0	408.0	816.0
525 16SIF	**1 408**	**960**	1 351 680	197.8	395.6	791.2
16CIF	**1 408**	**1 152**	1 622 016	164.8	329.6	659.3
4SVGA	**1 600**	**1 200**	1 945 600	137.4	274.8	549.7
1080 HD	**1 920**	**1 080**	2 088 960	128.0	256.0	512.0
2K×1K	**2 048**	**1 024**	2 097 152	127.5	255.0	510.0
2K×1080	**2 048**	**1 080**	2 228 224	120.0	240.0	480.0
4XGA	**2 048**	**1 536**	3 145 728	85.0	170.0	340.0
16VGA	**2 560**	**1 920**	4 915 200	54.4	108.8	217.6
3616×1536 (2.35:1)	**3 616**	**1 536**	5 603 328	47.7	95.4	190.8
3672×1536 (2.39:1)	**3 680**	**1 536**	5 701 632	46.8	93.7	187.5
3840×2160 (4*HD)	**3 840**	**2 160**	8 355 840	32.0	64.0	128.0
4K×2K	**4 096**	**2 048**	8 388 608	31.8	63.7	127.5
4096×2160	**4 096**	**2 160**	8 912 896	30.0	60.0	120.0
4096×2304 (16:9)	**4 096**	**2 304**	9 437 184	-	-	-
4096×3072	**4 096**	**3 072**	12 582 912	-	-	-
7680×4320	**7 680**	**4 320**	33 423 360	-	-	-
8192×4096	**8 192**	**4 096**	33 554 432	-	-	-
8192×4320	**8 192**	**4 320**	35 651 584	-	-	-

Table A.10 – Maximum picture rates (pictures per second) at the High tier, level 6.0 to 6.3 for some example picture sizes when MinCbSizeY is equal to 64

Level:				6.0	6.1	6.2	6.3
Max luma picture size (samples):				35 651 584	35 651 584	35 651 584	80 216 064
Max luma sample rate (samples/sec)				1 069 547 520	2 139 095 040	4 278 190 080	4 812 963 840
Format nickname	Luma width	Luma height	Luma picture size				
SQCIF	128	96	16 384	960.0	960.0	960.0	960.0
QCIF	176	144	36 864	960.0	960.0	960.0	960.0
QVGA	320	240	81 920	960.0	960.0	960.0	960.0
525 SIF	352	240	98 304	960.0	960.0	960.0	960.0
CIF	352	288	122 880	960.0	960.0	960.0	960.0
525 HHR	352	480	196 608	960.0	960.0	960.0	960.0
625 HHR	352	576	221 184	960.0	960.0	960.0	960.0
Q720p	640	360	245 760	960.0	960.0	960.0	960.0
VGA	640	480	327 680	960.0	960.0	960.0	960.0
525 4SIF	704	480	360 448	960.0	960.0	960.0	960.0
525 SD	720	480	393 216	960.0	960.0	960.0	960.0
4CIF	704	576	405 504	960.0	960.0	960.0	960.0
625 SD	720	576	442 368	960.0	960.0	960.0	960.0
480p (16:9)	864	480	458 752	960.0	960.0	960.0	960.0
SVGA	800	600	532 480	960.0	960.0	960.0	960.0
QHD	960	540	552 960	960.0	960.0	960.0	960.0
XGA	1 024	768	786 432	960.0	960.0	960.0	960.0
720p HD	1 280	720	983 040	960.0	960.0	960.0	960.0
4VGA	1 280	960	1 228 800	870.4	960.0	960.0	960.0
SXGA	1 280	1 024	1 310 720	816.0	960.0	960.0	960.0
525 16SIF	1 408	960	1 351 680	791.2	960.0	960.0	960.0
16CIF	1 408	1 152	1 622 016	659.3	960.0	960.0	960.0
4SVGA	1 600	1 200	1 945 600	549.7	960.0	960.0	960.0
1080 HD	1 920	1 080	2 088 960	512.0	960.0	960.0	960.0
2K×1K	2 048	1 024	2 097 152	510.0	960.0	960.0	960.0
2K×1080	2 048	1 080	2 228 224	480.0	960.0	960.0	960.0
4XGA	2 048	1 536	3 145 728	340.0	680.0	960.0	960.0
16VGA	2 560	1 920	4 915 200	217.6	435.2	870.4	960.0
3616×1536 (2.35:1)	3 616	1 536	5 603 328	190.8	381.7	763.5	858.9
3672×1536 (2.39:1)	3 680	1 536	5 701 632	187.5	375.1	750.3	844.1
3840×2160 (4*HD)	3 840	2 160	8 355 840	128.0	256.0	512.0	576.0
4K×2K	4 096	2 048	8 388 608	127.5	255.0	510.0	573.5
4096×2160	4 096	2 160	8 912 896	120.0	240.0	480.0	540.0
4096×2304 (16:9)	4 096	2 304	9 437 184	113.3	226.6	453.3	510.0
4096×3072	4 096	3 072	12 582 912	85.0	170.0	340.0	382.5
7680×4320	7 680	4 320	33 423 360	32.0	64.0	128.0	144.0
8192×4096	8 192	4 096	33 554 432	31.8	63.7	127.5	143.4
8192×4320	8 192	4 320	35 651 584	30.0	60.0	120.0	135.0
11520×6480	11 520	6 480	74 649 600	-	-	-	64.0
12288×6144	12 288	6 144	75 497 472	-	-	-	63.7
12288×6480	12 288	6 480	79 626 240	-	-	-	60.0

The following aspects are highlighted in regard to the examples shown in Table A.5 to Table A.10:

– This is a variable-picture-size specification. The specific listed picture sizes are illustrative examples only.

– The example luma picture sizes were computed by rounding up the luma width and luma height to multiples of 64 before computing the product of these quantities, to reflect the potential use of MinCbSizeY equal to 64 for these picture sizes, as pps_pic_width_in_luma_samples and pps_pic_height_in_luma_samples are each required to be a multiple of MinCbSizeY. For some illustrated values of luma width and luma height, a somewhat higher number of pictures per second can be supported when MinCbSizeY is less than 64.

– In cases where the maximum picture rate value is not an integer multiple of 0.1 pictures per second, the given maximum picture rate values have been rounded down to the largest integer multiple of 0.1 frames per second that does not exceed the exact value. For example, for level 3.1, the maximum picture rate for 720p HD has been rounded down to 33.7 from an exact value of 33.75.

– As used in the examples, "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.)

B.1 General

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 could (and under some circumstances, it shall) also contain an additional zero_byte syntax element. It could also contain one or more additional trailing_zero_8bits syntax elements. When it is the first byte stream NAL unit in the bitstream, it could also contain one or more additional leading_zero_8bits syntax elements.

B.2 Byte stream NAL unit syntax and semantics

B.2.1 Byte stream NAL unit syntax

byte_stream_nal_unit(NumBytesInNalUnit) {	**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.2.2 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 clause 7.4.2.4). The content of each byte stream NAL unit is associated with the same AU as the NAL unit contained in the byte stream NAL unit (see clause 7.4.2.4.4).

leading_zero_8bits is a byte equal to 0x00.

NOTE – The leading_zero_8bits syntax element could only be present in the first byte stream NAL unit of the bitstream, because (as shown in the syntax diagram of clause B.2.1) 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) would 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 one or more of the following conditions are true, the zero_byte syntax element shall be present:

– The nal_unit_type within the nal_unit( ) syntax structure is equal to DCI_NUT, OPI_NUT, VPS_NUT, SPS_NUT, PPS_NUT, PREFIX_APS_NUT, or SUFFIX_APS_NUT.

– The byte stream NAL unit syntax structure contains the first NAL unit of an AU in decoding order, as specified in clause 7.4.2.4.4.

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.3 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 initializes 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 (when 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 external 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 one or more of the following conditions:
   - 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 external 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 external 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 external means).

B.4 Decoder byte-alignment recovery (informative)

This clause does not form an integral part of this Specification.

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 clause.

A decoder is said to have byte alignment with a bitstream when the decoder has determined whether or not the positions of data in the bitstream are byte-aligned. When a decoder does not have byte alignment with the bitstream, 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 bitstream and positioned at the start of a NAL unit in the bitstream.

Once byte aligned with the bitstream, the decoder can examine the incoming bitstream data 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 clause 7.4.2.

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 clause 7.4.2), 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 in the manner described in this clause.

Annex C

Hypothetical reference decoder

(This annex forms an integral part of this Recommendation | International Standard.)

C.1 General

This annex specifies the hypothetical reference decoder (HRD) and its use to check bitstream and decoder conformance.

Two types of bitstreams or bitstream subsets are subject to HRD conformance checking for this Specification. The first type, called a Type I bitstream, is a NAL unit stream containing only the VCL NAL units, PH NAL units, and NAL units with nal_unit_type equal to FD_NUT (filler data NAL units) for all AUs in the bitstream. The second type, called a Type II bitstream, contains, in addition to the VCL NAL units, PH NAL units, and filler data NAL units for all AUs in the bitstream, at least one of the following:

– additional non-VCL NAL units other than filler data and PH 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.

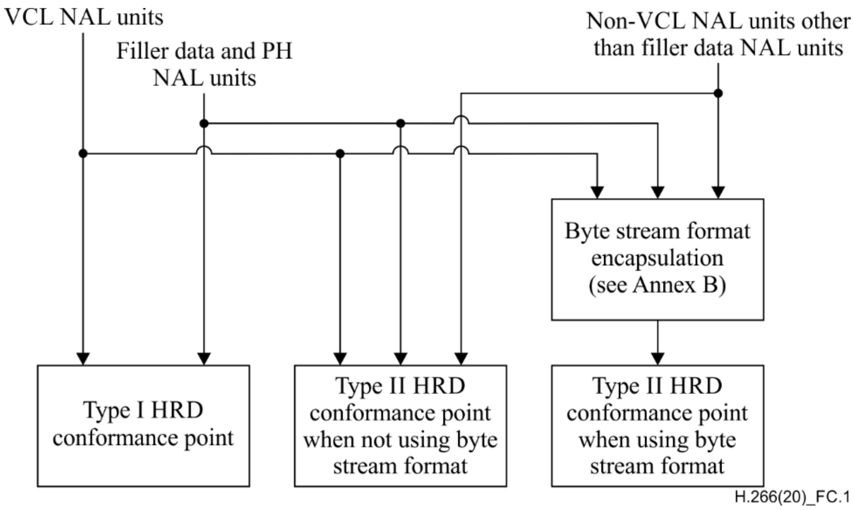

Figure C.1 – Flowchart of classification of byte streams and NAL unit streams for HRD conformance checks

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 clauses of clause 7 and Annex D.

Two sets of HRD parameters (NAL HRD parameters and VCL HRD parameters) are used. The HRD parameters are signalled through the general_timing_hrd_parameters( ) syntax structure and the ols_timing_hrd_parameters( ) syntax structure, which are either part of the referenced VPS (for multi-layer OLSs) or part of the referenced SPS (for single-layer OLSs).

A set of bitstream conformance tests is needed for checking the conformance of a bitstream, which is referred to as the entire bitstream, denoted as entireBitstream. The set of bitstream conformance tests are for testing the conformance of each OP of each OLS with OLS index in the range of 0 to TotalNumOlss − 1, respectively, and also for testing the conformance of each subpicture sequence specified by the subpicture level information SEI message.

For each test, the following ordered steps apply in the order listed, followed by the processes described after these steps in this clause:

1. An operation point under test, denoted as targetOp, is selected by selecting a target OLS with OLS index opOlsIdx, a highest TemporalId value opTid, and optionally, a list of target subpicture index values opSubpicIdxList[j] for j from 0 to NumLayersInOls[opOlsIdx] − 1, inclusive. The value of opOlsIdx is in the range of 0 to TotalNumOlss − 1, inclusive. The value of opTid is in the range of 0 to vps_max_sublayers_minus1, inclusive.

   If opSubpicIdxList[ ] is not present, targetOp consists of pictures, and each pair of the selected values of opOlsIdx and opTid shall be such that the sub-bitstream BitstreamToDecode that is the output by invoking the sub-bitstream extraction process as specified in clause C.6 with entireBitstream, opOlsIdx, and opTid as inputs satisifies the following condition:

– There is at least one VCL NAL unit with TemporalId equal to opTid in BitstreamToDecode.

Otherwise (opSubpicIdxList[ ] is present), targetOp consists of subpictures, and each set of the selected values of opOlsIdx, opTid, and opSubpicIdxList[j] for j from 0 to NumLayersInOls[opOlsIdx] − 1, inclusive, shall be such that the sub-bitstream BitstreamToDecode that is the output by invoking the subpicture sub-bitstream extraction process as specified in clause C.7 with entireBitstream, opOlsIdx, opTid, and opSubpicIdxList[j] for j from 0 to NumLayersInOls[opOlsIdx] − 1, inclusive, as inputs satisifies the following conditions:

– There is at least one VCL NAL unit with TemporalId equal to opTid in BitstreamToDecode.

– There is at least one VCL NAL unit with nuh_layer_id equal to LayerIdInOls[opOlsIdx][j] and sh_subpic_id equal to SubpicIdVal[opSubpicIdxList[j]] for each j in the range of 0 to NumLayersInOls[opOlsIdx] − 1, inclusive.

NOTE 1 – Regardless of whether opSubpicIdxList[ ] is present, due to the bitstream conformance requirement of each IRAP or GDR AU to be complete, there is at least one VCL NAL unit with nuh_layer_id equal to LayerIdInOls[opOlsIdx][j] for each j from 0 to NumLayersInOls[opOlsIdx] − 1, inclusive.

2. If opSubpicIdxList[ ] is not present the following applies:

– If the layers in targetOp include all layers in entireBitstream and opTid is equal to the highest TemporalId value among all NAL units in entireBitstream, BitstreamToDecode is set to be identical to entireBitstream.

– Otherwise, BitstreamToDecode is set to be the output by invoking the sub-bitstream extraction process as specified in clause C.6 with entireBitstream, opOlsIdx, and opTid as inputs.

Otherwise (opSubpicIdxList[ ] is present), BitstreamToDecode is set to be the output by invoking the subpicture sub-bitstream extraction process as specified in clause C.7 with entireBitstream, opOlsIdx, opTid and opSubpicIdxList[j] for j from 0 to NumLayersInOls[opOlsIdx] − 1, inclusive, as inputs.

3. The values of TargetOlsIdx and Htid are set equal to opOlsIdx and opTid, respectively, of targetOp.

4. The general_timing_hrd_parameters( ) syntax structure, the ols_timing_hrd_parameters( ) syntax structure, and the sublayer_hrd_parameters( ) syntax structure applicable to BitstreamToDecode are selected as follows:

– If NumLayersInOls[TargetOlsIdx] is equal to 1, the general_timing_hrd_parameters( ) syntax structure and the ols_timing_hrd_parameters( ) syntax structure in the SPS (or provided through an external means not specified in this Specification) are selected. Otherwise, the general_timing_hrd_parameters( ) syntax structure and the vps_ols_timing_hrd_idx[MultiLayerOlsIdx[TargetOlsIdx]]-th ols_timing_hrd_parameters( ) syntax structure in the VPS (or provided through an external means not specified in this Specification) are selected.

– Within the selected ols_timing_hrd_parameters( ) syntax structure, for testing of the Type I bitstream conformance point, the sublayer_hrd_parameters(Htid) syntax structure that immediately follows the condition "if(general_vcl_hrd_params_present_flag)" is selected and the variable NalHrdModeFlag is set equal to 0, and for testing of the Type II bitstream conformance point, the sublayer_hrd_parameters(Htid) syntax structure that immediately follows the condition "if(general_nal_hrd_params_present_flag)" is selected and the variable NalHrdModeFlag is set equal to 1. When BitstreamToDecode is a Type II bitstream and NalHrdModeFlag is equal to 0, all non-VCL NAL units except the PH and filler data NAL units, and 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), when present, are discarded from BitstreamToDecode and the remaining bitstream is assigned to BitstreamToDecode.

5. An AU associated with a BP SEI message (present in BitstreamToDecode or available through external means not specified in this Specification) applicable to TargetOp is selected as the HRD initialization point and referred to as AU 0.

6. When general_du_hrd_params_present_flag in the selected general_timing_hrd_parameters( ) syntax structure is equal to 1, the CPB is scheduled to operate either at the AU level (in which case the variable DecodingUnitHrdFlag is set equal to 0) or at the DU level (in which case the variable DecodingUnitHrdFlag is set equal to 1). Otherwise, DecodingUnitHrdFlag is set equal to 0 and the CPB is scheduled to operate at the AU level.

7. For each AU in BitstreamToDecode starting from AU 0, the BP SEI message (present in BitstreamToDecode or available through external means not specified in this Specification) that is associated with the AU and applies to TargetOlsIdx is selected, and the PT SEI message (present in BitstreamToDecode or available through external means not specified in this Specification) that is associated with the AU and applies to TargetOlsIdx is selected, and when DecodingUnitHrdFlag is equal to 1 and bp_du_cpb_params_in_pic_timing_sei_flag is equal to 0, the DUI SEI messages (present in BitstreamToDecode or available through external means not specified in this Specification) that are associated with DUs in the AU and apply to TargetOlsIdx are selected.

8. A value of ScIdx is selected. The selected ScIdx shall be in the range of 0 to hrd_cpb_cnt_minus1, inclusive.

9. When the BP SEI message associated with AU 0 has bp_alt_cpb_params_present_flag equal to 0, the variable DefaultInitCpbParamsFlag is set equal to 1. Otherwise, when the BP SEI message associated with AU 0 has bp_alt_cpb_params_present_flag equal to 1, either of the following applies for selection of the initial CPB removal delay and delay offset:

   – If NalHrdModeFlag is equal to 1, the default initial CPB removal delay and delay offset represented by bp_nal_initial_cpb_removal_delay[Htid][ScIdx] and bp_nal_initial_cpb_removal_offset[Htid][ScIdx], respectively, in the selected BP SEI message are selected. Otherwise, the default initial CPB removal delay and delay offset represented by bp_vcl_initial_cpb_removal_delay[Htid][ScIdx] and bp_vcl_initial_cpb_removal_offset[Htid][ScIdx], respectively, in the selected BP SEI message are selected. The variable DefaultInitCpbParamsFlag is set equal to 1.

   – If NalHrdModeFlag is equal to 1, the alternative initial CPB removal delay and delay offset represented by bp_nal_initial_cpb_removal_delay[Htid][ScIdx] and bp_nal_initial_cpb_removal_offset[Htid][ScIdx], respectively, in the selected BP SEI message and pt_nal_cpb_alt_initial_removal_delay_delta[Htid][ScIdx] and pt_nal_cpb_alt_initial_removal_offset_delta[Htid][ScIdx], respectively, in the PT SEI message associated with the AU following AU 0 in decoding order are selected. Otherwise, the alternative initial CPB removal delay and delay offset represented by bp_vcl_initial_cpb_removal_delay[Htid][ScIdx] and bp_vcl_initial_cpb_removal_offset[Htid][ScIdx], respectively, in the selected BP SEI message and pt_vcl_cpb_alt_initial_removal_delay_delta[Htid][ScIdx] and pt_vcl_cpb_alt_initial_removal_offset_delta[Htid][ScIdx], respectively, in the PT SEI message associated with the AU following AU 0 in decoding order are selected. The variable DefaultInitCpbParamsFlag is set equal to 0, and one of the following applies:

     – The RASL AUs that contain RASL pictures with pps_mixed_nalu_types_in_pic_flag equal to 0 and are associated with CRA pictures contained in AU 0 are discarded from BitstreamToDecode and the remaining bitstream is assigned to BitstreamToDecode.

     – All AUs following AU 0 in decoding order up to an AU associated with a dependent random access point (DRAP) indication SEI message are discarded from BitstreamToDecode and the remaining bitstream is assigned to BitstreamToDecode.

   NOTE 2 – A picture that follows the DRAP picture in decoding order and precedes the DRAP picture in output order could refer to a picture earlier than the DRAP picture in decoding order. Consequently, when the decoding starts from a DRAP picture, such a picture would not be correctly decodable.

Each conformance test consists of a combination of one option selected in each of these steps. When there is more than one option for a step, for any particular conformance test only one option is chosen. All possible combinations of all the steps form the entire set of conformance tests.

When BitstreamToDecode is a Type II bitstream, the following applies:

– If the sublayer_hrd_parameters(Htid) syntax structure that immediately follows the condition "if(general_vcl_hrd_params_present_flag)" is selected, the test is conducted at the Type I conformance point shown in Figure C.1, and only VCL and filler data NAL units are counted for the input bit rate and CPB storage.

– Otherwise (the sublayer_hrd_parameters(Htid) syntax structure that immediately follows the condition "if(general_nal_hrd_params_present_flag)" is selected), the test is conducted at the Type II conformance point shown in Figure C.1, and all bytes of the Type II bitstream, which could be a NAL unit stream or a byte stream, are counted for the input bit rate and CPB storage.

  NOTE 3 – NAL HRD parameters established by a value of ScIdx 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 InitCpbRemovalDelay[ScIdx], BitRate[Htid][ScIdx] and CpbSize[Htid][ScIdx] for the variable bit rate (VBR) case (cbr_flag[Htid][ScIdx] 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.

HRD conformance testing requires information that can be provided by PT and BP SEI messages. PT and BP SEI messages might not be present within the bitstream, so if HRD-related conformance is to be checked, equivalent information shall be made available and could be provided by external means outside of this Specification (such as system timing information).

When HRD conformance testing of a bitstream is performed, all DCI NAL units, when available, all VPSs, SPSs, PPSs, and APSs referenced by the VCL NAL units, and appropriate SLI, BP, PT, and DUI 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 Specification (such as system configuration and timing information).

In Annexes C and D, the specification for "presence" of non-VCL NAL units that contain DCI NAL units, VPSs, SPSs, PPSs, APSs, BP SEI messages, PT SEI messages, or DUI SEI messages, 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 in this Specification. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

NOTE 4 – As an example, synchronization of such a non-VCL NAL unit, conveyed by means other than presence in the bitstream, with the NAL units that are present in the bitstream, could 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 such 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 as specified in this Specification.

NOTE 5 – When HRD information is contained within the bitstream, it is possible to verify the conformance of a bitstream to the requirements of Annex C 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 could only be verified when the HRD data are supplied by some other means not specified in this Specification.

The HRD contains a bitstream extractor (optionally present), a CPB, an instantaneous decoding process, a DPB, and output cropping as shown in Figure C.2.

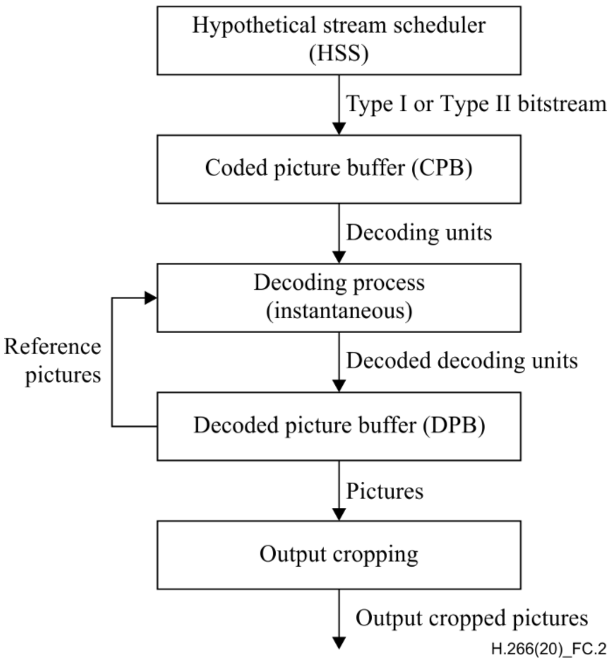

Figure C.2 – Flowchart of HRD buffer model

For each bitstream conformance test, the CPB size (number of bits) is CpbSize[Htid][ScIdx] as specified in clause 7.4.6.3, where ScIdx and the HRD parameters are specified above in this clause, and the DPB parameters dpb_max_dec_pic_buffering_minus1[Htid], dpb_max_num_reorder_pics[Htid], and MaxLatencyPictures[Htid] are found in or derived from the dpb_parameters( ) syntax structure that applies to the target OLS as follows:

– If NumLayersInOls[TargetOlsIdx] is equal to 1, the dpb_parameters( ) syntax structure is found in the SPS, and the variables PicWidthMaxInSamplesY, PicHeightMaxInSamplesY, MaxChromaFormat, and MaxBitDepthMinus8 are set equal to sps_pic_width_max_in_luma_samples, sps_pic_height_max_in_luma_samples, sps_chroma_format_idc, and sps_bitdepth_minus8, respectively, found in the SPS.

– Otherwise (NumLayersInOls[TargetOlsIdx] is greater than 1), the dpb_parameters( ) syntax structure is identified by vps_ols_dpb_params_idx[MultiLayerOlsIdx[TargetOlsIdx]] found in the VPS, and the variables PicWidthMaxInSamplesY, PicHeightMaxInSamplesY, MaxChromaFormat, and MaxBitDepthMinus8 are set equal to vps_ols_dpb_pic_width[MultiLayerOlsIdx[TargetOlsIdx]], vps_ols_dpb_pic_height[MultiLayerOlsIdx[TargetOlsIdx]], vps_ols_dpb_chroma_format[MultiLayerOlsIdx[TargetOlsIdx]], and vps_ols_dpb_bitdepth_minus8[MultiLayerOlsIdx[TargetOlsIdx]], respectively, found in the VPS.

If DecodingUnitHrdFlag is equal to 0, the HRD operates at the AU level and each DU is an AU. Otherwise the HRD operates at the DU level and each DU is a subset of an AU.

NOTE 6 – If the HRD operates at the AU level, each time when some bits are removed from the CPB, a DU that is an entire AU is removed from the CPB. Otherwise (the HRD operates at the DU level), each time when some bits are removed from the CPB, a DU that is a subset of an AU is removed from the CPB. Regardless of whether the HRD operates at access unt level or DU level, each time when some picture is output from the DPB, an entire decoded picture is output from the DPB, though the picture output time is derived based on the differently derived CPB removal times and the differently signalled DPB output delays.

The following is specified for expressing the constraints in this annex:

– Each AU is referred to as AU n, where the number n identifies the particular AU. AU 0 is selected per step 5 above. The value of n is incremented by 1 for each subsequent AU in decoding order.

– Each DU is referred to as DU m, where the number m identifies the particular DU. The first DU in decoding order in AU 0 is referred to as DU 0. The value of m is incremented by 1 for each subsequent DU in decoding order.

NOTE 7 – The numbering of DUs is relative to the first DU in AU 0.

– Picture n refers to the coded picture or the decoded picture of AU n.

The HRD operates as follows:

– The HRD is initialized at DU 0, with both the CPB and the DPB being set to be empty (the DPB fullness is set equal to 0).

NOTE 8 – After initialization, the HRD is not initialized again by subsequent BP SEI messages.

– Data associated with DUs that flow into the CPB according to a specified arrival schedule are delivered by the hypothetical stream scheduler (HSS).

– The data associated with each DU are removed and decoded instantaneously by the instantaneous decoding process at the CPB removal time of the DU.

– Each decoded picture is placed in the DPB.

– A decoded picture is removed from the DPB when it becomes no longer needed for inter prediction reference and no longer needed for output.

For each bitstream conformance test, the operation of the CPB is specified in clause C.2, the instantaneous decoder operation is specified in clauses 2 through 9, the operation of the DPB is specified in clause C.3 and the output cropping is specified in clauses C.3.3 and C.5.2.2.

HSS and HRD information concerning the number of enumerated delivery schedules and their associated bit rates and buffer sizes is specified in clauses 7.3.5.1 and 7.4.6.1. The HRD is initialized as specified by the BP SEI message (specified in clause D.3). The removal timing of DUs from the CPB and output timing of decoded pictures from the DPB is specified using information in PT SEI messages (specified in clause D.4) or in DUI SEI messages (specified in clause D.5). All timing information relating to a specific DU shall arrive prior to the CPB removal time of the DU.

The requirements for bitstream conformance are specified in clause C.4 and the HRD is used to check conformance of bitstreams as specified above in this clause and to check conformance of decoders as specified in clause C.5.

NOTE 9 – While conformance is guaranteed under the assumption that all picture rates and clocks used to generate the bitstream match exactly the values signalled in the bitstream, in a real system each of these might vary somewhat from the signalled or specified value.

All the arithmetic in this annex is performed 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 a DU is not necessarily an integer.

The variable ClockTick is derived as follows and is called a clock tick:

$$\text{ClockTick} = \text{num\_units\_in\_tick} \div \text{time\_scale} \quad (1590)$$

The variable ClockSubTick is derived as follows and is called a clock sub-tick:

$$\text{ClockSubTick} = \text{ClockTick} \div (\text{ tick\_divisor\_minus2} + 2 \text{ }) \quad (1591)$$

C.2 Operation of the CPB

C.2.1 General

The specifications in this clause 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 and the set of CPB parameters is selected as specified in clause C.1.

C.2.2 Timing of DU arrival

If DecodingUnitHrdFlag is equal to 0, the variable DecodingUnitParamsFlag is set equal to 0 and the process specified in the remainder of this clause is invoked with a DU being considered as an AU, for derivation of the initial and final CPB arrival times for AU n.

Otherwise (DecodingUnitHrdFlag is equal to 1), the process specified in the remainder of this clause is first invoked with DecodingUnitParamsFlag set equal to 0 and a DU being considered as an AU, for derivation of the initial and final CPB arrival times for AU n, and then invoked with DecodingUnitParamsFlag set equal to 1 and a DU being considered as a subset of an AU, for derivation of the initial and final CPB arrival times for the DUs in AU n.

The process specified in the remainder of this clause is invoked for derivation of the initial and final CPB arrival times for AU n.

The variables InitCpbRemovalDelay[ScIdx] and InitCpbRemovalDelayOffset[ScIdx] are derived as follows:

– If one or more of the following conditions are true, InitCpbRemovalDelay[ScIdx] and InitCpbRemovalDelayOffset[ScIdx] are set equal to the values of bp_nal_initial_cpb_removal_delay[Htid][ScIdx] and bp_nal_initial_cpb_removal_offset[Htid][ScIdx] minus the values of pt_nal_cpb_alt_initial_removal_delay_delta[Htid][ScIdx] and pt_nal_cpb_alt_initial_removal_offset_delta[Htid][ScIdx] of AU 1, respectively, when NalHrdModeFlag is equal to 1, or bp_vcl_initial_cpb_removal_delay[Htid][ScIdx] and bp_vcl_initial_cpb_removal_offset[Htid][ScIdx] minus the values of pt_vcl_cpb_alt_initial_removal_delay_delta[Htid][ScIdx] and pt_vcl_cpb_alt_initial_removal_offset_delta[Htid][ScIdx] of AU 1, respectively, when NalHrdModeFlag is equal to 0, where the BP SEI message and the PT SEI message is selected as specified in clause C.1:

  – UseAltCpbParamsFlag for AU 0 is equal to 1.

  – DefaultInitCpbParamsFlag is equal to 0.

– Otherwise, if the value of DecodingUnitParamsFlag is equal to 1, InitCpbRemovalDelay[ScIdx] and InitCpbRemovalDelayOffset[ScIdx] are set equal to the values of bp_nal_initial_alt_cpb_removal_delay[Htid][ScIdx] and bp_nal_initial_alt_cpb_removal_offset[Htid][ScIdx], respectively, when NalHrdModeFlag is equal to 1 or bp_vcl_initial_alt_cpb_removal_delay[Htid][ScIdx] and bp_vcl_initial_alt_cpb_removal_offset[Htid][ScIdx], respectively, when NalHrdModeFlag is equal to 0, where the BP SEI message is selected as specified in clause C.1.

– Otherwise, InitCpbRemovalDelay[ScIdx] and InitCpbRemovalDelayOffset[ScIdx] are set equal to the values of bp_nal_initial_cpb_removal_delay[Htid][ScIdx] and bp_nal_initial_cpb_removal_offset[Htid][ScIdx], respectively, when NalHrdModeFlag is equal to 1, or bp_vcl_initial_cpb_removal_delay[Htid][ScIdx] and bp_vcl_initial_cpb_removal_offset[Htid][ScIdx], respectively, when NalHrdModeFlag is equal to 0, where the BP SEI message is selected as specified in clause C.1.

The variables DpbDelayOffset and CpbDelayOffset are derived as follows with k being the AU associated with the BP SEI message:

– If one or more of the following conditions are true, DpbDelayOffset is set equal to the value of pt_nal_dpb_delay_offset[Htid] (when NalHrdModeFlag is equal to 1) or pt_vcl_dpb_delay_offset[Htid] (when NalHrdModeFlag is equal to 0) of AU k + 1, and CpbDelayOffset is set equal to the value of pt_nal_cpb_delay_offset[Htid] (when NalHrdModeFlag is equal to 1) or pt_vcl_cpb_delay_offset[Htid] (when NalHrdModeFlag is equal to 0) of AU k + 1, where the PT SEI message is selected as specified in clause C.1:

  – UseAltCpbParamsFlag for AU 0 is equal to 1.

  – DefaultInitCpbParamsFlag is equal to 0.

– Otherwise, DpbDelayOffset and CpbDelayOffset are set equal to 0.

The time at which the first bit of DU m begins to enter the CPB is referred to as the initial arrival time initArrivalTime[m].

The initial arrival time of DU m is derived as follows:

– If the AU is DU 0 (i.e., when m is equal to 0), initArrivalTime[0] is set equal to 0.

– Otherwise (the DU is DU m with m > 0), the following applies:

  – If cbr_flag[Htid][ScIdx] is equal to 1, the initial arrival time for DU m is equal to the final arrival time (which is derived in Equation 1597) of DU m − 1, i.e.,

```
if( !DecodingUnitParamsFlag )
    initArrivalTime[ m ] = AuFinalArrivalTime[ m − 1 ]                (1592)
else
    initArrivalTime[ m ] = DuFinalArrivalTime[ m − 1 ]
```

  – Otherwise (cbr_flag[Htid][ScIdx] is equal to 0), the initial arrival time for DU m is derived as follows:

```
if( !DecodingUnitParamsFlag )
    initArrivalTime[ m ] = Max( AuFinalArrivalTime[ m − 1 ], initArrivalEarliestTime[ m ] )      (1593)
else
    initArrivalTime[ m ] = Max( DuFinalArrivalTime[ m − 1 ], initArrivalEarliestTime[ m ] )
```

where initArrivalEarliestTime[m] is derived as follows:

– The variable tmpNominalRemovalTime is derived as follows:

```
if( !DecodingUnitParamsFlag )
    tmpNominalRemovalTime = AuNominalRemovalTime[ m ]      (1594)
else
    tmpNominalRemovalTime = DuNominalRemovalTime[ m ]
```

where AuNominalRemovalTime[m] and DuNominalRemovalTime[m] are the nominal CPB removal time of AU m and DU m, respectively, as specified in clause C.2.3.

– If DU m is not the first DU of a subsequent BP, initArrivalEarliestTime[m] is derived as follows:

```
initArrivalEarliestTime[ m ] = tmpNominalRemovalTime −
        ( InitCpbRemovalDelay[ ScIdx ] + InitCpbRemovalDelayOffset[ ScIdx ] ) ÷ 90000      (1595)
```

– Otherwise (DU m is the first DU of a subsequent BP), initArrivalEarliestTime[m] is derived as follows:

```
initArrivalEarliestTime[ m ] = tmpNominalRemovalTime −
        ( InitCpbRemovalDelay[ ScIdx ] ÷ 90000 )      (1596)
```

The final arrival time for DU m is derived as follows:

```
if( !decodingUnitParamsFlag )
    AuFinalArrivalTime[ m ] = initArrivalTime[ m ] + sizeInbits[ m ] ÷ BitRate[ Htid ][ ScIdx ]      (1597)
else
    DuFinalArrivalTime[ m ] = initArrivalTime[ m ] + sizeInbits[ m ] ÷ BitRate[ Htid ][ ScIdx ]
```

where sizeInbits[m] is the size in bits of DU m, counting the bits of the VCL NAL units, the PH 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 ScIdx, BitRate[Htid][ScIdx] and CpbSize[Htid][ScIdx] are constrained as follows:

– If the content of the selected general_timing_hrd_parameters( ) syntax structures for the AU containing AU m and the previous AU differ, the HSS selects a value ScIdx1 of ScIdx from among the values of ScIdx provided in the selected general_timing_hrd_parameters( ) syntax structures for the AU containing AU m that results in a BitRate[Htid][ScIdx1] or CpbSize[Htid][ScIdx1] for the AU containing AU m. The value of BitRate[Htid][ScIdx1] or CpbSize[Htid][ScIdx1] may differ from the value of BitRate[Htid][ScIdx0] or CpbSize[Htid][ScIdx0] for the value ScIdx0 of ScIdx that was in use for the previous AU.

– Otherwise, the HSS continues to operate with the previous values of ScIdx, BitRate[Htid][ScIdx] and CpbSize[Htid][ScIdx].

When the HSS selects values of BitRate[Htid][ScIdx] or CpbSize[Htid][ScIdx] that differ from those of the previous AU, the following applies:

– The variable BitRate[Htid][ScIdx] comes into effect at the initial CPB arrival time of the current AU.

– The variable CpbSize[Htid][ScIdx] comes into effect as follows:

  – If the new value of CpbSize[Htid][ScIdx] is greater than the old CPB size, it comes into effect at the initial CPB arrival time of the current AU.

  – Otherwise, the new value of CpbSize[Htid][ScIdx] comes into effect at the CPB removal time of the current AU.

C.2.3 Timing of DU removal and decoding of DU

The values of the variables InitCpbRemovalDelay[ScIdx] and InitCpbRemovalDelayOffset[ScIdx] are updated as follows:

– If one or more of the following conditions are true, InitCpbRemovalDelay[ScIdx] and InitCpbRemovalDelayOffset[ScIdx] are set equal to the values of bp_nal_initial_cpb_removal_delay[Htid][ScIdx] and bp_nal_initial_cpb_removal_offset[Htid][ScIdx] minus the values of pt_nal_cpb_alt_initial_removal_delay_delta[Htid][ScIdx] and pt_nal_cpb_alt_initial_removal_offset_delta[Htid][ScIdx] of AU n + 1, respectively, when NalHrdModeFlag is equal to 1, or the values of bp_vcl_initial_cpb_removal_delay[Htid][ScIdx] and pt_vcl_bp_vcl_initial_cpb_removal_offset[Htid][ScIdx] minus the values of pt_vcl_cpb_alt_initial_removal_delay_delta[Htid][ScIdx] and cpb_alt_initial_removal_offset_delta[Htid][ScIdx] of AU n + 1, respectively, when NalHrdModeFlag is equal to 0, where the BP SEI message and the PT SEI message are selected as specified in clause C.1:

  – UseAltCpbParamsFlag for AU n is equal to 1.

  – DefaultInitCpbParamsFlag is equal to 0.

– Otherwise, if the value of DecodingUnitParamsFlag is equal to 1, InitCpbRemovalDelay[ScIdx] and InitCpbRemovalDelayOffset[ScIdx] are set equal to the values of the BP SEI message syntax elements bp_nal_initial_alt_cpb_removal_delay[Htid][ScIdx] and bp_nal_initial_alt_cpb_removal_offset[Htid][ScIdx], respectively, when NalHrdModeFlag is equal to 1, or bp_vcl_initial_alt_cpb_removal_delay[Htid][ScIdx] and bp_vcl_initial_alt_cpb_removal_offset[Htid][ScIdx], respectively, when NalHrdModeFlag is equal to 0, where the BP SEI message containing the syntax elements is selected as specified in clause C.1.

– Otherwise, InitCpbRemovalDelay[ScIdx] and InitCpbRemovalDelayOffset[ScIdx] are set equal to the values of bp_nal_initial_cpb_removal_delay[Htid][ScIdx] and bp_nal_initial_cpb_removal_offset[Htid][ScIdx], respectively, when NalHrdModeFlag is equal to 1, or bp_vcl_initial_cpb_removal_delay[Htid][ScIdx] and bp_vcl_initial_cpb_removal_offset[Htid][ScIdx], respectively, when NalHrdModeFlag is equal to 0, where the BP SEI message is selected as specified in clause C.1.

The nominal removal time of the AU n from the CPB is specified as follows:

– If AU n is the AU with n equal to 0 (the AU that initializes the HRD), the nominal removal time of the AU from the CPB is specified by:

AuNominalRemovalTime[0] = InitCpbRemovalDelay[ScIdx] ÷ 90000 (1598)

– Otherwise, the following applies:

  – When AU n is the first AU of a BP that does not initialize the HRD, the following applies:

  The nominal removal time of the AU n from the CPB is specified by:

```
if( !concatenationFlag ) {
    baseTime = AuNominalRemovalTime[ firstAuInPrevBuffPeriod ]
    tmpCpbRemovalDelay = AuCpbRemovalDelayVal
    tmpCpbDelayOffset = CpbDelayOffset
} else {
    baseTime1 = AuNominalRemovalTime[ prevNonDiscardableAu ]
    tmpCpbRemovalDelay1 = ( auCpbRemovalDelayDeltaMinus1 + 1 )
    baseTime2 = AuNominalRemovalTime[ n − 1 ]
    tmpCpbRemovalDelay2 = Ceil( ( InitCpbRemovalDelay[ ScIdx ] ÷ 90000 +                    (1599)
            AuFinalArrivalTime[ n − 1 ] − AuNominalRemovalTime[ n − 1 ] ) ÷ ClockTick )
    if( baseTime1 + ClockTick * tmpCpbRemovalDelay1 <
            baseTime2 + ClockTick * tmpCpbRemovalDelay2 ) {
        baseTime = baseTime2
        tmpCpbRemovalDelay = tmpCpbRemovalDelay2
    } else {
        baseTime = baseTime1
        tmpCpbRemovalDelay = tmpCpbRemovalDelay1
    }
    tmpCpbDelayOffset = 0
}
AuNominalRemovalTime[ n ] =
baseTime + ( ClockTick * tmpCpbRemovalDelay − tmpCpbDelayOffset )
```

  where AuNominalRemovalTime[firstAuInPrevBuffPeriod] is the nominal removal time of the first AU of the previous BP, AuNominalRemovalTime[prevNonDiscardableAu] is the nominal removal time of the previous AU in decoding order with TemporalId equal to 0 that has at least one picture that has ph_non_ref_pic_flag equal to 0 that is not a RASL or RADL picture, AuCpbRemovalDelayVal is the value of CpbRemovalDelayVal[Htid] derived according to pt_cpb_removal_delay_minus1[Htid] and pt_cpb_removal_delay_delta_idx[Htid] in the PT SEI message, and bp_cpb_removal_delay_delta_val[pt_cpb_removal_delay_delta_idx[Htid]] in the BP SEI message, selected as specified in clause C.1, associated with AU n and concatenationFlag and auCpbRemovalDelayDeltaMinus1 are the values of the syntax elements bp_concatenation_flag and bp_cpb_removal_delay_delta_minus1, respectively, in the BP SEI message, selected as specified in clause C.1, associated with AU n.

After the derivation of the nominal CPB removal time and before the derivation of the DPB output time of access unit n, the variables DpbDelayOffset and CpbDelayOffset are derived as:

- If one or more of the following conditions are true, DpbDelayOffset is set equal to the value of the PT SEI message syntax element pt_nal_dpb_delay_offset[Htid] (when NalHrdModeFlag is equal to 1) or pt_vcl_dpb_delay_offset[Htid] (when NalHrdModeFlag is equal to 0) of AU n + 1, and CpbDelayOffset is set equal to the value of the PT SEI message syntax element pt_nal_cpb_delay_offset[Htid] (when NalHrdModeFlag is equal to 1) or pt_vcl_cpb_delay_offset[Htid] (when NalHrdModeFlag is equal to 0) of AU n + 1, where the PT SEI message containing the syntax elements is selected as specified in clause C.1:
  - UseAltCpbParamsFlag for AU n is equal to 1.
  - DefaultInitCpbParamsFlag is equal to 0.
- Otherwise, DpbDelayOffset and CpbDelayOffset are both set equal to 0.

– When AU n is not the first AU of a BP, the nominal removal time of the AU n from the CPB is specified by:

$$\text{AuNominalRemovalTime}[\ n\ ] = \text{AuNominalRemovalTime}[\ \text{firstAuInCurrBuffPeriod}\ ] + \text{ClockTick} * (\ \text{AuCpbRemovalDelayVal} - \text{CpbDelayOffset}\ ) \tag{1600}$$

where AuNominalRemovalTime[firstAuInCurrBuffPeriod] is the nominal removal time of the first AU of the current BP and AuCpbRemovalDelayVal is the value of CpbRemovalDelayVal[OpTid] derived according to pt_cpb_removal_delay_minus1[OpTid] and pt_cpb_removal_delay_delta_idx[OpTid] in the PT SEI message, and bp_cpb_removal_delay_delta_val[pt_cpb_removal_delay_delta_idx[OpTid]] in the BP SEI message, selected as specified in clause C.1, associated with AU n.

When DecodingUnitHrdFlag is equal to 1, the following applies:

- When pt_num_decoding_units_minus1 is greater than 0, the variable duCpbRemovalDelayInc is derived as follows:
  - If bp_du_cpb_params_in_pic_timing_sei_flag is equal to 0, duCpbRemovalDelayInc is set equal to the value of dui_du_cpb_removal_delay_increment[i] in the DUI SEI message, selected as specified in clause C.1, associated with DU m.
  - Otherwise, if pt_du_common_cpb_removal_delay_flag is equal to 0, duCpbRemovalDelayInc is set equal to the value of pt_du_cpb_removal_delay_increment_minus1[i][Htid] + 1 for DU m in the PT SEI message, selected as specified in clause C.1, associated with AU n, where the value of i is 0 for the first pt_num_nalus_in_du_minus1[0] + 1 consecutive NAL units in the AU that contains DU m, 1 for the subsequent pt_num_nalus_in_du_minus1[1] + 1 NAL units in the same AU, 2 for the subsequent pt_num_nalus_in_du_minus1[2] + 1 NAL units in the same AU, etc.
  - Otherwise, duCpbRemovalDelayInc is set equal to the value of pt_du_common_cpb_removal_delay_increment_minus1[Htid] + 1 in the PT SEI message, selected as specified in clause C.1, associated with AU n.
- The nominal removal time of DU m from the CPB is specified as follows, where AuNominalRemovalTime[n] is the nominal removal time of AU n:
  - If DU m is the last DU in AU n, the nominal removal time of DU m DuNominalRemovalTime[m] is set equal to AuNominalRemovalTime[n].
  - Otherwise (DU m is not the last DU in AU n), the nominal removal time of DU m DuNominalRemovalTime[m] is derived as follows:

```
if( bp_du_cpb_params_in_pic_timing_sei_flag )
    DuNominalRemovalTime[ m ] = DuNominalRemovalTime[ m + 1 ] −
        ClockSubTick * duCpbRemovalDelayInc                                  (1601)
else
```

DuNominalRemovalTime[m] = AuNominalRemovalTime[n] − ClockSubTick * duCpbRemovalDelayInc

If DecodingUnitHrdFlag is equal to 0, the removal time of AU n from the CPB is specified as follows, where AuFinalArrivalTime[n] and AuNominalRemovalTime[n] are the final CPB arrival time and nominal CPB removal time, respectively, of AU n:

```
if( !low_delay_hrd_flag[ Htid ] || AuNominalRemovalTime[ n ] >= AuFinalArrivalTime[ n ] )
    AuCpbRemovalTime[ n ] = AuNominalRemovalTime[ n ]
else                                                                                    (1602)
    AuCpbRemovalTime[ n ] = AuNominalRemovalTime[ n ] + ClockTick *
        Ceil( ( AuFinalArrivalTime[ n ] − AuNominalRemovalTime[ n ] ) ÷ ClockTick )
```

NOTE 1 – When low_delay_hrd_flag[Htid] is equal to 1 and AuNominalRemovalTime[n] is less than AuFinalArrivalTime[n], the size of AU n is so large that it prevents removal at the nominal removal time.

Otherwise (DecodingUnitHrdFlag is equal to 1), the removal time of DU m from the CPB is specified as follows:

```
if( !low_delay_hrd_flag[ Htid ] || DuNominalRemovalTime[ m ] >= DuFinalArrivalTime[ m ] )
    DuCpbRemovalTime[ m ] = DuNominalRemovalTime[ m ]
else                                                                                    (1603)
    DuCpbRemovalTime[ m ] = DuFinalArrivalTime[ m ]
```

NOTE 2 – When low_delay_hrd_flag[Htid] is equal to 1 and DuNominalRemovalTime[m] is less than DuFinalArrivalTime[m], the size of DU m is so large that it prevents removal at the nominal removal time.

If DecodingUnitHrdFlag is equal to 0, at the CPB removal time of AU n, the AU is instantaneously decoded.

Otherwise (DecodingUnitHrdFlag is equal to 1), at the CPB removal time of DU m, the DU is instantaneously decoded, and when DU m is the last DU of AU n, the following applies:

– Picture n is considered as decoded.

– The final CPB arrival time of AU n, i.e., AuFinalArrivalTime[n], is set equal to the final CPB arrival time of the last DU in AU n, i.e., DuFinalArrivalTime[m].

– The nominal CPB removal time of AU n, i.e., AuNominalRemovalTime[n], is set equal to the nominal CPB removal time of the last DU in AU n, i.e., DuNominalRemovalTime[m].

– The CPB removal time of AU n, i.e., AuCpbRemovalTime[m], is set equal to the CPB removal time of the last DU in AU n, i.e., DuCpbRemovalTime[m].

C.3 Operation of the DPB

C.3.1 General

The specifications in this clause apply independently to each set of DPB parameters selected as specified in clause C.1.

The DPB contains picture storage buffers for storage of decoded pictures. Each of the picture storage buffers could contain a decoded picture that is marked as "used for reference" or is held for future output. The processes specified in clauses C.3.2, C.3.3, and C.3.4 are sequentially applied, and are separately applied for each layer, starting from the lowest layer in the OLS, in increasing order of nuh_layer_id values of the layers in the OLS.

NOTE – In the operation of output timing DPB, decoded pictures with PictureOutputFlag equal to 1 in the same AU are output consecutively in ascending order of the nuh_layer_id values of the decoded pictures.

Let picture n and the current picture be the coded picture or decoded picture of the AU n for a particular value of nuh_layer_id, wherein n is a non-negative integer number.

C.3.2 Removal of pictures from the DPB before decoding of the current picture

The removal of pictures from the DPB before decoding of the current picture (but after parsing the slice header of the first slice of the current picture) happens instantaneously at the CPB removal time of the first DU of AU n (containing the current picture) and proceeds as follows:

– The decoding process for RPL construction as specified in clause 8.3.2 is invoked and the decoding process for reference picture marking as specified in clause 8.3.3 is invoked.

– When the current picture is the first picture of a CVSS AU that is not AU 0, the following ordered steps are applied:

1. The variable NoOutputOfPriorPicsFlag is derived for the decoder under test as follows:

   – If the value of PicWidthMaxInSamplesY, PicHeightMaxInSamplesY, MaxChromaFormat, MaxBitDepthMinus8, or dpb_max_dec_pic_buffering_minus1[Htid] derived for the current AU is different from the value of PicWidthMaxInSamplesY, PicHeightMaxInSamplesY, MaxChromaFormat, MaxBitDepthMinus8, or dpb_max_dec_pic_buffering_minus1[Htid], respectively, derived for the preceding AU in decoding order, NoOutputOfPriorPicsFlag may (but should not) be set equal to 1 by the decoder under test, regardless of the value of sh_no_output_of_prior_pics_flag of the current AU.

     NOTE – Although setting NoOutputOfPriorPicsFlag equal to sh_no_output_of_prior_pics_flag of the current AU is preferred under these conditions, the decoder under test is allowed to set NoOutputOfPriorPicsFlag equal to 1 in this case.

   – Otherwise, NoOutputOfPriorPicsFlag is set equal to sh_no_output_of_prior_pics_flag of the current AU.

2. The value of NoOutputOfPriorPicsFlag derived for the decoder under test is applied for the HRD, such that when the value of NoOutputOfPriorPicsFlag is equal to 1, all picture storage buffers in the DPB are emptied without output of the pictures they contain, and the DPB fullness is set equal to 0.

– When the current picture is not the first picture in AU 0 and both of the following conditions are true for any pictures k in the DPB, all such pictures k in the DPB are removed from the DPB and the DPB fullness is decremented by one for each of such pictures:

  – picture k is marked as "unused for reference".

  – picture k has PictureOutputFlag equal to 0 or its DPB output time is less than or equal to the CPB removal time of the first DU (denoted as DU m) of the current picture n; i.e., DpbOutputTime[k] is less than or equal to DuCpbRemovalTime[m].

C.3.3 Picture output

The processes specified in this clause happen instantaneously at the CPB removal time of AU n, AuCpbRemovalTime[n].

When picture n has PictureOutputFlag equal to 1, its DPB output time DpbOutputTime[n] is derived as follows, where the variable firstPicInBufferingPeriodFlag is equal to 1 if AU n is the first AU of a BP and 0 otherwise:

```
if( !DecodingUnitHrdFlag ) {
    DpbOutputTime[ n ] = AuCpbRemovalTime[ n ] + ClockTick * ( pt_dpb_output_delay –        (1604)
                AuDpbOutputDelta[ Htid ] )
    if( firstPicInBufferingPeriodFlag )
        DpbOutputTime[ n ] −= ClockTick * DpbDelayOffset
} else
    DpbOutputTime[ n ] = AuCpbRemovalTime[ n ] + ClockSubTick * dpbOutputDuDelay
```

where AuDpbOutputDelta[Htid] is derived according to pt_cpb_removal_delay_minus1[Htid] and pt_cpb_removal_delay_delta_idx[Htid] in the PT SEI message associated with AU n and bp_cpb_removal_delay_delta_val[pt_cpb_removal_delay_delta_idx[Htid]] and bp_dpb_output_tid_offset[Htid] in the BP SEI message associated with AU n, and dpbOutputDuDelay is the value of dui_dpb_output_du_delay in the DUI SEI messages associated with AU n when bp_du_dpb_params_in_pic_timing_sei_flag is equal to 0, or the value of pt_dpb_output_du_delay in the PT SEI message associated with AU n when bp_du_dpb_params_in_pic_timing_sei_flag is equal top 1.

The output of the current picture is specified as follows:

– If PictureOutputFlag is equal to 1 and DpbOutputTime[n] is equal to AuCpbRemovalTime[n], the current picture is output.

– Otherwise, if PictureOutputFlag is equal to 0, the current picture is not output, but will be stored in the DPB as specified in clause C.3.4.

– Otherwise (PictureOutputFlag is equal to 1 and DpbOutputTime[n] is greater than AuCpbRemovalTime[n]), the current picture is output later and will be stored in the DPB (as specified in clause C.3.4) and is output at time DpbOutputTime[n] unless indicated not to be output by NoOutputOfPriorPicsFlag equal to 1.

When output, the picture is cropped, using the conformance cropping window for the picture.

When picture n is a picture that is output and is not the last picture of the bitstream that is output, the value of the variable DpbOutputInterval[n] is derived as follows:

$$\text{DpbOutputInterval[n] = DpbOutputTime[nextPicInOutputOrder]} - \text{DpbOutputTime[n]} \quad (1605)$$

where nextPicInOutputOrder is the picture that follows picture n in output order and has PictureOutputFlag equal to 1.

C.3.4 Current decoded picture marking and storage

The current decoded picture is stored in the DPB in an empty picture storage buffer, the DPB fullness is incremented by one, and the current picture is marked as "used for short-term reference".

NOTE – Unless more memory than required by the level limit is available for storage of decoded pictures, decoders are expected to start storing decoded parts of the current picture into the DPB when the first slice is decoded and continue storing more decoded samples as the decoding process proceeds.

C.4 Bitstream conformance

A bitstream of coded data conforming to this Specification shall fulfil all requirements specified in this clause.

The bitstream shall be constructed according to the syntax, semantics and constraints specified in this Specification outside of this annex.

The first coded picture in a bitstream shall be an IRAP picture (i.e., an IDR picture or a CRA picture) or a GDR picture.

The bitstream is tested by the HRD for conformance as specified in clause C.1.

Let currPicLayerId be equal to the nuh_layer_id of the current picture.

For each current picture, let the variables maxPicOrderCnt and minPicOrderCnt be set equal to the maximum and the minimum, respectively, of the PicOrderCntVal values of the following pictures with nuh_layer_id equal to currPicLayerId:

– The current picture.

– The previous picture in decoding order that has TemporalId and ph_non_ref_pic_flag both equal to 0 and is not a RASL or RADL picture.

– The STRPs referred to by all entries in RefPicList[0] and all entries in RefPicList[1] of the current picture.

– All pictures n that have PictureOutputFlag equal to 1, AuCpbRemovalTime[n] less than AuCpbRemovalTime[currPic] and DpbOutputTime[n] greater than or equal to AuCpbRemovalTime[currPic], where currPic is the current picture.

All of the following conditions shall be fulfilled for each of the bitstream conformance tests:

1. For each AU n, with n greater than 0, associated with a BP SEI message, let the variable deltaTime90k[n] be specified as follows:

$$\text{deltaTime90k[n]} = 90000 * (\text{AuNominalRemovalTime[n]} - \text{AuFinalArrivalTime[n} - 1 \text{]}) \quad (1606)$$

   The value of InitCpbRemovalDelay[ScIdx] is constrained as follows:

   – If cbr_flag[Htid][ScIdx] is equal to 0, the following condition shall be true:

$$\text{InitCpbRemovalDelay[ScIdx]} <= \text{Ceil(deltaTime90k[n])} \quad (1607)$$

   – Otherwise (cbr_flag[Htid][ScIdx] is equal to 1), the following condition shall be true:

$$\text{Floor(deltaTime90k[n])} <= \text{InitCpbRemovalDelay[ScIdx]} <= \text{Ceil(deltaTime90k[n])} \quad (1608)$$

   NOTE 1 – The exact number of bits in the CPB at the removal time of each AU or DU could depend on which BP SEI message is selected to initialize the HRD. Encoders are expected to take this into account to ensure that all specified constraints are obeyed regardless of which BP SEI message is selected to initialize the HRD, as the HRD could be initialized at any one of the BP SEI messages.

2. A CPB overflow is specified as the condition in which the total number of bits in the CPB is greater than the CPB size. The CPB shall never overflow.

3. When low_delay_hrd_flag[Htid] is equal to 0, the CPB shall never underflow. A CPB underflow is specified as follows:

   – If DecodingUnitHrdFlag is equal to 0, a CPB underflow is specified as the condition in which the nominal CPB removal time of AU n AuNominalRemovalTime[n] is less than the final CPB arrival time of AU n AuFinalArrivalTime[n] for at least one value of n.

– Otherwise (DecodingUnitHrdFlag is equal to 1), a CPB underflow is specified as the condition in which the nominal CPB removal time of DU m DuNominalRemovalTime[m] is less than the final CPB arrival time of DU m DuFinalArrivalTime[m] for at least one value of m.

4. When DecodingUnitHrdFlag is equal to 1, low_delay_hrd_flag[Htid] is equal to 1 and the nominal removal time of a DU m of AU n is less than the final CPB arrival time of DU m (i.e., DuNominalRemovalTime[m] < DuFinalArrivalTime[m]), the nominal removal time of AU n shall be less than the final CPB arrival time of AU n (i.e., AuNominalRemovalTime[n] < AuFinalArrivalTime[n]).

5. The nominal removal times of AUs from the CPB (starting from the second AU in decoding order) shall satisfy the constraints on AuNominalRemovalTime[n] and AuCpbRemovalTime[n] expressed in clauses A.4.1 and A.4.2.

6. For each current picture, after invocation of the process for removal of pictures from the DPB as specified in clause C.3.2, the number of decoded pictures in the DPB, i.e., the number of all pictures n that are marked as "used for reference", or have PictureOutputFlag equal to 1 and DpbOutputTime[n] greater than AuCpbRemovalTime[currPic], where currPic is the current picture, shall be less than or equal to dpb_max_dec_pic_buffering_minus1[Htid].

7. All reference pictures shall be present in the DPB when needed for prediction. Each picture that has PictureOutputFlag equal to 1 shall be present in the DPB at its DPB output time unless it is removed from the DPB before its output time by one of the processes specified in clause C.3.

8. For each current picture that is not a CLVSS picture, the value of maxPicOrderCnt − minPicOrderCnt shall be less than MaxPicOrderCntLsb / 2.

9. The value of DpbOutputInterval[n] as given by Equation 1605, which is the difference between the output times of a picture and the first picture following it in output order and having PictureOutputFlag equal to 1, shall satisfy the constraint expressed in clause A.4.1 for the profile, tier and level specified in the bitstream using the decoding process specified in clauses 2 through 9.

10. For each current picture, when bp_du_cpb_params_in_pic_timing_sei_flag is equal to 1, let tmpCpbRemovalDelaySum be derived as follows:

```
tmpCpbRemovalDelaySum = 0
for( i = 0; i < pt_num_decoding_units_minus1; i++ )                                  (1609)
        tmpCpbRemovalDelaySum  +=  pt_du_cpb_removal_delay_increment_minus1[ i ][ Htid ] + 1
```

The value of ClockSubTick * tmpCpbRemovalDelaySum shall be equal to the difference between the nominal CPB removal time of the current AU and the nominal CPB removal time of the first DU in the current AU in decoding order.

11. For any two pictures m and n in the same CVS, when DpbOutputTime[m] is greater than DpbOutputTime[n], the PicOrderCntVal of picture m shall be greater than the PicOrderCntVal of picture n.

NOTE 2 – All pictures of an earlier CVS in decoding order that are output are output before any pictures of a later CVS in decoding order. Within any particular CVS, the pictures that are output are output in increasing PicOrderCntVal order.

12. The DPB output times derived for all pictures in any particular AU shall be the same.

C.5 Decoder conformance

C.5.1 General

A decoder conforming to this Specification shall fulfil all requirements specified in this clause.

A decoder claiming conformance to a specific profile, tier and level shall be able to successfully decode all bitstreams that conform to the bitstream conformance requirements specified in clause C.4, in the manner specified in Annex A, provided that all DCI NAL units, when available, all VPSs, SPSs, PPSs and APSs referred to in the VCL NAL units, and appropriate BP, PT, and DUI 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 in this Specification.

When a bitstream contains syntax elements that have values that are specified as reserved and it is specified that decoders shall ignore values of the syntax elements or NAL units containing the syntax elements having the reserved values, and the bitstream is otherwise conforming to this Specification, a conforming decoder shall decode the bitstream in the same manner as it would decode a conforming bitstream and shall ignore the syntax elements or the NAL units containing the syntax elements having the reserved values as specified.

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, tier and level, as specified in clause C.4 are delivered by a hypothetical stream scheduler (HSS) both to the HRD and to the decoder under test (DUT). All cropped decoded pictures output by the HRD shall also be output by the DUT, each cropped decoded picture output by the DUT shall be a picture with PictureOutputFlag equal to 1, and, for each such cropped decoded picture output by the DUT, the values of all samples that are output shall be equal to the values of the samples produced by the specified decoding process.

For output timing decoder conformance, the HSS operates as described in clause C.2, with delivery schedules selected only from the subset of values of ScIdx for which the bit rate and CPB size are restricted as specified in Annex A for the specified profile, tier and level or with "interpolated" delivery schedules as specified by Equations 1610, 1611, 1612, and 1613 for which the bit rate and CPB size are restricted as specified in Annex A. The same delivery schedule is used for both the HRD and the DUT.

When the HRD parameters and the BP SEI messages are present with hrd_cpb_cnt_minus1 and bp_cpb_cnt_minus1, respectively, 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), initial CPB removal delay (f(r) ÷ r), and initial CPB removal delay offset (o(r) ÷ r) as follows:

$$\alpha = (r - \text{BitRate}[\ \text{Htid}\ ][\ \text{ScIdx} - 1\ ]\ ) \div (\text{BitRate}[\ \text{Htid}\ ][\ \text{ScIdx}\ ] - \text{BitRate}[\ \text{Htid}\ ][\ \text{ScIdx} - 1\ ]\ ) \quad (1610)$$

$$c(r) = \alpha * \text{CpbSize}[\ \text{Htid}\ ][\ \text{ScIdx}\ ] + (1 - \alpha) * \text{CpbSize}[\ \text{Htid}\ ][\ \text{ScIdx} - 1\ ] \quad (1611)$$

$$f(r) = \alpha * \text{InitCpbRemovalDelay}[\ \text{ScIdx}\ ] * \text{BitRate}[\ \text{Htid}\ ][\ \text{ScIdx}\ ] + (1 - \alpha) * \text{InitCpbRemovalDelay}[\ \text{ScIdx} - 1\ ] * \text{BitRate}[\ \text{Htid}\ ][\ \text{ScIdx} - 1\ ] \quad (1612)$$

$$o(r) = \alpha * \text{InitCpbRemovalDelayOffset}\ [\ \text{ScIdx}\ ] * \text{BitRate}[\ \text{Htid}\ ][\ \text{ScIdx}\ ] + (1 - \alpha) * \text{InitCpbRemovalDelayOffset}\ [\ \text{ScIdx} - 1\ ] * \text{BitRate}[\ \text{Htid}\ ][\ \text{ScIdx} - 1\ ] \quad (1613)$$

for any ScIdx > 0 and r such that BitRate[Htid][ScIdx − 1] <= r <= BitRate[Htid][ScIdx] such that r and c(r) are within the limits as specified in Annex A for the maximum bit rate and buffer size for the specified profile, tier and level.

NOTE 1 – InitCpbRemovalDelay[ScIdx] could be different from one BP to another and have to be re-calculated.

For output timing decoder conformance, an HRD as specified in this annex is used and the timing (relative to the delivery time of the first bit) of picture output is the same for both the HRD and the DUT up to a fixed delay.

For output order decoder conformance, the following applies:

– The HSS delivers the bitstream BitstreamToDecode 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 CPB of the DUT could be as small as the size of the largest DU.

– A modified HRD as described in the next paragraph below is used, and the HSS delivers the bitstream to the HRD by one of the schedules specified in the bitstream BitstreamToDecode such that the bit rate and CPB size are restricted as specified in Annex A. The order of pictures output shall be the same for both the HRD and the DUT.

– The HRD CPB size is given by CpbSize[Htid][ScIdx] as specified in clause 7.4.6.3, where ScIdx and the HRD parameters are selected as specified in clause C.1. The DPB size is given by dpb_max_dec_pic_buffering_minus1[Htid] + 1. Removal time from the CPB for the HRD is the final bit arrival time and decoding is immediate. The operation of the DPB of this HRD is as described in clauses C.5.2 through C.5.2.3.

C.5.2 Operation of the output order DPB

C.5.2.1 General

The specifications in this clause apply independently to each set of DPB parameters selected as specified in clause C.1.

The DPB contains picture storage buffers for storage of decoded pictures. Each of the picture storage buffers contains a decoded picture that is marked as "used for reference" or is held for future output.

The process for output and removal of pictures from the DPB before decoding of the current picture as specified in clause C.5.2.2 is invoked, followed by the invocation of the process for current decoded picture marking and storage as specified in clause C.3.4, and finally followed by the invocation of the process for additional bumping as specified in clause C.5.2.3. The "bumping" process is specified in clause C.5.2.4 and is invoked as specified in clauses C.5.2.2 and C.5.2.3.

These processes are applied for each coded picture in the OLS.

NOTE – In the operation of output order DPB, same as in the operation of output timing DPB, decoded pictures with PictureOutputFlag equal to 1 in the same AU are also output consecutively in ascending order of the nuh_layer_id values of the decoded pictures.

Let the current picture be the coded picture or decoded picture of the AU n for a particular value of nuh_layer_id, wherein n is a non-negative integer number.

C.5.2.2 Output and removal of pictures from the DPB

The output and removal of pictures from the DPB before the decoding of the current picture (but after parsing the slice header of the first slice of the current picture) happens instantaneously when the first DU of the AU containing the current picture is removed from the CPB and proceeds as follows:

– The decoding process for RPL construction as specified in clause 8.3.2 and decoding process for reference picture marking as specified in clause 8.3.3 are invoked.

– If the current picture is the first picture of a CVSS AU that is not AU 0, the following ordered steps are applied:

1. The variable NoOutputOfPriorPicsFlag is derived for the decoder under test as follows:

   – If the value of PicWidthMaxInSamplesY, PicHeightMaxInSamplesY, MaxChromaFormat, MaxBitDepthMinus8, or dpb_max_dec_pic_buffering_minus1[Htid] derived for the current AU is different from the value of PicWidthMaxInSamplesY, PicHeightMaxInSamplesY, MaxChromaFormat, MaxBitDepthMinus8, or dpb_max_dec_pic_buffering_minus1[Htid], respectively, derived for the preceding AU in decoding order, NoOutputOfPriorPicsFlag may (but should not) be set equal to 1 by the decoder under test, regardless of the value of sh_no_output_of_prior_pics_flag of the current AU.

     NOTE – Although setting NoOutputOfPriorPicsFlag equal to sh_no_output_of_prior_pics_flag of the current AU is preferred under these conditions, the decoder under test is allowed to set NoOutputOfPriorPicsFlag equal to 1 in this case.

   – Otherwise, NoOutputOfPriorPicsFlag is set equal to sh_no_output_of_prior_pics_flag of the current AU.

2. The value of NoOutputOfPriorPicsFlag derived for the decoder under test is applied for the HRD as follows:

   – If NoOutputOfPriorPicsFlag is equal to 1, all picture storage buffers in the DPB are emptied without output of the pictures they contain and the DPB fullness is set equal to 0.

   – Otherwise (NoOutputOfPriorPicsFlag is equal to 0), all picture storage buffers containing a picture that is marked as "not needed for output" and "unused for reference" are emptied (without output) and all non-empty picture storage buffers in the DPB are emptied by repeatedly invoking the "bumping" process specified in clause C.5.2.4 and the DPB fullness is set equal to 0.

– Otherwise, when the current picture is not the first picture of AU 0, all picture storage buffers containing a picture which are marked as "not needed for output" and "unused for reference" are emptied (without output). For each picture storage buffer that is emptied, the DPB fullness is decremented by one. When the number of pictures in the DPB is greater than or equal to dpb_max_dec_pic_buffering_minus1[Htid] + 1, the "bumping" process specified in clause C.5.2.4 is invoked repeatedly until the number of pictures in the DPB is less than dpb_max_dec_pic_buffering_minus1[Htid] + 1.

C.5.2.3 Additional bumping

The processes specified in this clause happen instantaneously when the last DU of the current picture is removed from the CPB.

When the current picture has PictureOutputFlag equal to 1, for each picture in the DPB that is marked as "needed for output" and follows the current picture in output order, the associated variable PicLatencyCount is set equal to PicLatencyCount + 1.

The following applies:

– If the current decoded picture has PictureOutputFlag equal to 1, it is marked as "needed for output" and its associated variable PicLatencyCount is set equal to 0.

– Otherwise (the current decoded picture has PictureOutputFlag equal to 0), it is marked as "not needed for output".

When one or more of the following conditions are true, the "bumping" process specified in clause C.5.2.4 is invoked repeatedly until none of the following conditions are true:

– The number of pictures in the DPB that are marked as "needed for output" is greater than dpb_max_num_reorder_pics[Htid].

– dpb_max_latency_increase_plus1[Htid] is not equal to 0 and there is at least one picture in the DPB that is marked as "needed for output" for which the associated variable PicLatencyCount that is greater than or equal to MaxLatencyPictures[Htid].

C.5.2.4 "Bumping" process

The "bumping" process consists of the following ordered steps:

1. The picture or pictures that are first for output are selected as the one having the smallest value of PicOrderCntVal of all pictures in the DPB marked as "needed for output".
2. Each of these pictures, in ascending nuh_layer_id order, is cropped, using the conformance cropping window for the picture, the cropped picture is output, and the picture is marked as "not needed for output".
3. Each picture storage buffer that contains a picture marked as "unused for reference" and was one of the pictures cropped and output is emptied and the fullness of the DPB is decremented by one.

NOTE – For any two pictures picA and picB that belong to the same CVS and are output by the "bumping process", when picA is output earlier than picB, one of the following conditions applies:

– The value of PicOrderCntVal of picA and the value of PicOrderCntVal of picB are the same and the nuh_layer_id of picA is less than the nuh_layer_id_of picB.

– The value of PicOrderCntVal of picA is less than the value of PicOrderCntVal of picB.

C.6 General sub-bitstream extraction process

Inputs to this process are a bitstream inBitstream, a target OLS index targetOlsIdx, and a target highest TemporalId value tIdTarget.

Output of this process is a sub-bitstream outBitstream.

The OLS with OLS index targetOlsIdx is referred to as the target OLS.

It is a requirement of bitstream conformance for the input bitstream that any output sub-bitstream that satisfies all of the following conditions shall be a conforming bitstream:

– The output sub-bitstream is the output of the process specified in this clause with the bitstream, targetOlsIdx equal to an index to the list of OLSs specified by the VPS, and tIdTarget equal to any value in the range of 0 to vps_ptl_max_tid[vps_ols_ptl_idx[targetOlsIdx]], inclusive, as inputs.

– The output sub-bitstream contains at least one VCL NAL unit with nuh_layer_id equal to each of the nuh_layer_id values in LayerIdInOls[targetOlsIdx].

– The output sub-bitstream contains at least one VCL NAL unit with TemporalId equal to tIdTarget.

NOTE – A conforming bitstream contains one or more coded slice NAL units with TemporalId equal to 0, but does not have to contain coded slice NAL units with nuh_layer_id equal to 0.

The output sub-bitstream OutBitstream is derived by applying the following ordered steps:

1. The bitstream outBitstream is set to be identical to the bitstream inBitstream.
2. Remove from outBitstream all NAL units with TemporalId greater than tIdTarget.
3. Remove from outBitstream all NAL units that have nuh_layer_id not included in the list LayerIdInOls[targetOlsIdx], and are not DCI, OPI, VPS, AUD, or EOB NAL units, and are not SEI NAL units containing non-scalable-nested SEI messages with payload PayloadType equal to 0, 1, 130, or 203.
4. Remove from outBitstream all APS and VCL NAL units for which all of the following conditions are true, and the associated non-VCL NAL units of these VCL NAL units with nal_unit_type equal to PH_NUT or FD_NUT, or with nal_unit_type equal to SUFFIX_SEI_NUT or PREFIX_SEI_NUT and containing SEI messages with PayloadType not equal to any of 0 (BP), 1 (PT), 130 (DUI), and 203 (SLI):

   – nal_unit_type is equal to PREFIX_APS_NUT, SUFFIX_APS_NUT, TRAIL_NUT, STSA_NUT, RADL_NUT, or RASL_NUT, or nal_unit_type is equal to GDR_NUT and the associated ph_recovery_poc_cnt is greater than 0.

   – TemporalId is greater than or equal to NumSubLayersInLayerInOLS[targetOlsIdx][GeneralLayerIdx[nuh_layer_id]].

5. When all VCL NAL units of an AU are removed by steps 2, 3, or 4 above and an AUD or OPI NAL unit is present in the AU, remove the AUD or OPI NAL unit from outBitstream.

6. For each OPI NAL unit in outBitstream, set opi_htid_info_present_flag equal to 1, set opi_ols_info_present_flag equal to 1, set opi_htid_plus1 equal to tIdTarget + 1, and set opi_ols_idx equal to targetOlsIdx.
7. When an AUD NAL unit is present in an AU in outBitstream and the AU becomes an IRAP or GDR AU, set aud_irap_or_gdr_flag of the AUD NAL unit equal to 1.
8. Remove from outBitstream all SEI NAL units that contain a scalable nesting SEI message that has sn_ols_flag equal to 1 and there is no value of i in the range of 0 to sn_num_olss_minus1, inclusive, such that NestingOlsIdx[i] is equal to targetOlsIdx.
9. When LayerIdInOls[targetOlsIdx] does not include all values of nuh_layer_id in all VCL NAL units in the bitstream inBitstream, the following applies in the order listed:
   a. Remove from outBitstream all SEI NAL units that contain a non-scalable-nested SEI message with payloadType equal to 0 (BP), 130 (DUI), or 203 (SLI).
   b. When general_same_pic_timing_in_all_ols_flag is equal to 0, remove from outBitstream all SEI NAL units that contain a non-scalable-nested SEI message with payloadType equal to 1 (PT).
   c. When outBitstream contains an SEI NAL unit seiNalUnitA that contains a scalable nesting SEI message with sn_ols_flag equal to 1 and sn_subpic_flag equal to 0 that applies to the target OLS, or when NumLayersInOls[targetOlsIdx] is equal to 1 and outBitstream contains an SEI NAL unit seiNalUnitA that contains a scalable nesting SEI message with sn_ols_flag equal to 0 and sn_subpic_flag equal to 0 that applies to the layer in outBitstream, generate a new SEI NAL unit seiNalUnitB, include it in the PU containing seiNalUnitA immediately after seiNalUnitA, extract the scalable-nested SEI messages from the scalable nesting SEI message and include them directly in seiNalUnitB (as non-scalable-nested SEI messages), and remove seiNalUnitA from outBitstream.

C.7 Subpicture sub-bitstream extraction process

Inputs to this process are a bitstream inBitstream, a target OLS index targetOlsIdx, a target highest TemporalId value tIdTarget, and a list of target subpicture index values subpicIdxTarget[i] for i from 0 to NumLayersInOls[targetOlsIdx] − 1, inclusive.

Output of this process is a sub-bitstream outBitstream.

The OLS with OLS index targetOlsIdx is referred to as the target OLS. Among the layers in the target OLS, those for which the referenced SPSs have sps_num_subpics_minus1 greater than 0 are referred to as the multiSubpicLayers.

It is a requirement of bitstream conformance for the input bitstream that any output sub-bitstream that satisfies all of the following conditions shall be a conforming bitstream:

– The output sub-bitstream is the output of the process specified in this clause with the bitstream, targetOlsIdx equal to an index to the list of OLSs specified by the VPS, tIdTarget equal to any value in the range of 0 to vps_max_sublayers_minus1, inclusive, and the list subpicIdxTarget[i] for i from 0 to NumLayersInOls[targetOlsIdx] − 1, inclusive, satisfying the following conditions, as inputs:
  – The value of subpicIdxTarget[i] is equal to a value in the range of 0 to sps_num_subpics_minus1, inclusive, such that sps_subpic_treated_as_pic_flag[subpicIdxTarget[i]] is equal to 1, where sps_num_subpics_minus1 and sps_subpic_treated_as_pic_flag[subpicIdxTarget[i]] are found in or inferred based on the SPS referred to by the layer with nuh_layer_id equal to LayerIdInOls[targetOlsIdx][i].

    NOTE 1 – When the sps_num_subpics_minus1 for the layer with nuh_layer_id equal to LayerIdInOls[targetOlsIdx][i] is equal to 0, the value of subpicIdxTarget[i] is equal to 0.
  – For any two different integer values of m and n, when sps_num_subpics_minus1 is greater than 0 for both layers with nuh_layer_id equal to LayerIdInOls[targetOlsIdx][m] and LayerIdInOls[targetOlsIdx][n], respectively, subpicIdxTarget[m] is equal to subpicIdxTarget[n].
– The output sub-bitstream contains at least one VCL NAL unit with nuh_layer_id equal to each of the nuh_layer_id values in the list LayerIdInOls[targetOlsIdx].
– The output sub-bitstream contains at least one VCL NAL unit with TemporalId equal to tIdTarget.

  NOTE 2 – A conforming bitstream contains one or more coded slice NAL units with TemporalId equal to 0, but does not have to contain coded slice NAL units with nuh_layer_id equal to 0.
– The output sub-bitstream contains at least one VCL NAL unit with nuh_layer_id equal to LayerIdInOls[targetOlsIdx][i] and with sh_subpic_id equal to SubpicIdVal[subpicIdxTarget[i]] for each i in the range of 0 to NumLayersInOls[targetOlsIdx] − 1, inclusive.

The output sub-bitstream outBitstream is derived by the following order steps:

1. The sub-bitstream extraction process, specified in clause C.6, is invoked with inBitstream, targetOlsIdx, and tIdTarget as inputs and the output of the process is assigned to outBitstream.
2. For each value of i in the range of 0 to NumLayersInOls[targetOlsIdx] − 1, inclusive, remove from outBitstream all VCL NAL units with nuh_layer_id equal to LayerIdInOls[targetOlsIdx][i] and sh_subpic_id not equal to SubpicIdVal[subpicIdxTarget[i]], their associated filler data NAL units, and their associated SEI NAL units that contain filler payload SEI messages.
3. When an SLI SEI message that applies to the target OLS is present and sli_cbr_constraint_flag of the SLI SEI message is equal to 0, remove all NAL units with nal_unit_type equal to FD_NUT and SEI NAL units containing filler payload SEI messages.
4. Remove from outBitstream all SEI NAL units that contain scalable nesting SEI messages with sn_subpic_flag equal to 1 and none of the sn_subpic_id[j] values for j from 0 to sn_num_subpics_minus1, inclusive, is equal to any of the SubpicIdVal[subpicIdxTarget[i]] values for the layers in the multiSubpicLayers.
5. If some external means not specified in this Specification is available to provide replacement parameter sets for the sub-bitstream outBitstream, replace all parameter sets with the replacement parameter sets. Otherwise, the following ordered steps apply:
   a. The variable spIdx is set equal to the value of subpicIdxTarget[i] for any layer in the multiSubpicLayers.
   b. When an SLI SEI message that applies to the target OLS is present, set the values of general_level_idc and sublayer_level_idc[k] for k in the range of 0 to tIdTarget − 1, inclusive, when present, in the vps_ols_ptl_idx[targetOlsIdx]-th entry in the list of profile_tier_level( ) syntax structures in all the referenced VPSs, and, when NumLayersInOls[targetOlsIdx] is equal to 1, in the profile_tier_level( ) syntax structure in all the referenced SPSs, to be equal to SubpicLevelIdc[spIdx][tIdTarget] and SubpicLevelIdc[spIdx][k], respectively, derived by Equation 1635 for the spIdx-th subpicture sequence.
   c. When an SLI SEI message that applies to the target OLS is present, for k in the range of 0 to tIdTarget, inclusive, let spLvIdx be set equal to SubpicLevelIdx[spIdx][k], where SubpicLevelIdx[spIdx][k] is derived by Equation 1635 for the spIdx-th subpicture sequence. When VCL HRD parameters or NAL HRD parameters are present, for k in the range of 0 to tIdTarget, inclusive, set the respective values of cpb_size_value_minus1[k][j] and bit_rate_value_minus1[k][j] of the j-th CPB, when present, in the vps_ols_timing_hrd_idx[MultiLayerOlsIdx[targetOlsIdx]]-th ols_timing_hrd_parameters( ) syntax structure in all the referenced VPSs and, when NumLayersInOls[targetOlsIdx] is equal to 1, in the ols_timing_hrd_parameters( ) syntax structures in all the referenced SPSs, such that they correspond to SubpicCpbSizeVcl[spLvIdx][spIdx][k], and SubpicCpbSizeNal[spLvIdx][spIdx][k] as derived by Equations 1631 and 1632, respectively, SubpicBitrateVcl[spLvIdx][spIdx][k] and SubpicBitrateNal[spLvIdx][spIdx][k] as derived by Equations 1633 and 1634, respectively, where j is in the range of 0 to hrd_cpb_cnt_minus1, inclusive, and i is in the range of 0 to NumLayersInOls[targetOlsIdx] − 1, inclusive.
   d. For each layer in the multiSubpicLayers, the following ordered steps apply for rewriting of the SPSs and PPSs referenced by pictures in that layer:
      i. The variables subpicWidthInLumaSamples and subpicHeightInLumaSamples are derived as follows:

         subpicWidthInLumaSamples = Min((sps_subpic_ctu_top_left_x[spIdx] + (1614)
            sps_subpic_width_minus1[spIdx] + 1) * CtbSizeY, pps_pic_width_in_luma_samples) −
            sps_subpic_ctu_top_left_x[spIdx] * CtbSizeY

         subpicHeightInLumaSamples = Min((sps_subpic_ctu_top_left_y[spIdx] + (1615)
            sps_subpic_height_minus1[spIdx] + 1) * CtbSizeY, pps_pic_height_in_luma_samples) −
            sps_subpic_ctu_top_left_y[spIdx] * CtbSizeY

      ii. Set the values of the sps_pic_width_max_in_luma_samples and sps_pic_height_max_in_luma_samples in all the referenced SPSs and the values of pps_pic_width_in_luma_samples and pps_pic_height_in_luma_samples in all the referenced PPSs to be equal to subpicWidthInLumaSamples and subpicHeightInLumaSamples, respectively.
      iii. Set the value of sps_num_subpics_minus1 in all the referenced SPSs and pps_num_subpics_minus1 in all the referenced PPSs to be equal to 0.
      iv. Set the values of the syntax elements sps_subpic_ctu_top_left_x[spIdx] and sps_subpic_ctu_top_left_y[spIdx], when present, in all the referenced SPSs to be equal to 0.
      v. Remove the syntax elements sps_subpic_ctu_top_left_x[j], sps_subpic_ctu_top_left_y[j], sps_subpic_width_minus1[j], sps_subpic_height_minus1[j], sps_subpic_treated_as_pic_flag[j],

sps_loop_filter_across_subpic_enabled_flag[j], and sps_subpic_id[j], when present, in all the referenced SPSs for each j not equal to spIdx.

vi. When spIdx is greater than 0 and sps_subpic_id_mapping_explicitly_signalled_flag of a referenced SPS is equal to 0, set the values of sps_subpic_id_mapping_explicitly_signalled_flag and sps_subpic_id_mapping_present_flag both equal to 1, add sps_subpic_id[0] equal to spIdx into the SPS.

vii. Remove the syntax elements pps_subpic_id[j], when present, in all the referenced PPSs for each j that is not equal to spIdx.

viii. Set the syntax elements in all the referenced PPSs for signalling of tiles and slices to remove all tile rows, tile columns, and slices that are not associated with the subpicture with subpicture index equal to spIdx.

ix. The variables subpicConfWinLeftOffset, subpicConfWinRightOffset, subpicConfWinTopOffset and subpicConfWinBottomOffset are derived as follows:

```
subpicConfWinLeftOffset = sps_subpic_ctu_top_left_x[ spIdx ]  = =  0 ?                          (1616)
        sps_conf_win_left_offset : 0

subpicConfWinRightOffset = ( sps_subpic_ctu_top_left_x[ spIdx ] +                               (1617)
        sps_subpic_width_minus1[ spIdx ] + 1 ) *CtbSizeY  >=
        sps_pic_width_max_in_luma_samples ? sps_conf_win_right_offset : 0

subpicConfWinTopOffset = sps_subpic_ctu_top_left_y[ spIdx ]  = =  0 ?                           (1618)
        sps_conf_win_top_offset : 0

subpicConfWinBottomOffset = ( sps_subpic_ctu_top_left_y[ spIdx ] +                              (1619)
        sps_subpic_height_minus1[ spIdx ] + 1 ) * CtbSizeY  >=
        sps_pic_height_max_in_luma_samples ? sps_conf_win_bottom_offset : 0
```

Where the values of sps_subpic_ctu_top_left_x[spIdx], sps_subpic_width_minus1[spIdx], sps_subpic_ctu_top_left_y[spIdx], sps_subpic_height_minus1[spIdx], sps_pic_width_max_in_luma_samples, sps_pic_height_max_in_luma_samples, sps_conf_win_left_offset, sps_conf_win_right_offset, sps_conf_win_top_offset, and sps_conf_win_bottom_offset in these equations are the values from the original SPSs before they were rewritten.

NOTE 3 – For pictures in the layers in the multiSubpicLayers in both the input bitstream and the output bitstream, the values of sps_pic_width_max_in_luma_samples and sps_pic_height_max_in_luma_samples are equal to pps_pic_width_in_luma_samples and pps_pic_height_in_luma_samples, respectively. Thus in these equations, sps_pic_width_max_in_luma_samples and sps_pic_height_max_in_luma_samples could be replaced with pps_pic_width_in_luma_samples and pps_pic_height_in_luma_samples, respectively.

x. Set the values of sps_conf_win_left_offset, sps_conf_win_right_offset, sps_conf_win_top_offset, and sps_conf_win_bottom_offset in all the referenced SPSs to be equal to subpicConfWinLeftOffset, subpicConfWinRightOffset, subpicConfWinTopOffset, and subpicConfWinBottomOffset, respectively.

xi. The variables subpicScalWinLeftOffset, subpicScalWinRightOffset, subpicScalWinTopOffset and subpicScalWinBotOffset are derived as follows:

```
subpicScalWinLeftOffset = pps_scaling_win_left_offset −                                         (1620)
        sps_subpic_ctu_top_left_x[ spIdx ] * CtbSizeY / SubWidthC

rightSubpicBd = ( sps_subpic_ctu_top_left_x[ spIdx ] +
        sps_subpic_width_minus1[ spIdx ] + 1 ) * CtbSizeY
subpicScalWinRightOffset = ( rightSubpicBd  >=  sps_pic_width_max_in_luma_samples ) ?
        pps_scaling_win_right_offset : pps_scaling_win_right_offset −                          (1621)
        ( sps_pic_width_max_in_luma_samples − rightSubpicBd ) / SubWidthC

subpicScalWinTopOffset = pps_scaling_win_top_offset −                                           (1622)
        sps_subpic_ctu_top_left_y[ spIdx ] * CtbSizeY / SubHeightC

botSubpicBd = ( sps_subpic_ctu_top_left_y[ spIdx ] +
        sps_subpic_height_minus1[ spIdx ] + 1 ) * CtbSizeY
subpicScalWinBotOffset =  ( botSubpicBd  >=  sps_pic_height_max_in_luma_samples ) ?             (1623)
        pps_scaling_win_bottom_offset : pps_scaling_win_bottom_offset −
        ( sps_pic_height_max_in_luma_samples − botSubpicBd ) / SubHeightC
```

Where the values of sps_subpic_ctu_top_left_x[spIdx], sps_subpic_width_minus1[spIdx], sps_subpic_ctu_top_left_y[spIdx], sps_subpic_height_minus1[spIdx], sps_pic_width_max_in_luma_samples, and sps_pic_height_max_in_luma_samples in these equations are from the original SPSs before they were rewitten, and pps_scaling_win_left_offset, pps_scaling_win_right_offset, pps_scaling_win_top_offset, and pps_scaling_win_bottom_offset in these equations are the values from the original PPSs before they were rewitten.

xii. Set the values of pps_scaling_win_left_offset, pps_scaling_win_right_offset, pps_scaling_win_top_offset, and pps_scaling_win_bottom_offset in all the referenced PPS NAL units to be equal to subpicScalWinLeftOffset, subpicScalWinRightOffset, subpicScalWinTopOffset, and subpicScalWinBotOffset, respectively.

xiii. The variables numVerVbs, subpicVbx[i], numHorVbs, and subpicVby[i] are derived as follows:

```
numVerVbs = 0;
subpicX = sps_subpic_ctu_top_left_x[ spIdx ]
for( i = 0; i < sps_num_ver_virtual_boundaries; i++) {
    vbX = sps_virtual_boundary_pos_x_minus1[ i ] + 1
    if( vbX > ( subpicX * CtbSizeY / 8 )  &&  vbX < Min( ( subpicX +                    (1624)
        sps_subpic_width_minus1[ spIdx ] + 1 ) * CtbSizeY / 8,
            pps_pic_width_in_luma_samples / 8 ) )
        subpicVbx[ numVerVbs++ ] = vbX − subpicX * CtbSizeY / 8
}

numHorVbs = 0;
subpicY = sps_subpic_ctu_top_left_y[ spIdx ]
for( i = 0; i < sps_num_hor_virtual_boundaries; i++) {
    vbY = sps_virtual_boundary_pos_y_minus1[ i ] + 1
    if( vbY > ( subpicY * CtbSizeY / 8 )  &&  vbY < Min( ( subpicY +                    (1625)
        sps_subpic_height_minus1[ spIdx ] + 1 ) * CtbSizeY / 8,
        pps_pic_height_in_luma_samples / 8 ) )
        subpicVby[ numHorVbs++ ] = vbY − subpicY * CtbSizeY / 8
}
```

Where the values of sps_num_ver_virtual_boundaries, sps_virtual_boundary_pos_x_minus1[i], sps_subpic_ctu_top_left_x[spIdx], sps_subpic_width_minus1[spIdx], sps_num_hor_virtual_boundaries, sps_virtual_boundary_pos_y_minus1[i], sps_subpic_ctu_top_left_y[spIdx], and sps_subpic_height_minus1[spIdx] in these equations are the values from the original SPSs before they were rewritten.

xiv. When sps_virtual_boundaries_present_flag is equal to 1, set the values of the sps_num_ver_virtual_boundaries, sps_virtual_boundary_pos_x_minus1[i], sps_num_hor_virtual_boundaries, and sps_virtual_boundary_pos_y_minus1[j] syntax elements in all the reference SPSs to be equal to numVerVbs, subpicVbx[i] − 1, numHorVbs, and subpicVby[j] − 1, respectively, for i in the range of 0 to numVerVbs − 1, inclusive, and j in the range of 0 to numHorVbs − 1, inclusive. The virtual boundaries outside the extracted subpicture are removed. When both numVerVbs and numHorVbs are equal to 0, set the value of sps_virtual_boundaries_enabled_flag in all the referenced SPSs to be equal to 0 and remove the syntax elements sps_virtual_boundaries_present_flag, sps_num_ver_virtual_boundaries, sps_virtual_boundary_pos_x_minus1[i], sps_num_hor_virtual_boundaries, and sps_virtual_boundary_pos_y_minus1[i].

e. When an SLI SEI message that applies to the target OLS is present, the following applies:

i. If sli_cbr_constraint_flag is equal to 1, set cbr_flag[tIdTarget][j] equal to 1 of the j-th CPB in the vps_ols_timing_hrd_idx[MultiLayerOlsIdx[targetOlsIdx]]-th ols_timing_hrd_parameters( ) syntax structure in all the referenced VPSs and, when NumLayersInOls[targetOlsIdx] is equal to 1, in the ols_timing_hrd_parameters( ) syntax structure in all the referenced SPSs.

ii. Otherwise, (sli_cbr_constraint_flag is equal to 0), set cbr_flag[tIdTarget][j] equal to 0. In both cases, j is in the range of 0 to hrd_cpb_cnt_minus1, inclusive.

6. When at least one VCL NAL unit has been removed by step 2, remove from outBitstream all SEI NAL units that contain scalable nesting SEI messages with sn_subpic_flag equal to 0.

7. When at least one VCL NAL unit has been removed by step 2, the following applies in the order listed:

a. Remove from outBitstream all SEI NAL units that contain a non-scalable-nested SEI message with payloadType equal to 0 (BP), 130 (DUI), 203 (SLI), or 132 (decoded picture hash).

b. When general_same_pic_timing_in_all_ols_flag is equal to 0, remove from outBitstream all SEI NAL units that contain a non-scalable-nested SEI message with payloadType equal to 1 (PT).

c. When outBitstream contains an SEI NAL unit seiNalUnitA that contains a scalable nesting SEI message with sn_ols_flag equal to 1 and sn_subpic_flag equal to 1 that applies to the target OLS and the subpictures in outBitstream, or when NumLayersInOls[targetOlsIdx] is equal to 1 and outBitstream contains an SEI NAL unit seiNalUnitA that contains a scalable nesting SEI message with sn_ols_flag equal to 0 and sn_subpic_flag equal to 1 that applies to the layer and the subpictures in outBitstream, generate a new SEI NAL unit seiNalUnitB, include it in the PU containing seiNalUnitA immediately after seiNalUnitA, extract the scalable-nested SEI messages from the scalable nesting SEI message and include them directly in seiNalUnitB (as non-scalable-nested SEI messages), and remove seiNalUnitA from outBitstream.

Annex D

Supplemental enhancement information and use of SEI and VUI

(This annex forms an integral part of this Recommendation | International Standard.)

D.1 General

This annex specifies 1) the syntax and semantics for the SEI payload, which is the container of SEI messages, 2) the syntax and semantics for some SEI messages, and 3) the use of the VUI parameters and SEI messages for which the syntax and semantics are specified in Rec. ITU-T H.274 | ISO/IEC 23002-7.

When the VUI parameters or any SEI message specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 is included in a non-VCL NAL unit as specified in this Specification, its syntax elements and semantics shall be as specified in Rec. ITU-T H.274 | ISO/IEC 23002-7.

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 Specification (see Annex C for the specification of conformance). Some SEI messages are required for checking bitstream conformance and for output timing decoder conformance. Other SEI messages are not required for check bitstream conformance.

In clause C.5.2, the 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 in this Specification. When present in the bitstream, these SEI messages shall obey the syntax and semantics specified in clause 7.3.6 and this annex. When the content of such 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.2 General SEI payload

D.2.1 General SEI payload syntax

sei_payload(payloadType, payloadSize) {	**Descriptor**
SeiExtensionBitsPresentFlag = 0	
if(nal_unit_type = = PREFIX_SEI_NUT)	
if(payloadType = = 0)	
buffering_period(payloadSize)	
else if(payloadType = = 1)	
pic_timing(payloadSize)	
else if(payloadType = = 3) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
filler_payload(payloadSize)	
else if(payloadType = = 4) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
user_data_registered_itu_t_t35(payloadSize)	
else if(payloadType = = 5) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
user_data_unregistered(payloadSize)	
else if(payloadType = = 19) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
film_grain_characteristics(payloadSize)	
else if(payloadType = = 45) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
frame_packing_arrangement(payloadSize)	
else if(payloadType = = 47) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
display_orientation(payloadSize)	
else if(payloadType = = 56) /* Specified in ISO/IEC 23001-11 */	
green_metadata(payloadsize)	
else if(payloadType = = 129) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
parameter_sets_inclusion_indication(payloadSize)	
else if(payloadType = = 130)	
decoding_unit_info(payloadSize)	

else if(payloadType = = 133)	
scalable_nesting(payloadSize)	
else if(payloadType = = 137) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
mastering_display_colour_volume(payloadSize)	
else if(payloadType = = 142) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
colour_transform_info(payloadSize)	
else if(payloadType = = 144) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
content_light_level_info(payloadSize)	
else if(payloadType = = 145) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
dependent_rap_indication(payloadSize)	
else if(payloadType = = 147) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
alternative_transfer_characteristics(payloadSize)	
else if(payloadType = = 148) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
ambient_viewing_environment(payloadSize)	
else if(payloadType = = 149) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
content_colour_volume(payloadSize)	
else if(payloadType = = 150) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
equirectangular_projection(payloadSize)	
else if(payloadType = = 153) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
generalized_cubemap_projection(payloadSize)	
else if(payloadType = = 154) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
sphere_rotation(payloadSize)	
else if(payloadType = = 155) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
regionwise_packing(payloadSize)	
else if(payloadType = = 156) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
omni_viewport(payloadSize)	
else if(payloadType = = 165) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
alpha_channel_info(payloadSize)	
else if(payloadType = = 168) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
frame_field_info(payloadSize)	
else if(payloadType = = 177) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
depth_representation_info(payloadSize)	
else if(payloadType = = 179) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
multiview_acquisition_info(payloadSize)	
else if(payloadType = = 180) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
multiview_view_position(payloadSize)	
else if(payloadType = = 200)	
sei_manifest(payloadSize)	
else if(payloadType = = 201)	
sei_prefix_indication(payloadSize)	
else if(payloadType = = 202) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
annotated_regions(payloadSize)	
else if(payloadType = = 203)	
subpic_level_info(payloadSize)	
else if(payloadType = = 204) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
sample_aspect_ratio_info(payloadSize)	
else if(payloadType = = 205) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
shutter_interval_info(payloadSize)	
else if(payloadType = = 206) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	

extended_drap_indication(payloadSize)	
else if(payloadType = = 207)	
constrained_rasl_encoding_indication(payloadSize)	
else if(payloadType = = 208) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
scalability_dimension_info(payloadSize)	
else if(payloadType = = 209) /* Specified in ISO/IEC 23090-13 */	
vdi_sei_envelope(payloadsize)	
else if(payloadType = = 210) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
nn_post_filter_characteristics(payloadSize)	
else if(payloadType = = 211) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
nn_post_filter_activation(payloadSize)	
else if(payloadType = = 212) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
phase_indication(payloadSize)	
else /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
reserved_message(payloadSize)	
else /* nal_unit_type = = SUFFIX_SEI_NUT */	
if(payloadType = = 3) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
filler_payload(payloadSize)	
else if(payloadType = = 4) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
user_data_registered_itu_t_t35(payloadSize)	
else if(payloadType = = 5) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
user_data_unregistered(payloadSize)	
else if(payloadType = = 132) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
decoded_picture_hash(payloadSize)	
else if(payloadType = = 133)	
scalable_nesting(payloadSize)	
else if(payloadType = = 210) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
nn_post_filter_characteristics(payloadSize)	
else if(payloadType = = 211) /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
nn_post_filter_activation(payloadSize)	
else /* Specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 */	
reserved_message(payloadSize)	
if(SeiExtensionBitsPresentFlag || more_data_in_payload( )) {	
if(payload_extension_present( ))	
sei_reserved_payload_extension_data	u(v)
sei_payload_bit_equal_to_one /* equal to 1 */	f(1)
while(!byte_aligned( ))	
sei_payload_bit_equal_to_zero /* equal to 0 */	f(1)
}	
}	

D.2.2 General SEI payload semantics

sei_reserved_payload_extension_data shall not be present in bitstreams conforming to this version of this Specification. However, decoders conforming to this version of this Specification shall ignore the presence and value of sei_reserved_payload_extension_data. When present, the length, in bits, of sei_reserved_payload_extension_data is equal to 8 * payloadSize − nEarlierBits − nPayloadZeroBits − 1, where nEarlierBits is the number of bits in the sei_payload( ) syntax structure that precede the sei_reserved_payload_extension_data syntax element, and nPayloadZeroBits is the number of sei_payload_bit_equal_to_zero syntax elements at the end of the sei_payload( ) syntax structure.

If more_data_in_payload( ) is TRUE after the parsing of the SEI message syntax structure (e.g., the buffering_period( ) syntax structure) and nPayloadZeroBits is not equal to 7, PayloadBits is set equal to 8 * payloadSize − nPayloadZeroBits − 1; otherwise, PayloadBits is set equal to 8 * payloadSize.

payload_bit_equal_to_one shall be equal to 1.

payload_bit_equal_to_zero shall be equal to 0.

NOTE 1 – SEI messages with the same value of payloadType are conceptually the same SEI message regardless of whether they are contained in prefix or suffix SEI NAL units.

NOTE 2 – For SEI messages specified in this Specification and the VSEI specification (ITU-T H.274 | ISO/IEC 23002-7), the payloadType values are aligned with similar SEI messages specified in AVC (Rec. ITU-T H.264 | ISO/IEC 14496-10) and HEVC (Rec. ITU-T H.265 | ISO/IEC 23008-2).

The semantics and persistence scope for each SEI message are specified in the semantics specification for each particular SEI message.

NOTE 3 – Persistence information for SEI messages is informatively summarized in Table D.1.

Table D.1 – Persistence scope of SEI messages (informative)

SEI message	Persistence scope
Buffering period	The remainder of the bitstream
Picture timing	The AU containing the SEI message
DU information	The AU containing the SEI message
Scalable nesting	Depending on the scalable-nested SEI messages. Each scalable-nested SEI message has the same persistence scope as if the SEI message was not scalable-nested
SEI manifest	The CVS containing the SEI message
SEI prefix indication	The CVS containing the SEI message
Subpicture level information	The CVS containing the SLI SEI message and up to but not including the next CVS, in decoding order, that contains an SLI SEI message with different content
Constrained RASL encoding indication	The CVS containing the SEI message

Only filler payload, decoded picture hash, and scalable nesting SEI messages may be included in a suffix SEI NAL unit; all other SEI messages are not allowed to be included in a suffix SEI NAL unit. When there is a scalable nesting SEI message included in a suffix SEI NAL unit, it is only allowed to contain those SEI messages that are allowed to be included in a suffix NAL unit. When there is a scalable nesting SEI message included in a prefix SEI NAL unit, it is only allowed to contain those SEI messages that are allowed to be included in a prefix NAL unit.

The list VclAssociatedSeiList is set to consist of the payloadType values 3, 19, 45, 47, 56, 129, 132, 137, 142, 144, 145, 147 to 150, inclusive, 153 to 156, inclusive, 165, 168, 177, 179, 180, 200 to 202, inclusive, and 204 to 212, inclusive.

The list PicUnitRepConSeiList is set to consist of the payloadType values 0, 1, 19, 45, 47, 56, 129, 132, 133, 137, 142, 144, 145, 147 to 150, inclusive, 153 to 156, inclusive, 165, 168, 177, 179, 180, and 200 to 212, inclusive.

NOTE 4 – VclAssociatedSeiList consists of the payloadType values of the SEI messages that, when non-scalable-nested, infer constraints on the NAL unit header of the SEI NAL unit on the basis of the NAL unit header of the associated VCL NAL unit. PicUnitRepConSeiList consists of the payloadType values of the SEI messages that are subject to the restriction on 4 repetitions per PU.

It is a requirement of bitstream conformance that the following restrictions apply on containing of SEI messages in SEI NAL units:

– When general_same_pic_timing_in_all_ols_flag is equal to 1, there shall be no SEI NAL unit that contains a scalable-nested SEI message with payloadType equal to 1 (PT), and when an SEI NAL unit contains a non-scalable-nested SEI message with payloadType equal to 1 (PT), the SEI NAL unit shall not contain any other SEI message with payloadType not equal 1.

– When an SEI NAL unit contains a non-scalable-nested SEI message with payloadType equal to 0 (BP), 1 (PT), 130 (DUI), or 203 (SLI), the SEI NAL unit shall not contain any other SEI message with payloadType not equal to any of 0, 1, 130, and 203.

– When an SEI NAL unit contains a scalable-nested SEI message with payloadType equal to 0 (BP), 1 (PT), 130 (DUI), or 203 (SLI), the SEI NAL unit shall not contain any other SEI message with payloadType not equal to any of 0, 1, 130, 203, and 133 (scalable nesting).

– When an SEI NAL unit contains an SEI message with payloadType equal to 3 (filler payload), the SEI NAL unit shall not contain any other SEI message with payloadType not equal to 3.

It is a requirement of bitstream conformance that the following restrictions apply on containing of SEI messages in a scalable nesting SEI message:

– An SEI message that has payloadType equal to 3 (filler payload), 133 (scalable nesting), 179 (multiview acquisition information), 180 (multiview view position), or 208 (scalability dimension information) shall not be contained in a scalable nesting SEI message.

– When a scalable nesting SEI message contains a BP, PT, DUI, or SLI SEI message, the scalable nesting SEI message shall not contain any other SEI message with payloadType not equal to any of 0 (BP), 1 (PT), 130 (DUI), and 203 (SLI).

The following applies on the applicable OLSs or layers of non-scalable-nested SEI messages:

– For a non-scalable-nested SEI message, when payloadType is equal to 0 (BP), 1 (PT), 130 (DUI), or 203 (SLI), the non-scalable-nested SEI message applies to all the OLSs, when present, that consist of all layers in the current CVS in the entire bitstream. When there is no OLS that consists of all layers in the current CVS in the entire bitstream, there shall be no non-scalable-nested SEI message with payloadType equal to 0 (BP), 1 (PT), 130 (DUI), or 203 (SLI).

– For a non-scalable-nested SEI message, when payloadType is equal to any value among VclAssociatedSeiList, the non-scalable-nested SEI message applies only to the layer for which the VCL NAL units have nuh_layer_id equal to the nuh_layer_id of the SEI NAL unit containing the SEI message.

It is a requirement of bitstream conformance that the following restrictions apply on the value of nuh_layer_id of SEI NAL units:

– When a non-scalable-nested SEI message has payloadType equal to any value among VclAssociatedSeiList, the SEI NAL unit containing the non-scalable-nested SEI message shall have nuh_layer_id equal to the value of nuh_layer_id of the VCL NAL unit associated with the SEI NAL unit.

– An SEI NAL unit containing a scalable nesting SEI message shall have nuh_layer_id equal to the lowest value of nuh_layer_id of all layers to which the scalable-nested SEI messages apply (when sn_ols_flag of the scalable nesting SEI message is equal to 0) or the lowest value of nuh_layer_id of all layers in the OLSs to which the scalable-nested SEI message apply (when sn_ols_flag of the scalable nesting SEI message is equal to 1).

NOTE 5 – Same as for DCI, OPI, VPS, AUD, and EOB NAL units, the value of nuh_layer_id for SEI NAL units that contain non-scalable-nested SEI messages with payloadType equal to 0 (BP), 1 (PT), or 130 (DUI), or 203 (SLI) is not constrained.

It is a requirement of bitstream conformance that the following restrictions apply on repetition of SEI messages:

– For each of the payloadType values included in PicUnitRepConSeiList, there shall be less than or equal to 4 identical sei_payload( ) syntax structures within a PU.

– There shall be less than or equal to 4 identical sei_payload( ) syntax structures with payloadType equal to 130 within a DU.

The following applies on the content of SLI, BP, PT, and DUI SEI messages that are associated with an AU and apply to the OLSs consisting of the same set of layers and having the same values of NumSubLayersInLayerInOLS[olsIdx][i] for each value of i in the range of 0 to NumLayersInOls[olsIdx], inclusive, where olsIdx is the index of such an OLS:

– An SLI SEI message that is associated with an AU and applies to an OLS with OLS index olsIdxA consisting of a set of layers shall have the same SEI payload content as another SLI SEI message that is associated with the same AU and applies to another OLS with OLS index olsIdxB consisting of the same set of layers when the lists NumSubLayersInLayerInOLS[olsIdxA] and NumSubLayersInLayerInOLS[olsIdxB] are identical.

– A BP SEI message that is associated with an AU and applies to an OLS with OLS index olsIdxA consisting of a set of layers shall have the same SEI payload content as another BP SEI message that associated with in the same AU and applies to another OLS with OLS index olsIdxB consisting of the same set of layers when the lists NumSubLayersInLayerInOLS[olsIdxA] and NumSubLayersInLayerInOLS[olsIdxB] are identical.

– A PT SEI message that is associated with an AU and applies to an OLS with OLS index olsIdxA consisting of a set of layers shall have the same SEI payload content as another PT SEI message that is associated with the same AU and applies to another OLS with OLS index olsIdxB consisting of the same set of layers when the lists NumSubLayersInLayerInOLS[olsIdxA] and NumSubLayersInLayerInOLS[olsIdxB] are identical.

– A DUI SEI message that is associated with an AU for a DU and applies to an OLS with OLS index olsIdxA consisting of a set of layers shall have the same SEI payload content as another DUI SEI message that is associated with the same AU for the same DU and applies to another OLS with OLS index olsIdxB consisting of the same set of layers when the lists NumSubLayersInLayerInOLS[olsIdxA] and NumSubLayersInLayerInOLS[olsIdxB] are identical.

The following applies on the content of scalable-nested and non-scalable-nested SEI messages applying to the same OLS or layer:

– When there are multiple SEI messages with a particular value of payloadType not equal to any of 4, 5, 133, 210, and 211 that are associated with a particular AU or DU and apply to a particular OLS, layer, or subpicture, regardless of whether some or all of these SEI messages are scalable-nested, the SEI messages shall have the same SEI payload content.

– When there are multiple SEI messages with payloadType equal to 211 and the same nnpfa_target_id value that are associated with a particular AU or DU and apply to a particular OLS or layer, regardless of whether some or all of these SEI messages are scalable-nested, the SEI messages shall have the same SEI payload content.

The following applies on the order of SLI, BP, PT, and DUI SEI messages:

– When an SLI SEI message and a BP SEI message that apply to a particular OLS are present within an AU, the SLI SEI messages shall precede the BP SEI message in decoding order.

– When a BP SEI message and a PT SEI message that apply to a particular OLS are present within an AU, the BP SEI messages shall precede the PT SEI message in decoding order.

– When a BP SEI message and a DUI SEI message that apply to a particular OLS are present within an AU, the BP SEI messages shall precede the DUI SEI message in decoding order.

– When a PT SEI message and a DUI SEI message that apply to a particular OLS are present within an AU, the PT SEI messages shall precede the DUI SEI message in decoding order.

D.3 Buffering period SEI message

D.3.1 Buffering period SEI message syntax

buffering_period(payloadSize) {	**Descriptor**
bp_nal_hrd_params_present_flag	u(1)
bp_vcl_hrd_params_present_flag	u(1)
bp_cpb_initial_removal_delay_length_minus1	u(5)
bp_cpb_removal_delay_length_minus1	u(5)
bp_dpb_output_delay_length_minus1	u(5)
bp_du_hrd_params_present_flag	u(1)
if(bp_du_hrd_params_present_flag) {	
bp_du_cpb_removal_delay_increment_length_minus1	u(5)
bp_dpb_output_delay_du_length_minus1	u(5)
bp_du_cpb_params_in_pic_timing_sei_flag	u(1)
bp_du_dpb_params_in_pic_timing_sei_flag	u(1)
}	
bp_concatenation_flag	u(1)
bp_additional_concatenation_info_present_flag	u(1)
if(bp_additional_concatenation_info_present_flag)	
bp_max_initial_removal_delay_for_concatenation	u(v)
bp_cpb_removal_delay_delta_minus1	u(v)
bp_max_sublayers_minus1	u(3)
if(bp_max_sublayers_minus1 > 0)	
bp_cpb_removal_delay_deltas_present_flag	u(1)
if(bp_cpb_removal_delay_deltas_present_flag) {	
bp_num_cpb_removal_delay_deltas_minus1	ue(v)

for(i = 0; i <= bp_num_cpb_removal_delay_deltas_minus1; i++)	
bp_cpb_removal_delay_delta_val[i]	u(v)
}	
bp_cpb_cnt_minus1	ue(v)
if(bp_max_sublayers_minus1 > 0)	
bp_sublayer_initial_cpb_removal_delay_present_flag	u(1)
for(i = (bp_sublayer_initial_cpb_removal_delay_present_flag ? 0 : bp_max_sublayers_minus1); i <= bp_max_sublayers_minus1; i++) {	
if(bp_nal_hrd_params_present_flag)	
for(j = 0; j < bp_cpb_cnt_minus1 + 1; j++) {	
bp_nal_initial_cpb_removal_delay[i][j]	u(v)
bp_nal_initial_cpb_removal_offset[i][j]	u(v)
if(bp_du_hrd_params_present_flag) {	
bp_nal_initial_alt_cpb_removal_delay[i][j]	u(v)
bp_nal_initial_alt_cpb_removal_offset[i][j]	u(v)
}	
}	
if(bp_vcl_hrd_params_present_flag)	
for(j = 0; j < bp_cpb_cnt_minus1 + 1; j++) {	
bp_vcl_initial_cpb_removal_delay[i][j]	u(v)
bp_vcl_initial_cpb_removal_offset[i][j]	u(v)
if(bp_du_hrd_params_present_flag) {	
bp_vcl_initial_alt_cpb_removal_delay[i][j]	u(v)
bp_vcl_initial_alt_cpb_removal_offset[i][j]	u(v)
}	
}	
}	
if(bp_max_sublayers_minus1 > 0)	
bp_sublayer_dpb_output_offsets_present_flag	u(1)
if(bp_sublayer_dpb_output_offsets_present_flag)	
for(i = 0; i < bp_max_sublayers_minus1; i++)	
bp_dpb_output_tid_offset[i]	ue(v)
bp_alt_cpb_params_present_flag	u(1)
if(bp_alt_cpb_params_present_flag)	
bp_use_alt_cpb_params_flag	u(1)
}	

D.3.2 Buffering period SEI message semantics

A BP SEI message provides initial CPB removal delay and initial CPB removal delay offset information for initialization of the HRD at the position of the associated AU in decoding order.

When the BP SEI message is present, an AU is said to be a notDiscardableAu when the AU has TemporalId equal to 0 and has at least one picture that has ph_non_ref_pic_flag equal to 0 that is not a RASL or RADL picture.

When the current AU is not the first AU in the bitstream in decoding order, let the AU prevNonDiscardableAu be the previous AU in decoding order with TemporalId equal to 0 that has at least one picture that has ph_non_ref_pic_flag equal to 0 that is not a RASL or RADL picture.

The presence of BP SEI messages is specified as follows:

– If NalHrdBpPresentFlag is equal to 1 or VclHrdBpPresentFlag is equal to 1, the following applies for each AU in the CVS:

– If the AU is an IRAP or GDR AU, a BP SEI message applicable to the operation point shall be associated with the AU.

– Otherwise, if the AU is a notDiscardableAu, a BP SEI message applicable to the operation point might or might not be associated with the AU.

– Otherwise, the AU shall not be associated with a BP SEI message applicable to the operation point.

– Otherwise (NalHrdBpPresentFlag and VclHrdBpPresentFlag are both equal to 0), no AU in the CVS shall be associated with a BP SEI message.

NOTE 1 – For some applications, frequent presence of BP SEI messages could be desirable (e.g., for random access at an IRAP AU or a non-IRAP AU or for bitstream splicing).

bp_nal_hrd_params_present_flag equal to 1 specifies that a list of syntax element pairs bp_nal_initial_cpb_removal_delay[i][j] and bp_nal_initial_cpb_removal_offset[i][j] are present in the BP SEI message. bp_nal_hrd_params_present_flag equal to 0 specifies that no syntax element pairs bp_nal_initial_cpb_removal_delay[i][j] and bp_nal_initial_cpb_removal_offset[i][j] are present in the BP SEI message.

The value of bp_nal_hrd_params_present_flag shall be equal to general_nal_hrd_params_present_flag.

bp_vcl_hrd_params_present_flag equal to 1 specifies that a list of syntax element pairs bp_vcl_initial_cpb_removal_delay[i][j] and bp_vcl_initial_cpb_removal_offset[i][j] are present in the BP SEI message. bp_vcl_hrd_params_present_flag equal to 0 specifies that no syntax element pairs bp_vcl_initial_cpb_removal_delay[i][j] and bp_vcl_initial_cpb_removal_offset[i][j] are present in the BP SEI message.

The value of bp_vcl_hrd_params_present_flag shall be equal to general_vcl_hrd_params_present_flag.

bp_vcl_hrd_params_present_flag and bp_nal_hrd_params_present_flag in a BP SEI message shall not both be equal to 0.

bp_cpb_initial_removal_delay_length_minus1 plus 1 specifies the length, in bits, of the syntax elements bp_nal_initial_cpb_removal_delay[i][j], bp_nal_initial_cpb_removal_offset[i][j], bp_vcl_initial_cpb_removal_delay[i][j], and bp_vcl_initial_cpb_removal_offset[i][j] of the BP SEI messages, and the syntax elements pt_nal_cpb_alt_initial_removal_delay_delta[i][j], pt_vcl_cpb_alt_initial_removal_delay_delta[i][j], pt_nal_cpb_alt_initial_removal_offset_delta[i][j] and pt_vcl_cpb_alt_initial_removal_offset_delta[i][j] in the PT SEI messages in the current BP. When not present, the value of bp_cpb_initial_removal_delay_length_minus1 is inferred to be equal to 23.

bp_cpb_removal_delay_length_minus1 plus 1 specifies the length, in bits, of the syntax elements bp_cpb_removal_delay_delta_minus1 and bp_cpb_removal_delay_delta_val[i] in the BP SEI message and the syntax elements pt_cpb_removal_delay_minus1[i], pt_nal_cpb_delay_offset[i] and pt_vcl_cpb_delay_offset[i] in the PT SEI messages in the current BP. When not present, the value of bp_cpb_removal_delay_length_minus1 is inferred to be equal to 23.

bp_dpb_output_delay_length_minus1 plus 1 specifies the length, in bits, of the syntax elements pt_dpb_output_delay, pt_nal_dpb_delay_offset[i] and pt_vcl_dpb_delay_offset[i] in the PT SEI messages in the current BP. When not present, the value of bp_dpb_output_delay_length_minus1 is inferred to be equal to 23.

bp_du_hrd_params_present_flag equal to 1 specifies that DU level HRD parameters are present and the HRD can be operated at the AU level or DU level. bp_du_hrd_params_present_flag equal to 0 specifies that DU level HRD parameters are not present and the HRD operates at the AU level. When bp_du_hrd_params_present_flag is not present, its value is inferred to be equal to 0.

The value of bp_du_hrd_params_present_flag shall be equal to general_du_hrd_params_present_flag.

When bp_alt_cpb_params_present_flag is equal to 1, the value of bp_du_hrd_params_present_flag shall be equal to 0.

bp_du_cpb_removal_delay_increment_length_minus1 plus 1 specifies the length, in bits, of the pt_du_cpb_removal_delay_increment_minus1[ ][ ] and pt_du_common_cpb_removal_delay_increment_minus1[ ] syntax elements of the PT SEI messages in the current BP and the dui_du_cpb_removal_delay_increment[ ] syntax element in the DUI SEI messages in the current BP. When not present, the value of bp_du_cpb_removal_delay_increment_length_minus1 is inferred to be equal to 23.

bp_dpb_output_delay_du_length_minus1 plus 1 specifies the length, in bits, of the pt_dpb_output_du_delay syntax element in the PT SEI messages in the current BP and the dui_dpb_output_du_delay syntax element in the DUI SEI messages in the current BP. When not present, the value of bp_dpb_output_delay_du_length_minus1 is inferred to be equal to 23.

It is a requirement of bitstream conformance that all scalable-nested and non-scalable nested BP SEI messages in a CVS shall have the same value for each of the syntax elements bp_cpb_initial_removal_delay_length_minus1, bp_cpb_removal_delay_length_minus1, bp_dpb_output_delay_length_minus1, bp_du_cpb_removal_delay_increment_length_minus1, and bp_dpb_output_delay_du_length_minus1.

bp_du_cpb_params_in_pic_timing_sei_flag equal to 1 specifies that DU level CPB removal delay parameters are present in PT SEI messages and no DUI SEI message is available (in the CVS or provided through external means not specified in this Specification). bp_du_cpb_params_in_pic_timing_sei_flag equal to 0 specifies that DU level CPB removal delay parameters are present in DUI SEI messages and PT SEI messages do not include DU level CPB removal delay parameters. When the bp_du_cpb_params_in_pic_timing_sei_flag syntax element is not present, it is inferred to be equal to 0.

bp_du_dpb_params_in_pic_timing_sei_flag equal to 1 specifies that DU level DPB output delay parameters are present in PT SEI messages and not in DUI SEI messages. bp_du_dpb_params_in_pic_timing_sei_flag equal to 0 specifies that DU level DPB output delay parameters are present in DUI SEI messages and not in PT SEI messages. When the bp_du_dpb_params_in_pic_timing_sei_flag syntax element is not present, it is inferred to be equal to 0.

bp_concatenation_flag indicates, when the current AU is not the first AU in the bitstream in decoding order, whether the nominal CPB removal time of the current AU is determined relative to the nominal CPB removal time of the previous AU associated with a BP SEI message or relative to the nominal CPB removal time of the AU prevNonDiscardableAu.

bp_additional_concatenation_info_present_flag equal to 1 specifies that the syntax element bp_max_initial_removal_delay_for_concatenation is present in the BP SEI message and the syntax element pt_delay_for_concatenation_ensured_flag is present in the PT SEI messages. bp_additional_concatenation_info_present_flag equal to 0 specifies that the syntax element bp_max_initial_removal_delay_for_concatenation is not present in the BP SEI message and the syntax element pt_delay_for_concatenation_ensured_flag is not present in the PT SEI messages.

bp_max_initial_removal_delay_for_concatenation could be used together with pt_delay_for_concatenation_ensured_flag in a PT SEI message to identify whether the nominal removal time from the CPB of the first AU of a following BP computed with bp_cpb_removal_delay_delta_minus1 applies. The length of bp_max_initial_removal_delay_for_concatenation is bp_cpb_initial_removal_delay_length_minus1 + 1 bits.

bp_cpb_removal_delay_delta_minus1 plus 1, when the current AU is not the first AU in the bitstream in decoding order, specifies a CPB removal delay increment value relative to the nominal CPB removal time of the AU prevNonDiscardableAu. The length of this syntax element is bp_cpb_removal_delay_length_minus1 + 1 bits.

When the current AU is associated with a BP SEI message and bp_concatenation_flag is equal to 0 and the current AU is not the first AU in the bitstream in decoding order, it is a requirement of bitstream conformance that the following constraint applies:

– If the AU prevNonDiscardableAu is not associated with a BP SEI message, the pt_cpb_removal_delay_minus1 of the current AU shall be equal to the pt_cpb_removal_delay_minus1 of the AU prevNonDiscardableAu plus bp_cpb_removal_delay_delta_minus1 + 1.

– Otherwise, pt_cpb_removal_delay_minus1 shall be equal to bp_cpb_removal_delay_delta_minus1.

NOTE 2 – When the current AU is associated with a BP SEI message and bp_concatenation_flag is equal to 1, the pt_cpb_removal_delay_minus1 for the current AU is not used. The constraint expressed for pt_cpb_removal_delay_minus1 could, under some circumstances, make it possible to splice bitstreams (that use suitably-designed referencing structures) by simply changing the value of bp_concatenation_flag from 0 to 1 in the BP SEI message for an IRAP or GDR AU at the splicing point. When bp_concatenation_flag is equal to 0, the constraint expressed for pt_cpb_removal_delay_minus1 enables the decoder to check whether the constraint is satisfied as a way to detect the loss of the AU prevNonDiscardableAu.

bp_cpb_removal_delay_deltas_present_flag equal to 1 specifies that the BP SEI message contains CPB removal delay deltas. bp_cpb_removal_delay_deltas_present_flag equal to 0 specifies that no CPB removal delay deltas are present in the BP SEI message. When not present bp_cpb_removal_delay_deltas_present_flag is inferred to be equal to 0.

bp_num_cpb_removal_delay_deltas_minus1 plus 1 specifies the number of syntax elements bp_cpb_removal_delay_delta_val[i] in the BP SEI message. The value of bp_num_cpb_removal_delay_deltas_minus1 shall be in the range of 0 to 15, inclusive.

bp_cpb_removal_delay_delta_val[i] specifies the i-th CPB removal delay delta. The length of this syntax element is bp_cpb_removal_delay_length_minus1 + 1 bits.

bp_max_sublayers_minus1 plus 1 specifies the maximum number of temporal sublayers for which the initial CPB removal delay and the initial CPB removal offset are indicated in the BP SEI message. The value of bp_max_sublayers_minus1 shall be in the range of 0 to vps_max_sublayers_minus1, inclusive.

bp_cpb_cnt_minus1 plus 1 specifies the number of syntax element pairs bp_nal_initial_cpb_removal_delay[i][j] and bp_nal_initial_cpb_removal_offset[i][j] of the i-th temporal sublayer when bp_nal_hrd_params_present_flag is equal to 1, and the number of syntax element pairs bp_vcl_initial_cpb_removal_delay[i][j] and bp_vcl_initial_cpb_removal_offset[i][j] of the i-th temporal sublayer when bp_vcl_hrd_params_present_flag is equal to 1. The value of bp_cpb_cnt_minus1 shall be in the range of 0 to 31, inclusive.

The value of bp_cpb_cnt_minus1 shall be equal to the value of hrd_cpb_cnt_minus1.

bp_sublayer_initial_cpb_removal_delay_present_flag equal to 1 specifies that initial CPB removal delay related syntax elements are present for sublayer representation(s) in the range of 0 to bp_max_sublayers_minus1, inclusive. bp_sublayer_initial_cpb_removal_delay_present_flag equal to 0 specifies that initial CPB removal delay related syntax elements are present for the bp_max_sublayers_minus1-th sublayer representation. When not present, the value of bp_sublayer_initial_cpb_removal_delay_present_flag is inferred to be equal to 0.

bp_nal_initial_cpb_removal_delay[i][j] and **bp_nal_initial_alt_cpb_removal_delay**[i][j] specify the j-th default and alternative initial CPB removal delay for the NAL HRD in units of a 90 kHz clock of the i-th temporal sublayer. The length of bp_nal_initial_cpb_removal_delay[i][j] and bp_nal_initial_alt_cpb_removal_delay[i][j] is bp_cpb_initial_removal_delay_length_minus1 + 1 bits. The value of bp_nal_initial_cpb_removal_delay[i][j] and bp_nal_initial_alt_cpb_removal_delay[i][j] shall not be equal to 0 and shall be less than or equal to 90000 * (CpbSize[i][j] ÷ BitRate[i][j]), the time-equivalent of the CPB size in 90 kHz clock units. When not present, the values of bp_nal_initial_cpb_removal_delay[i][j] and bp_nal_initial_alt_cpb_removal_delay[i][j] are inferred to be equal to 90000 * (CpbSize[i][j] ÷ BitRate[i][j]).

bp_nal_initial_cpb_removal_offset[i][j] and **bp_nal_initial_alt_cpb_removal_offset**[i][j] specify the j-th default and alternative initial CPB removal offset of the i-th temporal sublayer for the NAL HRD in units of a 90 kHz clock. The length of bp_nal_initial_cpb_removal_offset[i][j] and bp_nal_initial_alt_cpb_removal_offset[i][j] is bp_cpb_initial_removal_delay_length_minus1 + 1 bits. When not present, the values of bp_nal_initial_cpb_removal_offset[i][j] and bp_nal_initial_alt_cpb_removal_offset[i][j] are inferred to be equal to 0.

Over the entire CVS, for each value pair of i and j, the sum of bp_nal_initial_cpb_removal_delay[i][j] and bp_nal_initial_cpb_removal_offset[i][j] shall be constant, and the sum of bp_nal_initial_alt_cpb_removal_delay[i][j] and bp_nal_initial_alt_cpb_removal_offset[i][j] shall be constant.

bp_vcl_initial_cpb_removal_delay[i][j] and **bp_vcl_initial_alt_cpb_removal_delay**[i][j] specify the j-th default and alternative initial CPB removal delay of the i-th temporal sublayer for the VCL HRD in units of a 90 kHz clock. The length of bp_vcl_initial_cpb_removal_delay[i][j] and bp_vcl_initial_alt_cpb_removal_delay[i][j] is bp_cpb_initial_removal_delay_length_minus1 + 1 bits. The value of bp_vcl_initial_cpb_removal_delay[i][j] and bp_vcl_initial_alt_cpb_removal_delay[i][j] shall not be equal to 0 and shall be less than or equal to 90000 * (CpbSize[i][j] ÷ BitRate[i][j]), the time-equivalent of the CPB size in 90 kHz clock units. When not present, the values of bp_vcl_initial_cpb_removal_delay[i][j] and bp_vcl_initial_alt_cpb_removal_delay[i][j] are inferred to be equal to 90000 * (CpbSize[i][j] ÷ BitRate[i][j]).

bp_vcl_initial_cpb_removal_offset[i][j] and **bp_vcl_initial_alt_cpb_removal_offset**[i][j] specify the j-th default and alternative initial CPB removal offset of the i-th temporal sublayer for the VCL HRD in units of a 90 kHz clock. The length of bp_vcl_initial_cpb_removal_offset[i] and bp_vcl_initial_alt_cpb_removal_offset[i][j] is bp_cpb_initial_removal_delay_length_minus1 + 1 bits. When not present, the values of bp_vcl_initial_cpb_removal_offset[i][j] and bp_vcl_initial_alt_cpb_removal_offset[i][j] are inferred to be equal to 0.

Over the entire CVS, for each value pair of i and j the sum of bp_vcl_initial_cpb_removal_delay[i][j] and bp_vcl_initial_cpb_removal_offset[i][j] shall be constant, and the sum of bp_vcl_initial_alt_cpb_removal_delay[i][j] and bp_vcl_initial_alt_cpb_removal_offset[i][j] shall be constant.

bp_sublayer_dpb_output_offsets_present_flag equal to 1 specifies that DPB output time offsets are present for sublayer representation(s) with TemporalId in the range of 0 to bp_max_sublayers_minus1 − 1, inclusive. bp_sublayer_dpb_output_offsets_present_flag equal to 0 specified that no such DPB output time offsets are present. When not present, the value of bp_sublayer_dpb_output_offsets_present_flag is inferred to be equal to 0.

bp_dpb_output_tid_offset[i] specifies the difference between the DPB output times for the i-th sublayer representation and the bp_max_sublayers_minus1-th sublayer representation. When bp_dpb_output_tid_offset[i] is not present, it is inferred to be equal to 0.

bp_alt_cpb_params_present_flag equal to 1 specifies the presence of the syntax element bp_use_alt_cpb_params_flag in the BP SEI message and the presence of the alternative timing information in the PT SEI messages in the current BP. When not present, the value of bp_alt_cpb_params_present_flag is inferred to be equal to 0. When the associated AU is not an IRAP or GDR AU, the value of bp_alt_cpb_params_present_flag shall be equal to 0.

bp_use_alt_cpb_params_flag could be used to derive the value of UseAltCpbParamsFlag. When bp_use_alt_cpb_params_flag is not present, it is inferred to be equal to 0.

When one or more of the following condtions apply, UseAltCpbParamsFlag is set equal to 1:

– bp_use_alt_cpb_params_flag is equal to 1.

– When some external means not specified in this Specification is available to set UseAltCpbParamsFlag and the value of UseAltCpbParamsFlag is set equal to 1 by the external means.

D.4 Picture timing SEI message

D.4.1 Picture timing SEI message syntax

pic_timing(payloadSize) {	**Descriptor**
pt_cpb_removal_delay_minus1[bp_max_sublayers_minus1]	u(v)
for(i = TemporalId; i < bp_max_sublayers_minus1; i++) {	
pt_sublayer_delays_present_flag[i]	u(1)
if(pt_sublayer_delays_present_flag[i]) {	
if(bp_cpb_removal_delay_deltas_present_flag)	
pt_cpb_removal_delay_delta_enabled_flag[i]	u(1)
if(pt_cpb_removal_delay_delta_enabled_flag[i]) {	
if(bp_num_cpb_removal_delay_deltas_minus1 > 0)	
pt_cpb_removal_delay_delta_idx[i]	u(v)
} else	
pt_cpb_removal_delay_minus1[i]	u(v)
}	
}	
pt_dpb_output_delay	u(v)
if(bp_alt_cpb_params_present_flag) {	
pt_cpb_alt_timing_info_present_flag	u(1)
if(pt_cpb_alt_timing_info_present_flag) {	
if(bp_nal_hrd_params_present_flag) {	
for(i = (bp_sublayer_initial_cpb_removal_delay_present_flag ? 0 : bp_max_sublayers_minus1); i <= bp_max_sublayers_minus1; i++) {	
for(j = 0; j < bp_cpb_cnt_minus1 + 1; j++) {	
pt_nal_cpb_alt_initial_removal_delay_delta[i][j]	u(v)
pt_nal_cpb_alt_initial_removal_offset_delta[i][j]	u(v)
}	
pt_nal_cpb_delay_offset[i]	u(v)
pt_nal_dpb_delay_offset[i]	u(v)
}	
}	
if(bp_vcl_hrd_params_present_flag) {	
for(i = (bp_sublayer_initial_cpb_removal_delay_present_flag ? 0 : bp_max_sublayers_minus1); i <= bp_max_sublayers_minus1; i++) {	
for(j = 0; j < bp_cpb_cnt_minus1 + 1; j++) {	
pt_vcl_cpb_alt_initial_removal_delay_delta[i][j]	u(v)
pt_vcl_cpb_alt_initial_removal_offset_delta[i][j]	u(v)
}	
pt_vcl_cpb_delay_offset[i]	u(v)

pt_vcl_dpb_delay_offset[i]	u(v)
}	
}	
}	
}	
if(bp_du_hrd_params_present_flag && bp_du_dpb_params_in_pic_timing_sei_flag)	
pt_dpb_output_du_delay	u(v)
if(bp_du_hrd_params_present_flag && bp_du_cpb_params_in_pic_timing_sei_flag) {	
pt_num_decoding_units_minus1	ue(v)
if(pt_num_decoding_units_minus1 > 0) {	
pt_du_common_cpb_removal_delay_flag	u(1)
if(pt_du_common_cpb_removal_delay_flag)	
for(i = TemporalId; i <= bp_max_sublayers_minus1; i++)	
if(pt_sublayer_delays_present_flag[i])	
pt_du_common_cpb_removal_delay_increment_minus1[i]	u(v)
for(i = 0; i <= pt_num_decoding_units_minus1; i++) {	
pt_num_nalus_in_du_minus1[i]	ue(v)
if(!pt_du_common_cpb_removal_delay_flag && i < pt_num_decoding_units_minus1)	
for(j = TemporalId; j <= bp_max_sublayers_minus1; j++)	
if(pt_sublayer_delays_present_flag[j])	
pt_du_cpb_removal_delay_increment_minus1[i][j]	u(v)
}	
}	
}	
if(bp_additional_concatenation_info_present_flag)	
pt_delay_for_concatenation_ensured_flag	u(1)
pt_display_elemental_periods_minus1	u(8)
}	

D.4.2 Picture timing SEI message semantics

The PT SEI message provides CPB removal delay and DPB output delay information for the AU associated with the SEI message.

If bp_nal_hrd_params_present_flag or bp_vcl_hrd_params_present_flag of the BP SEI message applicable for the current AU is equal to 1, the variable CpbDpbDelaysPresentFlag is set equal to 1. Otherwise, CpbDpbDelaysPresentFlag is set equal to 0.

The presence of PT SEI messages is specified as follows:

– If CpbDpbDelaysPresentFlag is equal to 1, a PT SEI message shall be associated with the current AU.

– Otherwise (CpbDpbDelaysPresentFlag is equal to 0), there shall not be a PT SEI message associated with the current AU.

The TemporalId in the PT SEI message syntax is the TemporalId of the SEI NAL unit containing the PT SEI message.

pt_cpb_removal_delay_minus1[i] plus 1 is used to calculate the number of clock ticks between the nominal CPB removal times of the AU associated with the PT SEI message and the preceding AU in decoding order that contains a BP SEI message when Htid is equal to i. This value is also used to calculate an earliest possible time of arrival of AU data into the CPB for the HSS. The length of pt_cpb_removal_delay_minus1[i] is bp_cpb_removal_delay_length_minus1 + 1 bits.

pt_sublayer_delays_present_flag[i] equal to 1 specifies that pt_cpb_removal_delay_delta_idx[i] or pt_cpb_removal_delay_minus1[i], and pt_du_common_cpb_removal_delay_increment_minus1[i] or pt_du_cpb_removal_delay_increment_minus1[ ][ ] are present for the sublayer with TemporalId equal to i. sublayer_delays_present_flag[i] equal to 0 specifies that neither pt_cpb_removal_delay_delta_idx[i] nor pt_cpb_removal_delay_minus1[i] and neither pt_du_common_cpb_removal_delay_increment_minus1[i] nor pt_du_cpb_removal_delay_increment_minus1[ ][ ] are present for the sublayer with TemporalId equal to i. The value of pt_sublayer_delays_present_flag[bp_max_sublayers_minus1] is inferred to be equal to 1. When not present, the value of pt_sublayer_delays_present_flag[i] for any i in the range of 0 to bp_max_sublayers_minus1 − 1, inclusive, is inferred to be equal to 0.

pt_cpb_removal_delay_delta_enabled_flag[i] equal to 1 specifies that pt_cpb_removal_delay_delta_idx[i] is present in the PT SEI message. pt_cpb_removal_delay_delta_enabled_flag[i] equal to 0 specifies that pt_cpb_removal_delay_delta_idx[i] is not present in the PT SEI message. When not present, the value of pt_cpb_removal_delay_delta_enabled_flag[i] is inferred to be equal to 0.

pt_cpb_removal_delay_delta_idx[i] specifies the index of the CPB removal delta that applies to Htid equal to i in the list of bp_cpb_removal_delay_delta_val[j] for j ranging from 0 to bp_num_cpb_removal_delay_deltas_minus1, inclusive. The length of pt_cpb_removal_delay_delta_idx[i] is Ceil(Log2(bp_num_cpb_removal_delay_deltas_minus1 + 1)) bits. When pt_cpb_removal_delay_delta_idx[i] is not present and pt_cpb_removal_delay_delta_enabled_flag[i] is equal to 1, the value of pt_cpb_removal_delay_delta_idx[i] is inferred to be equal to 0.

The variables CpbRemovalDelayMsb[i] and CpbRemovalDelayVal[i] of the current AU are derived as follows:

– If the current AU is the AU that initializes the HRD, CpbRemovalDelayMsb[i] and CpbRemovalDelayVal[i] are both set equal to 0, and the value of cpbRemovalDelayValTmp[i] is set equal to pt_cpb_removal_delay_minus1[i] + 1.

– Otherwise, let the AU prevNonDiscardableAu be the previous AU in decoding order with TemporalId equal to 0 that has at least one picture that has ph_non_ref_pic_flag equal to 0 that is not a RASL or RADL, let prevCpbRemovalDelayMinus1[i], prevCpbRemovalDelayMsb[i], and prevBpResetFlag be set equal to the values of cpbRemovalDelayValTmp[i] − 1, CpbRemovalDelayMsb[i], and BpResetFlag, respectively, for the AU prevNonDiscardablAu, and the following applies:

  – CpbRemovalDelayMsb[i] is derived as follows:

```
cpbRemovalDelayValTmp[ i ] = pt_cpb_removal_delay_delta_enabled_flag[ i ] ?
      pt_cpb_removal_delay_minus1[ bp_max_sublayers_minus1 ] + 1 +
      bp_cpb_removal_delay_delta_val[ pt_cpb_removal_delay_delta_idx[ i ] ] :
      pt_cpb_removal_delay_minus1[ i ] + 1
if( prevBpResetFlag )
   CpbRemovalDelayMsb[ i ] = 0
else if( cpbRemovalDelayValTmp[ i ] < prevCpbRemovalDelayMinus1[ i ] )
   CpbRemovalDelayMsb[ i ] = prevCpbRemovalDelayMsb[ i ] + 2^(bp_cpb_removal_delay_length_minus1 + 1)      (1626)
else
   CpbRemovalDelayMsb[ i ] = prevCpbRemovalDelayMsb[ i ]
```

  – CpbRemovalDelayVal is derived as follows:

```
if( pt_sublayer_delays_present_flag[ i ] )
   CpbRemovalDelayVal[ i ] = CpbRemovalDelayMsb[ i ] + cpbRemovalDelayValTmp[ i ]      (1627)
else
   CpbRemovalDelayVal[ i ] = CpbRemovalDelayVal[ i + 1 ]
```

The value of CpbRemovalDelayVal[i] shall be in the range of 1 to 2^{32}, inclusive.

The variable AuDpbOutputDelta[i] is derived as follows:

```
AuDpbOutputDelta[ i ] = CpbRemovalDelayVal[ i ] −
      CpbRemovalDelayVal[ bp_max_sublayers_minus1 ] −      (1628)
      ( i  = =  bp_max_sublayers_minus1 ? 0 : bp_dpb_output_tid_offset[ i ] )
```

Where the value of bp_dpb_output_tid_offset[i] is found in the associated BP SEI message.

pt_dpb_output_delay is used to compute the DPB output time of the AU. It specifies how many clock ticks to wait after removal of an AU from the CPB before the decoded pictures of the AU are output from the DPB.

NOTE 1 – A decoded 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".

The length of pt_dpb_output_delay is bp_dpb_output_delay_length_minus1 + 1 bits. When dpb_max_dec_pic_buffering_minus1[Htid] is equal to 0, the value of pt_dpb_output_delay shall be equal to 0.

The output time derived from the pt_dpb_output_delay of any picture that is output from an output timing conforming decoder shall precede the output time derived from the pt_dpb_output_delay of all pictures in any subsequent CVS in decoding order.

The picture output order established by the values of this syntax element shall be the same order as established by the values of PicOrderCntVal.

For pictures that are not output by the "bumping" process because they precede, in decoding order, a CVSS AU that has NoOutputOfPriorPicsFlag equal to 1, the output times derived from pt_dpb_output_delay shall be increasing with increasing value of PicOrderCntVal relative to all pictures within the same CVS.

pt_cpb_alt_timing_info_present_flag equal to 1 specifies that the syntax elements pt_nal_cpb_alt_initial_removal_delay_delta[i][j], pt_nal_cpb_alt_initial_removal_offset_delta[i][j], pt_nal_cpb_delay_offset[i], pt_nal_dpb_delay_offset[i], pt_vcl_cpb_alt_initial_removal_delay_delta[i][j], pt_vcl_cpb_alt_initial_removal_offset_delta[i][j], pt_vcl_cpb_delay_offset[i], and pt_vcl_dpb_delay_offset[i] could be present in the PT SEI message. pt_cpb_alt_timing_info_present_flag equal to 0 specifies that these syntax elements are not present in the PT SEI message. When all pictures in the associated AU are RASL pictures with pps_mixed_nalu_types_in_pic_flag equal to 0, the value of pt_cpb_alt_timing_info_present_flag shall be equal to 0.

NOTE 2 – The value of pt_cpb_alt_timing_info_present_flag could be equal to 1 for more than one AU following an IRAP AU in decoding order. However, the alternative timing is only applied to the first AU that has pt_cpb_alt_timing_info_present_flag equal to 1 and follows the IRAP AU in decoding order.

pt_nal_cpb_alt_initial_removal_delay_delta[i][j] specifies the alternative initial CPB removal delay delta for the i-th sublayer for the j-th CPB for the NAL HRD in units of a 90 kHz clock. The length of pt_nal_cpb_alt_initial_removal_delay_delta[i][j] is bp_cpb_initial_removal_delay_length_minus1 + 1 bits.

When pt_cpb_alt_timing_info_present_flag is equal to 1 and pt_nal_cpb_alt_initial_removal_delay_delta[i][j] is not present for any value of i less than bp_max_sublayers_minus1, its value is inferred to be equal to 0.

pt_nal_cpb_alt_initial_removal_offset_delta[i][j] specifies the alternative initial CPB removal offset delta for the i-th sublayer for the j-th CPB for the NAL HRD in units of a 90 kHz clock. The length of pt_nal_cpb_alt_initial_removal_offset_delta[i][j] is bp_cpb_initial_removal_delay_length_minus1 + 1 bits.

When pt_cpb_alt_timing_info_present_flag is equal to 1 and pt_nal_cpb_alt_initial_removal_offset_delta[i][j] is not present for any value of i less than bp_max_sublayers_minus1, its value is inferred to be equal to 0.

pt_nal_cpb_delay_offset[i] specifies, for the i-th sublayer for the NAL HRD, an offset to be used in the derivation of the nominal CPB removal times of the AU associated with the PT SEI message and of the AUs following in decoding order, when the AU associated with the PT SEI message directly follows in decoding order the AU associated with the BP SEI message. The length of pt_nal_cpb_delay_offset[i] is bp_cpb_removal_delay_length_minus1 + 1 bits. When not present, the value of pt_nal_cpb_delay_offset[i] is inferred to be equal to 0.

pt_nal_dpb_delay_offset[i] specifies, for the i-th sublayer for the NAL HRD, an offset to be used in the derivation of the DPB output times of the IRAP AU associated with the BP SEI message when the AU associated with the PT SEI message directly follows in decoding order the IRAP AU associated with the BP SEI message. The length of pt_nal_dpb_delay_offset[i] is bp_dpb_output_delay_length_minus1 + 1 bits. When not present, the value of pt_nal_dpb_delay_offset[i] is inferred to be equal to 0.

pt_vcl_cpb_alt_initial_removal_delay_delta[i][j] specifies the alternative initial CPB removal delay delta for the i-th sublayer for the j-th CPB for the VCL HRD in units of a 90 kHz clock. The length of pt_vcl_cpb_alt_initial_removal_delay_delta[i][j] is bp_cpb_initial_removal_delay_length_minus1 + 1 bits.

When pt_cpb_alt_timing_info_present_flag is equal to 1 and pt_vcl_cpb_alt_initial_removal_delay_delta[i][j] is not present for any value of i less than bp_max_sublayers_minus1, its value is inferred to be equal to 0.

pt_vcl_cpb_alt_initial_removal_offset_delta[i][j] specifies the alternative initial CPB removal offset delta for the i-th sublayer for the j-th CPB for the VCL HRD in units of a 90 kHz clock. The length of pt_vcl_cpb_alt_initial_removal_offset_delta[i][j] is bp_cpb_initial_removal_delay_length_minus1 + 1 bits.

When pt_cpb_alt_timing_info_present_flag is equal to 1 and pt_vcl_cpb_alt_initial_removal_offset_delta[i][j] is not present for any value of i less than bp_max_sublayers_minus1, its value is inferred to be equal to 0.

pt_vcl_cpb_delay_offset[i] specifies, for the i-th sublayer for the VCL HRD, an offset to be used in the derivation of the nominal CPB removal times of the AU associated with the PT SEI message and of the AUs following in decoding order, when the AU associated with the PT SEI message directly follows in decoding order the AU associated with the BP SEI message. The length of pt_vcl_cpb_delay_offset[i] is bp_cpb_removal_delay_length_minus1 + 1 bits. When not present, the value of pt_vcl_cpb_delay_offset[i] is inferred to be equal to 0.

pt_vcl_dpb_delay_offset[i] specifies, for the i-th sublayer for the VCL HRD, an offset to be used in the derivation of the DPB output times of the IRAP AU associated with the BP SEI message when the AU associated with the PT SEI message directly follows in decoding order the IRAP AU associated with the BP SEI message. The length of pt_vcl_dpb_delay_offset[i] is bp_dpb_output_delay_length_minus1 + 1 bits. When not present, the value of pt_vcl_dpb_delay_offset[i] is inferred to be equal to 0.

The variable BpResetFlag of the current AU is derived as follows:

– If the current AU is associated with a BP SEI message, BpResetFlag is set equal to 1.

– Otherwise, BpResetFlag is set equal to 0.

pt_dpb_output_du_delay is used to compute the DPB output time of the AU when DecodingUnitHrdFlag is equal to 1. It specifies how many sub clock ticks to wait after removal of the last DU in an AU from the CPB before the decoded pictures of the AU are output from the DPB.

The length of the syntax element pt_dpb_output_du_delay is given in bits by bp_dpb_output_delay_du_length_minus1 + 1.

The output time derived from the pt_dpb_output_du_delay of any picture that is output from an output timing conforming decoder shall precede the output time derived from the pt_dpb_output_du_delay of all pictures in any subsequent CVS in decoding order.

The picture output order established by the values of this syntax element shall be the same order as established by the values of PicOrderCntVal.

For pictures that are not output by the "bumping" process because they precede, in decoding order, a CVSS AU that has NoOutputOfPriorPicsFlag equal to 1, the output times derived from pt_dpb_output_du_delay shall be increasing with increasing value of PicOrderCntVal relative to all pictures within the same CVS.

For any two pictures in the CVS, the difference between the output times of the two pictures when DecodingUnitHrdFlag is equal to 1 shall be identical to the same difference when DecodingUnitHrdFlag is equal to 0.

pt_num_decoding_units_minus1 plus 1 specifies the number of DUs in the AU the PT SEI message is associated with. The value of pt_num_decoding_units_minus1 shall be in the range of 0 to PicSizeInCtbsY − 1, inclusive.

pt_du_common_cpb_removal_delay_flag equal to 1 specifies that the syntax elements pt_du_common_cpb_removal_delay_increment_minus1[i] are present. pt_du_common_cpb_removal_delay_flag equal to 0 specifies that the syntax elements pt_du_common_cpb_removal_delay_increment_minus1[i] are not present. When not present pt_du_common_cpb_removal_delay_flag is inferred to be equal to 0.

pt_du_common_cpb_removal_delay_increment_minus1[i] plus 1 specifies the duration, in units of clock sub-ticks (see clause C.1), between the nominal CPB removal times of any two consecutive DUs in decoding order in the AU associated with the PT SEI message when Htid is equal to i. This value is also used to calculate an earliest possible time of arrival of DU data into the CPB for the HSS, as specified in Annex C. The length of this syntax element is bp_du_cpb_removal_delay_increment_length_minus1 + 1 bits.

When pt_du_common_cpb_removal_delay_increment_minus1[i] is not present for any value of i less than bp_max_sublayers_minus1, its value is inferred to be equal to pt_du_common_cpb_removal_delay_increment_minus1[bp_max_sublayers_minus1].

pt_num_nalus_in_du_minus1[i] plus 1 specifies the number of NAL units in the i-th DU of the AU the PT SEI message is associated with. The value of pt_num_nalus_in_du_minus1[i] shall be in the range of 0 to PicSizeInCtbsY − 1, inclusive.

The first DU of the AU consists of the first pt_num_nalus_in_du_minus1[0] + 1 consecutive NAL units in decoding order in the AU. The i-th (with i greater than 0) DU of the AU consists of the pt_num_nalus_in_du_minus1[i] + 1 consecutive NAL units immediately following the last NAL unit in the previous DU of the AU, in decoding order. There shall be at least one VCL NAL unit in each DU. All non-VCL NAL units associated with a VCL NAL unit shall be included in the same DU as the VCL NAL unit.

pt_du_cpb_removal_delay_increment_minus1[i][j] plus 1 specifies the duration, in units of clock sub-ticks, between the nominal CPB removal times of the (i + 1)-th DU and the i-th DU, in decoding order, in the AU associated with the PT SEI message when Htid is equal to j. This value is also used to calculate an earliest possible time of arrival of DU data into the CPB for the HSS, as specified in Annex C. The length of this syntax element is bp_du_cpb_removal_delay_increment_length_minus1 + 1 bits.

When pt_du_cpb_removal_delay_increment_minus1[i][j] is not present for any value of j less than bp_max_sublayers_minus1, its value is inferred to be equal to pt_du_cpb_removal_delay_increment_minus1[i][bp_max_sublayers_minus1].

pt_delay_for_concatenation_ensured_flag equal to 1 specifies that the difference between the final arrival time and the CPB removal time of the AU associated with the PT SEI message is such that when followed by an AU with a BP SEI message with bp_concatenation_flag equal to 1 and InitCpbRemovalDelay[ScIdx] less than or equal to the value of bp_max_initial_removal_delay_for_concatenation, the nominal removal time of the following AU from the CPB computed with bp_cpb_removal_delay_delta_minus1 applies. pt_delay_for_concatenation_ensured_flag equal to 0 specifies that the difference between the final arrival time and the CPB removal time of the AU associated with the PT SEI message might or might not exceed the value of max_val_initial_removal_delay_for_splicing.

pt_display_elemental_periods_minus1 plus 1, when sps_field_seq_flag is equal to 0 and fixed_pic_rate_within_cvs_flag[Htid] is equal to 1, indicates the number of elemental picture period intervals that the decoded pictures of the current AU occupy for the display model.

When fixed_pic_rate_within_cvs_flag[Htid] is present and equal to 1 and both general_nal_hrd_params_present_flag and general_vcl_hrd_params_present_flag are equal to 0, the value of pt_display_elemental_periods_minus1, if provided by external means, shall be equal to 0.

When sps_field_seq_flag is equal to 1, the value of pt_display_elemental_periods_minus1 shall be equal to 0.

When sps_field_seq_flag is equal to 0 and fixed_pic_rate_within_cvs_flag[Htid] is equal to 1, a value of pt_display_elemental_periods_minus1 greater than 0 could be used to indicate a frame repetition period for displays that use a fixed frame refresh interval equal to DpbOutputElementalInterval[n] as given by Equation 108.

D.5 DU information SEI message

D.5.1 DU information SEI message syntax

decoding_unit_info(payloadSize) {	**Descriptor**
dui_decoding_unit_idx	ue(v)
if(!bp_du_cpb_params_in_pic_timing_sei_flag)	
for(i = TemporalId; i <= bp_max_sublayers_minus1; i++) {	
if(i < bp_max_sublayers_minus1)	
dui_sublayer_delays_present_flag[i]	u(1)
if(dui_sublayer_delays_present_flag[i])	
dui_du_cpb_removal_delay_increment[i]	u(v)
}	
if(!bp_du_dpb_params_in_pic_timing_sei_flag)	
dui_dpb_output_du_delay_present_flag	u(1)
if(dui_dpb_output_du_delay_present_flag)	
dui_dpb_output_du_delay	u(v)
}	

D.5.2 DU information SEI message semantics

The DUI SEI message provides CPB removal delay information for the DU associated with the SEI message.

The following applies for the DUI SEI message syntax and semantics:

– The syntax elements bp_du_hrd_params_present_flag, bp_du_cpb_params_in_pic_timing_sei_flag, bp_du_dpb_params_in_pic_timing_sei_flag, and bp_dpb_output_delay_du_length_minus1 are found in the BP SEI message that is applicable to at least one of the operation points to which the DUI SEI message applies.

– The bitstream (or a part thereof) refers to the bitstream subset (or a part thereof) associated with any of the operation points to which the DUI SEI message applies.

The presence of DUI SEI messages for an operation point is specified as follows:

– If CpbDpbDelaysPresentFlag is equal to 1, bp_du_hrd_params_present_flag is equal to 1 and bp_du_cpb_params_in_pic_timing_sei_flag or bp_du_dpb_params_in_pic_timing_sei_flag is equal to 0, one or more DUI SEI messages applicable to the operation point shall be associated with each DU in the CVS.
– Otherwise, in the CVS there shall be no DU that is associated with a DUI SEI message applicable to the operation point.

The set of NAL units associated with a DUI SEI message consists, in decoding order, of the SEI NAL unit containing the DUI SEI message and all subsequent NAL units in the AU up to but not including any subsequent SEI NAL unit containing a DUI SEI message with a different value of dui_decoding_unit_idx. Each DU shall include at least one VCL NAL unit. All non-VCL NAL units associated with a VCL NAL unit shall be included in the DU containing the VCL NAL unit.

The TemporalId in the DUI SEI message syntax is the TemporalId of the SEI NAL unit containing the DUI SEI message.

dui_decoding_unit_idx specifies the index, starting from 0, to the list of DUs in the current AU, of the DU associated with the DUI SEI message. The value of dui_decoding_unit_idx shall be in the range of 0 to PicSizeInCtbsY − 1, inclusive.

A DU identified by a particular value of duIdx includes and only includes all NAL units associated with all DUI SEI messages that have dui_decoding_unit_idx equal to duIdx. Such a DU is also referred to as associated with the DUI SEI messages having dui_decoding_unit_idx equal to duIdx.

For any two DUs duA and duB in one AU with dui_decoding_unit_idx equal to duIdxA and duIdxB, respectively, where duIdxA is less than duIdxB, duA shall precede duB in decoding order.

A NAL unit of one DU shall not be present, in decoding order, between any two NAL units of another DU.

dui_sublayer_delays_present_flag[i] equal to 1 specifies that dui_du_cpb_removal_delay_increment[i] is present for the sublayer with TemporalId equal to i. dui_sublayer_delays_present_flag[i] equal to 0 specifies that dui_du_cpb_removal_delay_increment[i] is not present for the sublayer with TemporalId equal to i.

When not present, the value of dui_sublayer_delays_present_flag[i] is inferred to be as follows:

– If bp_du_cpb_params_in_pic_timing_sei_flag is equal to 0 and i is equal to bp_max_sublayers_minus1, the value of dui_sublayer_delays_present_flag[i] is inferred to be equal to 1.
– Otherwise, the value of dui_sublayer_delays_present_flag[i] is inferred to be equal to 0.

dui_du_cpb_removal_delay_increment[i] specifies the duration, in units of clock sub-ticks, between the nominal CPB times of the last DU in decoding order in the current AU and the DU associated with the DUI SEI message when Htid is equal to i. This value is also used to calculate an earliest possible time of arrival of DU data into the CPB for the HSS, as specified in Annex C. The length of this syntax element is bp_du_cpb_removal_delay_increment_length_minus1 + 1. When the DU associated with the DUI SEI message is the last DU in the current AU, the value of dui_du_cpb_removal_delay_increment[i] shall be equal to 0. When dui_du_cpb_removal_delay_increment[i] is not present for any value of i less than bp_max_sublayers_minus1, its value is inferred to be equal to dui_du_cpb_removal_delay_increment[bp_max_sublayers_minus1].

dui_dpb_output_du_delay_present_flag equal to 1 specifies the presence of the dui_dpb_output_du_delay syntax element in the DUI SEI message. dui_dpb_output_du_delay_present_flag equal to 0 specifies the absence of the dui_dpb_output_du_delay syntax element in the DUI SEI message. When not present, the value of dui_dpb_output_du_delay_present_flag is inferred to be equal to 0. When bp_du_dpb_params_in_pic_timing_sei_flag is equal to 0, at least one DUI SEI message associated with the DUs of an AU shall have dui_dpb_output_du_delay_present_flag equal to 1.

dui_dpb_output_du_delay is used to compute the DPB output time of the AU when DecodingUnitHrdFlag is equal to 1 and bp_du_dpb_params_in_pic_timing_sei_flag is equal to 0. It specifies how many sub clock ticks to wait after removal of the last DU in an AU from the CPB before the decoded pictures of the AU are output from the DPB. When dui_dpb_output_du_delay is not present and bp_du_dpb_params_in_pic_timing_sei_flag is equal to 0, the value of dui_dpb_output_du_delay is inferred to be equal to dui_dpb_output_du_delay from any DU belonging to the same AU for which dui_dpb_output_du_delay_present_flag is equal to 1. The length of the syntax element dui_dpb_output_du_delay is given in bits by bp_dpb_output_delay_du_length_minus1 + 1.

It is a requirement of bitstream conformance that all DUI SEI messages that are associated with the same AU, apply to the same operation point, and have bp_du_dpb_params_in_pic_timing_sei_flag equal to 0 and dui_dpb_output_du_delay_present_flag equal to 1 shall have the same value of dui_dpb_output_du_delay.

The output time derived from the dui_dpb_output_du_delay of any picture that is output from an output timing conforming decoder shall precede the output time derived from the dui_dpb_output_du_delay of all pictures in any subsequent CVS in decoding order.

The picture output order established by the values of this syntax element shall be the same order as established by the values of PicOrderCntVal.

For pictures that are not output by the "bumping" process because they precede, in decoding order, a CVSS AU that has NoOutputOfPriorPicsFlag equal to 1, the output times derived from dui_dpb_output_du_delay shall be increasing with increasing value of PicOrderCntVal relative to all pictures within the same CVS.

For any two pictures in the CVS, the difference between the output times of the two pictures when DecodingUnitHrdFlag is equal to 1 shall be identical to the same difference when DecodingUnitHrdFlag is equal to 0.

D.6 Scalable nesting SEI message

D.6.1 Scalable nesting SEI message syntax

scalable_nesting(payloadSize) {	**Descriptor**
sn_ols_flag	u(1)
sn_subpic_flag	u(1)
if(sn_ols_flag) {	
sn_num_olss_minus1	ue(v)
for(i = 0; i <= sn_num_olss_minus1; i++)	
sn_ols_idx_delta_minus1[i]	ue(v)
} else {	
sn_all_layers_flag	u(1)
if(!sn_all_layers_flag) {	
sn_num_layers_minus1	ue(v)
for(i = 1; i <= sn_num_layers_minus1; i++)	
sn_layer_id[i]	u(6)
}	
}	
if(sn_subpic_flag) {	
sn_num_subpics_minus1	ue(v)
sn_subpic_id_len_minus1	ue(v)
for(i = 0; i <= sn_num_subpics_minus1; i++)	
sn_subpic_id[i]	u(v)
}	
sn_num_seis_minus1	ue(v)
while(!byte_aligned( ))	
sn_zero_bit /* equal to 0 */	u(1)
for(i = 0; i <= sn_num_seis_minus1; i++)	
sei_message( )	
}	

D.6.2 Scalable nesting SEI message semantics

The scalable nesting SEI message provides a mechanism to associate SEI messages with specific OLSs, specific layers, or specific sets of subpictures.

A scalable nesting SEI message contains one or more SEI messages. The SEI messages contained in the scalable nesting SEI message are also referred to as the scalable-nested SEI messages.

It is a requirement of bitstream conformance that the following restriction applies on the value of the nal_unit_type of the SEI NAL unit containing a scalable nesting SEI message:

– When a scalable nesting SEI message contains an SEI message that has payloadType not equal to 132 (decoded picture hash), the SEI NAL unit containing the scalable nesting SEI message shall have nal_unit_type equal to PREFIX_SEI_NUT.

– When a scalable nesting SEI message contains an SEI message that has payloadType equal to 132 (decoded picture hash), the SEI NAL unit containing the scalable nesting SEI message shall have nal_unit_type equal to SUFFIX_SEI_NUT.

sn_ols_flag equal to 1 specifies that the scalable-nested SEI messages apply to specific OLSs. sn_ols_flag equal to 0 specifies that the scalable-nested SEI messages apply to specific layers.

It is a requirement of bitstream conformance that the following restrictions apply on the value of sn_ols_flag:

– When the scalable nesting SEI message contains an SEI message that has payloadType equal to 0 (BP), 1 (PT), 130 (DUI), or 203 (SLI), the value of sn_ols_flag shall be equal to 1.

– When the scalable nesting SEI message contains an SEI message that has payloadType equal to a value in VclAssociatedSeiList, the value of sn_ols_flag shall be equal to 0.

sn_subpic_flag equal to 1 specifies that the scalable-nested SEI messages that apply to specified OLSs or layers apply only to specific subpictures of the specified OLSs or layers. sn_subpic_flag equal to 0 specifies that the scalable-nested SEI messages that apply to specific OLSs or layers apply to all subpictures of the specified OLSs or layers.

It is a requirement of bitstream conformance that the following restrictions apply on the value of sn_subpic_flag:

– When the scalable nesting SEI message contains an SEI message that has payloadType equal to 132 (decoded picture hash), the value of sn_subpic_flag shall be equal to 1.

– When the scalable nesting SEI message contains an SEI message that has payloadType equal to 203 (SLI), the value of sn_subpic_flag shall be equal to 0.

sn_num_olss_minus1 plus 1 specifies the number of OLSs to which the scalable-nested SEI messages apply. The value of sn_num_olss_minus1 shall be in the range of 0 to TotalNumOlss − 1, inclusive.

sn_ols_idx_delta_minus1[i] is used to derive the variable NestingOlsIdx[i] that specifies the OLS index of the i-th OLS to which the scalable-nested SEI messages apply when sn_ols_flag is equal to 1. The value of sn_ols_idx_delta_minus1[i] shall be in the range of 0 to TotalNumOlss − 2, inclusive.

The variable NestingOlsIdx[i] is derived as follows:

```
if( i = = 0 )
    NestingOlsIdx[ i ] = sn_ols_idx_delta_minus1[ i ]                                    (1629)
else
    NestingOlsIdx[ i ] = NestingOlsIdx[ i − 1 ] + sn_ols_idx_delta_minus1[ i ] + 1
```

sn_all_layers_flag equal to 1 specifies that the scalable-nested SEI messages apply to all layers that have nuh_layer_id greater than or equal to the nuh_layer_id of the current SEI NAL unit. sn_all_layers_flag equal to 0 specifies that the scalable-nested SEI messages might or might not apply to all layers that have nuh_layer_id greater than or equal to the nuh_layer_id of the current SEI NAL unit.

sn_num_layers_minus1 plus 1 specifies the number of layers to which the scalable-nested SEI messages apply. The value of sn_num_layers_minus1 shall be in the range of 0 to vps_max_layers_minus1 − GeneralLayerIdx[nuh_layer_id], inclusive, where nuh_layer_id is the nuh_layer_id of the current SEI NAL unit.

sn_layer_id[i] specifies the nuh_layer_id value of the i-th layer to which the scalable-nested SEI messages apply when sn_all_layers_flag is equal to 0. The value of sn_layer_id[i] shall be greater than nuh_layer_id, where nuh_layer_id is the nuh_layer_id of the current SEI NAL unit.

When sn_ols_flag is equal to 0, the variable nestingNumLayers, specifying the nubmer of layer to which the scalable-nested SEI messages apply, and the list NestingLayerId[i] for i in the range of 0 to nestingNumLayers − 1, inclusive, specifying the list of nuh_layer_id value of the layers to which the scalable-nested SEI messages apply, are derived as follows, where nuh_layer_id is the nuh_layer_id of the current SEI NAL unit:

```
if( sn_all_layers_flag ) {
    nestingNumLayers = vps_max_layers_minus1 + 1 − GeneralLayerIdx[ nuh_layer_id ]
    for( i = 0; i < nestingNumLayers; i ++)
        NestingLayerId[ i ] = vps_layer_id[ GeneralLayerIdx[ nuh_layer_id ] + i ]        (1630)
} else {
```

```
        nestingNumLayers = sn_num_layers_minus1 + 1
        for( i = 0; i < nestingNumLayers; i ++)
            NestingLayerId[ i ] = ( i  = = 0 ) ? nuh_layer_id : sn_layer_id[ i ]
    }
```

The layers that are referrred to as the multiSubpicLayers are defined as follows:

– If sn_ols_flag is equal to 1, the layers that are referrred to as the multiSubpicLayers are the layers in the OLSs to which the scalable-nested SEI messages apply for which the referenced SPSs have sps_num_subpics_minus1 greater than 0.

– Otherwise (sn_ols_flag is equal to 0), the layers that are referrred to as the multiSubpicLayers are the layers to which the scalable-nested SEI messages apply for which the referenced SPSs have sps_num_subpics_minus1 greater than 0.

It is a requirement of bitstream conformance that the value of sps_num_subpics_minus1 shall be the same in all SPSs referenced by pictures in the multiSubpicLayers.

sn_num_subpics_minus1 plus 1 specifies the number of subpictures in each picture in the multiSubpicLayers. The value of sn_num_subpics_minus1 shall be less than or equal to the value of sps_num_subpics_minus1 in the SPSs referred to by the pictures in the multiSubpicLayers.

sn_subpic_id_len_minus1 plus 1 specifies the number of bits used to represent the syntax element sn_subpic_id[i]. The value of sn_subpic_id_len_minus1 shall be in the range of 0 to 15, inclusive.

It is a requirement of bitstream conformance that the value of sn_subpic_id_len_minus1 shall be the same for all scalable nesting SEI messages that are associated with pictures in a CVS.

sn_subpic_id[i] indicates the subpicture ID of the i-th subpicture in each picture in the multiSubpicLayers. The length of the sn_subpic_id[i] syntax element is sn_subpic_id_len_minus1 + 1 bits. The scalable-nested SEI messages also apply to the single subpicture in each picture in the layers that are not in the multiSubpicLayers, but are among the layers in the OLSs (when sn_ols_flag is equal to 1) to which the scalable-nested SEI messages apply, or among the layers (when sn_ols_flag is equal to 0) to which the scalable-nested SEI messages apply.

sn_num_seis_minus1 plus 1 specifies the number of scalable-nested SEI messages. The value of sn_num_seis_minus1 shall be in the range of 0 to 63, inclusive.

sn_zero_bit shall be equal to 0.

D.7 Subpicture level information SEI message

D.7.1 Subpicture level information SEI message syntax

subpic_level_info(payloadSize) {	**Descriptor**
sli_num_ref_levels_minus1	u(3)
sli_cbr_constraint_flag	u(1)
sli_explicit_fraction_present_flag	u(1)
if(sli_explicit_fraction_present_flag)	
sli_num_subpics_minus1	ue(v)
sli_max_sublayers_minus1	u(3)
sli_sublayer_info_present_flag	u(1)
while(!byte_aligned( ))	
sli_alignment_zero_bit	f(1)
for(k = sli_sublayer_info_present_flag ? 0 : sli_max_sublayers_minus1; k <= sli_max_sublayers_minus1; k++)	
for(i = 0; i <= sli_num_ref_levels_minus1; i++) {	
sli_non_subpic_layers_fraction[i][k]	u(8)
sli_ref_level_idc[i][k]	u(8)
if(sli_explicit_fraction_present_flag)	
for(j = 0; j <= sli_num_subpics_minus1; j++)	

sli_ref_level_fraction_minus1[i][j][k]	u(8)
}	
}	

D.7.2 Subpicture level information SEI message semantics

The subpicture level information (SLI) SEI message contains information about the level that subpicture sequences in the set of CVSs of the OLSs to which the SEI message applies, denoted as targetCvss, conform to when testing the conformance of the extracted bitstreams containing the subpicture sequences according to Annex A. The OLSs to which the SLI message applies are also referred to as the applicable OLSs or the associated OLSs. A CVS in the remainder of this clause refers to a CVS of the applicable OLSs.

A subpicture sequence consists of all subpictures within targetCvss that have the same value of subpicture index subpicIdxA and belong to the layers in the multiSubpicLayers and all subpictures within targetCvss that have subpicture index equal to 0 and belong to the layers in the applicable OLSs but not in the multiSubpicLayers. A subpicture sequence is said to be associated with and identified by the subpicture index subpicIdxA.

When an SLI SEI message is present (either being in the bitstream or provided through an external means not specified in this Specification) for any AU of a CVS, an SLI SEI message shall be present for the first AU of the CVS. The SLI SEI message persists in decoding order from the current AU until the next AU containing an SLI SEI message for which the content differs from the current SLI SEI message or the end of the bitstream. All SLI SEI messages that apply to the same CVS shall have the same content.

Among the layers in the applicable OLSs, those for which the referenced SPSs have sps_num_subpics_minus1 greater than 0 are referred to as the multiSubpicLayers.

It is a requirement of bitstream conformance that, when an SLI SEI message applicable to an OLS is present for a CVS, for all the SPSs referenced by the pictures in the multiSubpicLayers in the OLS, the value of sps_num_subpics_minus1 shall be the same and the value of sps_subpic_treated_as_pic_flag[i] shall be equal to 1 for each value of i in the range of 0 to sps_num_subpics_minus1, inclusive.

sli_num_ref_levels_minus1 plus 1 specifies the number of reference levels signalled for each of the sli_num_subpics_minus1 + 1 subpicture sequences.

sli_cbr_constraint_flag equal to 0 specifies that to decode the sub-bitstreams resulting from extraction of any subpicture sequence according to clause C.7 by using the HRD using any CPB specification in the extracted sub-bitstream, the hypothetical stream scheduler (HSS) operates in an intermittent bit rate mode. sli_cbr_constraint_flag equal to 1 specifies that the HSS operates in a constant bit rate (CBR) mode in such a case.

sli_explicit_fraction_present_flag equal to 1 specifies that the syntax elements sli_ref_level_fraction_minus1[i] are present. sli_explicit_fraction_present_flag equal to 0 specifies that the syntax elements sli_ref_level_fraction_minus1[i] are not present.

sli_num_subpics_minus1 plus 1 specifies the number of subpictures in the pictures in the multiSubpicLayers in targetCvss. When present, the value of sli_num_subpics_minus1 shall be equal to the value of sps_num_subpics_minus1 in the SPSs referenced by the pictures in the multiSubpicLayers in targetCvss.

sli_max_sublayers_minus1 plus 1 specifies the maximum number of temporal sublayers in the subpicture sequences for which the level information is indicated in the SLI SEI message. The value of sli_max_sublayers_minus1 shall be equal to vps_max_sublayers_minus1.

sli_sublayer_info_present_flag equal to 1 specifies that the level information for subpicture sequences is present for sublayer representation(s) in the range of 0 to sli_max_sublayers_minus1, inclusive. sli_sublayer_info_present_flag equal to 0 specifies that the level information for subpicture sequences is present for the sli_max_sublayers_minus1-th sublayer representation. When not present, the value of sli_sublayer_info_present_flag is inferred to be equal to 0.

sli_alignment_zero_bit shall be equal to 0.

sli_non_subpic_layers_fraction[i][k] indicates the i-th fraction of the bitstream level limits associated with layers in targetCvss that have sps_num_subpics_minus1 equal to 0 when Htid is equal to k. When vps_max_layers_minus1 is equal to 0 or when no layer in the bitstream has sps_num_subpics_minus1 equal to 0, sli_non_subpic_layers_fraction[i][k] shall be equal to 0. When k is less than sli_max_sublayers_minus1 and sli_non_subpic_layers_fraction[i][k] is not present, it is inferred to be equal to sli_non_subpic_layers_fraction[i][k + 1].

sli_ref_level_idc[i][k] indicates the i-th level to which each subpicture sequence conforms as specified in Annex A when Htid is equal to k. Bitstreams shall not contain values of sli_ref_level_idc[i][k] other than those specified in Annex A. Other values of sli_ref_level_idc[i][k] are reserved for future use by ITU-T | ISO/IEC. It is a requirement of bitstream conformance that the value of sli_ref_level_idc[0][k] shall be equal to the value of general_level_idc of the bitstream and that the value of sli_ref_level_idc[i][k] shall be less than or equal to sli_ref_level_idc[m][k] for any value of i greater than 0 and m greater than i. When k is less than sli_max_sublayers_minus1 and sli_ref_level_idc[i][k] is not present, it is inferred to be equal to sli_ref_level_idc[i][k + 1].

sli_ref_level_fraction_minus1[i][j][k] plus 1 specifies the i-th fraction of the level limits, associated with sli_ref_level_idc[i][k], for the subpictures with subpicture index equal to j in layers in targetCvss that have sps_num_subpics_minus1 greater than 0 when Htid is equal to k. When k is less than sli_max_sublayers_minus1 and sli_ref_level_fraction_minus1[i][j][k] is not present, it is inferred to be equal to sli_ref_level_fraction_minus1[i][j][k + 1].

The variable SubpicSizeY[j] is set equal to (sps_subpic_width_minus1[j] + 1) * CtbSizeY * (sps_subpic_height_minus1[j] + 1) * CtbSizeY of the layers in the multiSubpicLayers.

When not present, the value of sli_ref_level_fraction_minus1[i][j][sli_max_sublayers_minus1] is inferred to be equal to Max(256, Ceil(256 * SubpicSizeY[j] ÷ PicSizeMaxInSamplesY * MaxLumaPs(general_level_idc) ÷ MaxLumaPs(sli_ref_level_idc[i][sli_max_sublayers_minus1])) − 1, where PicSizeMaxInSamplesY is the value of PicSizeMaxInSamplesY for the layers in the multiSubpicLayers.

The variable LayerRefLevelFraction[i][j][k] is set equal to sli_ref_level_fraction_minus1[i][j][k] + 1.

The variable OlsRefLevelFraction[i][j][k] is set equal to sli_non_subpic_layers_fraction[i][k] + (256 − sli_non_subpic_layers_fraction[i][k]) ÷ 256 * (sli_ref_level_fraction_minus1[i][j][k] + 1).

The variables SubpicCpbSizeVcl[i][j][k] and SubpicCpbSizeNal[i][j][k] are derived as follows:

$$\text{SubpicCpbSizeVcl[i][j][k]} = \text{Floor(CpbVclFactor * MaxCPB * OlsRefLevelFraction[i][j][k]} \div 256\text{)} \quad (1631)$$

$$\text{SubpicCpbSizeNal[i][j][k]} = \text{Floor(CpbNalFactor * MaxCPB * OlsRefLevelFraction[i][j][k]} \div 256\text{)} \quad (1632)$$

with MaxCPB derived from sli_ref_level_idc[i][k] as specified in clause A.4.2.

The variables SubpicBitRateVcl[i][j][k] and SubpicBitRateNal[i][j][k] are derived as follows:

$$\text{SubpicBitRateVcl[i][j][k]} = \text{Floor(CpbVclFactor * ValBR * OlsRefLevelFraction[0][j][k]} \div 256\text{)} \quad (1633)$$

$$\text{SubpicBitRateNal[i][j][k]} = \text{Floor(CpbNalFactor * ValBR * OlsRefLevelFraction[0][j][k]} \div 256\text{)} \quad (1634)$$

Where the value of ValBR is derived as follows:

– When bit_rate_value_minus1[k][ScIdx] is available in the respective HRD parameters in the VPS or SPS, ValBR is set equal to (bit_rate_value_minus1[k][ScIdx] + 1) * $2^{(6 + \text{bit\_rate\_scale})}$, where Htid is the considered sublayer index and ScIdx is the considered schedule index.

– Otherwise, ValBR is set equal to MaxBR derived from sli_ref_level_idc[0][k] as specified in clause A.4.2.

NOTE 1 – When a subpicture is extracted, the resulting bitstream has a CpbSize (either indicated in the VPS, SPS, or inferred) that is greater than or equal to SubpicCpbSizeVcl[i][j][k] and SubpicCpbSizeNal[i][j][k] and a BitRate (either indicated in the VPS, SPS, or inferred) that is greater than or equal to SubpicBitRateVcl[i][j][k] and SubpicBitRateNal[i][j][k].

It is a requirement of bitstream conformance that, for each value of k in the range of sli_max_sublayers_minus1, inclusive, each layer in the bitstream resulting from extracting the j-th subpicture sequence for j in the range of 0 to sli_num_subpics_minus1, inclusive, from a layer that had sps_num_subpics_minus1 greater than 0 in the input bitstream to the extraction process, and conforming to a profile with general_tier_flag equal to 0 and level equal to sli_ref_level_idc[i][k] for i in the range of 0 to sli_num_ref_levels_minus1, inclusive, shall obey the following constraints for each bitstream conformance test as specified in Annex C:

– Ceil(256 * SubpicSizeY[j] ÷ LayerRefLevelFraction[i][j][k]) shall be less than or equal to MaxLumaPs, where MaxLumaPs is specified in Table A.2 for level sli_ref_level_idc[i][k].

– The value of Ceil(256 * (sps_subpic_width_minus1[j] + 1) * CtbSizeY ÷ LayerRefLevelFraction[i][j][k]) shall be less than or equal to Sqrt(MaxLumaPs * 8).

– The value of Ceil(256 * (sps_subpic_height_minus1[j] + 1) * CtbSizeY ÷ LayerRefLevelFraction[i][j][k]) shall be less than or equal to Sqrt(MaxLumaPs * 8).

– The value of SubpicWidthInTiles[j] shall be less than or equal to MaxTileCols and the value of SubpicHeightInTiles[j] shall be less than or equal to MaxTilesPerAu / MaxTilesCols, where MaxTilesPerAu and MaxTileCols are specified in Table A.2 for level sli_ref_level_idc[i][k].

– The value of SubpicWidthInTiles[j] * SubpicHeightInTiles[j] shall be less than or equal to MaxTilesPerAu * LayerRefLevelFraction[i][j][k], where MaxTilesPerAu are specified in Table A.2 for level sli_ref_level_idc[i][k].

It is a requirement of bitstream conformance that, when Htid is equal to k, the bitstream resulting from extracting the j-th subpicture sequence for j in the range of 0 to sli_num_subpics_minus1, inclusive, and conforming to a profile with general_tier_flag equal to 0 and level equal to sli_ref_level_idc[i][k] for i in the range of 0 to sli_num_ref_levels_minus1, inclusive, shall obey the following constraints for each bitstream conformance test as specified in Annex C:

– The sum of the NumBytesInNalUnit variables for AU 0 corresponding to the j-th subpicture sequence shall be less than or equal to FormatCapabilityFactor * (Max(AuSizeMaxInSamplesY[0], FrVal * MaxLumaSr) + MaxLumaSr * (AuCpbRemovalTime[0] − AuNominalRemovalTime[0])) * OlsRefLevelFraction[i][j][k]) ÷ (256 * MinCr), where MaxLumaSr and FormatCapabilityFactor are the values specified in Table A.3 and Table A.4, respectively, that apply to AU 0, at level sli_ref_level_idc[i][k], and MinCr, AuSizeMaxInSamplesY[0] and FrVal are derived as specified in A.4.2.

– The sum of the NumBytesInNalUnit variables for AU n (with n greater than 0) corresponding to the j-th subpicture sequence shall be less than or equal to FormatCapabilityFactor * MaxLumaSr * (AuCpbRemovalTime[n] − AuCpbRemovalTime[n − 1]) * OlsRefLevelFraction[i][j][k] ÷ (256 * MinCr), where MaxLumaSr and FormatCapabilityFactor are the values specified in Table A.3 and Table A.4 respectively, that apply to AU n, at level sli_ref_level_idc[i][k], and MinCr is derived as indicated in A.4.2.

The value of the subpicture sequence level indicator, SubpicLevelIdc[j][k], for the i-th subpicture sequence when Htid is equal to k, is derived as follows:

```
for( j = 0; j < sli_num_subpics_minus1; j++ ) {
    SubpicLevelIdc[ j ][ k ] = general_level_idc
    SubpicLevelIdx[ j ][ k ] = 0
    for ( i = sli_num_ref_levels_minus1; i  >=  1; i− − )
        if( OlsRefLevelFraction[ i ][ j ][ k ]  <=  256 ) {                    (1635)
            SubpicLevelIdc[ j ][ k ] = sli_ref_level_idc[ i ][ k ]
            SubpicLevelIdx[ j ][ k ] = i
        }
}
```

The j-th subpicture sequence conforming to a profile with general_tier_flag equal to 0 and a level equal to SubpicLevelIdc[j][k] shall obey the following constraints for each bitstream conformance test as specified in Annex C when Htid is equal to k, and the variable spLvIdx is equal to SubpicLevelIdx[j][k]:

– For the VCL HRD parameters, SubpicCpbSizeVcl[spLvIdx][j][k] shall be less than or equal to CpbVclFactor * MaxCPB, where CpbVclFactor is specified in Table A.4 and MaxCPB is specified in Table A.2 in units of CpbVclFactor bits.

– For the NAL HRD parameters, SubpicCpbSizeNal[spLvIdx][j][k] shall be less than or equal to CpbNalFactor * MaxCPB, where CpbNalFactor is specified in Table A.4, and MaxCPB is specified in Table A.2 in units of CpbNalFactor bits.

– For the VCL HRD parameters, SubpicBitRateVcl[spLvIdx][j][k] shall be less than or equal to CpbVclFactor * MaxBR, where CpbVclFactor is specified in Table A.4 and MaxBR is specified in Table A.2 in units of CpbVclFactor bits.

– For the NAL HRD parameters, SubpicBitRateNal[spLvIdx][j][k] shall be less than or equal to CpbNalFactor * MaxBR, where CpbNalFactor is specified in Table A.4, and MaxBR is specified in Table A.2 in units of CpbNalFactor bits.

NOTE 2 – When the j-th subpicture sequence is extracted with tIdTarget equal to k, the resulting bitstream has a CPB size (either indicated in the VPS, SPS, or inferred) that is greater than or equal to SubpicCpbSizeVcl[spLvIdx][j][k] and SubpicCpbSizeNal[spLvIdx][j][k] and a bit rate (either indicated in the VPS, SPS, or inferred) that is greater than or equal to SubpicBitRateVcl[spLvIdx][j][k] and SubpicBitRateNal[spLvIdx][j][k].

D.8 SEI manifest SEI message

D.8.1 SEI manifest SEI message syntax

sei_manifest(payloadSize) {	**Descriptor**
manifest_num_sei_msg_types	u(16)
for(i = 0; i < manifest_num_sei_msg_types; i++) {	
manifest_sei_payload_type[i]	u(16)
manifest_sei_description[i]	u(8)
}	
}	

D.8.2 SEI manifest SEI message semantics

The SEI manifest SEI message conveys information on SEI messages that are indicated as expected (i.e., likely) to be present or not present. Such information may include the following:

– The indication that certain types of SEI messages are expected (i.e., likely) to be present (although not guaranteed to be present) in the CVS.

– For each type of SEI message that is indicated as expected (i.e., likely) to be present in the CVS, the degree of expressed necessity of interpretation of the SEI messages of this type, as follows:

  – The degree of necessity of interpretation of an SEI message type may be indicated as "necessary", "unnecessary", or "undetermined".

  – An SEI message is indicated by the encoder (i.e., the content producer) as being "necessary" when the information conveyed by the SEI message is considered as necessary for interpretation by the decoder or receiving system in order to properly process the content and enable an adequate user experience; it does not mean that the bitstream is required to contain the SEI message in order to be a conforming bitstream. It is at the discretion of the encoder to determine which SEI messages are to be considered as necessary in a particular CVS. However, it is suggested that some SEI messages, such as the frame packing arrangement, segmented rectangular frame packing arrangement, and omnidirectional projection indication SEI messages, should typically be considered as necessary.

– The indication that certain types of SEI messages are expected (i.e., likely) not to be present (although not guaranteed not to be present) in the CVS.

NOTE – An example of such a usage of an SEI manifest SEI message is to express the expectation that there are no frame packing arrangement SEI messages or omnidirectional projection indication SEI messages in the CVS, and therefore that the rendering of the decoded video pictures for display purposes would not need any of the additional post-processing that is commonly associated with the interpretation of these SEI messages.

The content of an SEI manifest SEI message may, for example, be used by transport-layer or systems-layer processing elements to determine whether the CVS is suitable for delivery to a receiving and decoding system, based on whether the receiving system can properly process the CVS to enable an adequate user experience or whether the CVS satisfies the application needs.

When an SEI manifest SEI message is present in any access unit of a CVS, an SEI manifest SEI message shall be present in the first access unit of the CVS. The SEI manifest SEI message persists in decoding order from the current access unit until the end of the CVS. When there are multiple SEI manifest SEI messages present in a CVS, they shall have the same content.

An SEI NAL unit containing an SEI manifest SEI message shall not contain any other SEI messages other than SEI prefix indication SEI messages. When present in an SEI NAL unit, the SEI manifest SEI message shall be the first SEI message in the SEI NAL unit.

manifest_num_sei_msg_types specifies the number of types of SEI messages for which information is provided in the SEI manifest SEI message.

manifest_sei_payload_type[i] indicates the payloadType value of the i-th type of SEI message for which information is provided in the SEI manifest SEI message. The values of manifest_sei_payload_type[m] and manifest_sei_payload_type[n] shall not be identical when m is not equal to n.

manifest_sei_description[i] provides information on SEI messages with payloadType equal to manifest_sei_payload_type[i] as specified in Table D.2.

Table D.2 – Interpretation of manifest_sei_description[i]

Value	Description
0	Indicates that there is no SEI message with payloadType equal to manifest_sei_payload_type[i] expected to be present in the CVS.
1	Indicates that there are SEI messages with payloadType equal to manifest_sei_payload_type[i] expected to be present in the CVS, and these SEI messages are considered as necessary.
2	Indicates that there are SEI messages with payloadType equal to manifest_sei_payload_type[i] expected to be present in the CVS, and these SEI messages are considered as unnecessary.
3	Indicates that there are SEI messages with payloadType equal to manifest_sei_payload_type[i] expected to be present in the CVS, and the necessity of these SEI messages is undetermined.
4..255	Reserved

The value of manifest_sei_description[i] shall be in the range of 0 to 3, inclusive, in bitstreams conforming to this version of this Specification. Other values for manifest_sei_description[i] are reserved for future use by ITU-T | ISO/IEC. Decoders shall also allow the value of manifest_sei_description[i] greater than or equal to 4 to appear in the syntax and shall ignore all information for payloadType equal to manifest_sei_payload_type[i] signalled in the SEI manifest SEI message and shall ignore all SEI prefix indication SEI messages with prefix_sei_payload_type equal to manifest_sei_payload_type[i] when manifest_sei_description[i] is greater than or equal to 4.

D.9 SEI prefix indication SEI message

D.9.1 SEI prefix indication SEI message syntax

sei_prefix_indication(payloadSize) {	**Descriptor**
prefix_sei_payload_type	u(16)
num_sei_prefix_indications_minus1	u(8)
for(i = 0; i <= num_sei_prefix_indications_minus1; i++) {	
num_bits_in_prefix_indication_minus1[i]	u(16)
for(j = 0; j <= num_bits_in_prefix_indication_minus1[i]; j++)	
sei_prefix_data_bit[i][j]	u(1)
while(!byte_aligned( ))	
byte_alignment_bit_equal_to_one /* equal to 1 */	f(1)
}	
}	

D.9.2 SEI prefix indication SEI message semantics

The SEI prefix indication SEI message carries one or more SEI prefix indications for SEI messages of a particular value of payloadType. Each SEI prefix indication is a bit string that follows the SEI payload syntax of that value of payloadType and contains a number of complete syntax elements starting from the first syntax element in the SEI payload.

Each SEI prefix indication for an SEI message of a particular value of payloadType indicates that one or more SEI messages of this value of payloadType are expected (i.e., likely) to be present in the CVS and to start with the provided bit string. A starting bit string would typically contain only a true subset of an SEI payload of the type of SEI message indicated by the payloadType, may contain a complete SEI payload, and shall not contain more than a complete SEI payload. It is not prohibited for SEI messages of the indicated value of payloadType to be present that do not start with any of the indicated bit strings.

These SEI prefix indications should provide sufficient information for indicating what type of processing is needed or what type of content is included. The former (type of processing) indicates decoder-side processing capability, e.g., whether some type of frame unpacking is needed. The latter (type of content) indicates, for example, whether the bitstream contains subtitle captions in a particular language.

The content of an SEI prefix indication SEI message may, for example, be used by transport-layer or systems-layer processing elements to determine whether the CVS is suitable for delivery to a receiving and decoding system, based on whether the receiving system can properly process the CVS to enable an adequate user experience or whether the CVS satisfies the application needs (as determined in some manner by external means outside the scope of this Specification).

In one example, when the payloadType indicates the frame packing arrangement SEI message, an SEI prefix indication should include up to at least the syntax element frame_packing_arrangement_type; and when the payloadType indicates the omnidirectional projection indication SEI message, an SEI prefix indication should include up to at least the syntax element projection_type.

In another example, for user data registered SEI messages that are used to carry captioning information, an SEI prefix indication should include up to at least the language code; and for user data unregistered SEI messages extended for private use, an SEI prefix indication should include up to at least the universally unique identifier (UUID).

When an SEI prefix indication SEI message is present in any access unit of a CVS, an SEI prefix indication SEI message shall be present in the first access unit of the CVS. The SEI prefix indication SEI message persists in decoding order from the current access unit until the end of the CVS. When there are multiple SEI prefix indication SEI messages present in a CVS for a particular value of payloadType, they shall have the same content.

An SEI NAL unit containing an SEI prefix indication SEI message for a particular value of payloadType shall not contain any other SEI messages other than an SEI manifest SEI message and SEI prefix indication SEI messages for other values of payloadType.

prefix_sei_payload_type indicates the payloadType value of the SEI messages for which one or more SEI prefix indications are provided in the SEI prefix indication SEI message. When an SEI manifest SEI message is also present for the CVS, the value of prefix_sei_payload_type shall be equal to one of the manifest_sei_payload_type[m] values for which manifest_sei_description[m] is equal to 1 to 3, inclusive, as indicated by an SEI manifest SEI message that applies to the CVS.

num_sei_prefix_indications_minus1 plus 1 specifies the number of SEI prefix indications.

num_bits_in_prefix_indication_minus1[i] plus 1 specifies the number of bits in the i-th SEI prefix indication.

sei_prefix_data_bit[i][j] specifies the j-th bit of the i-th SEI prefix indication.

The bits sei_prefix_data_bit[i][j] for j ranging from 0 to num_bits_in_prefix_indication_minus1[i], inclusive, follow the syntax of the SEI payload with payloadType equal to prefix_sei_payload_type, and contain a number of complete syntax elements starting from the first syntax element in the SEI payload syntax, and may or may not contain all the syntax elements in the SEI payload syntax. The last bit of these bits (i.e., the bit sei_prefix_data_bit[i][num_bits_in_prefix_indication_minus1[i]]) shall be the last bit of a syntax element in the SEI payload syntax, unless it is a bit within an itu_t_t35_payload_byte or user_data_payload_byte.

NOTE – The exception for itu_t_t35_payload_byte and user_data_payload_byte is provided because these syntax elements may contain externally-specified syntax elements, and the determination of the boundaries of such externally-specified syntax elements is a matter outside the scope of this Specification.

byte_alignment_bit_equal_to_one shall be equal to 1.

D.10 Constrained RASL encoding indication SEI message

D.10.1 Constrained RASL encoding indication SEI message syntax

constrained_rasl_encoding_indication(payloadSize) {	**Descriptor**
}	

D.10.2 Constrained RASL encoding indication SEI message semantics

The presence of the constrained RASL encoding indication (CREI) SEI message in a CVS indicates that a set of encoding constraints as described below applies for all RASL pictures in the CVS.

NOTE 1 – In some applications such as bit-rate adaptive streaming services, when this set of encoding constraints is applied, it is expected to be possible to perform bitstream switching between different bitstreams that represent the same source video content at CRA pictures that have associated RASL pictures with fewer visually noticeable or visually annoying artefacts in the reconstructed sample values of the RASL pictures than when the RASL pictures are encoded without applying this set of constraints.

NOTE 2 – CRA pictures with associated RASL pictures are sometimes referred to as open group of pictures (open-GOP) IRAP pictures.

When a CREI SEI message is present for any picture of an AU of a CVS, a CREI SEI message shall be present for the first picture of the CVSS AU. The CREI SEI message persists in decoding order from the current AU until the end of the CVS.

The presence of the CREI SEI message indicates that the following conditions all apply for each RASL picture in the CVS:

– The PH syntax structure has ph_dmvr_disabled_flag equal to 1.

– The PPS referred to by the RASL picture has pps_ref_wraparound_enabled_flag equal to 0.

– No CU in a slice with sh_slice_type equal to 0 (B) or 1 (P) has cclm_mode_flag equal to 1.

– No collocated reference picture precedes the CRA picture associated with the RASL picture in decoding order.

D.11 Use of ITU-T H.274 | ISO/IEC 23002-7 VUI parameters

The VUI parameters specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 may be used together with bitstreams specified by this Specification.

When present, the vui_parameters( ) syntax structure as specified in Rec. ITU-T H.274 | ISO/IEC 23002-7 is included in the vui_payload( ) syntax structure specified in clause 7.3.2.21, which can be included in the SPS syntax structure as specified in clause 7.3.2.4.

The value of PayloadBits, as specified in clause 7.4.3.21, is passed to the parser of the vui_parameters( ) syntax structure specified in Rec. ITU-T H.274 | ISO/IEC 23002-7.

When the SPS does not contain a vui_payload( ) syntax structure (i.e sps_vui_parameters_present_flag is equal to 0), the video usability information is inferred as follows unless determined by the application by external means:

– The values of vui_progressive_source_flag and vui_interlaced_source_flag are both inferred to be equal to 0 (source scan type interpreted as unknown or unspecified or specified by external means).

– The values of vui_non_packed_constraint_flag and vui_non_projected_constraint_flag are both inferred to be equal to 0 (no constraint imposed).

– The value of vui_aspect_ratio_constant_flag is inferred to be equal to 0 (no constraint imposed).

– The value of vui_aspect_ratio_idc is inferred to be equal to 0 (unknown or unspecified or specified by external means).

– The value of vui_overscan_info_present_flag is inferred to be equal to 0 (preferred display method for the video signal is unknown or unspecified or specified by external means).

– The value of vui_colour_primaries is inferred to be equal to 2 (unknown or unspecified or specified by external means).

– The value of vui_transfer_characteristics is inferred to be equal to 2 (unknown or unspecified or specified by external means).

– The value of vui_matrix_coeffs is inferred to be equal to 2 (unknown or unspecified or specified by external means).

– The value of vui_full_range_flag is inferred to be equal to 0 (a value that has no effect when vui_matrix_coeffs is equal to 2).

– The value of vui_chroma_sample_loc_type_frame, vui_chroma_sample_loc_type_top_field and vui_chroma_sample_loc_type_bottom_field, as applicable, are inferred to be equal to 6 (unknown or unspecified or specified by external means).

When the value of vui_chroma_sample_loc_type_frame, vui_chroma_sample_loc_type_top_field and vui_chroma_sample_loc_type_bottom_field, as applicable, are equal to 6 or are inferred to be equal to 6, the nominal vertical and horizontal relative locations of luma and chroma samples in pictures as shown in Figure 1 may be assumed.

D.12 Use of SEI messages specified in other specifications

D.12.1 General

The SEI messages having syntax structures identified in clause D.2.1 that are specified in other specifications, including Rec. ITU-T H.274 | ISO/IEC 23002-7, ISO/IEC 23001-11, or ISO/IEC 23090-13, may be used together with bitstreams specified by this Specification.

When any particular SEI message specified in Rec. ITU-T H.274 | ISO/IEC 23002-7, ISO/IEC 23001-11, or ISO/IEC 23090-13 is included in a bitstream specified by this Specification, the SEI payload syntax shall be as specified in Rec. ITU-T H.274 | ISO/IEC 23002-7, ISO/IEC 23001-11, or ISO/IEC 23090-13, respectively, that syntax shall be included into the sei_payload( ) syntax structure as specified in clause D.2.1 and shall use the payloadType value specified in clause D.2.1, the corresponding semantics specified in Rec. ITU-T H.274 | ISO/IEC 23002-7, ISO/IEC 23001-11, or ISO/IEC 23090-13 shall apply, and, additionally, any SEI-message-specific constraints, variables, and semantics specified in this annex for that particular SEI message shall apply.

The value of PayloadBits, as specified in clause D.2.2, is passed to the parser of the SEI message syntax structures specified in Rec. ITU-T H.274 | ISO/IEC 23002-7, ISO/IEC 23001-11, and ISO/IEC 23090-13.

NOTE – The definition of IRAP picture in the VSEI specification is as follows: A coded picture starting from which all pictures in the same layer in both decoding order and output order can be decoded without first decoding any picture in the same layer earlier in decoding order in the coded video bitstream. Consequently, a GDR picture with ph_recovery_poc_cnt equal to 0 in a VVC bitsream is an IRAP picture according to the IRAP picture definition in the VSEI specification.

D.12.2 Use of the film grain characteristics SEI message

For purposes of interpretation of the film grain characteristics SEI message, the following variables are specified:

– PicWidthInLumaSamples and PicHeightInLumaSamples are set equal to pps_pic_width_in_luma_samples and pps_pic_height_in_luma_samples, respectively.

– ChromaFormatIdc is set equal to sps_chroma_format_idc.

– $BitDepth_Y$ and $BitDepth_C$ are both set equal to BitDepth.

D.12.3 Use of the decoded picture hash SEI message

For purposes of interpretation of the decoded picture hash SEI message, the following variables are specified:

– PicWidthInLumaSamples and PicHeightInLumaSamples are set equal to pps_pic_width_in_luma_samples and pps_pic_height_in_luma_samples, respectively.

– ChromaFormatIdc is set equal to sps_chroma_format_idc.

– $BitDepth_Y$ and $BitDepth_C$ are both set equal to BitDepth.

– ComponentSample[cIdx] is set to be the 2-dimension array of decoded sample values of the cIdx-th component of a decoded picture.

D.12.4 Use of the dependent random access point (DRAP) indication SEI message

A picture that is associated with a DRAP indication SEI message is referred to as a DRAP picture.

The following constraints apply to a DRAP picture:

– The VCL NAL units of the DRAP picture shall have nal_unit_type equal to TRAIL_NUT.

– The DRAP picture shall have TemporalId equal to 0.

D.12.5 Use of the equirectangular projection, generalized cubemap projection, and region-wise packing SEI messages

For purposes of interpretation of the equirectangular projection, generalized cubemap projection, and region-wise packing SEI message, the following variable is specified:

– ChromaFormatIdc is set equal to sps_chroma_format_idc.

D.12.6 Use of the frame-field information SEI message

For purposes of interpretation of the frame-field information SEI message, the variable FixedPicRateWithinCvsFlag is set equal to fixed_pic_rate_within_cvs_flag[Htid].

When vui_progressive_source_flag and vui_interlaced_source_flag in the vui_parameters( ) syntax structure are both equal to 1, for each picture associated with the vui_parameters( ) syntax structure, a frame-field information SEI message associated with the picture shall be present.

When a frame-field information SEI message is present, the following constraints apply:

– The value of ffi_field_pic_flag shall be equal to sps_field_seq_flag.
– When a PT SEI message is present for picture n, the variable elementalOutputPeriods is set equal to the value of pt_display_elemental_periods_minus1 + 1.
– When FixedPicRateWithinCvsFlag is equal to 1, the following applies:
  – The variable displayElementalPeriods is set equal to ffi_display_elemental_periods_minus1 + 1.
  – The value of displayElementalPeriods shall be an integer multiple of elementalOutputPeriods.
  – For each AU, the smallest value of displayElementalPeriods among the frame-field information SEI messages that apply to the output layers of the OLS with OLS index equal to TargetOlsIdx shall be equal to elementalOutputPeriods.
– When vui_progressive_source_flag is equal to 0 or vui_interlaced_source_flag is equal to 0, the value of ffi_source_scan_type shall be constrained as follows:
  – If vui_progressive_source_flag is equal to 0 and vui_interlaced_source_flag is equal to 1, ffi_source_scan_type shall be equal to 0 (interlaced).
  – Otherwise, if vui_progressive_source_flag is equal to 1 and vui_interlaced_source_flag is equal to 0, ffi_source_scan_type shall be equal to 1 (progressive).
  – Otherwise (vui_progressive_source_flag is equal to 0 and vui_interlaced_source_flag is equal to 0), ffi_source_scan_type shall be equal to 2 (unknown or unspecified or specified by external means).

D.12.7 Use of the annotated regions SEI message

For purposes of interpretation of the annotated regions SEI message, the following variables are specified:

– CroppedWidth is set equal to (pps_pic_width_in_luma_samples – SubWidthC * (pps_conf_win_right_offset + pps_conf_win_left_offset)).
– CroppedHeight is set equal to (pps_pic_height_in_luma_samples – SubHeightC * (pps_conf_win_bottom_offset + pps_conf_win_top_offset)).
– ConfWinLeftOffset is set equal to pps_conf_win_left_offset.
– ConfWinTopOffset is set equal to pps_conf_win_top_offset.

D.12.8 Use of the extended dependent random access point (EDRAP) indication SEI message

A picture that is associated with an EDRAP indication SEI message is referred to as an EDRAP picture.

The following constraints apply to an EDRAP picture:

– The VCL NAL units of the EDRAP picture shall have nal_unit_type equal to TRAIL_NUT.
– The EDRAP picture shall have TemporalId equal to 0.

D.12.9 Use of the colour transform information SEI message

For purposes of interpretation of the colour transform information SEI message, the following variable is specified:

– ChromaFormatIdc is set equal to sps_chroma_format_idc.

D.12.10 Use of the shutter interval information SEI message

The following constraints apply to the shutter interval information SEI message:

– When the value of SpsMaxSubLayersMinus1 is equal to 0, the value of fixed_shutter_interval_within_clvs_flag shall be equal to 1.
– The value of sii_max_sub_layers_minus1 shall be equal to the value of SpsMaxSubLayersMinus1.

D.12.11 Use of the neural network post-filter characteristics SEI message and the neural network post-filter activation SEI message

Let currPic be the cropped decoded output picture for which the neural-network post-processing filter (NNPF) defined by the neural-network post-filter characteristics (NNPFC) SEI message is activated by a neural-network post-filter activation (NNPFA) SEI message, and currLayerId be the nuh_layer_id value of currPic.

It is a requirement of bitstream conformance that when a picture unit contains an NNPFA SEI message, the value of ph_pic_output_flag in the picture header contained in that picture unit shall be equal to 1.

NOTE – Since only cropped decoded output pictures are used as input pictures of the NNPF, the value of ph_pic_output_flag in the picture header of the coded picture corresponding to each input picture of the NNPF is equal to 1.

The variable pictureRateUpsamplingFlag is set equal to ((nnpfc_purpose & 0x08) > 0) ? 1 : 0.

The variable numInputPics is set equal to nnpfc_num_input_pics_minus1 + 1.

The variable numInferences is derived as follows:

- If all of the following conditions are true, the variable numPostRoll is set equal to the value of i such that nnpfc_interpolated_pics[i] is greater than 0 and the variable numInferences is set equal to 1 + numPostRoll:
  - nnpfc_purpose is equal to 8 (i.e., the only purpose for the NNPF is picture rate upsampling).
  - nnpfa_persistence_flag is equal to 1.
  - nnpfc_interpolated_pics[i] is greater than 0 only for a single value of i that is greater than 0.
  - Either of the following conditions is true:
    - currPic is the last picture of the bitstream in output order that has nuh_layer_id equal to currLayerId.
    - currPic is the last picture in the CLVS in output order and nnpfa_no_foll_clvs_flag is equal to 1.
- Otherwise, if all of the following conditions are true, the variable numPostRoll is set equal to InpIdx[i] for the value of i such that nnpfa_output_flag[i] is equal to 1, and the variable numInferences is set equal to 1 + numPostRoll:
  - pictureRateUpsamplingFlag is equal to 0.
  - numInputPics is greater than 1.
  - nnpfa_persistence_flag is equal to 1.
  - nnpfa_output_flag[idx] is equal to 1 for a single value of idx in the range of 0 to NumInpPicsInOutputTensor − 1, inclusive, and for that single value of idx, InpIdx[idx] is greater than 0.
  - Either of the following conditions is true:
    - currPic is the last picture of the bitstream in output order that has nuh_layer_id equal to currLayerId.
    - currPic is the last picture in the CLVS in output order and nnpfa_no_foll_clvs_flag is equal to 1.
- Otherwise, the variable numInferences is set equal to 1.

For each value of j in the range of 0 to numInferences − 1, inclusive, the following applies:

- The arrays inputPic[i] and inputPresentFlag[i] for i in the range of 0 to numInputPics − 1, inclusive, representing all the input pictures and the presence of input pictures, respectively, are specified as follows:
  - When j is greater than 0, for each value of k in the range of 0 to j − 1, inclusive, inputPic[k] is set to be currPic and inputPresentFlag[k] is set equal to 0.
  - The j-th input picture, inputPic[j], is set to be currPic and inputPresentFlag[j] is set equal to 1.
  - When numInputPics is greater than 1, the following applies for each value of i in the range of j + 1 to numInputPics − 1, inclusive, in increasing order of i:
    - If both of the following conditions are true, inputPic[i] is set to be prevPic and inputPresentFlag[i] is set equal to 1:
      - Either of the following conditions is true:
        - pictureRateUpsamplingFlag is equal to 1 and currPic is associated with a frame packing arrangement SEI message with frame_packing_arrangement_type equal to 5 and a particular

value of fp_current_frame_is_frame0_flag, and there is a cropped decoded output picture prevPic that is the last picture in output order among all cropped decoded output pictures that have nuh_layer_id equal to currLayerId, precede inputPic[i − 1] in output order, and are associated with a frame packing arrangement SEI message with frame_packing_arrangement_type equal to 5 and the same value of fp_current_frame_is_frame0_flag.

      – pictureRateUpsamplingFlag is equal to 0 or currPic is not associated with a frame packing arrangement SEI message with frame_packing_arrangement_type equal to 5, and there is a cropped decoded output picture prevPic that is the last picture in output order among all cropped decoded output pictures that have nuh_layer_id equal to currLayerId and precede inputPic[i − 1] in output order.

    – nnpfa_no_prev_clvs_flag is equal to 0 or the coded picture corresponding to prevPic and the current picture are present in the same CLVS.

  – Otherwise, the following applies:

    – inputPic[i] is set to be the same picture as inputPic[i − 1] and inputPresentFlag[i] is set equal to 0.

    – It is a requirement of bitstream conformance that, when pictureRateUpsamplingFlag is equal to 1, nnpfc_interpolated_pics[i − 1] shall be equal to 0.

    – It is a requirement of bitstream conformance that when inputPresentFlag[i] is equal to 0 and nnpfc_input_pic_output_flag[i] is equal to 1, the value of nnpfa_output_flag[idx] shall be equal to 0 for the value of idx such that InpIdx[idx] is equal to i.

– For purposes of interpretation of the NNPFC SEI message, the following variables are specified:

  – If numInputPics is greater than 1 and there is a second NNPF that is defined by at least one NNPFC SEI message, is activated by an NNPFA SEI message for currPic, and has nnpfc_purpose equal to 4, the following applies:

    – CroppedWidth is set equal to nnpfcOutputPicWidth defined for the second NNPF.

    – CroppedHeight is set equal to nnpfcOutputPicHeight defined for the second NNPF.

  – Otherwise, the following applies:

    – CroppedWidth is set equal to the value of pps_pic_width_in_luma_samples − SubWidthC * (pps_conf_win_left_offset + pps_conf_win_right_offset) for currPic.

    – CroppedHeight is set equal to the value of pps_pic_height_in_luma_samples − SubHeightC * (pps_conf_win_top_offset + pps_conf_win_bottom_offset) for currPic.

  – The luma sample arrays CroppedYPic[i] and the chroma sample arrays CroppedCbPic[i] and CroppedCrPic[i], when present, are derived as follows for each value of i in the range of 0 to numInputPics − 1, inclusive:

    – The variable sourcePic is derived as follows:

      – If inputPresentFlag[i] is equal to 1 or nnpfc_absent_input_pic_zero_flag is equal to 0, sourcePic is set to be inputPic[i].

      – Otherwise(inputPresentFlag[i] is equal to 0 and nnpfc_absent_input_pic_zero_flag is equal to 1), sourcePic is set to be a picture with a luma sample array of CroppedWidth × CroppedHeight samples equal to 0 and Cb and Cr sample arrays of (CroppedWidth / SubWidthC) × (CroppedHeight / SubHeightC) samples equal to 0.

    – If numInputPics is equal to 1, the following applies:

      – The luma sample array CroppedYPic[i] and the chroma sample arrays CroppedCbPic[i] and CroppedCrPic[i], when present, are set to be the 2-dimensional arrays of decoded sample values of the Y, Cb and Cr components, respectively, of sourcePic.

    – Otherwise (numInputPics is greater than 1), the following applies:

      – The variable sourceWidth is set equal to the value of pps_pic_width_in_luma_samples − SubWidthC * (pps_conf_win_left_offset + pps_conf_win_right_offset) for sourcePic.

– The variable sourceHeight is set equal to the value of pps_pic_height_in_luma_samples − SubHeightC * (pps_conf_win_top_offset + pps_conf_win_bottom_offset) for sourcePic.

– If sourceWidth is equal to CroppedWidth and sourceHeight is equal to CroppedHeight, resampledPic is set to be the same as sourcePic.

– Otherwise (sourceWidth is not equal to CroppedWidth or sourceHeight is not equal to CroppedHeight), the following applies:

  – There shall be an NNPF, hereafter referred to as the super resolution NNPF, that is defined by at least one NNPFC SEI message, is activated by an NNPFA SEI message for sourcePic, and has nnpfc_purpose equal to 4, nnpfcOutputPicWidth equal to CroppedWidth and nnpfcOutputPicHeight equal to CroppedHeight.

  – resampledPic is set to be the output of the neural-network inference of the super resolution NNPF with sourcePic being an input.

– The luma sample array CroppedYPic[i] and the chroma sample arrays CroppedCbPic[i] and CroppedCrPic[i], when present, are set to be the 2-dimensional arrays of decoded sample values of the Y, Cb and Cr components, respectively, of resampledPic.

– BitDepth_Y and BitDepth_C are both set equal to BitDepth.

– ChromaFormatIdc is set equal to sps_chroma_format_idc.

– The array StrengthControlVal[i] for all values of i in the range of 0 to numInputPics − 1, inclusive, specifying the filtering strength control value for the input pictures for the NNPF, is derived as follows:

  – StrengthControlVal[i] is set equal to the value of (firstSliceQp_Y + QpBdOffset) ÷ (63 + QpBdOffset), where firstSliceQp_Y is equal to SliceQp_Y of the first slice of inputPic[i].

There shall not be more than two NNPFC SEI messages present in a picture unit with the same value of nnpfc_id. When there are two NNPFC SEI messages present in a picture unit with the same value of nnpfc_id, these SEI messages shall have different content. When two NNPFC SEI messages with the same nnpfc_id and different content are present in the same picture unit, both of these NNPFC SEI messages shall be in the same SEI NAL unit.

D.12.12 Use of the phase indication SEI message

For purposes of interpretation of the phase indication SEI message, the following variables are specified:

– CroppedWidth is set equal to pps_pic_width_in_luma_samples − SubWidthC * (pps_conf_win_left_offset + pps_conf_win_right_offset).

– CroppedHeight is set equal to pps_pic_height_in_luma_samples − SubHeightC * (pps_conf_win_top_offset + pps_conf_win_bottom_offset).

Bibliography

[1] Rec. ITU-T H.222.0 | ISO/IEC 13818-1 (in force), *Information technology – Generic coding of moving pictures and associated audio information: Systems*.

[2] Rec. ITU-T H.264 | ISO/IEC 14496-10 (in force), *Advanced video coding for generic audiovisual services*.

[3] Rec. ITU-T H.265 | ISO/IEC 23008-2 (in force), *High efficiency video coding*.

[4] Rec. ITU-T H.273 | ISO/IEC 23091-2 (in force), *Coding-independent code points for video signal type identification*.

[5] Rec. ITU-T H.320 (in force), *Narrow-band visual telephone systems and terminal equipment*.

[6] Supplement ITU-T H-Suppl. 19 | ISO/IEC TR 23091-4 (in force), *Usage of video signal type code points*.

[7] Rec. ITU-R BT.2100 (in force), *Image parameter values for high dynamic range television for use in production and international programme exchange*.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	Tariff and accounting principles and international telecommunication/ICT economic and policy issues
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	Environment and ICTs, climate change, e-waste, energy efficiency; 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	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling, and associated measurements and tests
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, next-generation networks, Internet of Things and smart cities
Series Z	Languages and general software aspects for telecommunication systems

Published in Switzerland
Geneva, 2023