

---

---

**Information technology — Coded  
representation of immersive media —**

**Part 3:  
Versatile video coding**

*Technologies de l'information — Représentation codée de média  
immersifs —*

*Partie 3: Codage vidéo polyvalent*



STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 23090-3:2022



**COPYRIGHT PROTECTED DOCUMENT**

© ISO/IEC 2022

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office  
CP 401 • Ch. de Blandonnet 8  
CH-1214 Vernier, Geneva  
Phone: +41 22 749 01 11  
Email: [copyright@iso.org](mailto:copyright@iso.org)  
Website: [www.iso.org](http://www.iso.org)

Published in Switzerland

## Contents

|                                                                                                         |     |
|---------------------------------------------------------------------------------------------------------|-----|
| Foreword .....                                                                                          | vi  |
| Introduction .....                                                                                      | vii |
| 1 Scope .....                                                                                           | 1   |
| 2 Normative references .....                                                                            | 1   |
| 3 Terms and definitions .....                                                                           | 1   |
| 4 Abbreviated terms .....                                                                               | 16  |
| 5 Conventions .....                                                                                     | 18  |
| 5.1 General .....                                                                                       | 18  |
| 5.2 Arithmetic operators .....                                                                          | 19  |
| 5.3 Logical operators .....                                                                             | 19  |
| 5.4 Relational operators .....                                                                          | 19  |
| 5.5 Bit-wise operators .....                                                                            | 20  |
| 5.6 Assignment operators .....                                                                          | 20  |
| 5.7 Range notation .....                                                                                | 20  |
| 5.8 Mathematical functions .....                                                                        | 20  |
| 5.9 Order of operation precedence .....                                                                 | 21  |
| 5.10 Variables, syntax elements and tables .....                                                        | 22  |
| 5.11 Text description of logical operations .....                                                       | 23  |
| 5.12 Processes .....                                                                                    | 24  |
| 6 Bitstream and picture formats, partitionings, scanning processes and neighbouring relationships ..... | 25  |
| 6.1 Bitstream formats .....                                                                             | 25  |
| 6.2 Source, decoded and output picture formats .....                                                    | 25  |
| 6.3 Partitioning of pictures, subpictures, slices, tiles, and CTUs .....                                | 27  |
| 6.3.1 Partitioning of pictures into subpictures, slices, and tiles .....                                | 27  |
| 6.3.2 Block, quadtree and multi-type tree structures .....                                              | 30  |
| 6.3.3 Spatial or component-wise partitionings .....                                                     | 30  |
| 6.4 Availability processes .....                                                                        | 31  |
| 6.4.1 Allowed quad split process .....                                                                  | 31  |
| 6.4.2 Allowed binary split process .....                                                                | 32  |
| 6.4.3 Allowed ternary split process .....                                                               | 34  |
| 6.4.4 Derivation process for neighbouring block availability .....                                      | 35  |
| 6.5 Scanning processes .....                                                                            | 36  |
| 6.5.1 CTB raster scanning, tile scanning, and subpicture scanning processes .....                       | 36  |
| 6.5.2 Up-right diagonal scan order array initialization process .....                                   | 40  |
| 6.5.3 Horizontal and vertical traverse scan order array initialization process .....                    | 40  |
| 7 Syntax and semantics .....                                                                            | 41  |
| 7.1 Method of specifying syntax in tabular form .....                                                   | 41  |
| 7.2 Specification of syntax functions and descriptors .....                                             | 42  |
| 7.3 Syntax in tabular form .....                                                                        | 44  |
| 7.3.1 NAL unit syntax .....                                                                             | 44  |
| 7.3.2 Raw byte sequence payloads, trailing bits and byte alignment syntax .....                         | 44  |
| 7.3.3 Profile, tier, and level syntax .....                                                             | 64  |
| 7.3.4 DPB parameters syntax .....                                                                       | 67  |
| 7.3.5 Timing and HRD parameters syntax .....                                                            | 67  |
| 7.3.6 Supplemental enhancement information message syntax .....                                         | 68  |
| 7.3.7 Slice header syntax .....                                                                         | 68  |
| 7.3.8 Weighted prediction parameters syntax .....                                                       | 71  |
| 7.3.9 Reference picture lists syntax .....                                                              | 72  |

|        |                                                                                                            |     |
|--------|------------------------------------------------------------------------------------------------------------|-----|
| 7.3.10 | Reference picture list structure syntax.....                                                               | 72  |
| 7.3.11 | Slice data syntax .....                                                                                    | 73  |
| 7.4    | Semantics.....                                                                                             | 95  |
| 7.4.1  | General .....                                                                                              | 95  |
| 7.4.2  | NAL unit semantics.....                                                                                    | 95  |
| 7.4.3  | Raw byte sequence payloads, trailing bits and byte alignment semantics .....                               | 104 |
| 7.4.4  | Profile, tier, and level semantics.....                                                                    | 163 |
| 7.4.5  | DPB parameters semantics.....                                                                              | 169 |
| 7.4.6  | Timing and HRD parameters semantics .....                                                                  | 170 |
| 7.4.7  | Supplemental enhancement information message semantics.....                                                | 175 |
| 7.4.8  | Slice header semantics .....                                                                               | 175 |
| 7.4.9  | Weighted prediction parameters semantics .....                                                             | 185 |
| 7.4.10 | Reference picture lists semantics.....                                                                     | 187 |
| 7.4.11 | Reference picture list structure semantics.....                                                            | 188 |
| 7.4.12 | Slice data semantics .....                                                                                 | 189 |
| 8      | Decoding process.....                                                                                      | 215 |
| 8.1    | General decoding process.....                                                                              | 215 |
| 8.2    | NAL unit decoding process .....                                                                            | 218 |
| 8.3    | Slice decoding process .....                                                                               | 218 |
| 8.3.1  | Decoding process for picture order count.....                                                              | 218 |
| 8.3.2  | Decoding process for reference picture lists construction.....                                             | 220 |
| 8.3.3  | Decoding process for reference picture marking.....                                                        | 225 |
| 8.3.4  | Decoding process for generating unavailable reference pictures .....                                       | 226 |
| 8.3.5  | Decoding process for symmetric motion vector difference reference indices .....                            | 227 |
| 8.3.6  | Decoding process for collocated picture and no backward prediction .....                                   | 228 |
| 8.4    | Decoding process for coding units coded in intra prediction mode.....                                      | 228 |
| 8.4.1  | General decoding process for coding units coded in intra prediction mode .....                             | 228 |
| 8.4.2  | Derivation process for luma intra prediction mode .....                                                    | 230 |
| 8.4.3  | Derivation process for chroma intra prediction mode.....                                                   | 233 |
| 8.4.4  | Cross-component chroma intra prediction mode checking process.....                                         | 235 |
| 8.4.5  | Decoding process for intra blocks.....                                                                     | 236 |
| 8.5    | Decoding process for coding units coded in inter prediction mode.....                                      | 271 |
| 8.5.1  | General decoding process for coding units coded in inter prediction mode .....                             | 271 |
| 8.5.2  | Derivation process for motion vector components and reference indices .....                                | 276 |
| 8.5.3  | Decoder-side motion vector refinement process.....                                                         | 298 |
| 8.5.4  | Derivation process for geometric partitioning mode motion vector components and reference indices .....    | 304 |
| 8.5.5  | Derivation process for subblock motion vector components and reference indices .....                       | 306 |
| 8.5.6  | Decoding process for inter blocks.....                                                                     | 336 |
| 8.5.7  | Decoding process for geometric partitioning mode inter blocks .....                                        | 360 |
| 8.5.8  | Decoding process for the residual signal of coding blocks coded in inter prediction mode.....              | 366 |
| 8.5.9  | Decoding process for the reconstructed signal of chroma coding blocks coded in inter prediction mode ..... | 368 |
| 8.6    | Decoding process for coding units coded in IBC prediction mode.....                                        | 370 |
| 8.6.1  | General decoding process for coding units coded in IBC prediction mode.....                                | 370 |
| 8.6.2  | Derivation process for block vector components for IBC blocks.....                                         | 372 |
| 8.6.3  | Decoding process for IBC blocks .....                                                                      | 377 |
| 8.7    | Scaling, transformation and array construction process .....                                               | 378 |
| 8.7.1  | Derivation process for quantization parameters.....                                                        | 378 |
| 8.7.2  | Scaling and transformation process.....                                                                    | 380 |
| 8.7.3  | Scaling process for transform coefficients .....                                                           | 381 |
| 8.7.4  | Transformation process for scaled transform coefficients .....                                             | 384 |
| 8.7.5  | Picture reconstruction process .....                                                                       | 404 |
| 8.8    | In-loop filter process .....                                                                               | 408 |
| 8.8.1  | General .....                                                                                              | 408 |
| 8.8.2  | Picture inverse mapping process for luma samples .....                                                     | 409 |



|                     |                                                                   |     |
|---------------------|-------------------------------------------------------------------|-----|
| 8.8.3               | Deblocking filter process .....                                   | 410 |
| 8.8.4               | Sample adaptive offset process .....                              | 440 |
| 8.8.5               | Adaptive loop filter process .....                                | 442 |
| 9                   | Parsing process .....                                             | 455 |
| 9.1                 | General .....                                                     | 455 |
| 9.2                 | Parsing process for k-th order Exp-Golomb codes .....             | 455 |
| 9.2.1               | General .....                                                     | 455 |
| 9.2.2               | Mapping process for signed Exp-Golomb codes .....                 | 457 |
| 9.3                 | CABAC parsing process for slice data .....                        | 457 |
| 9.3.1               | General .....                                                     | 457 |
| 9.3.2               | Initialization process .....                                      | 459 |
| 9.3.3               | Binarization process .....                                        | 482 |
| 9.3.4               | Decoding process flow .....                                       | 492 |
| Annex A (normative) | Profiles, tiers and levels .....                                  | 510 |
| Annex B (normative) | Byte stream format .....                                          | 530 |
| Annex C (normative) | Hypothetical reference decoder .....                              | 533 |
| Annex D (normative) | Supplemental enhancement information and use of SEI and VUI ..... | 559 |

## Foreword

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

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see [www.iso.org/directives](http://www.iso.org/directives) or [www.iec.ch/members\\_experts/refdocs](http://www.iec.ch/members_experts/refdocs)).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see [www.iso.org/patents](http://www.iso.org/patents)) or the IEC list of patent declarations received (see <https://patents.iec.ch>).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see [www.iso.org/iso/foreword.html](http://www.iso.org/iso/foreword.html). In the IEC, see [www.iec.ch/understanding-standards](http://www.iec.ch/understanding-standards).

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*, in collaboration with ITU-T (as ITU-T H.266).

This second edition cancels and replaces the first edition (ISO/IEC 23090-3:2021), which has been technically revised.

The main changes are as follows:

- the specification of operation range extensions,
- the addition of level 6.3,
- the addition of the SEI manifest SEI message, the SEI prefix indication SEI message, and the constrained RASL encoding indication SEI message, and
- the specification of SEI payload type values and necessary variables for use of a number of SEI messages added to the second edition of Rec. ITU-T H.274 | ISO/IEC 23002-7.

A list of all parts in the ISO/IEC 23090 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at [www.iso.org/members.html](http://www.iso.org/members.html) and [www.iec.ch/national-committees](http://www.iec.ch/national-committees).

## Introduction

### Purpose

This document specifies 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 document particularly include ultra-high-definition video (e.g., with  $3840 \times 2160$  or  $7680 \times 4320$  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  $360^\circ$  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 document 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 document, 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 document is designed to facilitate video data interchange among different applications.

Considering the practicality of implementing the full syntax of this document, 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 document. 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 document 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 document is considered to be in conformance with the requirements of this document. 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 document.

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 document.

### Versions of this document

Rec. ITU-T H.266 | ISO/IEC 23090-3 version 1 refers to the first approved version of this document. The first edition published 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 (the current version) 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 document. This document corresponds to the second version. At the time of publication of this document, a corresponding second edition of Rec. ITU-T H.266 was in preparation for publication by ITU-T.

### 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 document) 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 subclauses 7.1 to 7.3 for syntactical order and subclause 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. Annexes A through D also form an integral part of this document.

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 document.

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

In this document, the following verbal forms are used:

- "shall" indicates a requirement;
- "should" indicates a recommendation;
- "may" indicates a permission;
- "can" indicates a possibility or a capability.

Information marked as "NOTE" is intended to assist the understanding or use of the document. "Notes to entry" used in Clause 3 provide additional information that supplements the terminological data and can contain provisions relating to the use of a term.

#### **Patent declarations**

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of patents.

ISO and IEC take no position concerning the evidence, validity and scope of these patent rights.

The holder of these patent rights have assured ISO and IEC that he/she is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patent rights are registered with ISO and IEC. Information may be obtained from the patent database available at [www.iso.org/patents](http://www.iso.org/patents).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those in the patent database. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 23090-3:2022

# Information technology — Coded representation of immersive media —

## Part 3: Versatile video coding

### 1 Scope

This document 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 document. Additionally, the internal processing steps performed within a decoder are also considered to be outside the scope of this document; only the externally observable output behaviour is required to conform to the specifications of this document.

This document 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 document, 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 document is designed to facilitate video data interchange among different applications.

### 2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

Rec. ITU-T H.274 | ISO/IEC 23002-7, *Versatile supplemental enhancement information messages for coded video bitstreams*

Rec. ITU-T T.35, *Procedure for the allocation of ITU-T defined codes for non standard facilities*

### 3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <https://www.electropedia.org/>

#### 3.1

##### access unit

set of *PUs* that belong to different *layers* and contain *coded pictures* associated with the same time for output from the *DPB*

#### 3.2

##### adaptive colour transform

*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



### 3.3

#### **adaptive loop filter**

filtering process that is applied as part of the *decoding process* and is controlled by parameters conveyed in an *APS*

### 3.4

#### **ALF APS**

*APS* that controls the *ALF* process

### 3.5

#### **adaptation parameter set**

*syntax structure* containing *syntax elements* that apply to zero or more *slices* as determined by zero or more *syntax elements* found in *slice headers*

### 3.6

#### **associated GDR picture**

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*

### 3.7

#### **associated IRAP picture**

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*

### 3.8

#### **associated IRAP subpicture**

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 *subpicture* in *decoding order* there is no *GDR subpicture* with *nuh\_layer\_id* equal to *layerId* and subpicture index equal to *subpicIdx*

### 3.9

#### **associated non-VCL NAL unit**

*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*

### 3.10

#### **associated VCL NAL unit**

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*

### 3.11

#### **bin**

bit of a *bin string*

### 3.12

#### **binarization**

set of *bin strings* for all possible values of a *syntax element*

### 3.13

#### **binarization process**

unique mapping process of all possible values of a *syntax element* onto a set of *bin strings*



**3.14****binary split**

split of a rectangular  $M \times N$  *block* of samples into two *blocks* where a vertical split results in a first  $(M / 2) \times N$  *block* and a second  $(M / 2) \times N$  *block*, and a horizontal split results in a first  $M \times (N / 2)$  *block* and a second  $M \times (N / 2)$  *block*

**3.15****bin string**

intermediate binary representation of values of *syntax elements* from the *binarization* of the *syntax element*

**3.16****bi-predictive slice****B slice**

*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*

**3.17****bitstream**

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*)

**3.18****block**

$M \times N$  ( $M$ -column by  $N$ -row) array of samples, or an  $M \times N$  array of *transform coefficients*

**3.19****block vector**

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*

**3.20****byte**

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

**3.21****byte-aligned**

<*bitstream*> positioned an integer multiple of 8 bits from the position of the first bit in the *bitstream*

**3.22****byte-aligned**

<bit or *byte* or *syntax element*> position at which it appears in a *bitstream* is byte-aligned

**3.23****byte stream**

encapsulation of a *NAL unit stream* into a series of *bytes* containing *start code prefixes* and *NAL units*

**3.24****chroma**

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 1 to entry: The term chroma is used rather than the term chrominance in order to avoid the implication of the use of linear light transfer characteristics that is often associated with the term chrominance.

**3.25****CRA PU**

*PU* in which the *coded picture* is a *CRA picture*

### 3.26

#### **CRA picture**

*IRAP picture* for which each *VCL NAL unit* has *nal\_unit\_type* equal to *CRA\_NUT*

Note 1 to entry: 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

#### **CRA subpicture**

*IRAP subpicture* for which each *VCL NAL unit* has *nal\_unit\_type* equal to *CRA\_NUT*

### 3.28

#### **coded layer video sequence:**

sequence of *PU*s with the same value of *nuh\_layer\_id* that consists, in *decoding order*, of a *CLVSS PU*, followed by zero or more *PU*s that are not *CLVSS PU*s, including all subsequent *PU*s up to but not including any subsequent *PU* that is a *CLVSS PU*

Note 1 to entry: 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.

### 3.29

#### **CLVSS PU**

*PU* in which the *coded picture* is a *CLVSS picture*

### 3.30

#### **CLVSS picture**

*coded picture* that is an *IRAP picture* with *NoOutputBeforeRecoveryFlag* equal to 1 or a *GDR picture* with *NoOutputBeforeRecoveryFlag* equal to 1

### 3.31

#### **coded picture**

*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*

### 3.32

#### **coded picture buffer**

first-in first-out buffer containing *DUs* in *decoding order* specified in the *hypothetical reference decoder*

Note 1 to entry: The hypothetical reference decoder is specified in Annex C.

### 3.33

#### **coded representation**

data element as represented in its coded form

### 3.34

#### **coded slice NAL unit**

*NAL unit* that contains a *coded slice*

### 3.35

#### **coded video sequence**

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*

### 3.36

#### **CVSS AU**

*IRAP AU* or *GDR AU* for which the *coded picture* in each *PU* is a *CLVSS picture*

**3.37****coding block**

$M \times N$  *block* of samples for some values of  $M$  and  $N$  such that the division of a *CTB* into *coding blocks* is a *partitioning*

**3.38****coding tree block**

$N \times N$  *block* of samples for some value of  $N$  such that the division of a *component* into *CTBs* is a *partitioning*

**3.39****coding tree unit**

*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

**3.40****coding unit**

*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

**3.41****component**

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

**3.42****context variable**

variable specified for the *adaptive binary arithmetic decoding process* of a *bin* by a formula containing recently decoded *bins*

**3.43****deblocking filter**

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*

**3.44****decoded picture**

*picture* produced by applying the *decoding process* to a *coded picture*

**3.45****decoded picture buffer**

buffer holding *decoded pictures* for reference, output reordering, or output delay specified for the *hypothetical reference decoder*

**3.46****decoder**

embodiment of a *decoding process*

**3.47****decoding order**

order in which *syntax elements* are processed by the *decoding process*

**3.48****decoding process**

process specified in this document that reads a *bitstream* and derives *decoded pictures* from it

**3.49**

**decoding unit**

*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*

**3.50**

**emulation prevention byte**

*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*

**3.51**

**encoder**

embodiment of an *encoding process*

**3.52**

**encoding process**

process not specified in this document that produces a *bitstream* conforming to this document

**3.53**

**filler data NAL units**

*NAL units* with *nal\_unit\_type* equal to FD\_NUT

**3.54**

**flag**

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

**3.55**

**frequency index**

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

**3.56**

**GDR AU**

*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*

**3.57**

**GDR PU**

*PU* in which the *coded picture* is a *GDR picture*

**3.58**

**GDR picture**

*picture* for which each *VCL NAL unit* has *nal\_unit\_type* equal to GDR\_NUT

Note 1 to entry: 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.

**3.59**

**GDR subpicture**

*subpicture* for which each *VCL NAL unit* has *nal\_unit\_type* equal to GDR\_NUT

**3.60**

**hypothetical reference decoder**

hypothetical *decoder* model that specifies constraints on the variability of conforming *NAL unit streams* or conforming *byte streams* that an encoding process may produce

**3.61****hypothetical stream scheduler**

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*

**3.62****IDR PU**

*PU* in which the *coded picture* is an *IDR picture*

**3.63****IDR picture**

*IRAP picture* for which each *VCL NAL unit* has *nal\_unit\_type* equal to *IDR\_W\_RADL* or *IDR\_N\_LP*

Note 1 to entry: 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****IDR subpicture**

*IRAP subpicture* for which each *VCL NAL unit* has *nal\_unit\_type* equal to *IDR\_W\_RADL* or *IDR\_N\_LP*

**3.65****inter-layer reference picture**

*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"

**3.66****inter coding**

coding of a *coding block*, *slice*, or *picture* that uses *inter prediction*

**3.67****inter prediction**

*prediction* derived from blocks of sample values of one or more *reference pictures* as determined by motion vectors

**3.68****intra block copy prediction**

*prediction* derived from blocks of sample values of the same decoded *slice* as determined by block vectors

**3.69****intra coding**

coding of a *coding block*, *slice*, or *picture* that uses *intra prediction*

**3.70****intra prediction**

*prediction* derived from neighbouring sample values of the same decoded *slice*

**3.71****IRAP AU**

*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*

**3.72****IRAP PU**

*PU* in which the *coded picture* is an *IRAP picture*

### 3.73

#### **IRAP picture**

*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 to entry: 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 to entry: 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

#### **IRAP subpicture**

*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

### 3.75

#### **intra slice**

##### **I slice**

*slice* that is decoded using *intra prediction* only

### 3.76

#### **layer**

set of *VCL NAL units* that all have a particular value of *nuh\_layer\_id* and the *associated non-VCL NAL units*

### 3.77

#### **leading picture**

*picture* that precedes the *associated IRAP picture* in output order

### 3.78

#### **leading subpicture**

*subpicture* that precedes the *associated IRAP subpicture* in output order

Note 1 to entry: As constrained in subclause 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

#### **leaf**

terminating node of a tree that is a root node of a tree of depth 0

### 3.80

#### **level <constraints>**

defined set of constraints on the values that may be taken by the *syntax elements* and variables of this document

Note 1 to entry: 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.

### 3.81

#### **level <transform coefficient value>**

value of a *transform coefficient* prior to *scaling*

### 3.82

#### **list 0 (list 1) motion vector**

*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**

APS that controls the LMCS process

**3.85****long-term reference picture**

*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**

sample array or single sample representing the monochrome signal related to the primary colours, represented by the symbol or subscript Y or L

Note 1 to entry: 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****LMCS****luma mapping with chroma scaling**

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**

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**

*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****NAL unit**

*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****NAL unit stream**

sequence of *NAL units*

**3.92****operation point**

temporal subset of an OLS, identified by an OLS index and a highest value of *TemporalId*

**3.93****output layer**

layer of an output layer set that is output

**3.94****output layer set**

set of layers for which one or more layers are specified as the output layers



**3.95**

**OLS layer index**

index, of a layer in an OLS, to the list of layers in the OLS

**3.96**

**output order**

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**

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**

set of representative *component* values

**3.99**

**palette prediction**

*prediction* derived from one or more *palettes*

**3.100**

**partitioning**

division of a set into subsets such that each element of the set is in exactly one of the subsets

**3.101**

**picture**

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 1 to entry: 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**

*syntax structure* containing *syntax elements* that apply to all *slices* of a coded picture

**3.103**

**picture-level slice index**

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**

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**

*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**

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**

embodiment of the *prediction process*

**3.108****prediction process**

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 slice****P slice**

*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**

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**

specified subset of the syntax of this document

**3.112****quadtree**

*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**

variable used by the *decoding process* for *scaling* of *transform coefficient levels*

**3.114****random access**

act of starting the decoding process for a *bitstream* at a point other than the beginning of the *bitstream*

**3.115****RADL PU**

*PU* in which the *coded picture* is a *RADL picture*

**3.116****RADL picture**

*coded picture* for which each *VCL NAL unit* has *nal\_unit\_type* equal to *RADL\_NUT*

Note 1 to entry: 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.

**3.117****RADL subpicture**

*subpicture* for which each *VCL NAL unit* has *nal\_unit\_type* equal to *RADL\_NUT*

**3.118****RASL PU**

*PU* in which the *coded picture* is a *RASL picture*

### 3.119

#### **RASL picture**

*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 1 to entry: 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.

### 3.120

#### **RASL subpicture**

*subpicture* for which each *VCL NAL unit* has *nal\_unit\_type* equal to *RASL\_NUT*

### 3.121

#### **raster scan**

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**

*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

#### **RBSP stop bit**

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**

index into a *reference picture list*

### 3.125

#### **reference picture**

*picture* that is a *short-term reference picture*, a *long-term reference picture*, or an *inter-layer reference picture*

Note 1 to entry: 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**

list of *reference pictures* that is used for *inter prediction* of a *P* or *B slice*

Note 1 to entry: 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**

*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**

second *reference picture list* used for *inter prediction* of a *B slice*

**3.129****residual**

decoded difference between a *prediction* of a sample or data element and its decoded value

**3.130****scaling**

process of multiplying *transform coefficient levels* by a factor, resulting in *transform coefficients*

**3.131****scaling list**

list that associates each *frequency index* with a scale factor for the *scaling* process

**3.132****scaling list APS**

*APS* with syntax elements used to construct the *scaling lists*

**3.133****sequence parameter set**

*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**

*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**

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**

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**

term used to describe the video material or some of its attributes before encoding

**3.138****start code prefix**

unique sequence of three *bytes* equal to 0x000001 embedded in the *byte stream* as a prefix to each *NAL unit*

Note 1 to entry: 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****STSA PU**

*PU* in which the *coded picture* is an *STSA picture*

### 3.140

#### STSA picture

*coded picture* for which each *VCL NAL unit* has *nal\_unit\_type* equal to STSA\_NUT

Note 1 to entry: 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.

### 3.141

#### STSA subpicture

*subpicture* for which each *VCL NAL unit* has *nal\_unit\_type* equal to STSA\_NUT

### 3.142

#### string of data bits

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

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

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

subset of the *bitstream* consisting of *NAL units* of a particular *sublayer* and the lower *sublayers*

### 3.146

#### subpicture

rectangular region of one or more *slices* within a *picture*

### 3.147

#### subpicture-level slice index

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

#### SEI message

*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

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**

split of a rectangular  $M \times N$  *block* of samples into three *blocks* where a vertical split results in a first  $(M / 4) \times N$  *block*, a second  $(M / 2) \times N$  *block*, a third  $(M / 4) \times N$  *block*, and a horizontal split results in a first  $M \times (N / 4)$  *block*, a second  $M \times (N / 2)$  *block*, a third  $M \times (N / 4)$  *block*

**3.152****tier**

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**

rectangular region of *CTUs* within a particular *tile column* and a particular *tile row* in a *picture*

**3.154****tile column**

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**

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**

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**

*picture* for which each *VCL NAL unit* has *nal\_unit\_type* equal to TRAIL\_NUT

Note 1 to entry: 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**

*subpicture* for which each *VCL NAL unit* has *nal\_unit\_type* equal to TRAIL\_NUT

Note 1 to entry: 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**

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**

rectangular  $M \times N$  *block* of samples resulting from a *transform* in the *decoding process*

**3.161****transform coefficient**

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

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

<single tree> *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*, and *syntax structures* used to transform the *transform block* samples

### 3.164

#### transform unit

<dual tree> *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

### 3.165

#### tree

finite set of nodes with a unique root node

### 3.166

#### VCL NAL unit

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 document

## 4 Abbreviated terms

|       |                                                             |
|-------|-------------------------------------------------------------|
| 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                        |
| CLVS  | coded layer video sequence                                  |
| CLVSS | coded layer video sequence start                            |
| 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                                              |
| CVSS  | coded video sequence start                                        |
| 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                                                                      |
| TU   | transform unit                                                                      |
| 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 (this document: Rec. ITU-T H.266   ISO/IEC 23090-3)          |

## 5 Conventions

### 5.1 General

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 document, but could potentially be used in syntax structures conforming to future versions of this document 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 document and are not expected to have a specified meaning in future editions of this document.

**NOTE** The mathematical operators used in this document 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-th, etc.

## 5.2 Arithmetic operators

|                     |                                                                                                                                                    |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| +                   | 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.                                               |
| $\div$              | division in mathematical formulae where no truncation or rounding is intended                                                                      |
| $\frac{x}{y}$       | division in mathematical formulae 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 $x \geq 0$ and $y > 0$                                                |

## 5.3 Logical operators

|                |                                                                                          |
|----------------|------------------------------------------------------------------------------------------|
| $x \ \&\& \ y$ | 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.

$x \gg y$  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.

$x \ll y$  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

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

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

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

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

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

$$\text{Floor}(x) \text{ largest integer less than or equal to } x \quad (6)$$

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

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

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

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

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

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

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

## 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 document 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)**

| <b>operations (with operands x, y, and z)</b>                          |
|------------------------------------------------------------------------|
| "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 subclause 7.2 and assume the existence of a bitstream pointer with an indication of the position of the next bit to be read by the decoding process from the bitstream. Syntax functions are described by their names, which are constructed as syntax element names and end with left and right round parentheses including zero or more variable names (for definition) or values (for usage), separated by commas (if more than one variable).

Functions that are not syntax functions (including mathematical functions specified in subclause 5.8) 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  $\{ \{ \dots \} \{ \dots \} \}$ , 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 subclause 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 document. The byte stream format is specified in Annex B.

### 6.2 Source, decoded and output picture formats

This subclause specifies the relationship between source and decoded 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 document, 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.

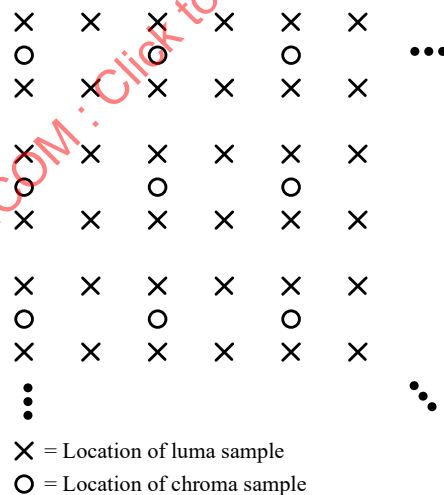
In 4:2:2 sampling, each of the two chroma arrays has the same height and half the width of the luma array.

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.



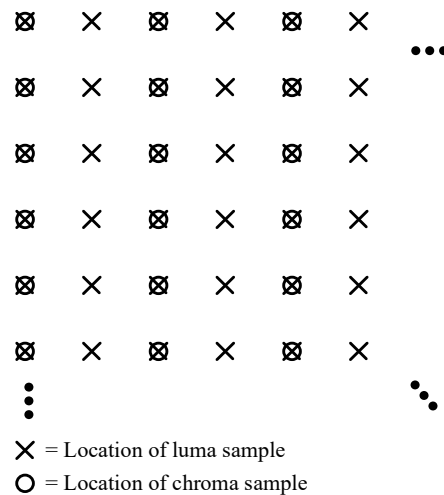


**Figure 2 — Nominal vertical and horizontal locations of 4:2:2 luma and chroma samples in a picture**

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.

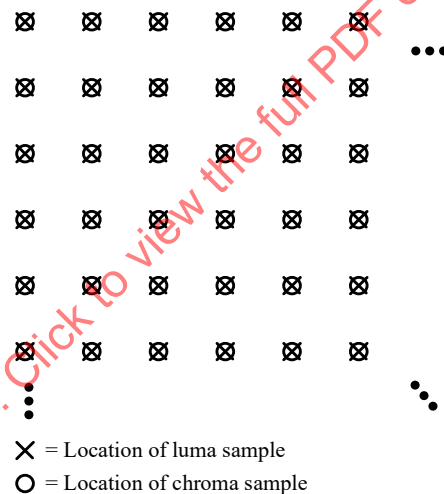


**Figure 3 — Nominal vertical and horizontal locations of 4:4:4 luma and chroma samples in a picture**

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

### 6.3.1 Partitioning of pictures into subpictures, slices, and tiles

This subclause 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.

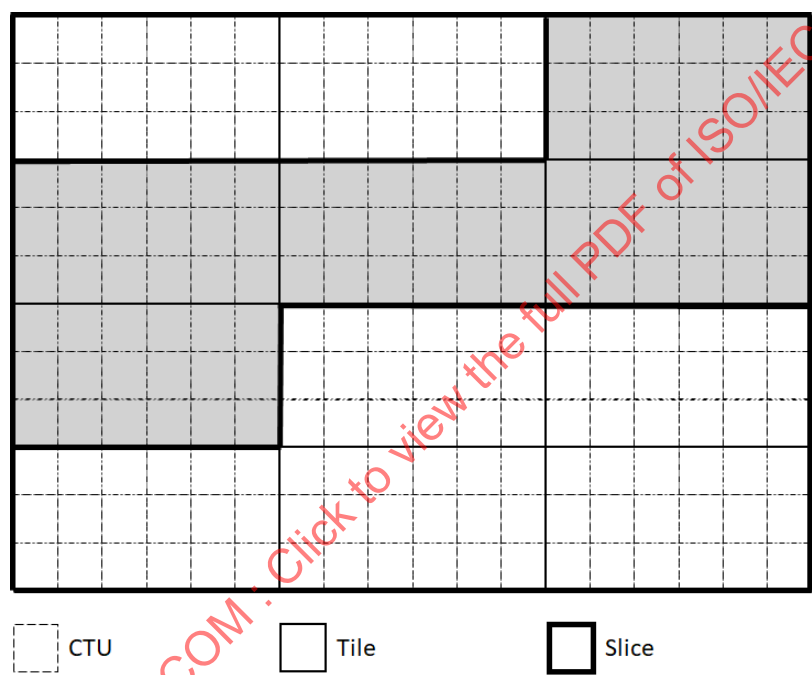


**Figure 4 — A picture with 18 by 12 luma CTUs that is partitioned into 12 tiles and 3 raster-scan slices**

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.

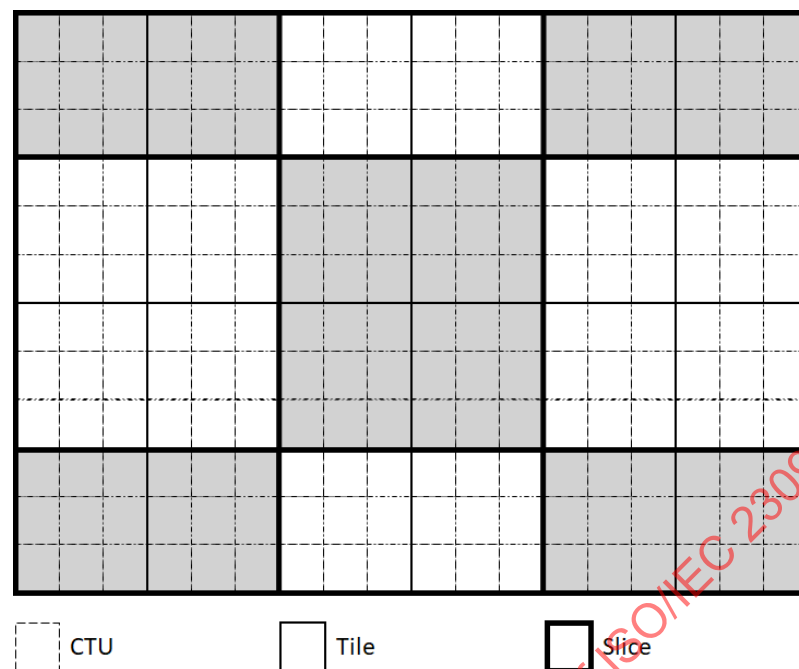


**Figure 5 — A picture with 18 by 12 luma CTUs that is partitioned into 24 tiles and 9 rectangular slices**

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.

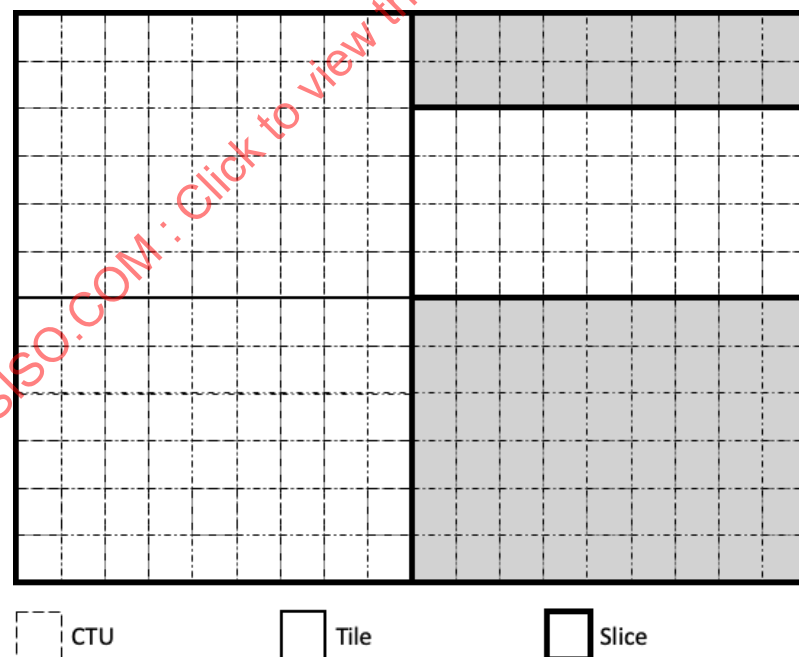


**Figure 6 — A picture that is partitioned into 4 tiles and 4 rectangular slices**

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).

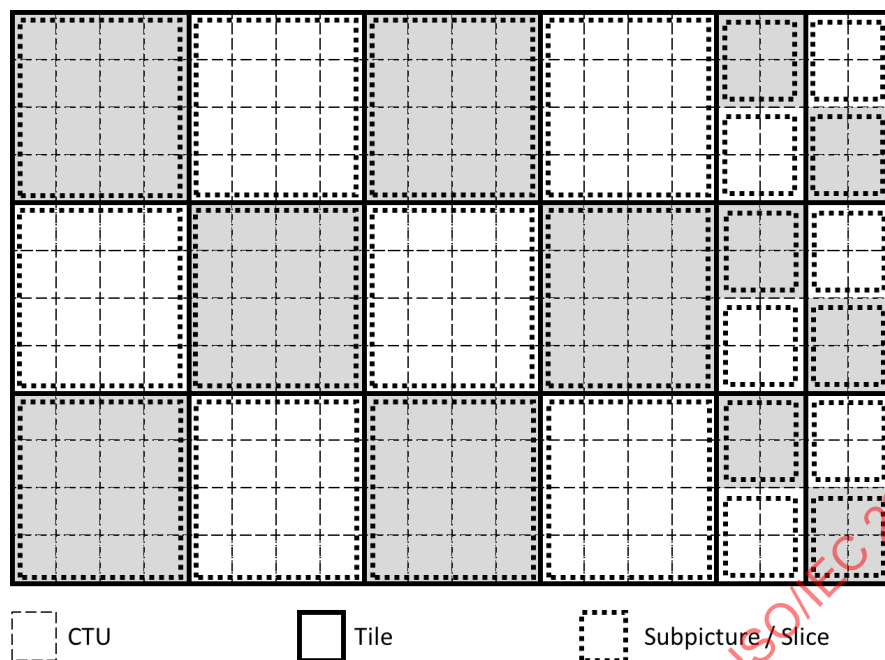


**Figure 7 — A picture that is partitioned into 18 tiles, 24 slices and 24 subpictures**

### 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 document 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 minimum 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`**

|                              | <code>btSplit</code> = <code>SPLIT_BT_VER</code> | <code>btSplit</code> = <code>SPLIT_BT_HOR</code> |
|------------------------------|--------------------------------------------------|--------------------------------------------------|
| <code>parallelTtSplit</code> | <code>SPLIT_TT_VER</code>                        | <code>SPLIT_TT_HOR</code>                        |
| <code>cbSize</code>          | <code>cbWidth</code>                             | <code>cbHeight</code>                            |

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  $\text{Min}(64, \text{maxTtSize})$ ;
- $\text{mttDepth}$  is greater than or equal to  $\text{maxMttDepth}$ ;
- $x0 + cbWidth$  is greater than  $\text{pps\_pic\_width\_in\_luma\_samples}$ ;
- $y0 + cbHeight$  is greater than  $\text{pps\_pic\_height\_in\_luma\_samples}$ ;
- $\text{treeType}$  is equal to  $\text{DUAL\_TREE\_CHROMA}$  and  $(cbWidth / \text{SubWidthC}) * (cbHeight / \text{SubHeightC})$  is less than or equal to 32;
- $\text{treeType}$  is equal to  $\text{DUAL\_TREE\_CHROMA}$  and  $(cbWidth / \text{SubWidthC})$  is equal to 8 and  $\text{ttSplit}$  is equal to  $\text{SPLIT\_TT\_VER}$ ;
- $\text{treeType}$  is equal to  $\text{DUAL\_TREE\_CHROMA}$  and  $\text{modeType}$  is equal to  $\text{MODE\_TYPE\_INTRA}$ ;
- $cbWidth * cbHeight$  is equal to 64 and  $\text{modeType}$  is equal to  $\text{MODE\_TYPE\_INTER}$ ;
- Otherwise,  $\text{allowTtSplit}$  is set equal to  $\text{TRUE}$ .

#### 6.4.4 Derivation process for neighbouring block availability

Inputs to this process are:

- the luma location  $(x_{\text{Curr}}, y_{\text{Curr}})$  of the top-left sample of the current block relative to the top-left luma sample of the current picture,
- the luma location  $(x_{\text{NbY}}, y_{\text{NbY}})$  covered by a neighbouring block relative to the top-left luma sample of the current picture,
- the variable  $\text{checkPredModeY}$  specifying whether availability depends on the prediction mode,
- the variable  $\text{cIdx}$  specifying the colour component of the current block.

Output of this process is the availability of the neighbouring block covering the location  $(x_{\text{NbY}}, y_{\text{NbY}})$ , denoted as  $\text{availableN}$ .

The neighbouring block availability  $\text{availableN}$  is derived as follows:

- If one or more of the following conditions are true,  $\text{availableN}$  is set equal to  $\text{FALSE}$ :
  - $x_{\text{NbY}}$  is less than 0;
  - $y_{\text{NbY}}$  is less than 0;
  - $x_{\text{NbY}}$  is greater than or equal to  $\text{pps\_pic\_width\_in\_luma\_samples}$ ;
  - $y_{\text{NbY}}$  is greater than or equal to  $\text{pps\_pic\_height\_in\_luma\_samples}$ ;
  - $(x_{\text{NbY}} \gg \text{CtbLog2SizeY})$  is greater than  $(x_{\text{Curr}} \gg \text{CtbLog2SizeY})$  and  $(y_{\text{NbY}} \gg \text{CtbLog2SizeY})$  is greater than or equal to  $(y_{\text{Curr}} \gg \text{CtbLog2SizeY})$ ;
  - $(y_{\text{NbY}} \gg \text{CtbLog2SizeY})$  is greater than or equal to  $(y_{\text{Curr}} \gg \text{CtbLog2SizeY}) + 1$ ;
  - $\text{IsAvailable}[\text{cIdx}][x_{\text{NbY}}][y_{\text{NbY}}]$  is equal to  $\text{FALSE}$ ;
  - 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;
  - $\text{sps\_entropy\_coding\_sync\_enabled\_flag}$  is equal to 1 and  $(x_{\text{NbY}} \gg \text{CtbLog2SizeY})$  is greater than or equal to  $(x_{\text{Curr}} \gg \text{CtbLog2SizeY}) + 1$ ;
- Otherwise,  $\text{availableN}$  is set equal to  $\text{TRUE}$ .

When all of the following conditions are true,  $\text{availableN}$  is set equal to  $\text{FALSE}$ :

- 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 ] )
        tileX++
    CtbToTileColBd[ ctbAddrX ] = TileColBdVal[ tileX ]
    ctbToTileColIdx[ ctbAddrX ] = tileX
}

```

(18)

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 ] )
        tileY++
    CtbToTileRowBd[ ctbAddrY ] = TileRowBdVal[ tileY ]
    ctbToTileRowIdx[ ctbAddrY ] = tileY
}

```

(19)

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 ]
    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
}

```

(20)

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 ) {
            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 ] ) &&
            ( 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 ]++
        }
    }
}

```

(23)

### 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 ) {
            diagScan[ i ][ 0 ] = x
            diagScan[ i ][ 1 ] = y
            i++
        }
        y--
        x++
    }
    y = x
    x = 0
    if( i >= blkWidth * blkHeight )
        stopLoop = TRUE
}

```

(24)

### 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

$(\text{blkWidth} * \text{blkHeight}) - 1$ , 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  $\text{blkWidth}$  and  $\text{blkHeight}$ , the array  $hTravScan$  and  $vTravScan$  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
        for( x = blkWidth - 1; x >= 0; x-- ) {
            hTravScan[ i ][ 0 ] = x
            hTravScan[ i ][ 1 ] = y
            i++
        }

```

(25)

```

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
        for( y = blkHeight - 1; y >= 0; y-- ) {
            vTravScan[ i ][ 0 ] = x
            vTravScan[ i ][ 1 ] = y
            i++
        }

```

(26)

## 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 document.

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 */ |            |
| <b>syntax_element</b>                                                                                                                                                                                                | ue(k)      |
| conditioning statement                                                                                                                                                                                               |            |
| /* A group of statements enclosed in curly brackets is a compound statement and is treated functionally as a single statement. */                                                                                    |            |



|                                                                                                                                                                                                                                                                                                                                                                 |  |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| {                                                                                                                                                                                                                                                                                                                                                               |  |
| statement                                                                                                                                                                                                                                                                                                                                                       |  |
| statement                                                                                                                                                                                                                                                                                                                                                       |  |
| ...                                                                                                                                                                                                                                                                                                                                                             |  |
| }                                                                                                                                                                                                                                                                                                                                                               |  |
| /* A "while" structure specifies a test of whether a condition is true, and if true, specifies evaluation of a statement (or compound statement) repeatedly until the condition is no longer true */                                                                                                                                                            |  |
| while( condition )                                                                                                                                                                                                                                                                                                                                              |  |
| statement                                                                                                                                                                                                                                                                                                                                                       |  |
| /* A "do ... while" structure specifies evaluation of a statement once, followed by a test of whether a condition is true, and if true, specifies repeated evaluation of the statement until the condition is no longer true */                                                                                                                                 |  |
| do                                                                                                                                                                                                                                                                                                                                                              |  |
| statement                                                                                                                                                                                                                                                                                                                                                       |  |
| while( condition )                                                                                                                                                                                                                                                                                                                                              |  |
| /* An "if ... else" structure specifies a test of whether a condition is true and, if the condition is true, specifies evaluation of a primary statement, otherwise, specifies evaluation of an alternative statement. The "else" part of the structure and the associated alternative statement is omitted if no alternative statement evaluation is needed */ |  |
| if( condition )                                                                                                                                                                                                                                                                                                                                                 |  |
| primary statement                                                                                                                                                                                                                                                                                                                                               |  |
| else                                                                                                                                                                                                                                                                                                                                                            |  |
| alternative statement                                                                                                                                                                                                                                                                                                                                           |  |
| /* A "for" structure specifies evaluation of an initial statement, followed by a test of a condition, and if the condition is true, specifies repeated evaluation of a primary statement followed by a subsequent statement until the condition is no longer true. */                                                                                           |  |
| for( initial statement; condition; subsequent statement )                                                                                                                                                                                                                                                                                                       |  |
| primary statement                                                                                                                                                                                                                                                                                                                                               |  |

## 7.2 Specification of syntax functions and descriptors

The functions presented in this subclause 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  $8 * \text{payloadSize}$  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 (`rbbsp_stop_one_bit`) of the `rbbsp_trailing_bits()` syntax structure, the following applies:
  - If there is more data in an Rbsp before the `rbbsp_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  $8 * \text{payloadSize}$  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 subclause 9.3.
- `b(8)`: byte having any pattern of bit string (8 bits). The parsing process for this descriptor is specified by the return value of the function `read_bits(8)`.
- `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 subclause 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 subclause 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

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

##### 7.3.1.2 NAL unit header syntax

| <code>nal_unit_header( ) {</code>  | Descriptor        |
|------------------------------------|-------------------|
| <code>forbidden_zero_bit</code>    | <code>f(1)</code> |
| <code>nuh_reserved_zero_bit</code> | <code>u(1)</code> |
| <code>nuh_layer_id</code>          | <code>u(6)</code> |
| <code>nal_unit_type</code>         | <code>u(5)</code> |
| <code>nuh_temporal_id_plus1</code> | <code>u(3)</code> |
| <code>}</code>                     |                   |

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

##### 7.3.2.1 Decoding capability information RBSP syntax

| <code>decoding_capability_information_rbsp( ) {</code> | Descriptor        |
|--------------------------------------------------------|-------------------|
| <code>dci_reserved_zero_4bits</code>                   | <code>u(4)</code> |
| <code>dci_num_ptls_minus1</code>                       | <code>u(4)</code> |

|                                             |      |
|---------------------------------------------|------|
| for( i = 0; i <= dci_num_ptls_minus1; i++ ) |      |
| profile_tier_level( 1, 0 )                  |      |
| <b>dci_extension_flag</b>                   | u(1) |
| if( dci_extension_flag )                    |      |
| while( more_rbsp_data( ) )                  |      |
| <b>dci_extension_data_flag</b>              | u(1) |
| rbsp_trailing_bits( )                       |      |
| }                                           |      |

### 7.3.2.2 Operating point information RBSP syntax

|                                       |                   |
|---------------------------------------|-------------------|
| operating_point_information_rbsp( ) { | <b>Descriptor</b> |
| <b>opi_ols_info_present_flag</b>      | u(1)              |
| <b>opi_htid_info_present_flag</b>     | u(1)              |
| if( opi_ols_info_present_flag )       |                   |
| <b>opi_ols_idx</b>                    | ue(v)             |
| if( opi_htid_info_present_flag )      |                   |
| <b>opi_htid_plus1</b>                 | u(3)              |
| <b>opi_extension_flag</b>             | u(1)              |
| if( opi_extension_flag )              |                   |
| while( more_rbsp_data( ) )            |                   |
| <b>opi_extension_data_flag</b>        | u(1)              |
| rbsp_trailing_bits( )                 |                   |
| }                                     |                   |

### 7.3.2.3 Video parameter set RBSP syntax

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

|                                                                                 |       |
|---------------------------------------------------------------------------------|-------|
| }                                                                               |       |
| if( vps_max_layers_minus1 > 0 ) {                                               |       |
| if( vps_all_independent_layers_flag )                                           |       |
| <b>vps_each_layer_is_an_ols_flag</b>                                            | u(1)  |
| if( !vps_each_layer_is_an_ols_flag ) {                                          |       |
| if( !vps_all_independent_layers_flag )                                          |       |
| <b>vps_ols_mode_idc</b>                                                         | u(2)  |
| if( vps_ols_mode_idc == 2 ) {                                                   |       |
| <b>vps_num_output_layer_sets_minus2</b>                                         | u(8)  |
| for( i = 1; i <= vps_num_output_layer_sets_minus2 + 1; i++ )                    |       |
| for( j = 0; j <= vps_max_layers_minus1; j++ )                                   |       |
| <b>vps_ols_output_layer_flag[ i ][ j ]</b>                                      | u(1)  |
| }                                                                               |       |
| }                                                                               |       |
| <b>vps_num_ptls_minus1</b>                                                      | u(8)  |
| }                                                                               |       |
| for( i = 0; i <= vps_num_ptls_minus1; i++ ) {                                   |       |
| if( i > 0 )                                                                     |       |
| <b>vps_pt_present_flag[ i ]</b>                                                 | u(1)  |
| if( !vps_default_ptl_dpb_hrd_max_tid_flag )                                     |       |
| <b>vps_ptl_max_tid[ i ]</b>                                                     | u(3)  |
| }                                                                               |       |
| while( !byte_aligned( ) )                                                       |       |
| <b>vps_ptl_alignment_zero_bit</b> /* 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 )        |       |
| <b>vps_ols_ptl_idx[ i ]</b>                                                     | u(8)  |
| if( !vps_each_layer_is_an_ols_flag ) {                                          |       |
| <b>vps_num_dpb_params_minus1</b>                                                | ue(v) |
| if( vps_max_sublayers_minus1 > 0 )                                              |       |
| <b>vps_sublayer_dpb_params_present_flag</b>                                     | u(1)  |
| for( i = 0; i < VpsNumDpbParams; i++ ) {                                        |       |
| if( !vps_default_ptl_dpb_hrd_max_tid_flag )                                     |       |
| <b>vps_dpb_max_tid[ i ]</b>                                                     | u(3)  |
| dpb_parameters( vps_dpb_max_tid[ i ],<br>vps_sublayer_dpb_params_present_flag ) |       |
| }                                                                               |       |
| for( i = 0; i < NumMultiLayerOlss; i++ ) {                                      |       |
| <b>vps_ols_dpb_pic_width[ i ]</b>                                               | ue(v) |
| <b>vps_ols_dpb_pic_height[ i ]</b>                                              | ue(v) |
| <b>vps_ols_dpb_chroma_format[ i ]</b>                                           | u(2)  |
| <b>vps_ols_dpb_bitdepth_minus8[ i ]</b>                                         | ue(v) |
| if( VpsNumDpbParams > 1 && VpsNumDpbParams != NumMultiLayerOlss )               |       |
| <b>vps_ols_dpb_params_idx[ i ]</b>                                              | ue(v) |
| }                                                                               |       |
| <b>vps_timing_hrd_params_present_flag</b>                                       | u(1)  |
| if( vps_timing_hrd_params_present_flag ) {                                      |       |
| general_timing_hrd_parameters( )                                                |       |

|                                                                                                                    |       |
|--------------------------------------------------------------------------------------------------------------------|-------|
| if( vps_max_sublayers_minus1 > 0 )                                                                                 |       |
| <b>vps_sublayer_cpb_params_present_flag</b>                                                                        | u(1)  |
| <b>vps_num_ols_timing_hrd_params_minus1</b>                                                                        | ue(v) |
| for( i = 0; i <= vps_num_ols_timing_hrd_params_minus1; i++ ) {                                                     |       |
| if( !vps_default_ptl_dpb_hrd_max_tid_flag )                                                                        |       |
| <b>vps_hrd_max_tid[ i ]</b>                                                                                        | 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 &&<br>vps_num_ols_timing_hrd_params_minus1 + 1 != NumMultiLayerOlss ) |       |
| for( i = 0; i < NumMultiLayerOlss; i++ )                                                                           |       |
| <b>vps_ols_timing_hrd_idx[ i ]</b>                                                                                 | ue(v) |
| }                                                                                                                  |       |
| }                                                                                                                  |       |
| <b>vps_extension_flag</b>                                                                                          | u(1)  |
| if( vps_extension_flag )                                                                                           |       |
| while( more_rbsp_data( ) )                                                                                         |       |
| <b>vps_extension_data_flag</b>                                                                                     | u(1)  |
| rbp_trailing_bits( )                                                                                               |       |
| }                                                                                                                  |       |

#### 7.3.2.4 Sequence parameter set RBSP syntax

|                                                   |                   |
|---------------------------------------------------|-------------------|
| seq_parameter_set_rbsp( ) {                       | <b>Descriptor</b> |
| <b>sps_seq_parameter_set_id</b>                   | u(4)              |
| <b>sps_video_parameter_set_id</b>                 | u(4)              |
| <b>sps_max_sublayers_minus1</b>                   | u(3)              |
| <b>sps_chroma_format_idc</b>                      | u(2)              |
| <b>sps_log2_ctu_size_minus5</b>                   | u(2)              |
| <b>sps_ptl_dpb_hrd_params_present_flag</b>        | u(1)              |
| if( sps_ptl_dpb_hrd_params_present_flag )         |                   |
| profile_tier_level( 1, sps_max_sublayers_minus1 ) |                   |
| <b>sps_gdr_enabled_flag</b>                       | u(1)              |
| <b>sps_ref_pic_resampling_enabled_flag</b>        | u(1)              |
| if( sps_ref_pic_resampling_enabled_flag )         |                   |
| <b>sps_res_change_in_clvs_allowed_flag</b>        | u(1)              |
| <b>sps_pic_width_max_in_luma_samples</b>          | ue(v)             |
| <b>sps_pic_height_max_in_luma_samples</b>         | ue(v)             |
| <b>sps_conformance_window_flag</b>                | u(1)              |
| if( sps_conformance_window_flag ) {               |                   |
| <b>sps_conf_win_left_offset</b>                   | ue(v)             |
| <b>sps_conf_win_right_offset</b>                  | ue(v)             |
| <b>sps_conf_win_top_offset</b>                    | ue(v)             |
| <b>sps_conf_win_bottom_offset</b>                 | ue(v)             |
| }                                                 |                   |
| <b>sps_subpic_info_present_flag</b>               | u(1)              |
| if( sps_subpic_info_present_flag ) {              |                   |
| <b>sps_num_subpics_minus1</b>                     | ue(v)             |

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

|                                                                                                 |       |
|-------------------------------------------------------------------------------------------------|-------|
| <b>sps_partition_constraints_override_enabled_flag</b>                                          | u(1)  |
| <b>sps_log2_diff_min_qt_min_cb_intra_slice_luma</b>                                             | ue(v) |
| <b>sps_max_mtt_hierarchy_depth_intra_slice_luma</b>                                             | ue(v) |
| if( sps_max_mtt_hierarchy_depth_intra_slice_luma != 0 ) {                                       |       |
| <b>sps_log2_diff_max_bt_min_qt_intra_slice_luma</b>                                             | ue(v) |
| <b>sps_log2_diff_max_tt_min_qt_intra_slice_luma</b>                                             | ue(v) |
| }                                                                                               |       |
| if( sps_chroma_format_idc != 0 )                                                                |       |
| <b>sps_qtbtt_dual_tree_intra_flag</b>                                                           | u(1)  |
| if( sps_qtbtt_dual_tree_intra_flag ) {                                                          |       |
| <b>sps_log2_diff_min_qt_min_cb_intra_slice_chroma</b>                                           | ue(v) |
| <b>sps_max_mtt_hierarchy_depth_intra_slice_chroma</b>                                           | ue(v) |
| if( sps_max_mtt_hierarchy_depth_intra_slice_chroma != 0 ) {                                     |       |
| <b>sps_log2_diff_max_bt_min_qt_intra_slice_chroma</b>                                           | ue(v) |
| <b>sps_log2_diff_max_tt_min_qt_intra_slice_chroma</b>                                           | ue(v) |
| }                                                                                               |       |
| }                                                                                               |       |
| <b>sps_log2_diff_min_qt_min_cb_inter_slice</b>                                                  | ue(v) |
| <b>sps_max_mtt_hierarchy_depth_inter_slice</b>                                                  | ue(v) |
| if( sps_max_mtt_hierarchy_depth_inter_slice != 0 ) {                                            |       |
| <b>sps_log2_diff_max_bt_min_qt_inter_slice</b>                                                  | ue(v) |
| <b>sps_log2_diff_max_tt_min_qt_inter_slice</b>                                                  | ue(v) |
| }                                                                                               |       |
| if( CtbSizeY > 32 )                                                                             |       |
| <b>sps_max_luma_transform_size_64_flag</b>                                                      | u(1)  |
| <b>sps_transform_skip_enabled_flag</b>                                                          | u(1)  |
| if( sps_transform_skip_enabled_flag ) {                                                         |       |
| <b>sps_log2_transform_skip_max_size_minus2</b>                                                  | ue(v) |
| <b>sps_bdpcm_enabled_flag</b>                                                                   | u(1)  |
| }                                                                                               |       |
| <b>sps_mts_enabled_flag</b>                                                                     | u(1)  |
| if( sps_mts_enabled_flag ) {                                                                    |       |
| <b>sps_explicit_mts_intra_enabled_flag</b>                                                      | u(1)  |
| <b>sps_explicit_mts_inter_enabled_flag</b>                                                      | u(1)  |
| }                                                                                               |       |
| <b>sps_lfnst_enabled_flag</b>                                                                   | u(1)  |
| if( sps_chroma_format_idc != 0 ) {                                                              |       |
| <b>sps_joint_cbr_enabled_flag</b>                                                               | u(1)  |
| <b>sps_same_qp_table_for_chroma_flag</b>                                                        | u(1)  |
| numQpTables = sps_same_qp_table_for_chroma_flag ? 1 :<br>( sps_joint_cbr_enabled_flag ? 3 : 2 ) |       |
| for( i = 0; i < numQpTables; i++ ) {                                                            |       |
| <b>sps_qp_table_start_minus26[ i ]</b>                                                          | se(v) |
| <b>sps_num_points_in_qp_table_minus1[ i ]</b>                                                   | ue(v) |
| for( j = 0; j <= sps_num_points_in_qp_table_minus1[ i ]; j++ ) {                                |       |
| <b>sps_delta_qp_in_val_minus1[ i ][ j ]</b>                                                     | ue(v) |
| <b>sps_delta_qp_diff_val[ i ][ j ]</b>                                                          | ue(v) |
| }                                                                                               |       |
| }                                                                                               |       |
| }                                                                                               |       |

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



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

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

### 7.3.2.5 Picture parameter set RBSP syntax

|                                                     |                   |
|-----------------------------------------------------|-------------------|
| pic_parameter_set_rbsp( ) {                         | <b>Descriptor</b> |
| <b>pps_pic_parameter_set_id</b>                     | u(6)              |
| <b>pps_seq_parameter_set_id</b>                     | u(4)              |
| <b>pps_mixed_nalu_types_in_pic_flag</b>             | u(1)              |
| <b>pps_pic_width_in_luma_samples</b>                | ue(v)             |
| <b>pps_pic_height_in_luma_samples</b>               | ue(v)             |
| <b>pps_conformance_window_flag</b>                  | u(1)              |
| if( pps_conformance_window_flag ) {                 |                   |
| <b>pps_conf_win_left_offset</b>                     | ue(v)             |
| <b>pps_conf_win_right_offset</b>                    | ue(v)             |
| <b>pps_conf_win_top_offset</b>                      | ue(v)             |
| <b>pps_conf_win_bottom_offset</b>                   | ue(v)             |
| }                                                   |                   |
| <b>pps_scaling_window_explicit_signalling_flag</b>  | u(1)              |
| if( pps_scaling_window_explicit_signalling_flag ) { |                   |
| <b>pps_scaling_win_left_offset</b>                  | se(v)             |

|                                                                                                                                                                        |       |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| <b>pps_scaling_win_right_offset</b>                                                                                                                                    | se(v) |
| <b>pps_scaling_win_top_offset</b>                                                                                                                                      | se(v) |
| <b>pps_scaling_win_bottom_offset</b>                                                                                                                                   | se(v) |
| }                                                                                                                                                                      |       |
| <b>pps_output_flag_present_flag</b>                                                                                                                                    | u(1)  |
| <b>pps_no_pic_partition_flag</b>                                                                                                                                       | u(1)  |
| <b>pps_subpic_id_mapping_present_flag</b>                                                                                                                              | u(1)  |
| if( pps_subpic_id_mapping_present_flag ) {                                                                                                                             |       |
| if( !pps_no_pic_partition_flag )                                                                                                                                       |       |
| <b>pps_num_subpics_minus1</b>                                                                                                                                          | ue(v) |
| <b>pps_subpic_id_len_minus1</b>                                                                                                                                        | ue(v) |
| for( i = 0; i <= pps_num_subpics_minus1; i++ )                                                                                                                         |       |
| <b>pps_subpic_id[ i ]</b>                                                                                                                                              | u(v)  |
| }                                                                                                                                                                      |       |
| if( !pps_no_pic_partition_flag ) {                                                                                                                                     |       |
| <b>pps_log2_ctu_size_minus5</b>                                                                                                                                        | u(2)  |
| <b>pps_num_exp_tile_columns_minus1</b>                                                                                                                                 | ue(v) |
| <b>pps_num_exp_tile_rows_minus1</b>                                                                                                                                    | ue(v) |
| for( i = 0; i <= pps_num_exp_tile_columns_minus1; i++ )                                                                                                                |       |
| <b>pps_tile_column_width_minus1[ i ]</b>                                                                                                                               | ue(v) |
| for( i = 0; i <= pps_num_exp_tile_rows_minus1; i++ )                                                                                                                   |       |
| <b>pps_tile_row_height_minus1[ i ]</b>                                                                                                                                 | ue(v) |
| if( NumTilesInPic > 1 ) {                                                                                                                                              |       |
| <b>pps_loop_filter_across_tiles_enabled_flag</b>                                                                                                                       | u(1)  |
| <b>pps_rect_slice_flag</b>                                                                                                                                             | u(1)  |
| }                                                                                                                                                                      |       |
| if( pps_rect_slice_flag )                                                                                                                                              |       |
| <b>pps_single_slice_per_subpic_flag</b>                                                                                                                                | u(1)  |
| if( pps_rect_slice_flag && !pps_single_slice_per_subpic_flag ) {                                                                                                       |       |
| <b>pps_num_slices_in_pic_minus1</b>                                                                                                                                    | ue(v) |
| if( pps_num_slices_in_pic_minus1 > 1 )                                                                                                                                 |       |
| <b>pps_tile_idx_delta_present_flag</b>                                                                                                                                 | u(1)  |
| for( i = 0; i < pps_num_slices_in_pic_minus1; i++ ) {                                                                                                                  |       |
| if( SliceTopLeftTileIdx[ i ] % NumTileColumns != NumTileColumns - 1 )                                                                                                  |       |
| <b>pps_slice_width_in_tiles_minus1[ i ]</b>                                                                                                                            | ue(v) |
| if( SliceTopLeftTileIdx[ i ] / NumTileColumns != NumTileRows - 1 &&<br>( pps_tile_idx_delta_present_flag   <br>SliceTopLeftTileIdx[ i ] % NumTileColumns == 0 ) )      |       |
| <b>pps_slice_height_in_tiles_minus1[ i ]</b>                                                                                                                           | ue(v) |
| if( pps_slice_width_in_tiles_minus1[ i ] == 0 &&<br>pps_slice_height_in_tiles_minus1[ i ] == 0 &&<br>RowHeightVal[ SliceTopLeftTileIdx[ i ] / NumTileColumns ] > 1 ) { |       |
| <b>pps_num_exp_slices_in_tile[ i ]</b>                                                                                                                                 | ue(v) |
| for( j = 0; j < pps_num_exp_slices_in_tile[ i ]; j++ )                                                                                                                 |       |
| <b>pps_exp_slice_height_in_ctus_minus1[ i ][ j ]</b>                                                                                                                   | ue(v) |
| i += NumSlicesInTile[ i ] - 1                                                                                                                                          |       |
| }                                                                                                                                                                      |       |
| if( pps_tile_idx_delta_present_flag && i < pps_num_slices_in_pic_minus1 )                                                                                              |       |
| <b>pps_tile_idx_delta_val[ i ]</b>                                                                                                                                     | se(v) |
| }                                                                                                                                                                      |       |
| }                                                                                                                                                                      |       |

|                                                                                                       |       |
|-------------------------------------------------------------------------------------------------------|-------|
| if( !pps_rect_slice_flag    pps_single_slice_per_subpic_flag   <br>pps_num_slices_in_pic_minus1 > 0 ) |       |
| <b>pps_loop_filter_across_slices_enabled_flag</b>                                                     | u(1)  |
| }                                                                                                     |       |
| <b>pps_cabac_init_present_flag</b>                                                                    | u(1)  |
| for( i = 0; i < 2; i++ )                                                                              |       |
| <b>pps_num_ref_idx_default_active_minus1[ i ]</b>                                                     | ue(v) |
| <b>pps_rpl1_idx_present_flag</b>                                                                      | u(1)  |
| <b>pps_weighted_pred_flag</b>                                                                         | u(1)  |
| <b>pps_weighted_bipred_flag</b>                                                                       | u(1)  |
| <b>pps_ref_wraparound_enabled_flag</b>                                                                | u(1)  |
| if( pps_ref_wraparound_enabled_flag )                                                                 |       |
| <b>pps_pic_width_minus_wraparound_offset</b>                                                          | ue(v) |
| <b>pps_init_qp_minus26</b>                                                                            | se(v) |
| <b>pps_cu_qp_delta_enabled_flag</b>                                                                   | u(1)  |
| <b>pps_chroma_tool_offsets_present_flag</b>                                                           | u(1)  |
| if( pps_chroma_tool_offsets_present_flag ) {                                                          |       |
| <b>pps_cb_qp_offset</b>                                                                               | se(v) |
| <b>pps_cr_qp_offset</b>                                                                               | se(v) |
| <b>pps_joint_cbr_qp_offset_present_flag</b>                                                           | u(1)  |
| if( pps_joint_cbr_qp_offset_present_flag )                                                            |       |
| <b>pps_joint_cbr_qp_offset_value</b>                                                                  | se(v) |
| <b>pps_slice_chroma_qp_offsets_present_flag</b>                                                       | u(1)  |
| <b>pps_cu_chroma_qp_offset_list_enabled_flag</b>                                                      | u(1)  |
| if( pps_cu_chroma_qp_offset_list_enabled_flag ) {                                                     |       |
| <b>pps_chroma_qp_offset_list_len_minus1</b>                                                           | ue(v) |
| for( i = 0; i <= pps_chroma_qp_offset_list_len_minus1; i++ ) {                                        |       |
| <b>pps_cb_qp_offset_list[ i ]</b>                                                                     | se(v) |
| <b>pps_cr_qp_offset_list[ i ]</b>                                                                     | se(v) |
| if( pps_joint_cbr_qp_offset_present_flag )                                                            |       |
| <b>pps_joint_cbr_qp_offset_list[ i ]</b>                                                              | se(v) |
| }                                                                                                     |       |
| }                                                                                                     |       |
| }                                                                                                     |       |
| <b>pps_deblocking_filter_control_present_flag</b>                                                     | u(1)  |
| if( pps_deblocking_filter_control_present_flag ) {                                                    |       |
| <b>pps_deblocking_filter_override_enabled_flag</b>                                                    | u(1)  |
| <b>pps_deblocking_filter_disabled_flag</b>                                                            | u(1)  |
| if( !pps_no_pic_partition_flag && pps_deblocking_filter_override_enabled_flag )                       |       |
| <b>pps_dbf_info_in_ph_flag</b>                                                                        | u(1)  |
| if( !pps_deblocking_filter_disabled_flag ) {                                                          |       |
| <b>pps_luma_beta_offset_div2</b>                                                                      | se(v) |
| <b>pps_luma_tc_offset_div2</b>                                                                        | se(v) |
| if( pps_chroma_tool_offsets_present_flag ) {                                                          |       |
| <b>pps_cb_beta_offset_div2</b>                                                                        | se(v) |
| <b>pps_cb_tc_offset_div2</b>                                                                          | se(v) |
| <b>pps_cr_beta_offset_div2</b>                                                                        | se(v) |
| <b>pps_cr_tc_offset_div2</b>                                                                          | 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 ) &&<br>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( ) {        | <b>Descriptor</b> |
| 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( ) {    | <b>Descriptor</b> |
| picture_header_structure( ) |                   |
| rbsp_trailing_bits( )       |                   |
| }                           |                   |

## 7.3.2.8 Picture header structure syntax

| picture_header_structure() {                            | Descriptor |
|---------------------------------------------------------|------------|
| <b>ph_gdr_or_irap_pic_flag</b>                          | u(1)       |
| <b>ph_non_ref_pic_flag</b>                              | u(1)       |
| if( ph_gdr_or_irap_pic_flag )                           |            |
| <b>ph_gdr_pic_flag</b>                                  | u(1)       |
| <b>ph_inter_slice_allowed_flag</b>                      | u(1)       |
| if( ph_inter_slice_allowed_flag )                       |            |
| <b>ph_intra_slice_allowed_flag</b>                      | u(1)       |
| <b>ph_pic_parameter_set_id</b>                          | ue(v)      |
| <b>ph_pic_order_cnt_lsb</b>                             | u(v)       |
| if( ph_gdr_pic_flag )                                   |            |
| <b>ph_recovery_poc_cnt</b>                              | ue(v)      |
| for( i = 0; i < NumExtraPhBits; i++ )                   |            |
| <b>ph_extra_bit[ i ]</b>                                | u(1)       |
| if( sps_poc_msb_cycle_flag ) {                          |            |
| <b>ph_poc_msb_cycle_present_flag</b>                    | u(1)       |
| if( ph_poc_msb_cycle_present_flag )                     |            |
| <b>ph_poc_msb_cycle_val</b>                             | u(v)       |
| }                                                       |            |
| if( sps_alf_enabled_flag && pps_alf_info_in_ph_flag ) { |            |
| <b>ph_alf_enabled_flag</b>                              | u(1)       |
| if( ph_alf_enabled_flag ) {                             |            |
| <b>ph_num_alf_aps_ids_luma</b>                          | u(3)       |
| for( i = 0; i < ph_num_alf_aps_ids_luma; i++ )          |            |
| <b>ph_alf_aps_id_luma[ i ]</b>                          | u(3)       |
| if( sps_chroma_format_idc != 0 ) {                      |            |
| <b>ph_alf_cb_enabled_flag</b>                           | u(1)       |
| <b>ph_alf_cr_enabled_flag</b>                           | u(1)       |
| }                                                       |            |
| if( ph_alf_cb_enabled_flag   ph_alf_cr_enabled_flag )   |            |
| <b>ph_alf_aps_id_chroma</b>                             | u(3)       |
| if( sps_ccalf_enabled_flag ) {                          |            |
| <b>ph_alf_cc_cb_enabled_flag</b>                        | u(1)       |
| if( ph_alf_cc_cb_enabled_flag )                         |            |
| <b>ph_alf_cc_cb_aps_id</b>                              | u(3)       |
| <b>ph_alf_cc_cr_enabled_flag</b>                        | u(1)       |
| if( ph_alf_cc_cr_enabled_flag )                         |            |
| <b>ph_alf_cc_cr_aps_id</b>                              | u(3)       |
| }                                                       |            |
| }                                                       |            |
| }                                                       |            |
| if( sps_lmcs_enabled_flag ) {                           |            |
| <b>ph_lmcs_enabled_flag</b>                             | u(1)       |
| if( ph_lmcs_enabled_flag ) {                            |            |
| <b>ph_lmcs_aps_id</b>                                   | u(2)       |
| if( sps_chroma_format_idc != 0 )                        |            |
| <b>ph_chroma_residual_scale_flag</b>                    | u(1)       |
| }                                                       |            |

|                                                                                     |       |
|-------------------------------------------------------------------------------------|-------|
| }                                                                                   |       |
| if( sps_explicit_scaling_list_enabled_flag ) {                                      |       |
| <b>ph_explicit_scaling_list_enabled_flag</b>                                        | u(1)  |
| if( ph_explicit_scaling_list_enabled_flag )                                         |       |
| <b>ph_scaling_list_aps_id</b>                                                       | u(3)  |
| }                                                                                   |       |
| if( sps_virtual_boundaries_enabled_flag && !sps_virtual_boundaries_present_flag ) { |       |
| <b>ph_virtual_boundaries_present_flag</b>                                           | u(1)  |
| if( ph_virtual_boundaries_present_flag ) {                                          |       |
| <b>ph_num_ver_virtual_boundaries</b>                                                | ue(v) |
| for( i = 0; i < ph_num_ver_virtual_boundaries; i++ )                                |       |
| <b>ph_virtual_boundary_pos_x_minus1[ i ]</b>                                        | ue(v) |
| <b>ph_num_hor_virtual_boundaries</b>                                                | ue(v) |
| for( i = 0; i < ph_num_hor_virtual_boundaries; i++ )                                |       |
| <b>ph_virtual_boundary_pos_y_minus1[ i ]</b>                                        | ue(v) |
| }                                                                                   |       |
| }                                                                                   |       |
| if( pps_output_flag_present_flag && !ph_non_ref_pic_flag )                          |       |
| <b>ph_pic_output_flag</b>                                                           | u(1)  |
| if( pps_rpl_info_in_ph_flag )                                                       |       |
| ref_pic_lists( )                                                                    |       |
| if( sps_partition_constraints_override_enabled_flag )                               |       |
| <b>ph_partition_constraints_override_flag</b>                                       | u(1)  |
| if( ph_intra_slice_allowed_flag ) {                                                 |       |
| if( ph_partition_constraints_override_flag ) {                                      |       |
| <b>ph_log2_diff_min_qt_min_cb_intra_slice_luma</b>                                  | ue(v) |
| <b>ph_max_mtt_hierarchy_depth_intra_slice_luma</b>                                  | ue(v) |
| if( ph_max_mtt_hierarchy_depth_intra_slice_luma != 0 ) {                            |       |
| <b>ph_log2_diff_max_bt_min_qt_intra_slice_luma</b>                                  | ue(v) |
| <b>ph_log2_diff_max_tt_min_qt_intra_slice_luma</b>                                  | ue(v) |
| }                                                                                   |       |
| if( sps_qtbt_dual_tree_intra_flag ) {                                               |       |
| <b>ph_log2_diff_min_qt_min_cb_intra_slice_chroma</b>                                | ue(v) |
| <b>ph_max_mtt_hierarchy_depth_intra_slice_chroma</b>                                | ue(v) |
| if( ph_max_mtt_hierarchy_depth_intra_slice_chroma != 0 ) {                          |       |
| <b>ph_log2_diff_max_bt_min_qt_intra_slice_chroma</b>                                | ue(v) |
| <b>ph_log2_diff_max_tt_min_qt_intra_slice_chroma</b>                                | ue(v) |
| }                                                                                   |       |
| }                                                                                   |       |
| }                                                                                   |       |
| if( pps_cu_qp_delta_enabled_flag )                                                  |       |
| <b>ph_cu_qp_delta_subdiv_intra_slice</b>                                            | ue(v) |
| if( pps_cu_chroma_qp_offset_list_enabled_flag )                                     |       |
| <b>ph_cu_chroma_qp_offset_subdiv_intra_slice</b>                                    | ue(v) |
| }                                                                                   |       |
| if( ph_inter_slice_allowed_flag ) {                                                 |       |
| if( ph_partition_constraints_override_flag ) {                                      |       |
| <b>ph_log2_diff_min_qt_min_cb_inter_slice</b>                                       | ue(v) |
| <b>ph_max_mtt_hierarchy_depth_inter_slice</b>                                       | ue(v) |
| if( ph_max_mtt_hierarchy_depth_inter_slice != 0 ) {                                 |       |

|                                                                                                                                                                                    |       |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| <b>ph_log2_diff_max_bt_min_qt_inter_slice</b>                                                                                                                                      | ue(v) |
| <b>ph_log2_diff_max_tt_min_qt_inter_slice</b>                                                                                                                                      | ue(v) |
| }                                                                                                                                                                                  |       |
| }                                                                                                                                                                                  |       |
| if( pps_cu_qp_delta_enabled_flag )                                                                                                                                                 |       |
| <b>ph_cu_qp_delta_subdiv_inter_slice</b>                                                                                                                                           | ue(v) |
| if( pps_cu_chroma_qp_offset_list_enabled_flag )                                                                                                                                    |       |
| <b>ph_cu_chroma_qp_offset_subdiv_inter_slice</b>                                                                                                                                   | ue(v) |
| if( sps_temporal_mvp_enabled_flag ) {                                                                                                                                              |       |
| <b>ph_temporal_mvp_enabled_flag</b>                                                                                                                                                | u(1)  |
| if( ph_temporal_mvp_enabled_flag && pps_rpl_info_in_ph_flag ) {                                                                                                                    |       |
| if( num_ref_entries[ 1 ][ RplIdx[ 1 ] ] > 0 )                                                                                                                                      |       |
| <b>ph_collocated_from_l0_flag</b>                                                                                                                                                  | u(1)  |
| if( ( ph_collocated_from_l0_flag &&<br>num_ref_entries[ 0 ][ RplIdx[ 0 ] ] > 1 )   <br>( !ph_collocated_from_l0_flag &&<br>num_ref_entries[ 1 ][ RplIdx[ 1 ] ] > 1 ) )             |       |
| <b>ph_collocated_ref_idx</b>                                                                                                                                                       | ue(v) |
| }                                                                                                                                                                                  |       |
| }                                                                                                                                                                                  |       |
| if( sps_mmvd_fullpel_only_enabled_flag )                                                                                                                                           |       |
| <b>ph_mmvd_fullpel_only_flag</b>                                                                                                                                                   | u(1)  |
| presenceFlag = 0                                                                                                                                                                   |       |
| if( !pps_rpl_info_in_ph_flag ) /* This condition is intentionally not merged into the<br>next,<br>to avoid possible interpretation of RplIdx[ i ] not having a specified value. */ |       |
| presenceFlag = 1                                                                                                                                                                   |       |
| else if( num_ref_entries[ 1 ][ RplIdx[ 1 ] ] > 0 )                                                                                                                                 |       |
| presenceFlag = 1                                                                                                                                                                   |       |
| if( presenceFlag ) {                                                                                                                                                               |       |
| <b>ph_mvd_l1_zero_flag</b>                                                                                                                                                         | u(1)  |
| if( sps_bdof_control_present_in_ph_flag )                                                                                                                                          |       |
| <b>ph_bdof_disabled_flag</b>                                                                                                                                                       | u(1)  |
| if( sps_dmvr_control_present_in_ph_flag )                                                                                                                                          |       |
| <b>ph_dmvr_disabled_flag</b>                                                                                                                                                       | u(1)  |
| }                                                                                                                                                                                  |       |
| if( sps_prof_control_present_in_ph_flag )                                                                                                                                          |       |
| <b>ph_prof_disabled_flag</b>                                                                                                                                                       | u(1)  |
| if( ( pps_weighted_pred_flag    pps_weighted_bipred_flag ) &&<br>pps_wp_info_in_ph_flag )                                                                                          |       |
| pred_weight_table( )                                                                                                                                                               |       |
| }                                                                                                                                                                                  |       |
| if( pps_qp_delta_info_in_ph_flag )                                                                                                                                                 |       |
| <b>ph_qp_delta</b>                                                                                                                                                                 | se(v) |
| if( sps_joint_cbr_enabled_flag )                                                                                                                                                   |       |
| <b>ph_joint_cbr_sign_flag</b>                                                                                                                                                      | u(1)  |
| if( sps_sao_enabled_flag && pps_sao_info_in_ph_flag ) {                                                                                                                            |       |
| <b>ph_sao_luma_enabled_flag</b>                                                                                                                                                    | u(1)  |
| if( sps_chroma_format_idc != 0 )                                                                                                                                                   |       |
| <b>ph_sao_chroma_enabled_flag</b>                                                                                                                                                  | u(1)  |
| }                                                                                                                                                                                  |       |



|                                                   |       |
|---------------------------------------------------|-------|
| if( pps_dbf_info_in_ph_flag ) {                   |       |
| <b>ph_deblocking_params_present_flag</b>          | u(1)  |
| if( ph_deblocking_params_present_flag ) {         |       |
| if( !pps_deblocking_filter_disabled_flag )        |       |
| <b>ph_deblocking_filter_disabled_flag</b>         | u(1)  |
| if( !ph_deblocking_filter_disabled_flag ) {       |       |
| <b>ph_luma_beta_offset_div2</b>                   | se(v) |
| <b>ph_luma_tc_offset_div2</b>                     | se(v) |
| if( pps_chroma_tool_offsets_present_flag ) {      |       |
| <b>ph_cb_beta_offset_div2</b>                     | se(v) |
| <b>ph_cb_tc_offset_div2</b>                       | se(v) |
| <b>ph_cr_beta_offset_div2</b>                     | se(v) |
| <b>ph_cr_tc_offset_div2</b>                       | se(v) |
| }                                                 |       |
| }                                                 |       |
| }                                                 |       |
| }                                                 |       |
| if( pps_picture_header_extension_present_flag ) { |       |
| <b>ph_extension_length</b>                        | ue(v) |
| for( i = 0; i < ph_extension_length; i++)         |       |
| <b>ph_extension_data_byte[ i ]</b>                | u(8)  |
| }                                                 |       |
| }                                                 |       |

### 7.3.2.9 Supplemental enhancement information RBSP syntax

|                            |                   |
|----------------------------|-------------------|
| sei_rbsp( ) {              | <b>Descriptor</b> |
| do                         |                   |
| sei_message( )             |                   |
| while( more_rbsp_data( ) ) |                   |
| rbsp_trailing_bits( )      |                   |
| }                          |                   |

### 7.3.2.10 AU delimiter RBSP syntax

|                                 |                   |
|---------------------------------|-------------------|
| access_unit_delimiter_rbsp( ) { | <b>Descriptor</b> |
| <b>aud_irap_or_gdr_flag</b>     | u(1)              |
| <b>aud_pic_type</b>             | u(3)              |
| rbsp_trailing_bits( )           |                   |
| }                               |                   |

### 7.3.2.11 End of sequence RBSP syntax

|                      |                   |
|----------------------|-------------------|
| end_of_seq_rbsp( ) { | <b>Descriptor</b> |
| }                    |                   |

## 7.3.2.12 End of bitstream RBSP syntax

|                           |                   |
|---------------------------|-------------------|
| end_of_bitstream_rbsp() { | <b>Descriptor</b> |
| }                         |                   |

## 7.3.2.13 Filler data RBSP syntax

|                                       |                   |
|---------------------------------------|-------------------|
| filler_data_rbsp() {                  | <b>Descriptor</b> |
| while( next_bits( 8 ) == 0xFF )       |                   |
| <b>fd_ff_byte</b> /* equal to 0xFF */ | f(8)              |
| rbsp_trailing_bits( )                 |                   |
| }                                     |                   |

## 7.3.2.14 Slice layer RBSP syntax

|                             |                   |
|-----------------------------|-------------------|
| slice_layer_rbsp() {        | <b>Descriptor</b> |
| slice_header( )             |                   |
| slice_data( )               |                   |
| rbsp_slice_trailing_bits( ) |                   |
| }                           |                   |

## 7.3.2.15 RBSP slice trailing bits syntax

|                                                   |                   |
|---------------------------------------------------|-------------------|
| rbsp_slice_trailing_bits() {                      | <b>Descriptor</b> |
| rbsp_trailing_bits( )                             |                   |
| while( more_rbsp_trailing_data( ) )               |                   |
| <b>rbsp_cabac_zero_word</b> /* equal to 0x0000 */ | f(16)             |
| }                                                 |                   |

## 7.3.2.16 RBSP trailing bits syntax

|                                                 |                   |
|-------------------------------------------------|-------------------|
| rbsp_trailing_bits() {                          | <b>Descriptor</b> |
| <b>rbsp_stop_one_bit</b> /* equal to 1 */       | f(1)              |
| while( !byte_aligned( ) )                       |                   |
| <b>rbsp_alignment_zero_bit</b> /* equal to 0 */ | f(1)              |
| }                                               |                   |

## 7.3.2.17 Byte alignment syntax

|                                                          |                   |
|----------------------------------------------------------|-------------------|
| byte_alignment() {                                       | <b>Descriptor</b> |
| <b>byte_alignment_bit_equal_to_one</b> /* equal to 1 */  | f(1)              |
| while( !byte_aligned() )                                 |                   |
| <b>byte_alignment_bit_equal_to_zero</b> /* equal to 0 */ | f(1)              |
| }                                                        |                   |

## 7.3.2.18 Adaptive loop filter data syntax

|                                                                              |                   |
|------------------------------------------------------------------------------|-------------------|
| alf_data() {                                                                 | <b>Descriptor</b> |
| <b>alf_luma_filter_signal_flag</b>                                           | u(1)              |
| if( aps_chroma_present_flag ) {                                              |                   |
| <b>alf_chroma_filter_signal_flag</b>                                         | u(1)              |
| <b>alf_cc_cb_filter_signal_flag</b>                                          | u(1)              |
| <b>alf_cc_cr_filter_signal_flag</b>                                          | u(1)              |
| }                                                                            |                   |
| if( alf_luma_filter_signal_flag ) {                                          |                   |
| <b>alf_luma_clip_flag</b>                                                    | u(1)              |
| <b>alf_luma_num_filters_signalled_minus1</b>                                 | ue(v)             |
| if( alf_luma_num_filters_signalled_minus1 > 0 )                              |                   |
| for( filtIdx = 0; filtIdx < NumAlfFilters; filtIdx++ )                       |                   |
| <b>alf_luma_coeff_delta_idx[ filtIdx ]</b>                                   | u(v)              |
| for( sflIdx = 0; sflIdx <= alf_luma_num_filters_signalled_minus1; sflIdx++ ) |                   |
| for( j = 0; j < 12; j++ ) {                                                  |                   |
| <b>alf_luma_coeff_abs[ sflIdx ][ j ]</b>                                     | ue(v)             |
| if( alf_luma_coeff_abs[ sflIdx ][ j ] )                                      |                   |
| <b>alf_luma_coeff_sign[ sflIdx ][ j ]</b>                                    | u(1)              |
| }                                                                            |                   |
| if( alf_luma_clip_flag )                                                     |                   |
| for( sflIdx = 0; sflIdx <= alf_luma_num_filters_signalled_minus1; sflIdx++ ) |                   |
| for( j = 0; j < 12; j++ )                                                    |                   |
| <b>alf_luma_clip_idx[ sflIdx ][ j ]</b>                                      | u(2)              |
| }                                                                            |                   |
| if( alf_chroma_filter_signal_flag ) {                                        |                   |
| <b>alf_chroma_clip_flag</b>                                                  | u(1)              |
| <b>alf_chroma_num_alt_filters_minus1</b>                                     | ue(v)             |
| for( altIdx = 0; altIdx <= alf_chroma_num_alt_filters_minus1; altIdx++ ) {   |                   |
| for( j = 0; j < 6; j++ ) {                                                   |                   |
| <b>alf_chroma_coeff_abs[ altIdx ][ j ]</b>                                   | ue(v)             |
| if( alf_chroma_coeff_abs[ altIdx ][ j ] > 0 )                                |                   |
| <b>alf_chroma_coeff_sign[ altIdx ][ j ]</b>                                  | u(1)              |
| }                                                                            |                   |
| if( alf_chroma_clip_flag )                                                   |                   |
| for( j = 0; j < 6; j++ )                                                     |                   |
| <b>alf_chroma_clip_idx[ altIdx ][ j ]</b>                                    | u(2)              |
| }                                                                            |                   |
| }                                                                            |                   |
| if( alf_cc_cb_filter_signal_flag ) {                                         |                   |

|                                                                 |       |
|-----------------------------------------------------------------|-------|
| <b>alf_cc_cb_filters_signalled_minus1</b>                       | ue(v) |
| for( k = 0; k < alf_cc_cb_filters_signalled_minus1 + 1; k++ ) { |       |
| for( j = 0; j < 7; j++ ) {                                      |       |
| <b>alf_cc_cb_mapped_coeff_abs[ k ][ j ]</b>                     | u(3)  |
| if( alf_cc_cb_mapped_coeff_abs[ k ][ j ] )                      |       |
| <b>alf_cc_cb_coeff_sign[ k ][ j ]</b>                           | u(1)  |
| }                                                               |       |
| }                                                               |       |
| if( alf_cc_cr_filter_signal_flag ) {                            |       |
| <b>alf_cc_cr_filters_signalled_minus1</b>                       | ue(v) |
| for( k = 0; k < alf_cc_cr_filters_signalled_minus1 + 1; k++ ) { |       |
| for( j = 0; j < 7; j++ ) {                                      |       |
| <b>alf_cc_cr_mapped_coeff_abs[ k ][ j ]</b>                     | u(3)  |
| if( alf_cc_cr_mapped_coeff_abs[ k ][ j ] )                      |       |
| <b>alf_cc_cr_coeff_sign[ k ][ j ]</b>                           | u(1)  |
| }                                                               |       |
| }                                                               |       |
| }                                                               |       |

### 7.3.2.19 Luma mapping with chroma scaling data syntax

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

### 7.3.2.20 Scaling list data syntax

|                                                            |                   |
|------------------------------------------------------------|-------------------|
| <b>scaling_list_data( ) {</b>                              | <b>Descriptor</b> |
| for( id = 0; id < 28; id ++ ) {                            |                   |
| matrixSize = id < 2 ? 2 : ( id < 8 ? 4 : 8 )               |                   |
| if( aps_chroma_present_flag    id % 3 == 2    id == 27 ) { |                   |
| <b>scaling_list_copy_mode_flag[ id ]</b>                   | u(1)              |
| if( !scaling_list_copy_mode_flag[ id ] )                   |                   |
| <b>scaling_list_pred_mode_flag[ id ]</b>                   | u(1)              |

|                                                                                                                   |       |
|-------------------------------------------------------------------------------------------------------------------|-------|
| if( ( scaling_list_copy_mode_flag[ id ]    scaling_list_pred_mode_flag[ id ] ) && id != 0 && id != 2 && id != 8 ) |       |
| <b>scaling_list_pred_id_delta</b> [ id ]                                                                          | ue(v) |
| if( !scaling_list_copy_mode_flag[ id ] ) {                                                                        |       |
| nextCoef = 0                                                                                                      |       |
| if( id > 13 ) {                                                                                                   |       |
| <b>scaling_list_dc_coef</b> [ 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 ) ) {                                                                          |       |
| <b>scaling_list_delta_coef</b> [ 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 ) {                                                        | <b>Descriptor</b> |
| 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( ) )                                                  |                   |
| <b>vui_reserved_payload_extension_data</b>                                          | u(v)              |
| <b>vui_payload_bit_equal_to_one</b> /* equal to 1 */                                | f(1)              |
| while( !byte_aligned( ) )                                                           |                   |
| <b>vui_payload_bit_equal_to_zero</b> /* equal to 0 */                               | f(1)              |
| }                                                                                   |                   |
| }                                                                                   |                   |

## 7.3.2.22 Sequence parameter set range extension syntax

| sps_range_extension( ) {                              | Descriptor |
|-------------------------------------------------------|------------|
| <b>sps_extended_precision_flag</b>                    | u(1)       |
| if( sps_transform_skip_enabled_flag )                 |            |
| <b>sps_ts_residual_coding_rice_present_in_sh_flag</b> | u(1)       |
| <b>sps_rrc_rice_extension_flag</b>                    | u(1)       |
| <b>sps_persistent_rice_adaptation_enabled_flag</b>    | u(1)       |
| <b>sps_reverse_last_sig_coeff_enabled_flag</b>        | 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 ) {                                        |            |
| <b>general_profile_idc</b>                                            | u(7)       |
| <b>general_tier_flag</b>                                              | u(1)       |
| }                                                                     |            |
| <b>general_level_idc</b>                                              | u(8)       |
| <b>ptl_frame_only_constraint_flag</b>                                 | u(1)       |
| <b>ptl_multilayer_enabled_flag</b>                                    | u(1)       |
| if( profileTierPresentFlag )                                          |            |
| <b>general_constraints_info( )</b>                                    |            |
| for( i = MaxNumSubLayersMinus1 - 1; i >= 0; i-- )                     |            |
| <b>ptl_sublayer_level_present_flag[ i ]</b>                           | u(1)       |
| while( !byte_aligned( ) )                                             |            |
| <b>ptl_reserved_zero_bit</b>                                          | u(1)       |
| for( i = MaxNumSubLayersMinus1 - 1; i >= 0; i-- )                     |            |
| if( ptl_sublayer_level_present_flag[ i ] )                            |            |
| <b>sublayer_level_idc[ i ]</b>                                        | u(8)       |
| if( profileTierPresentFlag ) {                                        |            |
| <b>ptl_num_sub_profiles</b>                                           | u(8)       |
| for( i = 0; i < ptl_num_sub_profiles; i++ )                           |            |
| <b>general_sub_profile_idc[ i ]</b>                                   | u(32)      |
| }                                                                     |            |
| }                                                                     |            |

## 7.3.3.2 General constraints information syntax

| general_constraints_info( ) {                     | Descriptor |
|---------------------------------------------------|------------|
| <b>gci_present_flag</b>                           | u(1)       |
| if( gci_present_flag ) {                          |            |
| /* general */                                     |            |
| <b>gci_intra_only_constraint_flag</b>             | u(1)       |
| <b>gci_all_layers_independent_constraint_flag</b> | u(1)       |
| <b>gci_one_au_only_constraint_flag</b>            | u(1)       |

|                                                              |      |
|--------------------------------------------------------------|------|
| /* picture format */                                         |      |
| <b>gci_sixteen_minus_max_bitdepth_constraint_idc</b>         | u(4) |
| <b>gci_three_minus_max_chroma_format_constraint_idc</b>      | u(2) |
| /* NAL unit type related */                                  |      |
| <b>gci_no_mixed_nalu_types_in_pic_constraint_flag</b>        | u(1) |
| <b>gci_no_trail_constraint_flag</b>                          | u(1) |
| <b>gci_no_stsa_constraint_flag</b>                           | u(1) |
| <b>gci_no_rasl_constraint_flag</b>                           | u(1) |
| <b>gci_no_radl_constraint_flag</b>                           | u(1) |
| <b>gci_no_idr_constraint_flag</b>                            | u(1) |
| <b>gci_no_cra_constraint_flag</b>                            | u(1) |
| <b>gci_no_gdr_constraint_flag</b>                            | u(1) |
| <b>gci_no_aps_constraint_flag</b>                            | u(1) |
| <b>gci_no_idr_rpl_constraint_flag</b>                        | u(1) |
| /* tile, slice, subpicture partitioning */                   |      |
| <b>gci_one_tile_per_pic_constraint_flag</b>                  | u(1) |
| <b>gci_pic_header_in_slice_header_constraint_flag</b>        | u(1) |
| <b>gci_one_slice_per_pic_constraint_flag</b>                 | u(1) |
| <b>gci_no_rectangular_slice_constraint_flag</b>              | u(1) |
| <b>gci_one_slice_per_subpic_constraint_flag</b>              | u(1) |
| <b>gci_no_subpic_info_constraint_flag</b>                    | u(1) |
| /* CTU and block partitioning */                             |      |
| <b>gci_three_minus_max_log2_ctu_size_constraint_idc</b>      | u(2) |
| <b>gci_no_partition_constraints_override_constraint_flag</b> | u(1) |
| <b>gci_no_mtt_constraint_flag</b>                            | u(1) |
| <b>gci_no_qtbtt_dual_tree_intra_constraint_flag</b>          | u(1) |
| /* intra */                                                  |      |
| <b>gci_no_palette_constraint_flag</b>                        | u(1) |
| <b>gci_no_ibc_constraint_flag</b>                            | u(1) |
| <b>gci_no_isp_constraint_flag</b>                            | u(1) |
| <b>gci_no_mrl_constraint_flag</b>                            | u(1) |
| <b>gci_no_mip_constraint_flag</b>                            | u(1) |
| <b>gci_no_cclm_constraint_flag</b>                           | u(1) |
| /* inter */                                                  |      |
| <b>gci_no_ref_pic_resampling_constraint_flag</b>             | u(1) |
| <b>gci_no_res_change_in_clvs_constraint_flag</b>             | u(1) |
| <b>gci_no_weighted_prediction_constraint_flag</b>            | u(1) |
| <b>gci_no_ref_wraparound_constraint_flag</b>                 | u(1) |
| <b>gci_no_temporal_mvp_constraint_flag</b>                   | u(1) |
| <b>gci_no_sbtmvp_constraint_flag</b>                         | u(1) |
| <b>gci_no_amvr_constraint_flag</b>                           | u(1) |
| <b>gci_no_bdof_constraint_flag</b>                           | u(1) |
| <b>gci_no_smvd_constraint_flag</b>                           | u(1) |
| <b>gci_no_dmvr_constraint_flag</b>                           | u(1) |
| <b>gci_no_mmvd_constraint_flag</b>                           | u(1) |
| <b>gci_no_affine_motion_constraint_flag</b>                  | u(1) |
| <b>gci_no_prof_constraint_flag</b>                           | u(1) |
| <b>gci_no_bcw_constraint_flag</b>                            | u(1) |
| <b>gci_no_ciip_constraint_flag</b>                           | u(1) |
| <b>gci_no_gpm_constraint_flag</b>                            | u(1) |

|                                                                        |      |
|------------------------------------------------------------------------|------|
| /* transform, quantization, residual */                                |      |
| <b>gci_no_luma_transform_size_64_constraint_flag</b>                   | u(1) |
| <b>gci_no_transform_skip_constraint_flag</b>                           | u(1) |
| <b>gci_no_bdpcm_constraint_flag</b>                                    | u(1) |
| <b>gci_no_mts_constraint_flag</b>                                      | u(1) |
| <b>gci_no_lfnst_constraint_flag</b>                                    | u(1) |
| <b>gci_no_joint_cbr_constraint_flag</b>                                | u(1) |
| <b>gci_no_sbt_constraint_flag</b>                                      | u(1) |
| <b>gci_no_act_constraint_flag</b>                                      | u(1) |
| <b>gci_no_explicit_scaling_list_constraint_flag</b>                    | u(1) |
| <b>gci_no_dep_quant_constraint_flag</b>                                | u(1) |
| <b>gci_no_sign_data_hiding_constraint_flag</b>                         | u(1) |
| <b>gci_no_cu_qp_delta_constraint_flag</b>                              | u(1) |
| <b>gci_no_chroma_qp_offset_constraint_flag</b>                         | u(1) |
| /* loop filter */                                                      |      |
| <b>gci_no_sao_constraint_flag</b>                                      | u(1) |
| <b>gci_no_alf_constraint_flag</b>                                      | u(1) |
| <b>gci_no_ccalf_constraint_flag</b>                                    | u(1) |
| <b>gci_no_lmcs_constraint_flag</b>                                     | u(1) |
| <b>gci_no_ladf_constraint_flag</b>                                     | u(1) |
| <b>gci_no_virtual_boundaries_constraint_flag</b>                       | u(1) |
| <b>gci_num_additional_bits</b>                                         | u(8) |
| if( gci_num_additional_bits > 5 ) {                                    |      |
| <b>gci_all_rap_pictures_constraint_flag</b>                            | u(1) |
| <b>gci_no_extended_precision_processing_constraint_flag</b>            | u(1) |
| <b>gci_no_ts_residual_coding_rice_constraint_flag</b>                  | u(1) |
| <b>gci_no_rrc_rice_extension_constraint_flag</b>                       | u(1) |
| <b>gci_no_persistent_rice_adaptation_constraint_flag</b>               | u(1) |
| <b>gci_no_reverse_last_sig_coeff_constraint_flag</b>                   | u(1) |
| numAdditionalBitsUsed = 6                                              |      |
| } else                                                                 |      |
| numAdditionalBitsUsed = 0                                              |      |
| for( i = 0; i < gci_num_additional_bits - numAdditionalBitsUsed; i++ ) |      |
| <b>gci_reserved_bit[ i ]</b>                                           | u(1) |
| }                                                                      |      |
| while( !byte_aligned( ) )                                              |      |
| <b>gci_alignment_zero_bit</b>                                          | f(1) |
| }                                                                      |      |



### 7.3.4 DPB parameters syntax

| dpb_parameters( MaxSubLayersMinus1, subLayerInfoFlag ) {                                    | Descriptor |
|---------------------------------------------------------------------------------------------|------------|
| for( i = ( subLayerInfoFlag ? 0 : MaxSubLayersMinus1 );<br>i <= MaxSubLayersMinus1; i++ ) { |            |
| <b>dpb_max_dec_pic_buffering_minus1</b> [ i ]                                               | ue(v)      |
| <b>dpb_max_num_reorder_pics</b> [ i ]                                                       | ue(v)      |
| <b>dpb_max_latency_increase_plus1</b> [ 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 |
|------------------------------------------------------------------------------------|------------|
| <b>num_units_in_tick</b>                                                           | u(32)      |
| <b>time_scale</b>                                                                  | u(32)      |
| <b>general_nal_hrd_params_present_flag</b>                                         | u(1)       |
| <b>general_vcl_hrd_params_present_flag</b>                                         | u(1)       |
| if( general_nal_hrd_params_present_flag    general_vcl_hrd_params_present_flag ) { |            |
| <b>general_same_pic_timing_in_all_ols_flag</b>                                     | u(1)       |
| <b>general_du_hrd_params_present_flag</b>                                          | u(1)       |
| if( general_du_hrd_params_present_flag )                                           |            |
| <b>tick_divisor_minus2</b>                                                         | u(8)       |
| <b>bit_rate_scale</b>                                                              | u(4)       |
| <b>cpb_size_scale</b>                                                              | u(4)       |
| if( general_du_hrd_params_present_flag )                                           |            |
| <b>cpb_size_du_scale</b>                                                           | u(4)       |
| <b>hrd_cpb_cnt_minus1</b>                                                          | 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++ ) {                                                                   |            |
| <b>fixed_pic_rate_general_flag</b> [ i ]                                                                                | u(1)       |
| if( !fixed_pic_rate_general_flag[ i ] )                                                                                 |            |
| <b>fixed_pic_rate_within_cvs_flag</b> [ i ]                                                                             | u(1)       |
| if( fixed_pic_rate_within_cvs_flag[ i ] )                                                                               |            |
| <b>elemental_duration_in_tc_minus1</b> [ i ]                                                                            | ue(v)      |
| else if( ( general_nal_hrd_params_present_flag   <br>general_vcl_hrd_params_present_flag ) && hrd_cpb_cnt_minus1 == 0 ) |            |
| <b>low_delay_hrd_flag</b> [ 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 ) {             | <b>Descriptor</b> |
| for( j = 0; j <= hrd_cpb_cnt_minus1; j++ ) {        |                   |
| <b>bit_rate_value_minus1</b> [ subLayerId ][ j ]    | ue(v)             |
| <b>cpb_size_value_minus1</b> [ subLayerId ][ j ]    | ue(v)             |
| if( general_du_hrd_params_present_flag ) {          |                   |
| <b>cpb_size_du_value_minus1</b> [ subLayerId ][ j ] | ue(v)             |
| <b>bit_rate_du_value_minus1</b> [ subLayerId ][ j ] | ue(v)             |
| }                                                   |                   |
| <b>cbr_flag</b> [ subLayerId ][ j ]                 | u(1)              |
| }                                                   |                   |
| }                                                   |                   |

### 7.3.6 Supplemental enhancement information message syntax

|                                         |                   |
|-----------------------------------------|-------------------|
| sei_message() {                         | <b>Descriptor</b> |
| payloadType = 0                         |                   |
| do {                                    |                   |
| <b>payload_type_byte</b>                | u(8)              |
| payloadType += payload_type_byte        |                   |
| } while( payload_type_byte == 0xFF )    |                   |
| payloadSize = 0                         |                   |
| do {                                    |                   |
| <b>payload_size_byte</b>                | u(8)              |
| payloadSize += payload_size_byte        |                   |
| } while( payload_size_byte == 0xFF )    |                   |
| sei_payload( payloadType, payloadSize ) |                   |
| }                                       |                   |

### 7.3.7 Slice header syntax

|                                                                                                                             |                   |
|-----------------------------------------------------------------------------------------------------------------------------|-------------------|
| slice_header() {                                                                                                            | <b>Descriptor</b> |
| <b>sh_picture_header_in_slice_header_flag</b>                                                                               | u(1)              |
| if( sh_picture_header_in_slice_header_flag )                                                                                |                   |
| picture_header_structure()                                                                                                  |                   |
| if( sps_subpic_info_present_flag )                                                                                          |                   |
| <b>sh_subpic_id</b>                                                                                                         | u(v)              |
| if( ( pps_rect_slice_flag && NumSlicesInSubpic[ CurrSubpicIdx ] > 1 )   <br>( !pps_rect_slice_flag && NumTilesInPic > 1 ) ) |                   |
| <b>sh_slice_address</b>                                                                                                     | u(v)              |
| for( i = 0; i < NumExtraShBits; i++ )                                                                                       |                   |
| <b>sh_extra_bit</b> [ i ]                                                                                                   | u(1)              |

|                                                                                                                                                     |       |
|-----------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| if( !pps_rect_slice_flag && NumTilesInPic – sh_slice_address > 1 )                                                                                  |       |
| <b>sh_num_tiles_in_slice_minus1</b>                                                                                                                 | ue(v) |
| if( ph_inter_slice_allowed_flag )                                                                                                                   |       |
| <b>sh_slice_type</b>                                                                                                                                | ue(v) |
| if( nal_unit_type == IDR_W_RADL    nal_unit_type == IDR_N_LP   <br>nal_unit_type == CRA_NUT    nal_unit_type == GDR_NUT )                           |       |
| <b>sh_no_output_of_prior_pics_flag</b>                                                                                                              | u(1)  |
| if( sps_alf_enabled_flag && !pps_alf_info_in_ph_flag ) {                                                                                            |       |
| <b>sh_alf_enabled_flag</b>                                                                                                                          | u(1)  |
| if( sh_alf_enabled_flag ) {                                                                                                                         |       |
| <b>sh_num_alf_aps_ids_luma</b>                                                                                                                      | u(3)  |
| for( i = 0; i < sh_num_alf_aps_ids_luma; i++ )                                                                                                      |       |
| <b>sh_alf_aps_id_luma[ i ]</b>                                                                                                                      | u(3)  |
| if( sps_chroma_format_idc != 0 ) {                                                                                                                  |       |
| <b>sh_alf_cb_enabled_flag</b>                                                                                                                       | u(1)  |
| <b>sh_alf_cr_enabled_flag</b>                                                                                                                       | u(1)  |
| }                                                                                                                                                   |       |
| if( sh_alf_cb_enabled_flag    sh_alf_cr_enabled_flag )                                                                                              |       |
| <b>sh_alf_aps_id_chroma</b>                                                                                                                         | u(3)  |
| if( sps_ccalf_enabled_flag ) {                                                                                                                      |       |
| <b>sh_alf_cc_cb_enabled_flag</b>                                                                                                                    | u(1)  |
| if( sh_alf_cc_cb_enabled_flag )                                                                                                                     |       |
| <b>sh_alf_cc_cb_aps_id</b>                                                                                                                          | u(3)  |
| <b>sh_alf_cc_cr_enabled_flag</b>                                                                                                                    | u(1)  |
| if( sh_alf_cc_cr_enabled_flag )                                                                                                                     |       |
| <b>sh_alf_cc_cr_aps_id</b>                                                                                                                          | u(3)  |
| }                                                                                                                                                   |       |
| }                                                                                                                                                   |       |
| }                                                                                                                                                   |       |
| if( ph_lmcs_enabled_flag && !sh_picture_header_in_slice_header_flag )                                                                               |       |
| <b>sh_lmcs_used_flag</b>                                                                                                                            | u(1)  |
| if( ph_explicit_scaling_list_enabled_flag && !sh_picture_header_in_slice_header_flag )                                                              |       |
| <b>sh_explicit_scaling_list_used_flag</b>                                                                                                           | u(1)  |
| if( !pps_rpl_info_in_ph_flag && ( ( nal_unit_type != IDR_W_RADL &&<br>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 )   <br>( sh_slice_type == B && num_ref_entries[ 1 ][ RplsIdx[ 1 ] ] > 1 ) ) { |       |
| <b>sh_num_ref_idx_active_override_flag</b>                                                                                                          | 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 )                                                                                                      |       |
| <b>sh_num_ref_idx_active_minus1[ i ]</b>                                                                                                            | ue(v) |
| }                                                                                                                                                   |       |
| if( sh_slice_type != I ) {                                                                                                                          |       |
| if( pps_cabac_init_present_flag )                                                                                                                   |       |
| <b>sh_cabac_init_flag</b>                                                                                                                           | u(1)  |
| if( ph_temporal_mvp_enabled_flag && !pps_rpl_info_in_ph_flag ) {                                                                                    |       |
| if( sh_slice_type == B )                                                                                                                            |       |
| <b>sh_collocated_from_I0_flag</b>                                                                                                                   | u(1)  |

|                                                                                                                                                   |       |
|---------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| if( ( sh_collocated_from_l0_flag && NumRefIdxActive[ 0 ] > 1 )   <br>( ! sh_collocated_from_l0_flag && NumRefIdxActive[ 1 ] > 1 ) )               |       |
| <b>sh_collocated_ref_idx</b>                                                                                                                      | ue(v) |
| }                                                                                                                                                 |       |
| if( !pps_wp_info_in_ph_flag &&<br>( ( pps_weighted_pred_flag && sh_slice_type == P )   <br>( pps_weighted_bipred_flag && sh_slice_type == B ) ) ) |       |
| pred_weight_table( )                                                                                                                              |       |
| }                                                                                                                                                 |       |
| if( !pps_qp_delta_info_in_ph_flag )                                                                                                               |       |
| <b>sh_qp_delta</b>                                                                                                                                | se(v) |
| if( pps_slice_chroma_qp_offsets_present_flag ) {                                                                                                  |       |
| <b>sh_cb_qp_offset</b>                                                                                                                            | se(v) |
| <b>sh_cr_qp_offset</b>                                                                                                                            | se(v) |
| if( sps_joint_cbr_enabled_flag )                                                                                                                  |       |
| <b>sh_joint_cbr_qp_offset</b>                                                                                                                     | se(v) |
| }                                                                                                                                                 |       |
| if( pps_cu_chroma_qp_offset_list_enabled_flag )                                                                                                   |       |
| <b>sh_cu_chroma_qp_offset_enabled_flag</b>                                                                                                        | u(1)  |
| if( sps_sao_enabled_flag && !pps_sao_info_in_ph_flag ) {                                                                                          |       |
| <b>sh_sao_luma_used_flag</b>                                                                                                                      | u(1)  |
| if( sps_chroma_format_idc != 0 )                                                                                                                  |       |
| <b>sh_sao_chroma_used_flag</b>                                                                                                                    | u(1)  |
| }                                                                                                                                                 |       |
| if( pps_deblocking_filter_override_enabled_flag && !pps_dbf_info_in_ph_flag )                                                                     |       |
| <b>sh_deblocking_params_present_flag</b>                                                                                                          | u(1)  |
| if( sh_deblocking_params_present_flag ) {                                                                                                         |       |
| if( !pps_deblocking_filter_disabled_flag )                                                                                                        |       |
| <b>sh_deblocking_filter_disabled_flag</b>                                                                                                         | u(1)  |
| if( !sh_deblocking_filter_disabled_flag ) {                                                                                                       |       |
| <b>sh_luma_beta_offset_div2</b>                                                                                                                   | se(v) |
| <b>sh_luma_tc_offset_div2</b>                                                                                                                     | se(v) |
| if( pps_chroma_tool_offsets_present_flag ) {                                                                                                      |       |
| <b>sh_cb_beta_offset_div2</b>                                                                                                                     | se(v) |
| <b>sh_cb_tc_offset_div2</b>                                                                                                                       | se(v) |
| <b>sh_cr_beta_offset_div2</b>                                                                                                                     | se(v) |
| <b>sh_cr_tc_offset_div2</b>                                                                                                                       | se(v) |
| }                                                                                                                                                 |       |
| }                                                                                                                                                 |       |
| }                                                                                                                                                 |       |
| if( sps_dep_quant_enabled_flag )                                                                                                                  |       |
| <b>sh_dep_quant_used_flag</b>                                                                                                                     | u(1)  |
| if( sps_sign_data_hiding_enabled_flag && !sh_dep_quant_used_flag )                                                                                |       |
| <b>sh_sign_data_hiding_used_flag</b>                                                                                                              | u(1)  |
| if( sps_transform_skip_enabled_flag && !sh_dep_quant_used_flag &&<br>!sh_sign_data_hiding_used_flag )                                             |       |
| <b>sh_ts_residual_coding_disabled_flag</b>                                                                                                        | u(1)  |
| if( !sh_ts_residual_coding_disabled_flag &&<br>sps_ts_residual_coding_rice_present_in_sh_flag )                                                   |       |
| <b>sh_ts_residual_coding_rice_idx_minus1</b>                                                                                                      | u(3)  |
| if( sps_reverse_last_sig_coeff_enabled_flag )                                                                                                     |       |

|                                                         |       |
|---------------------------------------------------------|-------|
| <b>sh_reverse_last_sig_coeff_flag</b>                   | u(1)  |
| if( pps_slice_header_extension_present_flag ) {         |       |
| <b>sh_slice_header_extension_length</b>                 | ue(v) |
| for( i = 0; i < sh_slice_header_extension_length; i++ ) |       |
| <b>sh_slice_header_extension_data_byte[ i ]</b>         | u(8)  |
| }                                                       |       |
| if( NumEntryPoints > 0 ) {                              |       |
| <b>sh_entry_offset_len_minus1</b>                       | ue(v) |
| for( i = 0; i < NumEntryPoints; i++ )                   |       |
| <b>sh_entry_point_offset_minus1[ i ]</b>                | u(v)  |
| }                                                       |       |
| byte_alignment( )                                       |       |
| }                                                       |       |

### 7.3.8 Weighted prediction parameters syntax

|                                                                                                        |                   |
|--------------------------------------------------------------------------------------------------------|-------------------|
| pred_weight_table( ) {                                                                                 | <b>Descriptor</b> |
| <b>luma_log2_weight_denom</b>                                                                          | ue(v)             |
| if( sps_chroma_format_idc != 0 )                                                                       |                   |
| <b>delta_chroma_log2_weight_denom</b>                                                                  | se(v)             |
| if( pps_wp_info_in_ph_flag )                                                                           |                   |
| <b>num_l0_weights</b>                                                                                  | ue(v)             |
| for( i = 0; i < NumWeightsL0; i++ )                                                                    |                   |
| <b>luma_weight_l0_flag[ i ]</b>                                                                        | u(1)              |
| if( sps_chroma_format_idc != 0 )                                                                       |                   |
| for( i = 0; i < NumWeightsL0; i++ )                                                                    |                   |
| <b>chroma_weight_l0_flag[ i ]</b>                                                                      | u(1)              |
| for( i = 0; i < NumWeightsL0; i++ ) {                                                                  |                   |
| if( luma_weight_l0_flag[ i ] ) {                                                                       |                   |
| <b>delta_luma_weight_l0[ i ]</b>                                                                       | se(v)             |
| <b>luma_offset_l0[ i ]</b>                                                                             | se(v)             |
| }                                                                                                      |                   |
| if( chroma_weight_l0_flag[ i ] )                                                                       |                   |
| for( j = 0; j < 2; j++ ) {                                                                             |                   |
| <b>delta_chroma_weight_l0[ i ][ j ]</b>                                                                | se(v)             |
| <b>delta_chroma_offset_l0[ i ][ j ]</b>                                                                | se(v)             |
| }                                                                                                      |                   |
| }                                                                                                      |                   |
| if( pps_weighted_bipred_flag && pps_wp_info_in_ph_flag &&<br>num_ref_entries[ 1 ][ RplIdx[ 1 ] ] > 0 ) |                   |
| <b>num_l1_weights</b>                                                                                  | ue(v)             |
| for( i = 0; i < NumWeightsL1; i++ )                                                                    |                   |
| <b>luma_weight_l1_flag[ i ]</b>                                                                        | u(1)              |
| if( sps_chroma_format_idc != 0 )                                                                       |                   |
| for( i = 0; i < NumWeightsL1; i++ )                                                                    |                   |
| <b>chroma_weight_l1_flag[ i ]</b>                                                                      | u(1)              |
| for( i = 0; i < NumWeightsL1; i++ ) {                                                                  |                   |
| if( luma_weight_l1_flag[ i ] ) {                                                                       |                   |
| <b>delta_luma_weight_l1[ i ]</b>                                                                       | se(v)             |
| }                                                                                                      |                   |

|                                         |       |
|-----------------------------------------|-------|
| <b>luma_offset_l1[ i ]</b>              | se(v) |
| }                                       |       |
| if( chroma_weight_l1_flag[ i ] )        |       |
| for( j = 0; j < 2; j++ ) {              |       |
| <b>delta_chroma_weight_l1[ i ][ j ]</b> | se(v) |
| <b>delta_chroma_offset_l1[ i ][ j ]</b> | se(v) |
| }                                       |       |
| }                                       |       |
| }                                       |       |

### 7.3.9 Reference picture lists syntax

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

### 7.3.10 Reference picture list structure syntax

|                                                                                                                               |                   |
|-------------------------------------------------------------------------------------------------------------------------------|-------------------|
| ref_pic_list_struct( listIdx, rplIdx ) {                                                                                      | <b>Descriptor</b> |
| <b>num_ref_entries[ listIdx ][ rplIdx ]</b>                                                                                   | ue(v)             |
| if( sps_long_term_ref_pics_flag && rplIdx < sps_num_ref_pic_lists[ listIdx ] &&<br>num_ref_entries[ listIdx ][ rplIdx ] > 0 ) |                   |
| <b>ltrp_in_header_flag[ listIdx ][ rplIdx ]</b>                                                                               | u(1)              |
| for( i = 0, j = 0; i < num_ref_entries[ listIdx ][ rplIdx ]; i++ ) {                                                          |                   |
| if( sps_inter_layer_prediction_enabled_flag )                                                                                 |                   |
| <b>inter_layer_ref_pic_flag[ listIdx ][ rplIdx ][ i ]</b>                                                                     | u(1)              |
| if( !inter_layer_ref_pic_flag[ listIdx ][ rplIdx ][ i ] ) {                                                                   |                   |
| if( sps_long_term_ref_pics_flag )                                                                                             |                   |
| <b>st_ref_pic_flag[ listIdx ][ rplIdx ][ i ]</b>                                                                              | u(1)              |
| if( st_ref_pic_flag[ listIdx ][ rplIdx ][ i ] ) {                                                                             |                   |
| <b>abs_delta_poc_st[ listIdx ][ rplIdx ][ i ]</b>                                                                             | ue(v)             |

|                                                         |       |
|---------------------------------------------------------|-------|
| if( AbsDeltaPocSt[ listIdx ][ rplsIdx ][ i ] > 0 )      |       |
| <b>strp_entry_sign_flag</b> [ listIdx ][ rplsIdx ][ i ] | u(1)  |
| } else if( !ltp_in_header_flag[ listIdx ][ rplsIdx ] )  |       |
| <b>rpls_poc_lsb_lt</b> [ listIdx ][ rplsIdx ][ j++ ]    | u(v)  |
| } else                                                  |       |
| <b>ilrp_idx</b> [ listIdx ][ rplsIdx ][ i ]             | ue(v) |
| }                                                       |       |
| }                                                       |       |

### 7.3.11 Slice data syntax

#### 7.3.11.1 General slice data syntax

|                                                             |                   |
|-------------------------------------------------------------|-------------------|
| slice_data( ) {                                             | <b>Descriptor</b> |
| 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 )                           |                   |
| <b>end_of_slice_one_bit</b> /* equal to 1 */                | ae(v)             |
| else if( CtbAddrX == CtbToTileColBd[ CtbAddrX + 1 ] - 1 ) { |                   |
| if( CtbAddrY == CtbToTileRowBd[ CtbAddrY + 1 ] - 1 ) {      |                   |
| <b>end_of_tile_one_bit</b> /* equal to 1 */                 | ae(v)             |
| byte_alignment( )                                           |                   |
| } else if( sps_entropy_coding_sync_enabled_flag ) {         |                   |
| <b>end_of_subset_one_bit</b> /* equal to 1 */               | ae(v)             |
| byte_alignment( )                                           |                   |
| }                                                           |                   |
| FirstCtbRowInSlice = 0                                      |                   |
| }                                                           |                   |
| }                                                           |                   |

#### 7.3.11.2 Coding tree unit syntax

|                                                        |                   |
|--------------------------------------------------------|-------------------|
| coding_tree_unit( ) {                                  | <b>Descriptor</b> |
| 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 ) {                            |                   |

|                                                                                                   |       |
|---------------------------------------------------------------------------------------------------|-------|
| <b>alf_ctb_flag</b> [ 0 ][ CtbAddrX ][ CtbAddrY ]                                                 | ae(v) |
| if( alf_ctb_flag[ 0 ][ CtbAddrX ][ CtbAddrY ] ) {                                                 |       |
| if( sh_num_alf_aps_ids_luma > 0 )                                                                 |       |
| <b>alf_use_aps_flag</b>                                                                           | ae(v) |
| if( alf_use_aps_flag ) {                                                                          |       |
| if( sh_num_alf_aps_ids_luma > 1 )                                                                 |       |
| <b>alf_luma_prev_filter_idx</b>                                                                   | ae(v) |
| } else                                                                                            |       |
| <b>alf_luma_fixed_filter_idx</b>                                                                  | ae(v) |
| }                                                                                                 |       |
| if( sh_alf_cb_enabled_flag ) {                                                                    |       |
| <b>alf_ctb_flag</b> [ 1 ][ CtbAddrX ][ CtbAddrY ]                                                 | ae(v) |
| if( alf_ctb_flag[ 1 ][ CtbAddrX ][ CtbAddrY ]<br>&& alf_chroma_num_alt_filters_minus1 > 0 )       |       |
| <b>alf_ctb_filter_alt_idx</b> [ 0 ][ CtbAddrX ][ CtbAddrY ]                                       | ae(v) |
| }                                                                                                 |       |
| if( sh_alf_cr_enabled_flag ) {                                                                    |       |
| <b>alf_ctb_flag</b> [ 2 ][ CtbAddrX ][ CtbAddrY ]                                                 | ae(v) |
| if( alf_ctb_flag[ 2 ][ CtbAddrX ][ CtbAddrY ]<br>&& alf_chroma_num_alt_filters_minus1 > 0 )       |       |
| <b>alf_ctb_filter_alt_idx</b> [ 1 ][ CtbAddrX ][ CtbAddrY ]                                       | ae(v) |
| }                                                                                                 |       |
| }                                                                                                 |       |
| if( sh_alf_cc_cb_enabled_flag )                                                                   |       |
| <b>alf_ctb_cc_cb_idc</b> [ CtbAddrX ][ CtbAddrY ]                                                 | ae(v) |
| if( sh_alf_cc_cr_enabled_flag )                                                                   |       |
| <b>alf_ctb_cc_cr_idc</b> [ CtbAddrX ][ CtbAddrY ]                                                 | ae(v) |
| if( sh_slice_type == 1 && sps_qtbt_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,<br>SINGLE_TREE, MODE_TYPE_ALL ) |       |
| }                                                                                                 |       |

|                                                                                      |                   |
|--------------------------------------------------------------------------------------|-------------------|
| dual_tree_implicit_qt_split( x0, y0, cbSize, cqtDepth ) {                            | <b>Descriptor</b> |
| 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 &&<br>cbSubdiv <= CuChromaQpOffsetSubdiv ) { |                   |
| IsCuChromaQpOffsetCoded = 0                                                          |                   |
| CuQpOffsetCb = 0                                                                     |                   |
| CuQpOffsetCr = 0                                                                     |                   |
| CuQpOffsetCbCr = 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

|                                                                        | Descriptor |
|------------------------------------------------------------------------|------------|
| sao( rx, ry ) {                                                        |            |
| if( rx > 0 ) {                                                         |            |
| leftCtbAvailable = rx != CtbToTileColBd[ rx ]                          |            |
| if( leftCtbAvailable )                                                 |            |
| <b>sao_merge_left_flag</b>                                             | ae(v)      |
| }                                                                      |            |
| if( ry > 0 && !sao_merge_left_flag ) {                                 |            |
| upCtbAvailable = ry != CtbToTileRowBd[ ry ] && !FirstCtbRowInSlice     |            |
| if( upCtbAvailable )                                                   |            |
| <b>sao_merge_up_flag</b>                                               | 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 )                                                        |            |
| <b>sao_type_idx_luma</b>                                               | ae(v)      |
| else if( cIdx == 1 )                                                   |            |
| <b>sao_type_idx_chroma</b>                                             | ae(v)      |
| if( SaoTypeIdx[ cIdx ][ rx ][ ry ] != 0 ) {                            |            |
| for( i = 0; i < 4; i++ )                                               |            |
| <b>sao_offset_abs</b> [ 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 )                     |            |
| <b>sao_offset_sign_flag</b> [ cIdx ][ rx ][ ry ][ i ]                  | ae(v)      |
| <b>sao_band_position</b> [ cIdx ][ rx ][ ry ]                          | ae(v)      |
| } else {                                                               |            |
| if( cIdx == 0 )                                                        |            |

|                            |       |
|----------------------------|-------|
| <b>sao_eo_class_luma</b>   | ae(v) |
| if( cldx == 1 )            |       |
| <b>sao_eo_class_chroma</b> | 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 ) ) |            |
| <b>split_cu_flag</b>                                                                                                                                                                                           | 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                                                                                                                                                                                    |            |
| CuQpOffsetCb = 0                                                                                                                                                                                               |            |
| CuQpOffsetCr = 0                                                                                                                                                                                               |            |
| CuQpOffsetCbCr = 0                                                                                                                                                                                             |            |
| }                                                                                                                                                                                                              |            |
| if( split_cu_flag ) {                                                                                                                                                                                          |            |
| if( ( allowSplitBtVer    allowSplitBtHor    allowSplitTtVer    allowSplitTtHor ) && allowSplitQt )                                                                                                             |            |
| <b>split_qt_flag</b>                                                                                                                                                                                           | ae(v)      |
| if( !split_qt_flag ) {                                                                                                                                                                                         |            |
| if( ( allowSplitBtHor    allowSplitTtHor ) && ( allowSplitBtVer    allowSplitTtVer ) )                                                                                                                         |            |
| <b>mtt_split_cu_vertical_flag</b>                                                                                                                                                                              | ae(v)      |
| if( ( allowSplitBtVer && allowSplitTtVer && mtt_split_cu_vertical_flag )    ( allowSplitBtHor && allowSplitTtHor && !mtt_split_cu_vertical_flag ) )                                                            |            |
| <b>mtt_split_cu_binary_flag</b>                                                                                                                                                                                | ae(v)      |
| }                                                                                                                                                                                                              |            |
| if( ModeTypeCondition == 1 )                                                                                                                                                                                   |            |
| modeType = MODE_TYPE_INTRA                                                                                                                                                                                     |            |
| else if( ModeTypeCondition == 2 ) {                                                                                                                                                                            |            |
| <b>non_inter_flag</b>                                                                                                                                                                                          | 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 &&<br>x1 < pps_pic_width_in_luma_samples )                                         |  |
| coding_tree( x1, y1, cbWidth / 2, cbHeight / 2, qgOnY, qgOnC, cbSubdiv + 2,<br>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,<br>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 &&<br>( !( cbWidth == 4 && cbHeight == 4 ) &&<br>modeType != MODE_TYPE_INTRA )   <br>( sps_ibc_enabled_flag && cbWidth <= 64 && cbHeight <= 64 ) )                                                                                                                                                                                                                      |            |
| <b>cu_skip_flag</b> [ x0 ][ y0 ]                                                                                                                                                                                                                                                                                                                                                                         | ae(v)      |
| if( cu_skip_flag[ x0 ][ y0 ] == 0 && sh_slice_type != I &&<br>!( cbWidth == 4 && cbHeight == 4 ) && modeType == MODE_TYPE_ALL )                                                                                                                                                                                                                                                                          |            |
| <b>pred_mode_flag</b>                                                                                                                                                                                                                                                                                                                                                                                    | ae(v)      |
| if( ( ( sh_slice_type == I && cu_skip_flag[ x0 ][ y0 ] == 0 )   <br>( sh_slice_type != I && ( CuPredMode[ chType ][ x0 ][ y0 ] != MODE_INTRA<br>   ( ( cbWidth == 4 && cbHeight == 4 )   <br>modeType == MODE_TYPE_INTRA ) && cu_skip_flag[ x0 ][ y0 ] == 0 ) ) ) )<br>&& cbWidth <= 64 && cbHeight <= 64 &&<br>modeType != MODE_TYPE_INTER &&<br>sps_ibc_enabled_flag && treeType != DUAL_TREE_CHROMA ) |            |
| <b>pred_mode_ibc_flag</b>                                                                                                                                                                                                                                                                                                                                                                                | ae(v)      |
| }                                                                                                                                                                                                                                                                                                                                                                                                        |            |
| if( CuPredMode[ chType ][ x0 ][ y0 ] == MODE_INTRA && sps_palette_enabled_flag<br>&& cbWidth <= 64 && cbHeight <= 64 && cu_skip_flag[ x0 ][ y0 ] == 0 &&<br>modeType != MODE_TYPE_INTER && ( ( cbWidth * cbHeight ) ><br>( treeType != DUAL_TREE_CHROMA ? 16 : 16 * SubWidthC * SubHeightC ) )<br>&& modeType != MODE_TYPE_INTRA   <br>treeType != DUAL_TREE_CHROMA )                                    |            |
| <b>pred_mode_plt_flag</b>                                                                                                                                                                                                                                                                                                                                                                                | ae(v)      |
| if( CuPredMode[ chType ][ x0 ][ y0 ] == MODE_INTRA && sps_act_enabled_flag &&<br>treeType == SINGLE_TREE )                                                                                                                                                                                                                                                                                               |            |
| <b>cu_act_enabled_flag</b>                                                                                                                                                                                                                                                                                                                                                                               | ae(v)      |
| if( CuPredMode[ chType ][ x0 ][ y0 ] == MODE_INTRA   <br>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 &&<br>cbWidth <= MaxTsSize && cbHeight <= MaxTsSize )                                                                                                                  |       |
| <b>intra_bdpcm_luma_flag</b>                                                                                                                                                                      | ae(v) |
| if( intra_bdpcm_luma_flag )                                                                                                                                                                       |       |
| <b>intra_bdpcm_luma_dir_flag</b>                                                                                                                                                                  | ae(v) |
| else {                                                                                                                                                                                            |       |
| if( sps_mip_enabled_flag )                                                                                                                                                                        |       |
| <b>intra_mip_flag</b>                                                                                                                                                                             | ae(v) |
| if( intra_mip_flag ) {                                                                                                                                                                            |       |
| <b>intra_mip_transposed_flag</b> [ x0 ][ y0 ]                                                                                                                                                     | ae(v) |
| <b>intra_mip_mode</b> [ x0 ][ y0 ]                                                                                                                                                                | ae(v) |
| } else {                                                                                                                                                                                          |       |
| if( sps_mrl_enabled_flag && ( ( y0 % CtbSizeY ) > 0 ) )                                                                                                                                           |       |
| <b>intra_luma_ref_idx</b>                                                                                                                                                                         | ae(v) |
| if( sps_isp_enabled_flag && intra_luma_ref_idx == 0 &&<br>( cbWidth <= MaxTbSizeY && cbHeight <= MaxTbSizeY ) &&<br>( cbWidth * cbHeight > MinTbSizeY * MinTbSizeY ) &&<br>!cu_act_enabled_flag ) |       |
| <b>intra_subpartitions_mode_flag</b>                                                                                                                                                              | ae(v) |
| if( intra_subpartitions_mode_flag == 1 )                                                                                                                                                          |       |
| <b>intra_subpartitions_split_flag</b>                                                                                                                                                             | ae(v) |
| if( intra_luma_ref_idx == 0 )                                                                                                                                                                     |       |
| <b>intra_luma_mpm_flag</b> [ x0 ][ y0 ]                                                                                                                                                           | ae(v) |
| if( intra_luma_mpm_flag[ x0 ][ y0 ] ) {                                                                                                                                                           |       |
| if( intra_luma_ref_idx == 0 )                                                                                                                                                                     |       |
| <b>intra_luma_not_planar_flag</b> [ x0 ][ y0 ]                                                                                                                                                    | ae(v) |
| if( intra_luma_not_planar_flag[ x0 ][ y0 ] )                                                                                                                                                      |       |
| <b>intra_luma_mpm_idx</b> [ x0 ][ y0 ]                                                                                                                                                            | ae(v) |
| } else                                                                                                                                                                                            |       |
| <b>intra_luma_mpm_remainder</b> [ x0 ][ y0 ]                                                                                                                                                      | ae(v) |
| }                                                                                                                                                                                                 |       |
| }                                                                                                                                                                                                 |       |
| }                                                                                                                                                                                                 |       |
| }                                                                                                                                                                                                 |       |
| if( ( treeType == SINGLE_TREE    treeType == DUAL_TREE_CHROMA ) &&<br>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 &&<br>cbHeight / SubHeightC <= MaxTsSize && sps_bdpcm_enabled_flag )                                                                                         |       |
| <b>intra_bdpcm_chroma_flag</b>                                                                                                                                                                    | ae(v) |
| if( intra_bdpcm_chroma_flag )                                                                                                                                                                     |       |
| <b>intra_bdpcm_chroma_dir_flag</b>                                                                                                                                                                | ae(v) |
| else {                                                                                                                                                                                            |       |
| if( CclmEnabled )                                                                                                                                                                                 |       |
| <b>cclm_mode_flag</b>                                                                                                                                                                             | ae(v) |
| if( cclm_mode_flag )                                                                                                                                                                              |       |
| <b>cclm_mode_idx</b>                                                                                                                                                                              | ae(v) |
| else                                                                                                                                                                                              |       |

|                                                                                                                                                                              |       |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| <b>intra_chroma_pred_mode</b>                                                                                                                                                | ae(v) |
| }                                                                                                                                                                            |       |
| }                                                                                                                                                                            |       |
| }                                                                                                                                                                            |       |
| }                                                                                                                                                                            |       |
| } else if( treeType != DUAL_TREE_CHROMA ) { /* MODE_INTER or MODE_IBC */                                                                                                     |       |
| if( cu_skip_flag[ x0 ][ y0 ] == 0 )                                                                                                                                          |       |
| <b>general_merge_flag[ x0 ][ y0 ]</b>                                                                                                                                        | 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 )                                                                                                                                                 |       |
| <b>mvp_l0_flag[ x0 ][ y0 ]</b>                                                                                                                                               | ae(v) |
| if( sps_amvr_enabled_flag &&<br>( MvdL0[ x0 ][ y0 ][ 0 ] != 0    MvdL0[ x0 ][ y0 ][ 1 ] != 0 ) )                                                                             |       |
| <b>amvr_precision_idx[ x0 ][ y0 ]</b>                                                                                                                                        | ae(v) |
| } else {                                                                                                                                                                     |       |
| if( sh_slice_type == B )                                                                                                                                                     |       |
| <b>inter_pred_idc[ x0 ][ y0 ]</b>                                                                                                                                            | ae(v) |
| if( sps_affine_enabled_flag && cbWidth >= 16 && cbHeight >= 16 ) {                                                                                                           |       |
| <b>inter_affine_flag[ x0 ][ y0 ]</b>                                                                                                                                         | ae(v) |
| if( sps_6param_affine_enabled_flag && inter_affine_flag[ x0 ][ y0 ] )                                                                                                        |       |
| <b>cu_affine_type_flag[ x0 ][ y0 ]</b>                                                                                                                                       | ae(v) |
| }                                                                                                                                                                            |       |
| if( sps_smvd_enabled_flag && !ph_mvd_l1_zero_flag &&<br>inter_pred_idc[ x0 ][ y0 ] == PRED_BI &&<br>!inter_affine_flag[ x0 ][ y0 ] && RefIdxSymL0 > -1 && RefIdxSymL1 > -1 ) |       |
| <b>sym_mvd_flag[ x0 ][ y0 ]</b>                                                                                                                                              | ae(v) |
| if( inter_pred_idc[ x0 ][ y0 ] != PRED_L1 ) {                                                                                                                                |       |
| if( NumRefIdxActive[ 0 ] > 1 && !sym_mvd_flag[ x0 ][ y0 ] )                                                                                                                  |       |
| <b>ref_idx_l0[ x0 ][ y0 ]</b>                                                                                                                                                | ae(v) |
| mvd_coding( x0, y0, 0, 0 )                                                                                                                                                   |       |
| if( MotionModelIdx[ x0 ][ y0 ] > 0 )                                                                                                                                         |       |
| mvd_coding( x0, y0, 0, 1 )                                                                                                                                                   |       |
| if( MotionModelIdx[ x0 ][ y0 ] > 1 )                                                                                                                                         |       |
| mvd_coding( x0, y0, 0, 2 )                                                                                                                                                   |       |
| <b>mvp_l0_flag[ x0 ][ y0 ]</b>                                                                                                                                               | ae(v) |
| } else {                                                                                                                                                                     |       |
| MvdL0[ x0 ][ y0 ][ 0 ] = 0                                                                                                                                                   |       |
| MvdL0[ x0 ][ y0 ][ 1 ] = 0                                                                                                                                                   |       |
| }                                                                                                                                                                            |       |
| if( inter_pred_idc[ x0 ][ y0 ] != PRED_L0 ) {                                                                                                                                |       |
| if( NumRefIdxActive[ 1 ] > 1 && !sym_mvd_flag[ x0 ][ y0 ] )                                                                                                                  |       |
| <b>ref_idx_l1[ x0 ][ y0 ]</b>                                                                                                                                                | ae(v) |
| if( ph_mvd_l1_zero_flag && inter_pred_idc[ x0 ][ y0 ] == PRED_BI ) {                                                                                                         |       |
| MvdL1[ x0 ][ y0 ][ 0 ] = 0                                                                                                                                                   |       |
| MvdL1[ x0 ][ y0 ][ 1 ] = 0                                                                                                                                                   |       |
| MvdCpL1[ x0 ][ y0 ][ 0 ][ 0 ] = 0                                                                                                                                            |       |
| MvdCpL1[ x0 ][ y0 ][ 0 ][ 1 ] = 0                                                                                                                                            |       |
| MvdCpL1[ x0 ][ y0 ][ 1 ][ 0 ] = 0                                                                                                                                            |       |

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |       |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| MvdCpL1[ x0 ][ y0 ][ 1 ][ 1 ] = 0                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |       |
| MvdCpL1[ x0 ][ y0 ][ 2 ][ 0 ] = 0                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |       |
| MvdCpL1[ x0 ][ y0 ][ 2 ][ 1 ] = 0                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |       |
| } else {                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |       |
| if( sym_mvd_flag[ x0 ][ y0 ] ) {                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |       |
| MvdL1[ x0 ][ y0 ][ 0 ] = -MvdL0[ x0 ][ y0 ][ 0 ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |       |
| MvdL1[ x0 ][ y0 ][ 1 ] = -MvdL0[ x0 ][ y0 ][ 1 ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |       |
| } else                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |       |
| mvd_coding( x0, y0, 1, 0 )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |       |
| if( MotionModelIdc[ x0 ][ y0 ] > 0 )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |       |
| mvd_coding( x0, y0, 1, 1 )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |       |
| if( MotionModelIdc[ x0 ][ y0 ] > 1 )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |       |
| mvd_coding( x0, y0, 1, 2 )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |       |
| }                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |       |
| <b>mvp_l1_flag</b> [ x0 ][ y0 ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | ae(v) |
| } else {                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |       |
| MvdL1[ x0 ][ y0 ][ 0 ] = 0                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |       |
| MvdL1[ x0 ][ y0 ][ 1 ] = 0                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |       |
| }                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |       |
| if( ( sps_amvr_enabled_flag && inter_affine_flag[ x0 ][ y0 ] == 0 &&<br>( MvdL0[ x0 ][ y0 ][ 0 ] != 0    MvdL0[ x0 ][ y0 ][ 1 ] != 0   <br>MvdL1[ x0 ][ y0 ][ 0 ] != 0    MvdL1[ x0 ][ y0 ][ 1 ] != 0 ) )   <br>( sps_affine_amvr_enabled_flag && inter_affine_flag[ x0 ][ y0 ] == 1 &&<br>( MvdCpL0[ x0 ][ y0 ][ 0 ][ 0 ] != 0    MvdCpL0[ x0 ][ y0 ][ 0 ][ 1 ] != 0   <br>MvdCpL1[ x0 ][ y0 ][ 0 ][ 0 ] != 0    MvdCpL1[ x0 ][ y0 ][ 0 ][ 1 ] != 0   <br>MvdCpL0[ x0 ][ y0 ][ 1 ][ 0 ] != 0    MvdCpL0[ x0 ][ y0 ][ 1 ][ 1 ] != 0   <br>MvdCpL1[ x0 ][ y0 ][ 1 ][ 0 ] != 0    MvdCpL1[ x0 ][ y0 ][ 1 ][ 1 ] != 0   <br>MvdCpL0[ x0 ][ y0 ][ 2 ][ 0 ] != 0    MvdCpL0[ x0 ][ y0 ][ 2 ][ 1 ] != 0   <br>MvdCpL1[ x0 ][ y0 ][ 2 ][ 0 ] != 0    MvdCpL1[ x0 ][ y0 ][ 2 ][ 1 ] != 0 ) ) ) { |       |
| <b>amvr_flag</b> [ x0 ][ y0 ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | ae(v) |
| if( amvr_flag[ x0 ][ y0 ] )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |       |
| <b>amvr_precision_idx</b> [ x0 ][ y0 ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | ae(v) |
| }                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |       |
| if( sps_bcw_enabled_flag && inter_pred_idc[ x0 ][ y0 ] == PRED_BI &&<br>luma_weight_l0_flag[ ref_idx_l0 [ x0 ][ y0 ] ] == 0 &&<br>luma_weight_l1_flag[ ref_idx_l1 [ x0 ][ y0 ] ] == 0 &&<br>chroma_weight_l0_flag[ ref_idx_l0 [ x0 ][ y0 ] ] == 0 &&<br>chroma_weight_l1_flag[ ref_idx_l1 [ x0 ][ y0 ] ] == 0 &&<br>cbWidth * cbHeight >= 256 )                                                                                                                                                                                                                                                                                                                                                                                                                                          |       |
| <b>bcw_idx</b> [ x0 ][ y0 ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | ae(v) |
| }                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |       |
| }                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |       |
| if( CuPredMode[ chType ][ x0 ][ y0 ] != MODE_INTRA && !pred_mode_plt_flag &&<br>general_merge_flag[ x0 ][ y0 ] == 0 )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |       |
| <b>cu_coded_flag</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | ae(v) |
| if( cu_coded_flag ) {                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |       |
| if( CuPredMode[ chType ][ x0 ][ y0 ] == MODE_INTER &&<br>sps_sbt_enabled_flag && !ciip_flag[ x0 ][ y0 ] &&<br>cbWidth <= MaxTbSizeY && cbHeight <= MaxTbSizeY ) {                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |       |
| allowSbtVerH = cbWidth >= 8                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |       |
| allowSbtVerQ = cbWidth >= 16                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |       |
| allowSbtHorH = cbHeight >= 8                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |       |
| allowSbtHorQ = cbHeight >= 16                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |       |



|                                                                                                                                                                                                                                                                                                                                                               |       |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| if( allowSbtVerH    allowSbtHorH )                                                                                                                                                                                                                                                                                                                            |       |
| <b>cu_sbt_flag</b>                                                                                                                                                                                                                                                                                                                                            | ae(v) |
| if( cu_sbt_flag ) {                                                                                                                                                                                                                                                                                                                                           |       |
| if( ( allowSbtVerH    allowSbtHorH ) && ( allowSbtVerQ    allowSbtHorQ ) )                                                                                                                                                                                                                                                                                    |       |
| <b>cu_sbt_quad_flag</b>                                                                                                                                                                                                                                                                                                                                       | ae(v) |
| if( ( cu_sbt_quad_flag && allowSbtVerQ && allowSbtHorQ )   <br>( !cu_sbt_quad_flag && allowSbtVerH && allowSbtHorH ) )                                                                                                                                                                                                                                        |       |
| <b>cu_sbt_horizontal_flag</b>                                                                                                                                                                                                                                                                                                                                 | ae(v) |
| <b>cu_sbt_pos_flag</b>                                                                                                                                                                                                                                                                                                                                        | ae(v) |
| }                                                                                                                                                                                                                                                                                                                                                             |       |
| }                                                                                                                                                                                                                                                                                                                                                             |       |
| if( sps_act_enabled_flag && CuPredMode[ chType ][ x0 ][ y0 ] != MODE_INTRA<br>&& treeType == SINGLE_TREE )                                                                                                                                                                                                                                                    |       |
| <b>cu_act_enabled_flag</b>                                                                                                                                                                                                                                                                                                                                    | 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 :<br>( ( IntraSubPartitionsSplitType == ISP_VER_SPLIT ) ?<br>cbWidth / NumIntraSubPartitions : cbWidth )                                                                                                                                                                                  |       |
| lfnstHeight = ( treeType == DUAL_TREE_CHROMA ) ? cbHeight / SubHeightC :<br>( ( IntraSubPartitionsSplitType == ISP_HOR_SPLIT ) ?<br>cbHeight / NumIntraSubPartitions : cbHeight )                                                                                                                                                                             |       |
| lfnstNotTsFlag = ( treeType == DUAL_TREE_CHROMA   <br>!tu_y_coded_flag[ x0 ][ y0 ]   <br>transform_skip_flag[ x0 ][ y0 ][ 0 ] == 0 ) &&<br>( treeType == DUAL_TREE_LUMA   <br>( ( !tu_cb_coded_flag[ x0 ][ y0 ]   <br>transform_skip_flag[ x0 ][ y0 ][ 1 ] == 0 ) &&<br>( !tu_cr_coded_flag[ x0 ][ y0 ]   <br>transform_skip_flag[ x0 ][ y0 ][ 2 ] == 0 ) ) ) |       |
| if( Min( lfnstWidth, lfnstHeight ) >= 4 && sps_lfnst_enabled_flag == 1 &&<br>CuPredMode[ chType ][ x0 ][ y0 ] == MODE_INTRA && lfnstNotTsFlag == 1<br>&& ( treeType == DUAL_TREE_CHROMA    !IntraMipFlag[ x0 ][ y0 ]   <br>Min( lfnstWidth, lfnstHeight ) >= 16 ) &&<br>Max( cbWidth, cbHeight ) <= MaxTbSizeY ) {                                            |       |
| if( ( IntraSubPartitionsSplitType != ISP_NO_SPLIT    LfnstDcOnly == 0 ) &&<br>LfnstZeroOutSigCoeffFlag == 1 )                                                                                                                                                                                                                                                 |       |
| <b>lfnst_idx</b>                                                                                                                                                                                                                                                                                                                                              | ae(v) |
| }                                                                                                                                                                                                                                                                                                                                                             |       |
| if( treeType != DUAL_TREE_CHROMA && lfnst_idx == 0 &&<br>transform_skip_flag[ x0 ][ y0 ][ 0 ] == 0 && Max( cbWidth, cbHeight ) <= 32<br>&& IntraSubPartitionsSplitType == ISP_NO_SPLIT && cu_sbt_flag == 0<br>&& MtsZeroOutSigCoeffFlag == 1 && MtsDcOnly == 0 ) {                                                                                            |       |
| if( ( ( CuPredMode[ chType ][ x0 ][ y0 ] == MODE_INTER &&<br>sps_explicit_mts_inter_enabled_flag )   <br>( CuPredMode[ chType ][ x0 ][ y0 ] == MODE_INTRA &&<br>sps_explicit_mts_intra_enabled_flag ) ) )                                                                                                                                                     |       |
| <b>mts_idx</b>                                                                                                                                                                                                                                                                                                                                                | 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 ) :<br>( treeType == DUAL_TREE_CHROMA ) ? 2 : 1                                                                       |            |
| maxNumPaletteEntries = ( treeType == SINGLE_TREE ) ? 31 : 15                                                                                                                                        |            |
| palettePredictionFinished = 0                                                                                                                                                                       |            |
| NumPredictedPaletteEntries = 0                                                                                                                                                                      |            |
| for( predictorEntryIdx = 0; predictorEntryIdx < PredictorPaletteSize[ startComp ] &&<br>!palettePredictionFinished &&<br>NumPredictedPaletteEntries < maxNumPaletteEntries; predictorEntryIdx++ ) { |            |
| <b>palette_predictor_run</b>                                                                                                                                                                        | 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 )                                                                                                                                             |            |
| <b>num_signalled_palette_entries</b>                                                                                                                                                                | ae(v)      |
| for( cIdx = startComp; cIdx < ( startComp + numComps ); cIdx++ )                                                                                                                                    |            |
| for( i = 0; i < num_signalled_palette_entries; i++ )                                                                                                                                                |            |
| <b>new_palette_entries</b> [ cIdx ][ i ]                                                                                                                                                            | ae(v)      |
| if( CurrentPaletteSize[ startComp ] > 0 )                                                                                                                                                           |            |
| <b>palette_escape_val_present_flag</b>                                                                                                                                                              | ae(v)      |
| if( MaxPaletteIndex > 0 ) {                                                                                                                                                                         |            |
| adjust = 0                                                                                                                                                                                          |            |
| <b>palette_transpose_flag</b>                                                                                                                                                                       | ae(v)      |
| }                                                                                                                                                                                                   |            |
| if( treeType != DUAL_TREE_CHROMA && palette_escape_val_present_flag )                                                                                                                               |            |
| if( pps_cu_qp_delta_enabled_flag && !IsCuQpDeltaCoded ) {                                                                                                                                           |            |
| <b>cu_qp_delta_abs</b>                                                                                                                                                                              | ae(v)      |
| if( cu_qp_delta_abs )                                                                                                                                                                               |            |
| <b>cu_qp_delta_sign_flag</b>                                                                                                                                                                        | ae(v)      |
| }                                                                                                                                                                                                   |            |
| if( treeType != DUAL_TREE_LUMA && palette_escape_val_present_flag )                                                                                                                                 |            |
| if( sh_cu_chroma_qp_offset_enabled_flag && !IsCuChromaQpOffsetCoded ) {                                                                                                                             |            |
| <b>cu_chroma_qp_offset_flag</b>                                                                                                                                                                     | ae(v)      |
| if( cu_chroma_qp_offset_flag && pps_chroma_qp_offset_list_len_minus1 > 0 )                                                                                                                          |            |
| <b>cu_chroma_qp_offset_idx</b>                                                                                                                                                                      | 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 ) {                                                                                                                  |       |
| <b>run_copy_flag</b>                                                                                                                                               | 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 ) { |       |
| <b>copy_above_palette_indices_flag</b>                                                                                                                             | 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 )                                                                                                                                 |       |
| <b>palette_idx_idc</b>                                                                                                                                             | 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 ) {                  |       |
| <b>palette_escape_val</b>                                               | ae(v) |
| PaletteEscapeVal[ cIdx ][ xC ][ yC ] = palette_escape_val               |       |
| }                                                                       |       |
| }                                                                       |       |
| }                                                                       |       |
| }                                                                       |       |
| }                                                                       |       |
| }                                                                       |       |
| }                                                                       |       |
| }                                                                       |       |
| }                                                                       |       |
| }                                                                       |       |

### 7.3.11.7 Merge data syntax

|                                                                    |                   |
|--------------------------------------------------------------------|-------------------|
| merge_data( x0, y0, cbWidth, cbHeight, chType ) {                  | <b>Descriptor</b> |
| if( CuPredMode[ chType ][ x0 ][ y0 ] == MODE_IBC ) {               |                   |
| if( MaxNumIbcMergeCand > 1 )                                       |                   |
| <b>merge_idx</b> [ x0 ][ y0 ]                                      | ae(v)             |
| } else {                                                           |                   |
| if( MaxNumSubblockMergeCand > 0 && cbWidth >= 8 && cbHeight >= 8 ) |                   |
| <b>merge_subblock_flag</b> [ x0 ][ y0 ]                            | ae(v)             |
| if( merge_subblock_flag[ x0 ][ y0 ] == 1 ) {                       |                   |
| if( MaxNumSubblockMergeCand > 1 )                                  |                   |
| <b>merge_subblock_idx</b> [ 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 ) ) )       |                   |
| <b>regular_merge_flag</b> [ x0 ][ y0 ]                             | ae(v)             |
| if( regular_merge_flag[ x0 ][ y0 ] == 1 ) {                        |                   |
| if( sps_mmvd_enabled_flag )                                        |                   |
| <b>mmvd_merge_flag</b> [ x0 ][ y0 ]                                | ae(v)             |
| if( mmvd_merge_flag[ x0 ][ y0 ] == 1 ) {                           |                   |

|                                                                                                                                                                                                                                                        |       |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| if( MaxNumMergeCand > 1 )                                                                                                                                                                                                                              |       |
| <b>mmvd_cand_flag</b> [ x0 ][ y0 ]                                                                                                                                                                                                                     | ae(v) |
| <b>mmvd_distance_idx</b> [ x0 ][ y0 ]                                                                                                                                                                                                                  | ae(v) |
| <b>mmvd_direction_idx</b> [ x0 ][ y0 ]                                                                                                                                                                                                                 | ae(v) |
| } else if( MaxNumMergeCand > 1 )                                                                                                                                                                                                                       |       |
| <b>merge_idx</b> [ x0 ][ y0 ]                                                                                                                                                                                                                          | ae(v) |
| } else {                                                                                                                                                                                                                                               |       |
| if( sps_ciip_enabled_flag && sps_gpm_enabled_flag &&<br>sh_slice_type == B &&<br>cu_skip_flag[ x0 ][ y0 ] == 0 && cbWidth >= 8 && cbHeight >= 8 &&<br>cbWidth < ( 8 * cbHeight ) && cbHeight < ( 8 * cbWidth ) &&<br>cbWidth < 128 && cbHeight < 128 ) |       |
| <b>ciip_flag</b> [ x0 ][ y0 ]                                                                                                                                                                                                                          | ae(v) |
| if( ciip_flag[ x0 ][ y0 ] && MaxNumMergeCand > 1 )                                                                                                                                                                                                     |       |
| <b>merge_idx</b> [ x0 ][ y0 ]                                                                                                                                                                                                                          | ae(v) |
| if( !ciip_flag[ x0 ][ y0 ] ) {                                                                                                                                                                                                                         |       |
| <b>merge_gpm_partition_idx</b> [ x0 ][ y0 ]                                                                                                                                                                                                            | ae(v) |
| <b>merge_gpm_idx0</b> [ x0 ][ y0 ]                                                                                                                                                                                                                     | ae(v) |
| if( MaxNumGpmMergeCand > 2 )                                                                                                                                                                                                                           |       |
| <b>merge_gpm_idx1</b> [ x0 ][ y0 ]                                                                                                                                                                                                                     | ae(v) |
| }                                                                                                                                                                                                                                                      |       |
| }                                                                                                                                                                                                                                                      |       |
| }                                                                                                                                                                                                                                                      |       |
| }                                                                                                                                                                                                                                                      |       |
| }                                                                                                                                                                                                                                                      |       |

### 7.3.11.8 Motion vector difference syntax

| mvd_coding( x0, y0, refList, cpIdx ) { | Descriptor |
|----------------------------------------|------------|
| <b>abs_mvd_greater0_flag</b> [ 0 ]     | ae(v)      |
| <b>abs_mvd_greater0_flag</b> [ 1 ]     | ae(v)      |
| if( abs_mvd_greater0_flag[ 0 ] )       |            |
| <b>abs_mvd_greater1_flag</b> [ 0 ]     | ae(v)      |
| if( abs_mvd_greater0_flag[ 1 ] )       |            |
| <b>abs_mvd_greater1_flag</b> [ 1 ]     | ae(v)      |
| if( abs_mvd_greater0_flag[ 0 ] ) {     |            |
| if( abs_mvd_greater1_flag[ 0 ] )       |            |
| <b>abs_mvd_minus2</b> [ 0 ]            | ae(v)      |
| <b>mvd_sign_flag</b> [ 0 ]             | ae(v)      |
| }                                      |            |
| if( abs_mvd_greater0_flag[ 1 ] ) {     |            |
| if( abs_mvd_greater1_flag[ 1 ] )       |            |
| <b>abs_mvd_minus2</b> [ 1 ]            | ae(v)      |
| <b>mvd_sign_flag</b> [ 1 ]             | ae(v)      |
| }                                      |            |
| }                                      |            |

## 7.3.11.9 Transform tree syntax

|                                                                                              | Descriptor |
|----------------------------------------------------------------------------------------------|------------|
| transform_tree( x0, y0, tbWidth, tbHeight, treeType, chType ) {                              |            |
| InferTuCbflLuma = 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

|                                                                                                                              | Descriptor |
|------------------------------------------------------------------------------------------------------------------------------|------------|
| transform_unit( x0, y0, tbWidth, tbHeight, treeType, subTuIndex, chType ) {                                                  |            |
| if( IntraSubPartitionsSplitType != ISP_NO_SPLIT &&<br>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 {                                                                                                                     |            |

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                    |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|
| <code>xC = x0</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                    |
| <code>yC = y0</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                    |
| <code>wC = tbWidth / SubWidthC</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                    |
| <code>hC = tbHeight / SubHeightC</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                    |
| <code>}</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                    |
| <code>chromaAvailable = treeType != DUAL_TREE_LUMA &amp;&amp; sps_chroma_format_idc != 0<br/>&amp;&amp; ( IntraSubPartitionsSplitType == ISP_NO_SPLIT   <br/>( IntraSubPartitionsSplitType != ISP_NO_SPLIT &amp;&amp;<br/>subTuIndex == NumIntraSubPartitions - 1 ) )</code>                                                                                                                                                                                                                                                                                                                                                                                                                             |                    |
| <code>if( ( treeType == SINGLE_TREE    treeType == DUAL_TREE_CHROMA ) &amp;&amp;<br/>sps_chroma_format_idc != 0 &amp;&amp;<br/>( ( IntraSubPartitionsSplitType == ISP_NO_SPLIT &amp;&amp; !( cu_sbt_flag &amp;&amp;<br/>( subTuIndex == 0 &amp;&amp; cu_sbt_pos_flag )   <br/>( subTuIndex == 1 &amp;&amp; !cu_sbt_pos_flag ) ) )   <br/>( IntraSubPartitionsSplitType != ISP_NO_SPLIT &amp;&amp;<br/>( subTuIndex == NumIntraSubPartitions - 1 ) ) ) ) {</code>                                                                                                                                                                                                                                         |                    |
| <code>tu_cb_coded_flag[ xC ][ yC ]</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | <code>ae(v)</code> |
| <code>tu_cr_coded_flag[ xC ][ yC ]</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | <code>ae(v)</code> |
| <code>}</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                    |
| <code>if( treeType == SINGLE_TREE    treeType == DUAL_TREE_LUMA ) {</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                    |
| <code>if( ( IntraSubPartitionsSplitType == ISP_NO_SPLIT &amp;&amp; !( cu_sbt_flag &amp;&amp;<br/>( subTuIndex == 0 &amp;&amp; cu_sbt_pos_flag )   <br/>( subTuIndex == 1 &amp;&amp; !cu_sbt_pos_flag ) ) ) &amp;&amp;<br/>( ( CuPredMode[ chType ][ x0 ][ y0 ] == MODE_INTRA &amp;&amp;<br/>!cu_act_enabled_flag[ x0 ][ y0 ] )   <br/>( chromaAvailable &amp;&amp; ( tu_cb_coded_flag[ xC ][ yC ]   <br/>tu_cr_coded_flag[ xC ][ yC ] ) )   <br/>CbWidth[ chType ][ x0 ][ y0 ] &gt; MaxTbSizeY   <br/>CbHeight[ chType ][ x0 ][ y0 ] &gt; MaxTbSizeY ) )   <br/>( IntraSubPartitionsSplitType != ISP_NO_SPLIT &amp;&amp;<br/>( subTuIndex &lt; NumIntraSubPartitions - 1    !InferTuCbfLuma ) ) )</code> |                    |
| <code>tu_y_coded_flag[ x0 ][ y0 ]</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | <code>ae(v)</code> |
| <code>if(IntraSubPartitionsSplitType != ISP_NO_SPLIT)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                    |
| <code>InferTuCbfLuma = InferTuCbfLuma &amp;&amp; !tu_y_coded_flag[ x0 ][ y0 ]</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                    |
| <code>}</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                    |
| <code>if( ( CbWidth[ chType ][ x0 ][ y0 ] &gt; 64    CbHeight[ chType ][ x0 ][ y0 ] &gt; 64   <br/>tu_y_coded_flag[ x0 ][ y0 ]    ( chromaAvailable &amp;&amp;<br/>( tu_cb_coded_flag[ xC ][ yC ]    tu_cr_coded_flag[ xC ][ yC ] ) ) ) &amp;&amp;<br/>treeType != DUAL_TREE_CHROMA &amp;&amp;<br/>pps_cu_qp_delta_enabled_flag &amp;&amp; !IsCuQpDeltaCoded ) {</code>                                                                                                                                                                                                                                                                                                                                  |                    |
| <code>cu_qp_delta_abs</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | <code>ae(v)</code> |
| <code>if( cu_qp_delta_abs )</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                    |
| <code>cu_qp_delta_sign_flag</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | <code>ae(v)</code> |
| <code>}</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                    |
| <code>if( ( CbWidth[ chType ][ x0 ][ y0 ] &gt; 64    CbHeight[ chType ][ x0 ][ y0 ] &gt; 64   <br/>( chromaAvailable &amp;&amp; ( tu_cb_coded_flag[ xC ][ yC ]   <br/>tu_cr_coded_flag[ xC ][ yC ] ) ) ) &amp;&amp;<br/>treeType != DUAL_TREE_LUMA &amp;&amp; sh_cu_chroma_qp_offset_enabled_flag &amp;&amp;<br/>!IsCuChromaQpOffsetCoded ) {</code>                                                                                                                                                                                                                                                                                                                                                     |                    |
| <code>cu_chroma_qp_offset_flag</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | <code>ae(v)</code> |
| <code>if( cu_chroma_qp_offset_flag &amp;&amp; pps_chroma_qp_offset_list_len_minus1 &gt; 0 )</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                    |
| <code>cu_chroma_qp_offset_idx</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | <code>ae(v)</code> |
| <code>}</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                    |

|                                                                                                                                                                                                                                                                   |       |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| if( sps_joint_cbr_enabled_flag &&<br>( ( CuPredMode[ chType ][ x0 ][ y0 ] == MODE_INTRA &&<br>( tu_cb_coded_flag[ xC ][ yC ]    tu_cr_coded_flag[ xC ][ yC ] ) )   <br>( tu_cb_coded_flag[ xC ][ yC ] && tu_cr_coded_flag[ xC ][ yC ] ) ) &&<br>chromaAvailable ) |       |
| <b>tu_joint_cbr_residual_flag[ xC ][ yC ]</b>                                                                                                                                                                                                                     | ae(v) |
| if( tu_y_coded_flag[ x0 ][ y0 ] && treeType != DUAL_TREE_CHROMA ) {                                                                                                                                                                                               |       |
| if( sps_transform_skip_enabled_flag && !BdpcmFlag[ x0 ][ y0 ][ 0 ] &&<br>tbWidth <= MaxTsSize && tbHeight <= MaxTsSize &&<br>( IntraSubPartitionsSplitType == ISP_NO_SPLIT ) && !cu_sbt_flag )                                                                    |       |
| <b>transform_skip_flag[ x0 ][ y0 ][ 0 ]</b>                                                                                                                                                                                                                       | 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 ] &&<br>wC <= MaxTsSize && hC <= MaxTsSize && !cu_sbt_flag )                                                                                                                                     |       |
| <b>transform_skip_flag[ xC ][ yC ][ 1 ]</b>                                                                                                                                                                                                                       | 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 &&<br>!( tu_cb_coded_flag[ xC ][ yC ] && tu_joint_cbr_residual_flag[ xC ][ yC ] ) ) {                                                                                                              |       |
| if( sps_transform_skip_enabled_flag && !BdpcmFlag[ x0 ][ y0 ][ 2 ] &&<br>wC <= MaxTsSize && hC <= MaxTsSize && !cu_sbt_flag )                                                                                                                                     |       |
| <b>transform_skip_flag[ xC ][ yC ][ 2 ]</b>                                                                                                                                                                                                                       | 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 ) {                                      | <b>Descriptor</b> |
| if( sps_mts_enabled_flag && cu_sbt_flag && cIdx == 0 &&<br>log2TbWidth == 5 && log2TbHeight < 6 ) |                   |
| Log2ZoTbWidth = 4                                                                                 |                   |
| else                                                                                              |                   |
| Log2ZoTbWidth = Min( log2TbWidth, 5 )                                                             |                   |
| if( sps_mts_enabled_flag && cu_sbt_flag && cIdx == 0 &&<br>log2TbWidth < 6 && log2TbHeight == 5 ) |                   |
| Log2ZoTbHeight = 4                                                                                |                   |
| else                                                                                              |                   |
| Log2ZoTbHeight = Min( log2TbHeight, 5 )                                                           |                   |
| if( log2TbWidth > 0 )                                                                             |                   |
| <b>last_sig_coeff_x_prefix</b>                                                                    | ae(v)             |



|                                                                                                                                                                                   |       |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| if( log2TbHeight > 0 )                                                                                                                                                            |       |
| <b>last_sig_coeff_y_prefix</b>                                                                                                                                                    | ae(v) |
| if( last_sig_coeff_x_prefix > 3 )                                                                                                                                                 |       |
| <b>last_sig_coeff_x_suffix</b>                                                                                                                                                    | ae(v) |
| if( last_sig_coeff_y_prefix > 3 )                                                                                                                                                 |       |
| <b>last_sig_coeff_y_suffix</b>                                                                                                                                                    | ae(v) |
| Log2FullTbWidth = log2TbWidth                                                                                                                                                     |       |
| Log2FullTbHeight = log2TbHeight                                                                                                                                                   |       |
| 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[ cldx ] ) : 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 ][ cldx ] && 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 ) && cldx == 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 ) {                                                                                                              |       |
| <b>sb_coded_flag</b> [ xS ][ yS ]                                                                                                              | ae(v) |
| inferSbDcSigCoeffFlag = 1                                                                                                                      |       |
| }                                                                                                                                              |       |
| if( sb_coded_flag[ xS ][ yS ] && ( xS > 3    yS > 3 ) && cldx == 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 ) ) {     |       |
| <b>sig_coeff_flag</b> [ xC ][ yC ]                                                                                                             | ae(v) |
| remBinsPass1--                                                                                                                                 |       |
| if( sig_coeff_flag[ xC ][ yC ] )                                                                                                               |       |
| inferSbDcSigCoeffFlag = 0                                                                                                                      |       |
| }                                                                                                                                              |       |
| if( sig_coeff_flag[ xC ][ yC ] ) {                                                                                                             |       |
| <b>abs_level_gtx_flag</b> [ n ][ 0 ]                                                                                                           | ae(v) |
| remBinsPass1--                                                                                                                                 |       |
| if( abs_level_gtx_flag[ n ][ 0 ] ) {                                                                                                           |       |
| <b>par_level_flag</b> [ n ]                                                                                                                    | ae(v) |
| remBinsPass1--                                                                                                                                 |       |
| <b>abs_level_gtx_flag</b> [ 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 ] ) {                                                                                                           |       |
| <b>abs_remainder</b> [ n ]                                                                                                                     | ae(v) |
| if( updateHist && abs_remainder[ n ] > 0 ) {                                                                                                   |       |

|                                                                                                                                                  |       |
|--------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| StatCoeff[ cIdx ] = ( StatCoeff[ cIdx ] +<br>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 ] ) {                                                                                                                |       |
| <b>dec_abs_level[ n ]</b>                                                                                                                        | ae(v) |
| if( updateHist && dec_abs_level[ n ] > 0 ) {                                                                                                     |       |
| StatCoeff[ cIdx ] = ( StatCoeff[ cIdx ] +<br>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 &&<br>( 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 ) &&<br>( !signHiddenFlag    ( n != firstSigScanPosSb ) ) )                                                       |       |
| <b>coeff_sign_flag[ n ]</b>                                                                                                                      | 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 ] =<br>( 2 * AbsLevel[ xC ][ yC ] - ( QState > 1 ? 1 : 0 ) ) *<br>( 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 ]        |  |
| }                                                       |  |
| }                                                       |  |
| }                                                       |  |
| }                                                       |  |
| }                                                       |  |
| }                                                       |  |

|                                                                                    | Descriptor |
|------------------------------------------------------------------------------------|------------|
| residual_ts_coding( x0, y0, log2TbWidth, log2TbHeight, cIdx ) {                    |            |
| 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                                                                     |            |
| RemCpbs = ( ( 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 )                                             |            |
| <b>sb_coded_flag</b> [ 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 && RemCpbs >= 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 ) ) {                                |            |
| <b>sig_coeff_flag</b> [ xC ][ yC ]                                                 | ae(v)      |
| RemCpbs--                                                                          |            |

|                                                                                 |       |
|---------------------------------------------------------------------------------|-------|
| if( sig_coeff_flag[ xC ][ yC ] )                                                |       |
| inferSbSigCoeffFlag = 0                                                         |       |
| }                                                                               |       |
| CoeffSignLevel[ xC ][ yC ] = 0                                                  |       |
| if( sig_coeff_flag[ xC ][ yC ] ) {                                              |       |
| <b>coeff_sign_flag</b> [ n ]                                                    | ae(v) |
| RemCcbs--                                                                       |       |
| CoeffSignLevel[ xC ][ yC ] = ( coeff_sign_flag[ n ] > 0 ? -1 : 1 )              |       |
| <b>abs_level_gtx_flag</b> [ n ][ 0 ]                                            | ae(v) |
| RemCcbs--                                                                       |       |
| if( abs_level_gtx_flag[ n ][ 0 ] ) {                                            |       |
| <b>par_level_flag</b> [ 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 ] ) {                                        |       |
| <b>abs_level_gtx_flag</b> [ 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 ] ) )                         |       |
| <b>abs_remainder</b> [ 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 ] )                                                        |       |
| <b>coeff_sign_flag</b> [ 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 ] ) *<br>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 subclause 7.4. When the semantics of a syntax element are specified using a table or a set of tables, any values that are not specified in the table(s) shall not be present in the bitstream unless otherwise specified in this document.

### 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 document.

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:

- a) If the SODB is empty (i.e., zero bits in length), the RBSP is also empty.
- b) 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 document, decoders conforming to this version of this document 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 document, decoders conforming to this version of this document 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 document.

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 document. 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 document 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 document.

**Table 5 — NAL unit type codes and NAL unit type classes**

| <code>nal_unit_type</code> | Name of <code>nal_unit_type</code> | Content of NAL unit and RBSP syntax structure                                          | NAL unit type class |
|----------------------------|------------------------------------|----------------------------------------------------------------------------------------|---------------------|
| 0                          | TRAIL_NUT                          | Coded slice of a trailing picture or subpicture*<br><code>slice_layer_rbsp()</code>    | VCL                 |
| 1                          | STSA_NUT                           | Coded slice of an STSA picture or subpicture*<br><code>slice_layer_rbsp()</code>       | VCL                 |
| 2                          | RADL_NUT                           | Coded slice of a RADL picture or subpicture*<br><code>slice_layer_rbsp()</code>        | VCL                 |
| 3                          | RASL_NUT                           | Coded slice of a RASL picture or subpicture*<br><code>slice_layer_rbsp()</code>        | VCL                 |
| 4..6                       | RSV_VCL_4..<br>RSV_VCL_6           | Reserved non-IRAP VCL NAL unit types                                                   | VCL                 |
| 7<br>8                     | IDR_W_RADL<br>IDR_N_LP             | Coded slice of an IDR picture or subpicture*<br><code>slice_layer_rbsp()</code>        | VCL                 |
| 9                          | CRA_NUT                            | Coded slice of a CRA picture or subpicture*<br><code>slice_layer_rbsp()</code>         | VCL                 |
| 10                         | GDR_NUT                            | Coded slice of a GDR picture or subpicture*<br><code>slice_layer_rbsp()</code>         | VCL                 |
| 11                         | RSV_IRAP_11                        | Reserved IRAP VCL NAL unit type                                                        | VCL                 |
| 12                         | OPI_NUT                            | Operating point information<br><code>operating_point_information_rbsp()</code>         | non-VCL             |
| 13                         | DCI_NUT                            | Decoding capability information<br><code>decoding_capability_information_rbsp()</code> | non-VCL             |
| 14                         | VPS_NUT                            | Video parameter set<br><code>video_parameter_set_rbsp()</code>                         | non-VCL             |
| 15                         | SPS_NUT                            | Sequence parameter set<br><code>seq_parameter_set_rbsp()</code>                        | non-VCL             |



| <b>nal_unit_type</b>                                                                                                                                                          | <b>Name of nal_unit_type</b>     | <b>Content of NAL unit and RBSP syntax structure</b>         | <b>NAL unit type class</b> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------|--------------------------------------------------------------|----------------------------|
| 16                                                                                                                                                                            | PPS_NUT                          | Picture parameter set<br>pic_parameter_set_rbsp( )           | non-VCL                    |
| 17<br>18                                                                                                                                                                      | PREFIX_APS_NUT<br>SUFFIX_APS_NUT | Adaptation parameter set<br>adaptation_parameter_set_rbsp( ) | non-VCL                    |
| 19                                                                                                                                                                            | PH_NUT                           | Picture header<br>picture_header_rbsp( )                     | non-VCL                    |
| 20                                                                                                                                                                            | AUD_NUT                          | AU delimiter<br>access_unit_delimiter_rbsp( )                | non-VCL                    |
| 21                                                                                                                                                                            | EOS_NUT                          | End of sequence<br>end_of_seq_rbsp( )                        | non-VCL                    |
| 22                                                                                                                                                                            | EOB_NUT                          | End of bitstream<br>end_of_bitstream_rbsp( )                 | non-VCL                    |
| 23<br>24                                                                                                                                                                      | PREFIX_SEI_NUT<br>SUFFIX_SEI_NUT | Supplemental enhancement information<br>sei_rbsp( )          | non-VCL                    |
| 25                                                                                                                                                                            | FD_NUT                           | Filler data<br>filler_data_rbsp( )                           | non-VCL                    |
| 26<br>27                                                                                                                                                                      | RSV_NVCL_26<br>RSV_NVCL_27       | Reserved non-VCL NAL unit types                              | non-VCL                    |
| 28..31                                                                                                                                                                        | UNSPEC_28..<br>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 document) 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

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 subclauses specifies constraints on the order of NAL units in the bitstream.

Any order of NAL units in the bitstream obeying these constraints is referred to in the text as the decoding order of NAL units.

Within a NAL unit, the syntax in subclauses 7.3 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 document, 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 CVSSs

A bitstream consists of one or more CVSSs.

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

The first AU of a CVSS 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 CVSS and each picture in an AU in a CVSS shall have nuh\_layer\_id equal to the nuh\_layer\_id of one of the pictures present in the first AU of the CVSS.

#### 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 subclause 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 subclause 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 subclause 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 index 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 document.

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 document. 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 document. However, some use of **dci\_extension\_flag** equal to 1 could be specified in some future version of this document, and decoders conforming to this version of this document 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 document. Decoders conforming to this version of this document 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 document. However, some use of **opi\_extension\_flag** equal to 1 could be specified in some future version of this document, and decoders conforming to this version of this document 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 document. Decoders conforming to this version of this document 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++ ) {
        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
}

```

(28)

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++ )
    GeneralLayerIdx[vps_layer_id[i]] = i

```

(29)

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.  $\text{vps\_max\_tid\_il\_ref\_pics\_plus1}[i][j]$  greater than 0 specifies that, for decoding pictures of the  $i$ -th layer, no picture from the  $j$ -th layer with  $\text{TemporalId}$  greater than  $\text{vps\_max\_tid\_il\_ref\_pics\_plus1}[i][j] - 1$  is used as ILRP and no APS with  $\text{nuh\_layer\_id}$  equal to  $\text{vps\_layer\_id}[j]$  and  $\text{TemporalId}$  greater than  $\text{vps\_max\_tid\_il\_ref\_pics\_plus1}[i][j] - 1$  is referenced. When not present, the value of  $\text{vps\_max\_tid\_il\_ref\_pics\_plus1}[i][j]$  is inferred to be equal to  $\text{vps\_max\_sublayers\_minus1} + 1$ .

**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  $\text{vps\_max\_layers\_minus1}$  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  $\text{vps\_max\_layers\_minus1} + 1$ , 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  $\text{vps\_max\_layers\_minus1} + 1$ , 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 document 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
else
    olsModeIdc = 4
```

(30)

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
else if( olsModeIdc == 2 )
    TotalNumOlss = vps_num_output_layer_sets_minus2 + 2
```

(31)

**vps\_ols\_output\_layer\_flag** $[i][j]$  equal to 1 specifies that the layer with  $\text{nuh\_layer\_id}$  equal to  $\text{vps\_layer\_id}[j]$  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  $\text{nuh\_layer\_id}$  equal to  $\text{vps\_layer\_id}[j]$  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  $\text{nuh\_layer\_id}$  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++ )
        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++ ) {

```

(32)

```

    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  $TotalNumOls - 1$ , 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 ]
    } 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++
    }
}

```

(33)

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  $\text{TotalNumOls} - 1$ , inclusive, the value of  $\text{vps\_independent\_layer\_flag}[\text{GeneralLayerIdx}[\text{LayerIdInOls}[i][0]]]$  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  $\text{nuh\_layer\_id}$   $\text{nuhLayerId}$  equal to one of  $\text{vps\_layer\_id}[k]$  for  $k$  in the range of 0 to  $\text{vps\_max\_layers\_minus1}$ , inclusive, there shall be at least one pair of values of  $i$  and  $j$ , where  $i$  is in the range of 0 to  $\text{TotalNumOls} - 1$ , inclusive, and  $j$  is in the range of  $\text{NumLayersInOls}[i] - 1$ , inclusive, such that the value of  $\text{LayerIdInOls}[i][j]$  is equal to  $\text{nuhLayerId}$ .

**vps\_num\_ptls\_minus1** plus 1 specifies the number of  $\text{profile\_tier\_level}()$  syntax structures in the VPS. The value of  $\text{vps\_num\_ptls\_minus1}$  shall be less than  $\text{TotalNumOls}$ . When not present, the value of  $\text{vps\_num\_ptls\_minus1}$  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  $\text{profile\_tier\_level}()$  syntax structure in the VPS.  $\text{vps\_pt\_present\_flag}[i]$  equal to 0 specifies that profile, tier, and general constraints information are not present in the  $i$ -th  $\text{profile\_tier\_level}()$  syntax structure in the VPS. The value of  $\text{vps\_pt\_present\_flag}[0]$  is inferred to be equal to 1. When  $\text{vps\_pt\_present\_flag}[i]$  is equal to 0, the profile, tier, and general constraints information for the  $i$ -th  $\text{profile\_tier\_level}()$  syntax structure in the VPS are inferred to be the same as that for the  $(i - 1)$ -th  $\text{profile\_tier\_level}()$  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  $\text{profile\_tier\_level}()$  syntax structure in the VPS and the TemporalId of the highest sublayer representation that is present in the OLSs with OLS index  $\text{olsIdx}$  such that  $\text{vps\_ols\_ptl\_idx}[\text{olsIdx}]$  is equal to  $i$ . The value of  $\text{vps\_ptl\_max\_tid}[i]$  shall be in the range of 0 to  $\text{vps\_max\_sublayers\_minus1}$ , inclusive. When  $\text{vps\_default\_ptl\_dpb\_hrd\_max\_tid\_flag}$  is equal to 1, the value of  $\text{vps\_ptl\_max\_tid}[i]$  is inferred to be equal to  $\text{vps\_max\_sublayers\_minus1}$ .

**vps\_ptl\_alignment\_zero\_bit** shall be equal to 0.

**vps\_ols\_ptl\_idx** $[i]$  specifies the index, to the list of  $\text{profile\_tier\_level}()$  syntax structures in the VPS, of the  $\text{profile\_tier\_level}()$  syntax structure that applies to the  $i$ -th OLS. When present, the value of  $\text{vps\_ols\_ptl\_idx}[i]$  shall be in the range of 0 to  $\text{vps\_num\_ptls\_minus1}$ , inclusive.

When not present, the value of  $\text{vps\_ols\_ptl\_idx}[i]$  is inferred as follows:

- If  $\text{vps\_num\_ptls\_minus1}$  is equal to 0, the value of  $\text{vps\_ols\_ptl\_idx}[i]$  is inferred to be equal to 0.
- Otherwise ( $\text{vps\_num\_ptls\_minus1}$  is greater than 0 and  $\text{vps\_num\_ptls\_minus1} + 1$  is equal to  $\text{TotalNumOls}$ ), the value of  $\text{vps\_ols\_ptl\_idx}[i]$  is inferred to be equal to  $i$ .

When  $\text{NumLayersInOls}[i]$  is equal to 1, the  $\text{profile\_tier\_level}()$  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  $\text{NumLayersInOls}[i]$  is equal to 1, the  $\text{profile\_tier\_level}()$  syntax structures signalled in the VPS and in the SPS for the  $i$ -th OLS shall be identical.

Each  $\text{profile\_tier\_level}()$  syntax structure in the VPS shall be referred to by at least one value of  $\text{vps\_ols\_ptl\_idx}[i]$  for  $i$  in the range of 0 to  $\text{TotalNumOls} - 1$ , inclusive.

**vps\_num\_dpb\_params\_minus1** plus 1, when present, specifies the number of  $\text{dpb\_parameters}()$  syntax structures in the VPS. The value of  $\text{vps\_num\_dpb\_params\_minus1}$  shall be in the range of 0 to  $\text{NumMultiLayerOls} - 1$ , inclusive.

The variable  $\text{VpsNumDpbParams}$ , specifying the number of  $\text{dpb\_parameters}()$  syntax structures in the VPS, is derived as follows:

```

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

```

(34)

**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 structures 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 structure 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 **NumLayersInOlss[ 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 document. However, some use of `vps_extension_flag` equal to 1 could be specified in some future version of this document, and decoders conforming to this version of this document 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 document. Decoders conforming to this version of this document 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 `TotalNumOls` 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 document 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 document 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 subclause 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 document 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  $\text{Max}(8, \text{MinCbSizeY})$ .

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  $\text{NumMultiLayerOlss} - 1$ , 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  $\text{Max}(8, \text{MinCbSizeY})$ .

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  $\text{NumMultiLayerOlss} - 1$ , 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  $\text{SubWidthC} * \text{sps\_conf\_win\_left\_offset}$  to  $\text{sps\_pic\_width\_max\_in\_luma\_samples} - (\text{SubWidthC} * \text{sps\_conf\_win\_right\_offset} + 1)$  and vertical picture coordinates from  $\text{SubHeightC} * \text{sps\_conf\_win\_top\_offset}$  to  $\text{sps\_pic\_height\_max\_in\_luma\_samples} - (\text{SubHeightC} * \text{sps\_conf\_win\_bottom\_offset} + 1)$ , inclusive.

The value of  $\text{SubWidthC} * (\text{sps\_conf\_win\_left\_offset} + \text{sps\_conf\_win\_right\_offset})$  shall be less than `sps_pic_width_max_in_luma_samples`, and the value of  $\text{SubHeightC} * (\text{sps\_conf\_win\_top\_offset} + \text{sps\_conf\_win\_bottom\_offset})$  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  $(x / \text{SubWidthC}, y / \text{SubHeightC})$ , 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  $(\text{sps\_pic\_width\_max\_in\_luma\_samples} + \text{CtbSizeY} - 1) / \text{CtbSizeY}$ , and the variable `tmpHeightVal` be set equal to  $(\text{sps\_pic\_height\_max\_in\_luma\_samples} + \text{CtbSizeY} - 1) / \text{CtbSizeY}$ .

**`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  $\text{Ceil}(\text{Log2}(\text{tmpWidthVal}))$  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  $(i \% \text{numSubpicCols}) * (\text{sps\_subpic\_width\_minus1}[0] + 1)$ .

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  $\text{Ceil}(\text{Log2}(\text{tmpHeightVal}))$  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  $(i / \text{numSubpicCols}) * (\text{sps\_subpic\_height\_minus1}[0] + 1)$ .

**`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  $\text{Ceil}(\text{Log2}(\text{tmpWidthVal}))$  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  $\text{Ceil}(\text{Log2}(\text{tmpHeightVal}))$  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  $(\text{sps\_subpic\_ctu\_top\_left\_x}[i] * \text{CtbSizeY})$  shall be less than  $(\text{sps\_pic\_width\_max\_in\_luma\_samples} - \text{sps\_conf\_win\_right\_offset} * \text{SubWidthC})$ .
- The value of  $((\text{sps\_subpic\_ctu\_top\_left\_x}[i] + \text{sps\_subpic\_width\_minus1}[i] + 1) * \text{CtbSizeY})$  shall be greater than  $(\text{sps\_conf\_win\_left\_offset} * \text{SubWidthC})$ .
- The value of  $(\text{sps\_subpic\_ctu\_top\_left\_y}[i] * \text{CtbSizeY})$  shall be less than  $(\text{sps\_pic\_height\_max\_in\_luma\_samples} - \text{sps\_conf\_win\_bottom\_offset} * \text{SubHeightC})$ .
- The value of  $((\text{sps\_subpic\_ctu\_top\_left\_y}[i] + \text{sps\_subpic\_height\_minus1}[i] + 1) * \text{CtbSizeY})$  shall be greater than  $(\text{sps\_conf\_win\_top\_offset} * \text{SubHeightC})$ .

**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_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  $1 \ll (\text{sps\_subpic\_id\_len\_minus1} + 1)$  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 `NumMultiLayerOls - 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 document. Although the value of **sps\_num\_extra\_ph\_bytes** is required to be equal to 0 in this version of this document, decoders conforming to this version of this document 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 ] )
        NumExtraPhBits++

```

(41)

**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 document. Although the value of **sps\_num\_extra\_sh\_bytes** is required to be equal to 0 in this version of this document, decoders conforming to this version of this document 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 ] )
        NumExtraShBits++

```

(42)

**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 structure 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:

$$\text{MinCbLog2SizeY} = \text{sps\_log2\_min\_luma\_coding\_block\_size\_minus2} + 2 \quad (43)$$

$$\text{MinCbSizeY} = 1 \ll \text{MinCbLog2SizeY} \quad (44)$$

$$\text{IbcBufWidthY} = 256 * 128 / \text{CtbSizeY} \quad (45)$$

$$\text{IbcBufWidthC} = \text{IbcBufWidthY} / \text{SubWidthC} \quad (46)$$

$$\text{VSize} = \text{Min}( 64, \text{CtbSizeY} ) \quad (47)$$

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:

$$\text{CtbWidthC} = \text{CtbSizeY} / \text{SubWidthC} \quad (48)$$

$$\text{CtbHeightC} = \text{CtbSizeY} / \text{SubHeightC} \quad (49)$$

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 subclause 6.5.2 is invoked with  $1 << \text{log2BlockWidth}$  and  $1 << \text{log2BlockHeight}$  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 subclause 6.5.3 is invoked with  $1 << \text{log2BlockWidth}$  and  $1 << \text{log2BlockHeight}$  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  $\text{Min}(6, \text{CtbLog2SizeY}) - \text{MinCbLog2SizeY}$ , 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:

$$\text{MinQtLog2SizeIntraY} = \text{sps\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_luma} + \text{MinCbLog2SizeY} \quad (50)$$

**`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  $2 * (\text{CtbLog2SizeY} - \text{MinCbLog2SizeY})$ , 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  $\text{CtbLog2SizeY} - \text{MinQtLog2SizeIntraY}$ , 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  $\text{Min}(6, \text{CtbLog2SizeY}) - \text{MinQtLog2SizeIntraY}$ , 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  $64 \times 64$  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  $\text{Min}(6, \text{CtbLog2SizeY}) - \text{MinQtLog2SizeIntraY}$ , 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  $\text{Min}(6, \text{CtbLog2SizeY}) - \text{MinCbLog2SizeY}$ , 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  $2 * (\text{CtbLog2SizeY} - \text{MinCbLog2SizeY})$ , 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  $\text{Min}(6, \text{CtbLog2SizeY}) - \text{MinQtLog2SizeIntraC}$ , 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  $\text{Min}(6, \text{CtbLog2SizeY}) - \text{MinQtLog2SizeIntraC}$ , 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  $\text{Min}(6, \text{CtbLog2SizeY}) - \text{MinCbLog2SizeY}$ , 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  $2 * (\text{CtbLog2SizeY} - \text{MinCbLog2SizeY})$ , 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  $\text{CtbLog2SizeY} - \text{MinQtLog2SizeInterY}$ , 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  $\text{Min}(6, \text{CtbLog2SizeY}) - \text{MinQtLog2SizeInterY}$ , 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  $\text{MinTbLog2SizeY}$ ,  $\text{MaxTbLog2SizeY}$ ,  $\text{MinTbSizeY}$ , and  $\text{MaxTbSizeY}$  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  $\text{MaxTsSize}$  is set equal to  $1 \ll (\text{sps\_log2\_transform\_skip\_max\_size\_minus2} + 2)$ .

**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\_cbr\_enabled\_flag** equal to 1 specifies that the joint coding of chroma residuals is enabled for the CLVS. **sps\_joint\_cbr\_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\_cbr\_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\_cbr\_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\_cbr\_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  $-26 - \text{QpBdOffset}$  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 )
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, rplIdx)** 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, rplIdx)** is inferred to be equal to the value of corresponding syntax element in **ref\_pic\_list\_struct(0, rplIdx)** for **rplIdx** 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, rplIdx)** 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, rplIdx)** syntax structures since there could be one **ref\_pic\_list\_struct(listIdx, rplIdx)** 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] + 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 CLVS. **sps\_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\_only\_enabled\_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:

$$\text{MaxNumMergeCand} = 6 - \text{sps\_six\_minus\_max\_num\_merge\_cand} \quad (58)$$

**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
else if( sps_gpm_enabled_flag && MaxNumMergeCand == 2 )
    MaxNumGpmMergeCand = 2
else
    MaxNumGpmMergeCand = 0

```

(59)

**`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 subclause 8.5.2.3, the derivation process for motion vectors and reference indices in subblock merge mode as specified in subclause 8.5.5.2, and to control the invocation of the updating process for the history-based motion vector predictor list in subclause 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:

$$\text{Log2ParMrgLevel} = \text{sps\_log2\_parallel\_merge\_level\_minus2} + 2 \quad (60)$$

**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 specifies 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:

$$QpPrimeTsMin = 4 + 6 * \text{sps\_min\_qp\_prime\_ts} \quad (61)$$

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
else
    MaxNumIbcMergeCand = 0

```

(62)

**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 subclause 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 subclause 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  $2^{\text{BitDepth}} - 3$ , 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:

$$\text{SpsLadfIntervalLowerBound}[i + 1] = \text{SpsLadfIntervalLowerBound}[i] + \text{sps\_ladf\_delta\_threshold\_minus1}[i] + 1 \quad (63)$$

**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**  $\leq 8 ? 0 : 3$ ), 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  $\text{Ceil}(\text{sps\_pic\_width\_max\_in\_luma\_samples} \div 8) - 2$ , 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**  $\leq 8 ? 0 : 3$ ), 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  $\text{Ceil}(\text{sps\_pic\_height\_max\_in\_luma\_samples} \div 8) - 2$ , 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 subclause D.8 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 referring 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  $\text{Max}(8, \text{MinCbSizeY})$ , 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  $(\text{CtbSizeY} / \text{MinCbSizeY} + 1)$  shall be less than or equal to the value of  $(\text{pps\_pic\_width\_in\_luma\_samples} / \text{MinCbSizeY} - 1)$ .

**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  $\text{Max}(8, \text{MinCbSizeY})$ , 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:

$$\text{PicWidthInCtbsY} = \text{Ceil}(\text{pps\_pic\_width\_in\_luma\_samples} \div \text{CtbSizeY}) \quad (64)$$

$$\text{PicHeightInCtbsY} = \text{Ceil}(\text{pps\_pic\_height\_in\_luma\_samples} \div \text{CtbSizeY}) \quad (65)$$

$$\text{PicSizeInCtbsY} = \text{PicWidthInCtbsY} * \text{PicHeightInCtbsY} \quad (66)$$

$$\text{PicWidthInMinCbsY} = \text{pps\_pic\_width\_in\_luma\_samples} / \text{MinCbSizeY} \quad (67)$$

$$\text{PicHeightInMinCbsY} = \text{pps\_pic\_height\_in\_luma\_samples} / \text{MinCbSizeY} \quad (68)$$

$$\text{PicSizeInMinCbsY} = \text{PicWidthInMinCbsY} * \text{PicHeightInMinCbsY} \quad (69)$$

$$\text{PicSizeInSamplesY} = \text{pps\_pic\_width\_in\_luma\_samples} * \text{pps\_pic\_height\_in\_luma\_samples} \quad (70)$$

$$\text{PicWidthInSamplesC} = \text{pps\_pic\_width\_in\_luma\_samples} / \text{SubWidthC} \quad (71)$$

$$\text{PicHeightInSamplesC} = \text{pps\_pic\_height\_in\_luma\_samples} / \text{SubHeightC} \quad (72)$$

**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  $\text{SubWidthC} * \text{pps\_conf\_win\_left\_offset}$  to  $\text{pps\_pic\_width\_in\_luma\_samples} - (\text{SubWidthC} * \text{pps\_conf\_win\_right\_offset} + 1)$  and vertical picture coordinates from  $\text{SubHeightC} * \text{pps\_conf\_win\_top\_offset}$  to  $\text{pps\_pic\_height\_in\_luma\_samples} - (\text{SubHeightC} * \text{pps\_conf\_win\_bottom\_offset} + 1)$ , inclusive.

The value of  $\text{SubWidthC} * (\text{pps\_conf\_win\_left\_offset} + \text{pps\_conf\_win\_right\_offset})$  shall be less than **pps\_pic\_width\_in\_luma\_samples**, and the value of  $\text{SubHeightC} * (\text{pps\_conf\_win\_top\_offset} + \text{pps\_conf\_win\_bottom\_offset})$  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  $(x / \text{SubWidthC}, y / \text{SubHeightC})$ , 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  $\text{SubWidthC} * \text{pps\_scaling\_win\_left\_offset}$  and  $\text{SubWidthC} * \text{pps\_scaling\_win\_right\_offset}$  shall both be greater than or equal to  $-\text{pps\_pic\_width\_in\_luma\_samples} * 15$  and less than  $\text{pps\_pic\_width\_in\_luma\_samples}$ . The values of  $\text{SubHeightC} * \text{pps\_scaling\_win\_top\_offset}$  and  $\text{SubHeightC} * \text{pps\_scaling\_win\_bottom\_offset}$  shall both be greater than or equal to  $-\text{pps\_pic\_height\_in\_luma\_samples} * 15$  and less than  $\text{pps\_pic\_height\_in\_luma\_samples}$ .

The value of  $\text{SubWidthC} * (\text{pps\_scaling\_win\_left\_offset} + \text{pps\_scaling\_win\_right\_offset})$  shall be greater than or equal to  $-\text{pps\_pic\_width\_in\_luma\_samples} * 15$  and less than  $\text{pps\_pic\_width\_in\_luma\_samples}$ , and the value of  $\text{SubHeightC} * (\text{pps\_scaling\_win\_top\_offset} + \text{pps\_scaling\_win\_bottom\_offset})$  shall be greater than or equal to  $-\text{pps\_pic\_height\_in\_luma\_samples} * 15$  and less than  $\text{pps\_pic\_height\_in\_luma\_samples}$ .

The variables **CurrPicScalWinWidthL** and **CurrPicScalWinHeightL** are derived as follows:

$$\text{CurrPicScalWinWidthL} = \text{pps\_pic\_width\_in\_luma\_samples} - \text{SubWidthC} * (\text{pps\_scaling\_win\_right\_offset} + \text{pps\_scaling\_win\_left\_offset}) \quad (73)$$

$$\text{CurrPicScalWinHeightL} = \text{pps\_pic\_height\_in\_luma\_samples} - \text{SubHeightC} * (\text{pps\_scaling\_win\_bottom\_offset} + \text{pps\_scaling\_win\_top\_offset}) \quad (74)$$

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 ]
    else
        SubpicIdVal[ i ] = i

```

(75)

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 **num\_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 subclause 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 subclause 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 subclause 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 bitstream 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  $(\text{pps\_pic\_width\_in\_luma\_samples} / \text{MinCbSizeY}) - (\text{CtbSizeY} / \text{MinCbSizeY}) - 2$ .

The variable PpsRefWraparoundOffset is set equal to  $\text{pps\_pic\_width\_in\_luma\_samples} / \text{MinCbSizeY} - \text{pps\_pic\_width\_minus\_wraparound\_offset}$ .

**pps\_init\_qp\_minus26** plus 26 specifies the initial value of SliceQp<sub>Y</sub> for each slice referring to the PPS. The initial value of SliceQp<sub>Y</sub> 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  $-(26 + \text{QpBdOffset})$  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<sub>c</sub> 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<sub>c</sub> 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'<sub>Y</sub> used for deriving Qp'<sub>cb</sub> and Qp'<sub>cr</sub>, 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\_cbr\_qp\_offset\_present\_flag** equal to 1 specifies that pps\_joint\_cbr\_qp\_offset\_value and pps\_joint\_cbr\_qp\_offset\_list[i] are present in the PPS RBSP syntax structure. pps\_joint\_cbr\_qp\_offset\_present\_flag equal to 0 specifies that pps\_joint\_cbr\_qp\_offset\_value and pps\_joint\_cbr\_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\_cbr\_enabled\_flag is equal to 0, the value of pps\_joint\_cbr\_qp\_offset\_present\_flag shall be equal to 0. When not present, the value of pps\_joint\_cbr\_qp\_offset\_present\_flag is inferred to be equal to 0.

**pps\_joint\_cbr\_qp\_offset\_value** specifies the offset to the luma quantization parameter Qp'<sub>Y</sub> used for deriving Qp'<sub>cbCr</sub>. The value of pps\_joint\_cbr\_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\_cbr\_enabled\_flag is equal to 0, pps\_joint\_cbr\_qp\_offset\_value is not used in the decoding process and decoders shall ignore its value. When pps\_joint\_cbr\_qp\_offset\_present\_flag is equal to 0, pps\_joint\_cbr\_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\_cbr\_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\_cbr\_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\_cbr\_qp\_offset\_list[i]** shall be in the range of -12 to +12, inclusive. When **pps\_joint\_cbr\_qp\_offset\_present\_flag** is equal to 0, **pps\_joint\_cbr\_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  $\beta$  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  $\beta$  and  $t_c$  (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  $\beta$  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  $\beta$  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 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 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 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 document. However, some use of `pps_picture_header_extension_present_flag` equal to  $1$  could be specified in some future version of this

document, and decoders conforming to this version of this document 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 document. However, some use of `pps_slice_header_extension_present_flag` equal to 1 could be specified in some future version of this document, and decoders conforming to this version of this document 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 document. However, some use of `pps_extension_flag` equal to 1 could be specified in some future version of this document, and decoders conforming to this version of this document 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 document. Decoders conforming to this version of this document 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 subclause 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 document. Other values of `aps_params_type` are reserved for future use by ITU-T | ISO/IEC. Decoders conforming to this version of this document shall ignore APS NAL units with reserved values of `aps_params_type`.

**Table 6 — APS parameters type codes and types of APS parameters**

| <code>aps_params_type</code> | Name of <code>aps_params_type</code> | 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 document. However, some use of **aps\_extension\_data\_flag** equal to 1 could be specified in some future version of this document, and decoders conforming to this version of this document 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 document. Decoders conforming to this version of this document 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 supposed 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  $\text{sps\_log2\_max\_pic\_order\_cnt\_lsb\_minus4} + 4$  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 document 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 document.

**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  $\text{sps\_poc\_msb\_cycle\_len\_minus1} + 1$  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-component 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 ) :  
( ph\_virtual\_boundary\_pos\_x\_minus1[ i ] + 1 ) ) \* 8 (79)

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 ) :  
( ph\_virtual\_boundary\_pos\_y\_minus1[ i ] + 1 ) ) \* 8 (81)



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 bitstream 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  $\text{Min}(6, CtbLog2SizeY) - \text{MinCbLog2SizeY}$ , 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:

$$\text{MinQtLog2SizeIntraY} = \text{ph\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_luma} + \text{MinCbLog2SizeY} \quad (82)$$

**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  $2 * (CtbLog2SizeY - \text{MinCbLog2SizeY})$ , 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  $(\text{sps\_qtbtt\_dual\_tree\_intra\_flag} ? \text{Min}(6, CtbLog2SizeY) : CtbLog2SizeY) - \text{MinQtLog2SizeIntraY}$ , 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  $\text{Min}(6, CtbLog2SizeY) - \text{MinQtLog2SizeIntraY}$ , 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  $\text{Min}(6, \text{CtbLog2SizeY}) - \text{MinCbLog2SizeY}$ , 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:

$$\text{MinQtLog2SizeIntraC} = \text{ph\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_chroma} + \text{MinCbLog2SizeY} \quad (83)$$

**`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  $2 * (\text{CtbLog2SizeY} - \text{MinCbLog2SizeY})$ , 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  $\text{Min}(6, \text{CtbLog2SizeY}) - \text{MinQtLog2SizeIntraC}$ , 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  $\text{Min}(6, \text{CtbLog2SizeY}) - \text{MinQtLog2SizeIntraC}$ , 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  $2 * (\text{CtbLog2SizeY} - \text{MinQtLog2SizeIntraY} + \text{ph\_max\_mtt\_hierarchy\_depth\_intra\_slice\_luma})$ , 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  $2 * (\text{CtbLog2SizeY} - \text{MinQtLog2SizeIntraY} + \text{ph\_max\_mtt\_hierarchy\_depth\_intra\_slice\_luma})$ , 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  $\text{Min}(6, \text{CtbLog2SizeY}) - \text{MinCbLog2SizeY}$ , 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:

$$\text{MinQtLog2SizeInterY} = \text{ph\_log2\_diff\_min\_qt\_min\_cb\_inter\_slice} + \text{MinCbLog2SizeY} \quad (84)$$

**`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  $2 * (\text{CtbLog2SizeY} - \text{MinCbLog2SizeY})$ , 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  $\text{CtbLog2SizeY} - \text{MinQtLog2SizeInterY}$ , 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  $\text{Min}(6, \text{CtbLog2SizeY}) - \text{MinQtLog2SizeInterY}$ , 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  $2 * (\text{CtbLog2SizeY} - \text{MinQtLog2SizeInterY} + \text{ph\_max\_mtt\_hierarchy\_depth\_inter\_slice})$ , 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  $2 * (\text{CtbLog2SizeY} - \text{MinQtLog2SizeInterY} + \text{ph\_max\_mtt\_hierarchy\_depth\_inter\_slice})$ , 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:

$$\text{if}(\text{sps\_affine\_enabled\_flag}) \\ \text{MaxNumSubblockMergeCand} = 5 - \text{sps\_five\_minus\_max\_num\_subblock\_merge\_cand} \quad (85)$$

```

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 ][ RplIdx[ 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 ][ RplIdx[ 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 ][ RplIdx[ 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\_mvdl1\_zero\_flag** equal to 1 specifies that the **mvdl\_coding( x0, y0, 1, cpldx )** syntax structure is not parsed and **MvdlL1[ x0 ][ y0 ][ compldx ]** and **MvdlCpL1[ x0 ][ y0 ][ cpldx ][ compldx ]** are set equal to 0 for **compldx = 0..1** and **cpldx = 0..2**. **ph\_mvdl1\_zero\_flag** equal to 0 specifies that the **mvdl\_coding( x0, y0, 1, cpldx )** syntax structure is parsed. When not present, the value of **ph\_mvdl1\_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  $Q_{pY}$  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  $Q_{pY}$  quantization parameter for all slices of the picture,  $\text{SliceQ}_{pY}$ , is derived as follows:

$$\text{SliceQ}_{pY} = 26 + \text{pps\_init\_qp\_minus26} + \text{ph\_qp\_delta} \quad (86)$$

The value of  $\text{SliceQ}_{pY}$  shall be in the range of  $-\text{QpBdOffset}$  to  $+63$ , inclusive.

**ph\_joint\_cbr\_sign\_flag** specifies whether, in transform units with **tu\_joint\_cbr\_residual\_flag**[x0][y0] equal to 1, the collocated residual samples of both chroma components have inverted signs. When **tu\_joint\_cbr\_residual\_flag**[x0][y0] equal to 1 for a transform unit, **ph\_joint\_cbr\_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\_cbr\_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  $\beta$  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  $\beta$  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  $\beta$  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 document, some use of `ph_extension_length` could be specified in some future version of this document, and decoders conforming to this version of this document 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 document. Decoders conforming to this version of this document 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 subclause 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 document. Other values of **aud\_pic\_type** are reserved for future use by ITU-T/ISO/IEC. Decoders conforming to this version of this document shall ignore reserved values of **aud\_pic\_type**.

**Table 7 — Interpretation of **aud\_pic\_type****

| <b>aud_pic_type</b> | <b>sh_slice_type values that could be present in the AU</b> |
|---------------------|-------------------------------------------------------------|
| 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 subclause 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:

$$\text{RawMinCuBits} = \text{MinCbSizeY} * \text{MinCbSizeY} * (\text{BitDepth} + 2 * \text{BitDepth} / (\text{SubWidthC} * \text{SubHeightC})) \quad (87)$$

Let the variable `vclByteScaleFactor` be derived to be equal to  $(32 + 4 * \text{general\_tier\_flag}) \div 3$ .

The value of `BinCountsInPicNalUnits` shall be less than or equal to  $\text{vclByteScaleFactor} * \text{NumBytesInPicVclNalUnits} + (\text{RawMinCuBits} * \text{PicSizeInMinCbsY}) \div 32$ .

NOTE The constraint on the maximum number of bins resulting from decoding the contents of the coded slice NAL units could be met by inserting a number of `rbasp_cabac_zero_word` syntax elements to increase the value of `NumBytesInPicVclNalUnits`. Each `rbasp_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 `rbasp_cabac_zero_word`).

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 subclause 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  $((\text{sps\_subpic\_width\_minus1}[\text{subpicIdxA}] + 1) * \text{CtbSizeY} / \text{MinCbSizeY} * (\text{sps\_subpic\_height\_minus1}[\text{subpicIdxA}] + 1) * \text{CtbSizeY} / \text{MinCbSizeY})$ .

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  $\text{vclByteScaleFactor} * \text{NumBytesInSubpicVclNalUnits} + (\text{RawMinCuBits} * \text{subpicSizeInMinCbsY}) \div 32$ .

#### 7.4.3.16 RBSP trailing bits semantics

**`rbasp_stop_one_bit`** shall be equal to 1.

**`rbasp_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 adaptive 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  $\text{Ceil}(\text{Log2}(\text{alf\_luma\_num\_filters\_signalled\_minus1} + 1))$  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**[ **sflDx** ][ **j** ] specifies the absolute value of the *j*-th coefficient of the signalled luma filter indicated by **sflDx**. When **alf\_luma\_coeff\_abs**[ **sflDx** ][ **j** ] is not present, it is inferred to be equal 0. The value of **alf\_luma\_coeff\_abs**[ **sflDx** ][ **j** ] shall be in the range of 0 to 128, inclusive.

**alf\_luma\_coeff\_sign**[ **sflDx** ][ **j** ] specifies the sign of the *j*-th luma coefficient of the filter indicated by **sflDx** as follows:

- If **alf\_luma\_coeff\_sign**[ **sflDx** ][ **j** ] is equal to 0, the corresponding luma filter coefficient has a positive value.
- Otherwise (**alf\_luma\_coeff\_sign**[ **sflDx** ][ **j** ] is equal to 1), the corresponding luma filter coefficient has a negative value.

When **alf\_luma\_coeff\_sign**[ **sflDx** ][ **j** ] is not present, it is inferred to be equal to 0.

The variable **filtCoeff**[ **sflDx** ][ **j** ] with **sflDx** = 0..**alf\_luma\_num\_filters\_signalled\_minus1**, **j** = 0..11 is initialized as follows:

$$\text{filtCoeff}[ \text{sflDx} ][ j ] = \text{alf\_luma\_coeff\_abs}[ \text{sflDx} ][ j ] * ( 1 - 2 * \text{alf\_luma\_coeff\_sign}[ \text{sflDx} ][ j ] ) \quad (88)$$

The luma filter coefficients **AlfCoeff<sub>L</sub>**[ **aps\_adaptation\_parameter\_set\_id** ] with elements **AlfCoeff<sub>L</sub>**[ **aps\_adaptation\_parameter\_set\_id** ][ **filtIdx** ][ **j** ], with **filtIdx** = 0..**NumAlfFilters** – 1 and **j** = 0..11 are derived as follows:

$$\text{AlfCoeff}_L[ \text{aps\_adaptation\_parameter\_set\_id} ][ \text{filtIdx} ][ j ] = \text{filtCoeff}[ \text{alf\_luma\_coeff\_delta\_idx}[ \text{filtIdx} ] ][ j ] \quad (89)$$

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:

$$\text{AlfFixFiltCoeff} = \quad (90)$$

```
{
{ 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  $\text{AlfCoeff}_L[\text{aps\_adaptation\_parameter\_set\_id}][\text{filtIdx}][j]$  with  $\text{filtIdx} = 0.. \text{NumAlfFilters} - 1$ ,  $j = 0..11$  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  $\text{sfIdx}$ . When **alf\_luma\_clip\_idx[sfIdx][j]** is not present, it is inferred to be equal to 0.

The luma filter clipping values  $\text{AlfClip}_L[\text{aps\_adaptation\_parameter\_set\_id}]$  with elements  $\text{AlfClip}_L[\text{aps\_adaptation\_parameter\_set\_id}][\text{filtIdx}][j]$ , with  $\text{filtIdx} = 0..\text{NumAlfFilters} - 1$  and  $j = 0..11$  are derived as specified in Table 8 depending on BitDepth and clipIdx set equal to  $\text{alf\_luma\_clip\_idx}[\text{alf\_luma\_coeff\_delta\_idx}[\text{filtIdx}][j]]$ .

**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** $[\text{altIdx}][j]$  specifies the absolute value of the  $j$ -th chroma filter coefficient for the alternative chroma filter with index  $\text{altIdx}$ . When **alf\_chroma\_coeff\_abs** $[\text{altIdx}][j]$  is not present, it is inferred to be equal 0. The value of **alf\_chroma\_coeff\_abs** $[\text{sIdx}][j]$  shall be in the range of 0 to 128, inclusive.

**alf\_chroma\_coeff\_sign** $[\text{altIdx}][j]$  specifies the sign of the  $j$ -th chroma filter coefficient for the alternative chroma filter with index  $\text{altIdx}$  as follows:

- If **alf\_chroma\_coeff\_sign** $[\text{altIdx}][j]$  is equal to 0, the corresponding chroma filter coefficient has a positive value.
- Otherwise (**alf\_chroma\_coeff\_sign** $[\text{altIdx}][j]$  is equal to 1), the corresponding chroma filter coefficient has a negative value.

When **alf\_chroma\_coeff\_sign** $[\text{altIdx}][j]$  is not present, it is inferred to be equal to 0.

The chroma filter coefficients  $\text{AlfCoeff}_c[\text{aps\_adaptation\_parameter\_set\_id}][\text{altIdx}]$  with elements  $\text{AlfCoeff}_c[\text{aps\_adaptation\_parameter\_set\_id}][\text{altIdx}][j]$ , with  $\text{altIdx} = 0..\text{alf\_chroma\_num\_alt\_filters\_minus1}$ ,  $j = 0..5$  are derived as follows:

$$\text{AlfCoeff}_c[\text{aps\_adaptation\_parameter\_set\_id}][\text{altIdx}][j] = \text{alf\_chroma\_coeff\_abs}[\text{altIdx}][j] * (92 - 2 * \text{alf\_chroma\_coeff\_sign}[\text{altIdx}][j])$$

It is a requirement of bitstream conformance that the values of  $\text{AlfCoeff}_c[\text{aps\_adaptation\_parameter\_set\_id}][\text{altIdx}][j]$  with  $\text{altIdx} = 0..\text{alf\_chroma\_num\_alt\_filters\_minus1}$ ,  $j = 0..5$  shall be in the range of  $-2^7$  to  $2^7 - 1$ , inclusive.

**alf\_chroma\_clip\_idx** $[\text{altIdx}][j]$  specifies the clipping index of the clipping value to use before multiplying by the  $j$ -th coefficient of the alternative chroma filter with index  $\text{altIdx}$ . When **alf\_chroma\_clip\_idx** $[\text{altIdx}][j]$  is not present, it is inferred to be equal to 0.

The chroma filter clipping values  $\text{AlfClip}_c[\text{aps\_adaptation\_parameter\_set\_id}][\text{altIdx}]$  with elements  $\text{AlfClip}_c[\text{aps\_adaptation\_parameter\_set\_id}][\text{altIdx}][j]$ , with  $\text{altIdx} = 0..\text{alf\_chroma\_num\_alt\_filters\_minus1}$ ,  $j = 0..5$  are derived as specified in Table 8 depending on BitDepth and clipIdx set equal to  $\text{alf\_chroma\_clip\_idx}[\text{altIdx}][j]$ .

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 **CcAlfApsCoeffCb[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, **CcAlfApsCoeffCb[aps\_adaptation\_parameter\_set\_id][k][j]** is set equal to 0.
- Otherwise, **CcAlfApsCoeffCb[aps\_adaptation\_parameter\_set\_id][k][j]** is set equal to  $(1 - 2 * \text{alf\_cc\_cb\_coeff\_sign}[k][j]) * 2^{\text{alf\_cc\_cb\_mapped\_coeff\_abs}[k][j]} - 1$ .

**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 ( $\text{alf\_cc\_cr\_sign}[k][j]$  is equal to 1), the corresponding cross-component filter coefficient has a negative value.

When  $\text{alf\_cc\_cr\_coeff\_sign}[k][j]$  is not present, it is inferred to be equal to 0.

The signalled  $k$ -th cross-component filter coefficients for the Cr colour component  $\text{CcAlfApsCoeff}_{\text{Cr}}[\text{aps\_adaptation\_parameter\_set\_id}][k][j]$ , with  $j = 0..6$  are derived as follows:

- If  $\text{alf\_cc\_cr\_mapped\_coeff\_abs}[k][j]$  is equal to 0,  $\text{CcAlfApsCoeff}_{\text{Cr}}[\text{aps\_adaptation\_parameter\_set\_id}][k][j]$  is set equal to 0.
- Otherwise,  $\text{CcAlfApsCoeff}_{\text{Cr}}[\text{aps\_adaptation\_parameter\_set\_id}][k][j]$  is set equal to  $(1 - 2 * \text{alf\_cc\_cr\_coeff\_sign}[k][j]) * 2^{\text{alf\_cc\_cr\_mapped\_coeff\_abs}[k][j] - 1}$ .

#### 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  $15 - \text{lmcs\_delta\_max\_bin\_idx}$ . 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  $\text{lmcs\_delta\_abs\_cw}[i]$ . 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  $i$ th 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:

$$\text{OrgCW} = (1 \ll \text{BitDepth}) / 16 \quad (93)$$

The variable **lmcsDeltaCW** $[i]$ , with  $i = \text{lmcs\_min\_bin\_idx}.. \text{LmcsMaxBinIdx}$ , is derived as follows:

$$\text{lmcsDeltaCW}[i] = (1 - 2 * \text{lmcs\_delta\_sign\_cw\_flag}[i]) * \text{lmcs\_delta\_abs\_cw}[i] \quad (94)$$

The variable **lmcsCW** $[i]$  is derived as follows:

- For  $i = 0.. \text{lmcs\_min\_bin\_idx} - 1$ , **lmcsCW** $[i]$  is set equal 0.
- For  $i = \text{lmcs\_min\_bin\_idx}.. \text{LmcsMaxBinIdx}$ , the following applies:

$$\text{lmcsCW}[i] = \text{OrgCW} + \text{lmcsDeltaCW}[i] \quad (95)$$

The value of **lmcsCW** $[i]$  shall be in the range of  $\text{OrgCW} \gg 3$  to  $(\text{OrgCW} \ll 3) - 1$ , inclusive.

- For  $i = \text{LmcsMaxBinIdx} + 1..15$ , **lmcsCW** $[i]$  is set equal 0.

It is a requirement of bitstream conformance that the following condition is true:

$$\sum_{i=0}^{15} \text{lmcsCW}[i] \leq (1 \ll \text{BitDepth}) - 1 \quad (96)$$

The variable **InputPivot** $[i]$ , with  $i = 0..15$ , is derived as follows:

$$\text{InputPivot}[i] = i * \text{OrgCW} \quad (97)$$

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 )
    InvScaleCoeff[ i ] = 0
else
    InvScaleCoeff[ i ] = OrgCW * ( 1 << 11 ) / lmcscw[ i ]

```

(98)

It is a requirement of bitstream conformance that, for  $i = lmc\_min\_bin\_idx..LmcsMaxBinIdx$ , 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  $(LmcsPivot[i + 1] >> (BitDepth - 5))$ .

**lmc\_delta\_abs\_crs** specifies the absolute codeword value of the variable  $lmcDeltaCrs$ . When not present,  $lmc\_delta\_abs\_crs$  is inferred to be equal to 0.

**lmc\_delta\_sign\_crs\_flag** specifies the sign of the variable  $lmcDeltaCrs$ . When not present,  $lmc\_delta\_sign\_crs\_flag$  is inferred to be equal to 0.

The variable  $lmcDeltaCrs$  is derived as follows:

$$lmcDeltaCrs = (1 - 2 * lmc\_delta\_sign\_crs\_flag) * lmc\_delta\_abs\_crs \quad (99)$$

It is a requirement of bitstream conformance that, when  $lmcCW[i]$  is not equal to 0,  $(lmcCW[i] + lmcDeltaCrs)$  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( lmcCW[ i ] == 0 )
    ChromaScaleCoeff[ i ] = ( 1 << 11 )
else
    ChromaScaleCoeff[ i ] = OrgCW * ( 1 << 11 ) / ( lmcCW[ i ] + lmcDeltaCrs )

```

(100)

#### 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  $scaling\_list\_pred\_mode\_flag$  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:

$$maxIdDelta = (id < 2) ? id : ((id < 8) ? (id - 2) : (id - 8)) \quad (101)$$

The variables  $refId$  and  $matrixSize$  are derived as follows:

$$refId = id - scaling\_list\_pred\_id\_delta[id] \quad (102)$$

$$\text{matrixSize} = (\text{id} < 2) ? 2 : ((\text{id} < 8) ? 4 : 8) \quad (103)$$

The  $(\text{matrixSize}) \times (\text{matrixSize})$  array  $\text{scalingMatrixPred}[x][y]$  with  $x = 0..\text{matrixSize} - 1$ ,  $y = 0..\text{matrixSize} - 1$  and the variable  $\text{scalingMatrixDcPred}$  are derived as follows:

- When both  $\text{scaling\_list\_copy\_mode\_flag}[\text{id}]$  and  $\text{scaling\_list\_pred\_mode\_flag}[\text{id}]$  are equal to 0, all elements of  $\text{scalingMatrixPred}$  are set equal to 8, and the value of  $\text{scalingMatrixDcPred}$  is set equal to 8.
- Otherwise, if  $\text{scaling\_list\_pred\_id\_delta}[\text{id}]$  is equal to 0, all elements of  $\text{scalingMatrixPred}$  are set equal to 16, and  $\text{scalingMatrixDcPred}$  is set equal to 16.
- Otherwise (either  $\text{scaling\_list\_copy\_mode\_flag}[\text{id}]$  or  $\text{scaling\_list\_pred\_mode\_flag}[\text{id}]$  is equal to 1 and  $\text{scaling\_list\_pred\_id\_delta}[\text{id}]$  is greater than 0),  $\text{scalingMatrixPred}$  is set equal to  $\text{ScalingMatrixRec}[\text{refId}]$ , and the following applies for  $\text{scalingMatrixDcPred}$ :
  - If  $\text{refId}$  is greater than 13,  $\text{scalingMatrixDcPred}$  is set equal to  $\text{ScalingMatrixDcRec}[\text{refId} - 14]$ .
  - Otherwise ( $\text{refId}$  is less than or equal to 13),  $\text{scalingMatrixDcPred}$  is set equal to  $\text{scalingMatrixPred}[0][0]$ .

$\text{scaling\_list\_dc\_coef}[\text{id} - 14]$  is used to derive the value of the variable  $\text{ScalingMatrixDcRec}[\text{id} - 14]$  when  $\text{id}$  is greater than 13 as follows:

$$\text{ScalingMatrixDcRec}[\text{id} - 14] = (\text{scalingMatrixDcPred} + \text{scaling\_list\_dc\_coef}[\text{id} - 14]) \& 255 \quad (104)$$

When not present, the value of  $\text{scaling\_list\_dc\_coef}[\text{id} - 14]$  is inferred to be equal to 0. The value of  $\text{scaling\_list\_dc\_coef}[\text{id} - 14]$  shall be in the range of  $-128$  to  $127$ , inclusive. The value of  $\text{ScalingMatrixDcRec}[\text{id} - 14]$  shall be greater than 0.

$\text{scaling\_list\_delta\_coef}[\text{id}][i]$  specifies the difference between the current matrix coefficient  $\text{ScalingList}[\text{id}][i]$  and the previous matrix coefficient  $\text{ScalingList}[\text{id}][i - 1]$ , when  $\text{scaling\_list\_copy\_mode\_flag}[\text{id}]$  is equal to 0. The value of  $\text{scaling\_list\_delta\_coef}[\text{id}][i]$  shall be in the range of  $-128$  to  $127$ , inclusive. When  $\text{scaling\_list\_copy\_mode\_flag}[\text{id}]$  is equal to 1, all elements of  $\text{ScalingList}[\text{id}]$  are set equal to 0.

The  $(\text{matrixSize}) \times (\text{matrixSize})$  array  $\text{ScalingMatrixRec}[\text{id}]$  is derived as follows:

$$\begin{aligned} \text{ScalingMatrixRec}[\text{id}][x][y] &= (\text{scalingMatrixPred}[x][y] + \text{ScalingList}[\text{id}][k]) \& 255 \quad (105) \\ \text{with } k &= 0..(\text{matrixSize} * \text{matrixSize} - 1), \\ x &= \text{DiagScanOrder}[\text{Log2}(\text{matrixSize})][\text{Log2}(\text{matrixSize})][k][0], \text{ and} \\ y &= \text{DiagScanOrder}[\text{Log2}(\text{matrixSize})][\text{Log2}(\text{matrixSize})][k][1] \end{aligned}$$

The value of  $\text{ScalingMatrixRec}[\text{id}][x][y]$  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 document. However, decoders conforming to this version of this document 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  $8 * \text{payloadSize} - \text{nEarlierBits} - \text{nPayloadZeroBits} - 1$ , 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  $8 * \text{payloadSize} - \text{nPayloadZeroBits} - 1$ ; otherwise, **PayloadBits** is equal to  $8 * \text{payloadSize}$ .

**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:

$$\text{Log2TransformRange} = \text{sps\_extended\_precision\_flag} ? \text{Max}(15, \text{Min}(20, \text{BitDepth} + 6)) : 15 \quad (106)$$

**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 document).

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 document 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 document.

#### 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\_qtbt\_dual\_tree\_intra\_constraint\_flag** equal to 1 specifies that `sps_qtbt_dual_tree_intra_flag` for all pictures in `OlsInScope` shall be equal to 0. `gci_no_qtbt_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\_cbr\_constraint\_flag** equal to 1 specifies that `sps_joint_cbr_enabled_flag` for all pictures in `OlsInScope` shall be equal to 0. `gci_no_joint_cbr_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 document. Decoders conforming to this version of this document 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 subclause 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:

$$\text{MaxLatencyPictures}[i] = \text{dpb\_max\_num\_reorder\_pics}[i] + \text{dpb\_max\_latency\_increase\_plus1}[i] - 1 \quad (107)$$

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 document.

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 document.
- 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 document.

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 document.
- 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 document.
- 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 subclause 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 subclause 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:

$$\text{DpbOutputElementalInterval}[n] = \text{DpbOutputInterval}[n] \square (\text{pt\_display\_elemental\_periods\_minus1} + 1) \quad (108)$$

where  $DpbOutputInterval[n]$  is specified in Formula (1603).

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 Formula (1588) (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 Formula (1603):

- 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:

$$BitRate[i][j] = (bit\_rate\_value\_minus1[i][j] + 1) * 2^{(6 + bit\_rate\_scale)} \quad (109)$$

- 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 applies:

- The CPB size in bits is given by:



$$\text{CpbSize}[i][j] = (\text{cpb\_size\_value\_minus1}[i][j] + 1) * 2^{(4 + \text{cpb\_size\_scale})} \quad (110)$$

- When the  $\text{cpb\_size\_value\_minus1}[i][j]$  syntax element is not present, it is inferred as follows:
  - If  $\text{timingHrdParamsPresentFlag}$  is equal to 1,  $\text{cpb\_size\_value\_minus1}[i][j]$  is inferred to be equal to  $\text{cpb\_size\_value\_minus1}[\text{maxSubLayersMinus1}][j]$ .
  - Otherwise ( $\text{timingHrdParamsPresentFlag}$  is equal to 0), the value of  $\text{CpbSize}[i][j]$  is inferred to be equal to  $\text{BrVclFactor} * \text{MaxCPB}$  for VCL HRD parameters and to be equal to  $\text{BrNalFactor} * \text{MaxCPB}$  for NAL HRD parameters, where  $\text{MaxCPB}$ ,  $\text{BrVclFactor}$  and  $\text{BrNalFactor}$  are specified in Annex A.

**cpb\_size\_du\_value\_minus1** $[i][j]$  is used together with  $\text{cpb\_size\_du\_scale}$  to specify the  $i$ -th CPB size with  $\text{Htid}$  equal to  $i$  when the CPB operates at the DU level.  $\text{cpb\_size\_du\_value\_minus1}[i][j]$  shall be in the range of 0 to  $2^{32} - 2$ , inclusive. For any  $j$  greater than 0 and any particular value of  $i$ ,  $\text{cpb\_size\_du\_value\_minus1}[i][j]$  shall be less than or equal to  $\text{cpb\_size\_du\_value\_minus1}[i][j - 1]$ .

When  $\text{DecodingUnitHrdFlag}$  is equal to 1, the following applies:

- The CPB size in bits is given by:

$$\text{CpbSize}[i][j] = (\text{cpb\_size\_du\_value\_minus1}[i][j] + 1) * 2^{(4 + \text{cpb\_size\_du\_scale})} \quad (111)$$

- When the  $\text{cpb\_size\_du\_value\_minus1}[i][j]$  syntax element is not present, it is inferred as follows:
  - If  $\text{timingHrdParamsPresentFlag}$  is equal to 1,  $\text{cpb\_size\_du\_value\_minus1}[i][j]$  is inferred to be equal to  $\text{cpb\_size\_du\_value\_minus1}[\text{maxSubLayersMinus1}][j]$ .
  - Otherwise ( $\text{timingHrdParamsPresentFlag}$  is equal to 0), the value of  $\text{CpbSize}[i][j]$  is inferred to be equal to  $\text{CpbVclFactor} * \text{MaxCPB}$  for VCL HRD parameters and to be equal to  $\text{CpbNalFactor} * \text{MaxCPB}$  for NAL HRD parameters, where  $\text{MaxCPB}$ ,  $\text{CpbVclFactor}$  and  $\text{CpbNalFactor}$  are specified in Annex A.

**bit\_rate\_du\_value\_minus1** $[i][j]$  (together with  $\text{bit\_rate\_scale}$ ) specifies the maximum input bit rate for the  $j$ -th CPB with  $\text{Htid}$  equal to  $i$  when the CPB operates at the DU level.  $\text{bit\_rate\_du\_value\_minus1}[i][j]$  shall be in the range of 0 to  $2^{32} - 2$ , inclusive. For any  $j$  greater than 0 and any particular value of  $i$ ,  $\text{bit\_rate\_du\_value\_minus1}[i][j]$  shall be greater than  $\text{bit\_rate\_du\_value\_minus1}[i][j - 1]$ .

When  $\text{DecodingUnitHrdFlag}$  is equal to 1, the following applies:

- The bit rate in bits per second is given by:

$$\text{BitRate}[i][j] = (\text{bit\_rate\_du\_value\_minus1}[i][j] + 1) * 2^{(6 + \text{bit\_rate\_scale})} \quad (112)$$

- When the  $\text{bit\_rate\_du\_value\_minus1}[i][j]$  syntax element is not present, it is inferred as follows:
  - If  $\text{timingHrdParamsPresentFlag}$  is equal to 1,  $\text{bit\_rate\_du\_value\_minus1}[i][j]$  is inferred to be equal to  $\text{bit\_rate\_du\_value\_minus1}[\text{maxSubLayersMinus1}][j]$ .
  - Otherwise ( $\text{timingHrdParamsPresentFlag}$  is equal to 0), the value of  $\text{BitRate}[i][j]$  is inferred to be equal to  $\text{BrVclFactor} * \text{MaxBR}$  for VCL HRD parameters and to be equal to  $\text{BrNalFactor} * \text{MaxBR}$  for NAL HRD parameters, where  $\text{MaxBR}$ ,  $\text{BrVclFactor}$  and  $\text{BrNalFactor}$  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.  $\text{cbr\_flag}[i][j]$  equal to 1 specifies that the HSS operates in a constant bit rate (CBR) mode.

When not present, the value of  $\text{cbr\_flag}[i][j]$  it is inferred as follows:

- When the  $\text{cbr\_flag}[i][j]$  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 `CuQpOffsetCb`, `CuQpOffsetCr`, and `CuQpOffsetCbCr`, 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 `SubpicIdxVal[ 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  $\text{Ceil}(\text{Log}_2(\text{NumTilesInPic}))$  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  $\text{Ceil}(\text{Log}_2(\text{NumSlicesInSubpic}[\text{CurrSubpicIdx}]))$  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 document 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 document.

`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 ]
} 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
            }
        }
    }
}
    
```

(113)

```

        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
    SubpicBotBoundaryPos = Min( pps_pic_height_in_luma_samples - 1,
        ( sps_subpic_ctu_top_left_y[ CurrSubpicIdx ] +
          sps_subpic_height_minus1[ CurrSubpicIdx ] + 1 ) * CtbSizeY - 1 )
}

```

(114)

**sh\_slice\_type** specifies the coding type of the slice according to Table 9.

**Table 9 — Name association to `sh_slice_type`**

| <code>sh_slice_type</code> | Name of <code>sh_slice_type</code> |
|----------------------------|------------------------------------|
| 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 value of `sh_no_output_of_prior_pics_flag` in the SHs is also referred to as the value `sh_no_output_of_prior_pics_flag` of the AU.

The variables  $\text{MinQtLog2SizeY}$ ,  $\text{MinQtLog2SizeC}$ ,  $\text{MinQtSizeY}$ ,  $\text{MinQtSizeC}$ ,  $\text{MaxBtSizeY}$ ,  $\text{MaxBtSizeC}$ ,  $\text{MinBtSizeY}$ ,  $\text{MaxTtSizeY}$ ,  $\text{MaxTtSizeC}$ ,  $\text{MinTtSizeY}$ ,  $\text{MaxMttDepthY}$  and  $\text{MaxMttDepthC}$  are derived as follows:

— If  $\text{sh\_slice\_type}$  equal to 2 (I), the following applies:

$$\text{MinQtLog2SizeY} = \text{MinCbLog2SizeY} + \text{ph\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_luma} \quad (115)$$

$$\text{MinQtLog2SizeC} = \text{MinCbLog2SizeY} + \text{ph\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_chroma} \quad (116)$$

$$\text{MaxBtSizeY} = 1 \ll (\text{MinQtLog2SizeY} + \text{ph\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_luma}) \quad (117)$$

$$\text{MaxBtSizeC} = 1 \ll (\text{MinQtLog2SizeC} + \text{ph\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_chroma}) \quad (118)$$

$$\text{MaxTtSizeY} = 1 \ll (\text{MinQtLog2SizeY} + \text{ph\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_luma}) \quad (119)$$

$$\text{MaxTtSizeC} = 1 \ll (\text{MinQtLog2SizeC} + \text{ph\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_chroma}) \quad (120)$$

$$\text{MaxMttDepthY} = \text{ph\_max\_mtt\_hierarchy\_depth\_intra\_slice\_luma} \quad (121)$$

$$\text{MaxMttDepthC} = \text{ph\_max\_mtt\_hierarchy\_depth\_intra\_slice\_chroma} \quad (122)$$

$$\text{CuQpDeltaSubdiv} = \text{ph\_cu\_qp\_delta\_subdiv\_intra\_slice} \quad (123)$$

$$\text{CuChromaQpOffsetSubdiv} = \text{ph\_cu\_chroma\_qp\_offset\_subdiv\_intra\_slice} \quad (124)$$

— Otherwise ( $\text{sh\_slice\_type}$  equal to 0 (B) or 1 (P)), the following applies:

$$\text{MinQtLog2SizeY} = \text{MinCbLog2SizeY} + \text{ph\_log2\_diff\_min\_qt\_min\_cb\_inter\_slice} \quad (125)$$

$$\text{MinQtLog2SizeC} = \text{MinCbLog2SizeY} + \text{ph\_log2\_diff\_min\_qt\_min\_cb\_inter\_slice} \quad (126)$$

$$\text{MaxBtSizeY} = 1 \ll (\text{MinQtLog2SizeY} + \text{ph\_log2\_diff\_max\_bt\_min\_qt\_inter\_slice}) \quad (127)$$

$$\text{MaxBtSizeC} = 1 \ll (\text{MinQtLog2SizeC} + \text{ph\_log2\_diff\_max\_bt\_min\_qt\_inter\_slice}) \quad (128)$$

$$\text{MaxTtSizeY} = 1 \ll (\text{MinQtLog2SizeY} + \text{ph\_log2\_diff\_max\_tt\_min\_qt\_inter\_slice}) \quad (129)$$

$$\text{MaxTtSizeC} = 1 \ll (\text{MinQtLog2SizeC} + \text{ph\_log2\_diff\_max\_tt\_min\_qt\_inter\_slice}) \quad (130)$$

$$\text{MaxMttDepthY} = \text{ph\_max\_mtt\_hierarchy\_depth\_inter\_slice} \quad (131)$$

$$\text{MaxMttDepthC} = \text{ph\_max\_mtt\_hierarchy\_depth\_inter\_slice} \quad (132)$$

$$\text{CuQpDeltaSubdiv} = \text{ph\_cu\_qp\_delta\_subdiv\_inter\_slice} \quad (133)$$

$$\text{CuChromaQpOffsetSubdiv} = \text{ph\_cu\_chroma\_qp\_offset\_subdiv\_inter\_slice} \quad (134)$$

— The following applies:

$$\text{MinQtSizeY} = 1 \ll \text{MinQtLog2SizeY} \quad (135)$$

$$\text{MinQtSizeC} = 1 \ll \text{MinQtLog2SizeC} \quad (136)$$

$$\text{MinBtSizeY} = 1 \ll \text{MinCbLog2SizeY} \quad (137)$$

$$\text{MinTtSizeY} = 1 \ll \text{MinCbLog2SizeY} \quad (138)$$

**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][RplIdx[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][RplIdx[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 Formula (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
        else {
            if( num_ref_entries[ i ][ RplIdx[ 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 ][ RplIdx[ i ] ]
        }
    } else /* sh_slice_type == I || ( sh_slice_type == P && i == 1 ) */
        NumRefIdxActive[ i ] = 0
}

```

(139)

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\_cbr\_qp\_offset** specifies a difference to be added to the value of  $pps\_joint\_cbr\_qp\_offset\_value$  when determining the value of the  $Qp'_{CbCr}$ . The value of  $sh\_joint\_cbr\_qp\_offset$  shall be in the range of  $-12$  to  $+12$ , inclusive. When  $sh\_joint\_cbr\_qp\_offset$  is not present, it is inferred to be equal to 0. The value of  $pps\_joint\_cbr\_qp\_offset\_value + sh\_joint\_cbr\_qp\_offset$  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  $\beta$  and  $t_C$  (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  $\beta$  and  $t_C$  (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  $\beta$  and  $t_C$  (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  $((\text{Log2ZoTbWidth} \ll 1) - 1, (\text{Log2ZoTbHeight} \ll 1) - 1)$  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 document, some use of **sh\_slice\_header\_extension\_length** could be specified in some future version of this document, and decoders conforming to this version of this document 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 document. Decoders conforming to this version of this document 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 document.

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
        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++
    }

```

(141)

**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  $\text{sh\_entry\_point\_offset\_minus1}[0]$ , inclusive, of the coded slice data, subset  $k$ , with  $k$  in the range of 1 to  $\text{NumEntryPoints} - 1$ , inclusive, consists of bytes  $\text{firstByte}[k]$  to  $\text{lastByte}[k]$ , inclusive, of the coded slice data with  $\text{firstByte}[k]$  and  $\text{lastByte}[k]$  derived as follows:

$$\text{firstByte}[k] = \sum_{n=1}^k (\text{sh\_entry\_point\_offset\_minus1}[n - 1] + 1) \quad (142)$$

$$\text{lastByte}[k] = \text{firstByte}[k] + \text{sh\_entry\_point\_offset\_minus1}[k] \quad (143)$$

The last subset (with subset index equal to  $\text{NumEntryPoints}$ ) consists of the remaining bytes of the coded slice data.

When  $\text{sps\_entropy\_coding\_sync\_enabled\_flag}$  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  $\text{NumEntryPoints} + 1$ ) shall be equal to the number of tiles in the slice.

When  $\text{sps\_entropy\_coding\_sync\_enabled\_flag}$  is equal to 0 and the slice contains a subset of CTU rows from a single tile, the value of  $\text{NumEntryPoints}$  shall be equal to 0. The subset shall consist of all coded bits of all CTUs in the slice.

When  $\text{sps\_entropy\_coding\_sync\_enabled\_flag}$  is equal to 1, each subset  $k$  with  $k$  in the range of 0 to  $\text{NumEntryPoints}$ , 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  $\text{NumEntryPoints} + 1$ ) 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  $\text{pps\_wp\_info\_in\_ph\_flag}$  is equal to 1. The value of **num\_l0\_weights** shall be in the range of 0 to  $\text{Min}(15, \text{num\_ref\_entries}[0][\text{RplIdx}[0]])$ , inclusive.

If  $\text{pps\_wp\_info\_in\_ph\_flag}$  is equal to 1, the variable **NumWeightsL0** is set equal to **num\_l0\_weights**. Otherwise ( $\text{pps\_wp\_info\_in\_ph\_flag}$  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  $(1 \ll \text{luma\_log2\_weight\_denom}) + \text{delta\_luma\_weight\_l0}[i]$ . When **luma\_weight\_l0\_flag**[ $i$ ] is equal to 1, the value of



$\text{delta\_luma\_weight\_l0}[i]$  shall be in the range of  $-128$  to  $127$ , inclusive. When  $\text{luma\_weight\_l0\_flag}[i]$  is equal to 0,  $\text{LumaWeightL0}[i]$  is inferred to be equal to  $2^{\text{luma\_log2\_weight\_denom}}$ .

**$\text{luma\_offset\_l0}[i]$**  is the additive offset applied to the luma prediction value for list 0 prediction using  $\text{RefPicList}[0][i]$ . The value of  $\text{luma\_offset\_l0}[i]$  shall be in the range of  $-128$  to  $127$ , inclusive. When  $\text{luma\_weight\_l0\_flag}[i]$  is equal to 0,  $\text{luma\_offset\_l0}[i]$  is inferred to be equal to 0.

**$\text{delta\_chroma\_weight\_l0}[i][j]$**  is the difference of the weighting factor applied to the chroma prediction values for list 0 prediction using  $\text{RefPicList}[0][i]$  with  $j$  equal to 0 for Cb and  $j$  equal to 1 for Cr.

The variable  $\text{ChromaWeightL0}[i][j]$  is derived to be equal to  $(1 \ll \text{ChromaLog2WeightDenom}) + \text{delta\_chroma\_weight\_l0}[i][j]$ . When  $\text{chroma\_weight\_l0\_flag}[i]$  is equal to 1, the value of  $\text{delta\_chroma\_weight\_l0}[i][j]$  shall be in the range of  $-128$  to  $127$ , inclusive. When  $\text{chroma\_weight\_l0\_flag}[i]$  is equal to 0,  $\text{ChromaWeightL0}[i][j]$  is inferred to be equal to  $2^{\text{ChromaLog2WeightDenom}}$ .

**$\text{delta\_chroma\_offset\_l0}[i][j]$**  is the difference of the additive offset applied to the chroma prediction values for list 0 prediction using  $\text{RefPicList}[0][i]$  with  $j$  equal to 0 for Cb and  $j$  equal to 1 for Cr.

The variable  $\text{ChromaOffsetL0}[i][j]$  is derived as follows:

$$\begin{aligned} \text{ChromaOffsetL0}[i][j] = & \text{Clip3}(-128, 127, \\ & (128 + \text{delta\_chroma\_offset\_l0}[i][j] - \\ & ((128 * \text{ChromaWeightL0}[i][j]) \gg \text{ChromaLog2WeightDenom}))) \end{aligned} \quad (144)$$

The value of  $\text{delta\_chroma\_offset\_l0}[i][j]$  shall be in the range of  $-4 * 128$  to  $4 * 127$ , inclusive. When  $\text{chroma\_weight\_l0\_flag}[i]$  is equal to 0,  $\text{ChromaOffsetL0}[i][j]$  is inferred to be equal to 0.

**$\text{num\_l1\_weights}$**  specifies the number of weights signalled for entries in RPL 1 when  $\text{pps\_weighted\_bipred\_flag}$  and  $\text{pps\_wp\_info\_in\_ph\_flag}$  are both equal to 1. The value of  $\text{num\_l1\_weights}$  shall be in the range of 0 to  $\text{Min}(15, \text{num\_ref\_entries}[1][\text{RplIdx}[1]])$ , inclusive.

The variable  $\text{NumWeightsL1}$  is derived as follows:

$$\begin{aligned} & \text{if}(\text{!pps\_weighted\_bipred\_flag} \mid \\ & \quad (\text{pps\_wp\_info\_in\_ph\_flag} \ \&\& \ \text{num\_ref\_entries}[1][\text{RplIdx}[1]] == 0)) \\ & \quad \text{NumWeightsL1} = 0 \\ & \text{else if}(\text{pps\_wp\_info\_in\_ph\_flag}) \\ & \quad \text{NumWeightsL1} = \text{num\_l1\_weights} \\ & \text{else} \\ & \quad \text{NumWeightsL1} = \text{NumRefIdxActive}[1] \end{aligned} \quad (145)$$

**$\text{luma\_weight\_l1\_flag}[i]$**  equal to 1 specifies that weighting factors for the luma component of list 1 prediction using  $\text{RefPicList}[1][i]$  are present.  $\text{luma\_weight\_l1\_flag}[i]$  equal to 0 specifies that these weighting factors are not present. When not present, the value of  $\text{luma\_weight\_l1\_flag}[i]$  is inferred to be equal to 0.

**$\text{chroma\_weight\_l1\_flag}[i]$** ,  **$\text{delta\_luma\_weight\_l1}[i]$** ,  **$\text{luma\_offset\_l1}[i]$** ,  **$\text{delta\_chroma\_weight\_l1}[i][j]$** , and  **$\text{delta\_chroma\_offset\_l1}[i][j]$**  have the same semantics as  $\text{chroma\_weight\_l0\_flag}[i]$ ,  $\text{delta\_luma\_weight\_l0}[i]$ ,  $\text{luma\_offset\_l0}[i]$ ,  $\text{delta\_chroma\_weight\_l0}[i][j]$  and  $\text{delta\_chroma\_offset\_l0}[i][j]$ , respectively, with l0, L0, list 0 and List0 replaced by l1, L1, list 1 and List1, respectively.

The variable  $\text{sumWeightL0Flags}$  is derived to be equal to the sum of  $\text{luma\_weight\_l0\_flag}[i] + 2 * \text{chroma\_weight\_l0\_flag}[i]$ , for  $i = 0.. \text{NumRefIdxActive}[0] - 1$ .

When  $\text{sh\_slice\_type}$  is equal to B, the variable  $\text{sumWeightL1Flags}$  is derived to be equal to the sum of  $\text{luma\_weight\_l1\_flag}[i] + 2 * \text{chroma\_weight\_l1\_flag}[i]$ , for  $i = 0.. \text{NumRefIdxActive}[1] - 1$ .



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, rplIdx)` 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, rplIdx)` 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, rplIdx)` syntax structures with `listIdx` equal to `i` included in the SPS, of the `ref_pic_list_struct(listIdx, rplIdx)` 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  $\text{Ceil}(\text{Log2}(\text{sps\_num\_ref\_pic\_lists}[i]))$  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:

$$\text{RplsIdx}[i] = \text{rpl\_sps\_flag}[i] ? \text{rpl\_idx}[i] : \text{sps\_num\_ref\_pic\_lists}[i] \quad (146)$$

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:

$$\text{PocLsbLt}[i][j] = \text{ltrp\_in\_header\_flag}[i][\text{RplsIdx}[i]] ? \text{poc\_lsb\_lt}[i][j] : \text{rpls\_poc\_lsb\_lt}[i][\text{RplsIdx}[i]][j] \quad (147)$$

`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
    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]
    
```

(148)

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, rplIdx)` 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, rplIdx)` 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, rplIdx)` syntax structure specifies a candidate for RPL `listIdx`, and the term "the current picture" in the semantics specified in the remainder of this subclause 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, rplIdx)` syntax structures included in the SPS, and 2) is in a CLVS that refers to the SPS.

**num\_ref\_entries[listIdx][rplIdx]** specifies the number of entries in the `ref_pic_list_struct(listIdx, rplIdx)` syntax structure. The value of `num_ref_entries[listIdx][rplIdx]` shall be in the range of 0 to `MaxDpbSize + 13`, inclusive, where `MaxDpbSize` is as specified in subclause A.4.2.

**ltrp\_in\_header\_flag[listIdx][rplIdx]** equal to 0 specifies that the POC LSBs of the LTRP entries indicated in the `ref_pic_list_struct(listIdx, rplIdx)` syntax structure are present in the same syntax structure. `ltrp_in_header_flag[listIdx][rplIdx]` equal to 1 specifies that the POC LSBs of the LTRP entries indicated in the `ref_pic_list_struct(listIdx, rplIdx)` syntax structure are not present in the same syntax structure. When `sps_long_term_ref_pics_flag` is equal to 1 and `rplIdx` is equal to `sps_num_ref_pic_lists[listIdx]`, the value of `ltrp_in_header_flag[listIdx][rplIdx]` is inferred to be equal to 1.

**inter\_layer\_ref\_pic\_flag[listIdx][rplIdx][i]** equal to 1 specifies that the *i*-th entry in the `ref_pic_list_struct(listIdx, rplIdx)` syntax structure is an ILRP entry. `inter_layer_ref_pic_flag[listIdx][rplIdx][i]` equal to 0 specifies that the *i*-th entry in the `ref_pic_list_struct(listIdx, rplIdx)` syntax structure is not an ILRP entry. When not present, the value of `inter_layer_ref_pic_flag[listIdx][rplIdx][i]` is inferred to be equal to 0.

**st\_ref\_pic\_flag[listIdx][rplIdx][i]** equal to 1 specifies that the *i*-th entry in the `ref_pic_list_struct(listIdx, rplIdx)` syntax structure is an STRP entry. `st_ref_pic_flag[listIdx][rplIdx][i]` equal to 0 specifies that the *i*-th entry in the `ref_pic_list_struct(listIdx, rplIdx)` syntax structure is an LTRP entry. When `inter_layer_ref_pic_flag[listIdx][rplIdx][i]` is equal to 0 and `st_ref_pic_flag[listIdx][rplIdx][i]` is not present, the value of `st_ref_pic_flag[listIdx][rplIdx][i]` is inferred to be equal to 1.

The variable NumLtrpEntries[ listIdx ][ rplIdx ] is derived as follows:

```
for( i = 0, NumLtrpEntries[ listIdx ][ rplIdx ] = 0; i < num_ref_entries[ listIdx ][ rplIdx ]; i++ )
    if( !inter_layer_ref_pic_flag[ listIdx ][ rplIdx ][ i ] && !st_ref_pic_flag[ listIdx ][ rplIdx ][ i ] ) (149)
        NumLtrpEntries[ listIdx ][ rplIdx ]++
```

**abs\_delta\_poc\_st**[ listIdx ][ rplIdx ][ i ] specifies the value of the variable AbsDeltaPocSt[ listIdx ][ rplIdx ][ i ] as follows:

```
if( ( sps_weighted_pred_flag || sps_weighted_bipred_flag ) && i != 0 )
    AbsDeltaPocSt[ listIdx ][ rplIdx ][ i ] = abs_delta_poc_st[ listIdx ][ rplIdx ][ i ] (150)
else
    AbsDeltaPocSt[ listIdx ][ rplIdx ][ i ] = abs_delta_poc_st[ listIdx ][ rplIdx ][ i ] + 1
```

The value of abs\_delta\_poc\_st[ listIdx ][ rplIdx ][ i ] shall be in the range of 0 to  $2^{15} - 1$ , inclusive.

**strp\_entry\_sign\_flag**[ listIdx ][ rplIdx ][ i ] equal to 0 specifies that DeltaPocValSt[ listIdx ][ rplIdx ] is greater than or equal to 0. strp\_entry\_sign\_flag[ listIdx ][ rplIdx ][ i ] equal to 1 specifies that DeltaPocValSt[ listIdx ][ rplIdx ] is less than 0. When not present, the value of strp\_entry\_sign\_flag[ listIdx ][ rplIdx ][ i ] is inferred to be equal to 0.

The list DeltaPocValSt[ listIdx ][ rplIdx ] is derived as follows:

```
for( i = 0; i < num_ref_entries[ listIdx ][ rplIdx ]; i++ )
    if( !inter_layer_ref_pic_flag[ listIdx ][ rplIdx ][ i ] && st_ref_pic_flag[ listIdx ][ rplIdx ][ i ] ) (151)
        DeltaPocValSt[ listIdx ][ rplIdx ][ i ] = ( 1 - 2 * strp_entry_sign_flag[ listIdx ][ rplIdx ][ i ] ) *
            AbsDeltaPocSt[ listIdx ][ rplIdx ][ i ]
```

**rpls\_poc\_lsb\_lt**[ listIdx ][ rplIdx ][ 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, rplIdx ) syntax structure. The length of the rpls\_poc\_lsb\_lt[ listIdx ][ rplIdx ][ i ] syntax element is sps\_log2\_max\_pic\_order\_cnt\_lsb\_minus4 + 4 bits.

**ilrp\_idx**[ listIdx ][ rplIdx ][ 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, rplIdx ) syntax structure. The value of ilrp\_idx[ listIdx ][ rplIdx ][ 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 subclause 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 `cldx` of the coding tree unit at luma location ( `xCtb`, `yCtb` ).

When `alf_ctb_flag[ cldx ][ 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 to 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[ cldx ][ rx ][ ry ]` specifies the offset type as specified in Table 10 for the CTB at the location ( rx, ry ) for the colour component `cldx`. 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[ cldx ][ rx ][ ry ]` are derived as follows for `cldx` equal to 1..2:

- If `sao_type_idx_chroma` is present, `SaoTypeIdx[ cldx ][ rx ][ ry ]` is set equal to `sao_type_idx_chroma`.
- Otherwise (`sao_type_idx_chroma` is not present), `SaoTypeIdx[ cldx ][ rx ][ ry ]` is derived as follows:
  - If `sao_merge_left_flag` is equal to 1, `SaoTypeIdx[ cldx ][ rx ][ ry ]` is set equal to `SaoTypeIdx[ cldx ][ rx - 1 ][ ry ]`.
  - Otherwise, if `sao_merge_up_flag` is equal to 1, `SaoTypeIdx[ cldx ][ rx ][ ry ]` is set equal to `SaoTypeIdx[ cldx ][ rx ][ ry - 1 ]`.
  - Otherwise, `SaoTypeIdx[ cldx ][ rx ][ ry ]` is set equal to 0.

**Table 10 — Specification of the SAO type**

| <code>SaoTypeIdx[ cldx ][ rx ][ ry ]</code> | SAO type    |
|---------------------------------------------|-------------|
| 0                                           | Not applied |
| 1                                           | Band offset |
| 2                                           | Edge offset |

**sao\_offset\_abs[ cldx ][ rx ][ ry ][ i ]** specifies the offset value of *i*-th category for the CTB at the location ( rx, ry ) for the colour component `cldx`.

When `sao_offset_abs[ cldx ][ rx ][ ry ][ i ]` is not present, it is inferred as follows:

- If `sao_merge_left_flag` is equal to 1, `sao_offset_abs[ cldx ][ rx ][ ry ][ i ]` is inferred to be equal to `sao_offset_abs[ cldx ][ rx - 1 ][ ry ][ i ]`.
- Otherwise, if `sao_merge_up_flag` is equal to 1, `sao_offset_abs[ cldx ][ rx ][ ry ][ i ]` is inferred to be equal to `sao_offset_abs[ cldx ][ rx ][ ry - 1 ][ i ]`.
- Otherwise, `sao_offset_abs[ cldx ][ rx ][ ry ][ i ]` is inferred to be equal to 0.

**sao\_offset\_sign\_flag[ cldx ][ 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 `cldx`.



When `sao_offset_sign_flag[ cldx ][ rx ][ ry ][ i ]` is not present, it is inferred as follows:

- If `sao_merge_left_flag` is equal to 1, `sao_offset_sign_flag[ cldx ][ rx ][ ry ][ i ]` is inferred to be equal to `sao_offset_sign_flag[ cldx ][ rx - 1 ][ ry ][ i ]`.
- Otherwise, if `sao_merge_up_flag` is equal to 1, `sao_offset_sign_flag[ cldx ][ rx ][ ry ][ i ]` is inferred to be equal to `sao_offset_sign_flag[ cldx ][ rx ][ ry - 1 ][ i ]`.
- Otherwise, if `SaoTypeIdx[ cldx ][ rx ][ ry ]` is equal to 2, the following applies:
  - If `i` is equal to 0 or 1, `sao_offset_sign_flag[ cldx ][ rx ][ ry ][ i ]` is inferred to be equal 0.
  - Otherwise (`i` is equal to 2 or 3), `sao_offset_sign_flag[ cldx ][ rx ][ ry ][ i ]` is inferred to be equal 1.
- Otherwise, `sao_offset_sign_flag[ cldx ][ rx ][ ry ][ i ]` is inferred to be equal 0.

The list `SaoOffsetVal[ cldx ][ rx ][ ry ][ i ]` for `i` ranging from 0 to 4, inclusive, is derived as follows:

$$\begin{aligned} \text{SaoOffsetVal}[ \text{cldx} ][ \text{rx} ][ \text{ry} ][ 0 ] &= 0 \\ \text{for}( i = 0; i < 4; i++ ) \\ \text{SaoOffsetVal}[ \text{cldx} ][ \text{rx} ][ \text{ry} ][ i + 1 ] &= ( 1 - 2 * \text{sao\_offset\_sign\_flag}[ \text{cldx} ][ \text{rx} ][ \text{ry} ][ i ] ) * \\ &\quad ( \text{sao\_offset\_abs}[ \text{cldx} ][ \text{rx} ][ \text{ry} ][ i ] < ( \text{BitDepth} - \text{Min}( 10, \text{BitDepth} ) ) ) \end{aligned} \quad (153)$$

**sao\_band\_position**`[ cldx ][ rx ][ ry ]` specifies the displacement of the band offset of the sample range when `SaoTypeIdx[ cldx ][ rx ][ ry ]` is equal to 1.

When `sao_band_position[ cldx ][ rx ][ ry ]` is not present, it is inferred as follows:

- If `sao_merge_left_flag` is equal to 1, `sao_band_position[ cldx ][ rx ][ ry ]` is inferred to be equal to `sao_band_position[ cldx ][ rx - 1 ][ ry ]`.
- Otherwise, if `sao_merge_up_flag` is equal to 1, `sao_band_position[ cldx ][ rx ][ ry ]` is inferred to be equal to `sao_band_position[ cldx ][ rx ][ ry - 1 ]`.
- Otherwise, `sao_band_position[ cldx ][ 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[ cldx ][ rx ][ ry ]` specifies the offset type as specified in Table 11 for the CTB at the location `( rx, ry )` for the colour component `cldx`. 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[ cldx ][ rx ][ ry ]` are derived as follows for `cldx` equal to 1..2:

- If `sao_eo_class_chroma` is present, `SaoEoClass[ cldx ][ rx ][ ry ]` is set equal to `sao_eo_class_chroma`.
- Otherwise (`sao_eo_class_chroma` is not present), `SaoEoClass[ cldx ][ rx ][ ry ]` is derived as follows:
  - If `sao_merge_left_flag` is equal to 1, `SaoEoClass[ cldx ][ rx ][ ry ]` is set equal to `SaoEoClass[ cldx ][ rx - 1 ][ ry ]`.
  - Otherwise, if `sao_merge_up_flag` is equal to 1, `SaoEoClass[ cldx ][ rx ][ ry ]` is set equal to `SaoEoClass[ cldx ][ rx ][ ry - 1 ]`.



- Otherwise,  $\text{SaoEoClass}[\text{cIdx}][\text{rx}][\text{ry}]$  is set equal to 0.

**Table 11 — Specification of the SAO edge offset class**

| $\text{SaoEoClass}[\text{cIdx}][\text{rx}][\text{ry}]$ | SAO edge offset class     |
|--------------------------------------------------------|---------------------------|
| 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  $\text{allowSplitQt}$ ,  $\text{allowSplitBtVer}$ ,  $\text{allowSplitBtHor}$ ,  $\text{allowSplitTtVer}$ , and  $\text{allowSplitTtHor}$  are derived as follows:

- The allowed quad split process as specified in subclause 6.4.1 is invoked with the coding block size  $\text{cbSize}$  set equal to  $\text{cbWidth}$ , the current multi-type tree depth  $\text{mttDepth}$ ,  $\text{treeTypeCurr}$  and  $\text{modeTypeCurr}$  as inputs, and the output is assigned to  $\text{allowSplitQt}$ .
- The variables  $\text{minQtSize}$ ,  $\text{maxBtSize}$ ,  $\text{maxTtSize}$  and  $\text{maxMttDepth}$  are derived as follows:
  - If  $\text{treeTypeCurr}$  is equal to  $\text{DUAL\_TREE\_CHROMA}$ ,  $\text{minQtSize}$ ,  $\text{maxBtSize}$ ,  $\text{maxTtSize}$  and  $\text{maxMttDepth}$  are set equal to  $\text{MinQtSizeC}$ ,  $\text{MaxBtSizeC}$ ,  $\text{MaxTtSizeC}$  and  $\text{MaxMttDepthC} + \text{depthOffset}$ , respectively.
  - Otherwise,  $\text{minQtSize}$ ,  $\text{maxBtSize}$ ,  $\text{maxTtSize}$  and  $\text{maxMttDepth}$  are set equal to  $\text{MinQtSizeY}$ ,  $\text{MaxBtSizeY}$ ,  $\text{MaxTtSizeY}$  and  $\text{MaxMttDepthY} + \text{depthOffset}$ , respectively.
- The allowed binary split process as specified in subclause 6.4.2 is invoked with the binary split mode  $\text{SPLIT\_BT\_VER}$ , the coding block width  $\text{cbWidth}$ , the coding block height  $\text{cbHeight}$ , the location  $(x_0, y_0)$ , the current multi-type tree depth  $\text{mttDepth}$ , the maximum multi-type tree depth with offset  $\text{maxMttDepth}$ , the maximum binary tree size  $\text{maxBtSize}$ , the minimum quadtree size  $\text{minQtSize}$ , the current partition index  $\text{partIdx}$ ,  $\text{treeTypeCurr}$  and  $\text{modeTypeCurr}$  as inputs, and the output is assigned to  $\text{allowSplitBtVer}$ .
- The allowed binary split process as specified in subclause 6.4.2 is invoked with the binary split mode  $\text{SPLIT\_BT\_HOR}$ , the coding block height  $\text{cbHeight}$ , the coding block width  $\text{cbWidth}$ , the location  $(x_0, y_0)$ , the current multi-type tree depth  $\text{mttDepth}$ , the maximum multi-type tree depth with offset  $\text{maxMttDepth}$ , the maximum binary tree size  $\text{maxBtSize}$ , the minimum quadtree size  $\text{minQtSize}$ , the current partition index  $\text{partIdx}$ ,  $\text{treeTypeCurr}$  and  $\text{modeTypeCurr}$  as inputs, and the output is assigned to  $\text{allowSplitBtHor}$ .
- The allowed ternary split process as specified in subclause 6.4.3 is invoked with the ternary split mode  $\text{SPLIT\_TT\_VER}$ , the coding block width  $\text{cbWidth}$ , the coding block height  $\text{cbHeight}$ , the location  $(x_0, y_0)$ , the current multi-type tree depth  $\text{mttDepth}$ , the maximum multi-type tree depth with offset  $\text{maxMttDepth}$ , the maximum ternary tree size  $\text{maxTtSize}$ ,  $\text{treeTypeCurr}$  and  $\text{modeTypeCurr}$  as inputs, and the output is assigned to  $\text{allowSplitTtVer}$ .
- The allowed ternary split process as specified in subclause 6.4.3 is invoked with the ternary split mode  $\text{SPLIT\_TT\_HOR}$ , the coding block height  $\text{cbHeight}$ , the coding block width  $\text{cbWidth}$ , the location  $(x_0, y_0)$ , the current multi-type tree depth  $\text{mttDepth}$ , the maximum multi-type tree depth with offset  $\text{maxMttDepth}$ , the maximum ternary tree size  $\text{maxTtSize}$ ,  $\text{treeTypeCurr}$  and  $\text{modeTypeCurr}$  as inputs, and the output is assigned to  $\text{allowSplitTtHor}$ .

**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  $y = y0..y0 + cbHeight - 1$ .

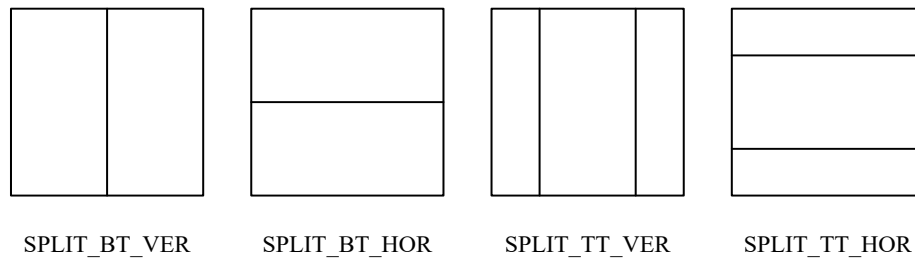


**Figure 8 — Multi-type tree splitting modes indicated by MttSplitMode**

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\_qtbt\_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  $y = y0..y0 + cbHeight - 1$ :

$$CbPosX[chType][x][y] = x0 \quad (154)$$

$$CbPosY[chType][x][y] = y0 \quad (155)$$

$$CbWidth[chType][x][y] = cbWidth \quad (156)$$

$$CbHeight[chType][x][y] = cbHeight \quad (157)$$

$$CqtDepth[chType][x][y] = cqtDepth \quad (158)$$

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 subclause 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  $c = (\text{treeType} == \text{SINGLE\_TREE} ? (0..1) : (\text{chType}..chType))$ ,  $x = x0..x0 + cbWidth - 1$  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  $c = (\text{treeType} == \text{SINGLE\_TREE} ? (0..1) : (\text{chType}.. \text{chType}))$ ,  $x = x0..x0 + \text{cbWidth} - 1$  and  $y = y0..y0 + \text{cbHeight} - 1$ .

**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  $c = (\text{treeType} == \text{SINGLE\_TREE} ? (0..1) : (\text{chType}.. \text{chType}))$ ,  $x = x0..x0 + \text{cbWidth} - 1$  and  $y = y0..y0 + \text{cbHeight} - 1$ .

**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  $x = x0..x0 + \text{cbWidth} - 1$ ,  $y = y0..y0 + \text{cbHeight} - 1$  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  $x = x0..x0 + \text{cbWidth} - 1$ ,  $y = y0..y0 + \text{cbHeight} - 1$  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  $1 + \text{intra\_subpartitions\_split\_flag}$ .

**Table 13 — Name association to `IntraSubPartitionsSplitType`**

| <code>IntraSubPartitionsSplitType</code> | Name of <code>IntraSubPartitionsSplitType</code> |
|------------------------------------------|--------------------------------------------------|
| 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 subclause 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 ][ cldx ] is set equal to **intra\_bdpcm\_chroma\_flag** for  $x = x0..x0 + cbWidth - 1$ ,  $y = y0..y0 + cbHeight - 1$  and  $cldx = 1..2$ .

**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 ][ cldx ] is set equal to **intra\_bdpcm\_chroma\_dir\_flag** for  $x = x0..x0 + cbWidth - 1$ ,  $y = y0..y0 + cbHeight - 1$  and  $cldx = 1..2$ .

**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, cpldx )` 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:

$$\text{MotionModelIdc}[ x ][ y ] = \text{merge\_subblock\_flag}[ x0 ][ y0 ] \quad (159)$$

— Otherwise (`general_merge_flag[ x0 ][ y0 ]` is equal to 0), the following applies:

$$\text{MotionModelIdc}[ x ][ y ] = \text{inter\_affine\_flag}[ x0 ][ y0 ] + \text{cu\_affine\_type\_flag}[ x0 ][ y0 ] \quad (160)$$

**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  $x_0, y_0$  specify the location (  $x_0, y_0$  ) 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  $x_0, y_0$  specify the location (  $x_0, y_0$  ) 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:

$$\text{MvdL0}[x_0][y_0][0] = \text{MvdL0}[x_0][y_0][0] \ll \text{AmvrShift} \quad (161)$$

$$\text{MvdL0}[x_0][y_0][1] = \text{MvdL0}[x_0][y_0][1] \ll \text{AmvrShift} \quad (162)$$

$$\text{MvdL1}[x_0][y_0][0] = \text{MvdL1}[x_0][y_0][0] \ll \text{AmvrShift} \quad (163)$$

$$\text{MvdL1}[x_0][y_0][1] = \text{MvdL1}[x_0][y_0][1] \ll \text{AmvrShift} \quad (164)$$

- 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:

$$\text{MvdCpL0}[x_0][y_0][0][0] = \text{MvdCpL0}[x_0][y_0][0][0] \ll \text{AmvrShift} \quad (165)$$

$$\text{MvdCpL1}[x_0][y_0][0][1] = \text{MvdCpL1}[x_0][y_0][0][1] \ll \text{AmvrShift} \quad (166)$$

$$\text{MvdCpL0}[x_0][y_0][1][0] = \text{MvdCpL0}[x_0][y_0][1][0] \ll \text{AmvrShift} \quad (167)$$

$$\text{MvdCpL1}[x_0][y_0][1][1] = \text{MvdCpL1}[x_0][y_0][1][1] \ll \text{AmvrShift} \quad (168)$$

$$\text{MvdCpL0}[x_0][y_0][2][0] = \text{MvdCpL0}[x_0][y_0][2][0] \ll \text{AmvrShift} \quad (169)$$

$$\text{MvdCpL1}[x_0][y_0][2][1] = \text{MvdCpL1}[x_0][y_0][2][1] \ll \text{AmvrShift} \quad (170)$$

Table 16 — Specification of AmvrShift

| amvr_flag | amvr_precision_idx | AmvrShift                              |                                                   |                                                                                          |
|-----------|--------------------|----------------------------------------|---------------------------------------------------|------------------------------------------------------------------------------------------|
|           |                    | inter_affine_flag[ x0 ]<br>[ y0 ] == 1 | CuPredMode[ chType ][ x0 ][ y0 ]<br>== MODE_IBC ) | inter_affine_flag[ x0 ][ y0 ] == 0<br>&&<br>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:

$$\text{ApplyLfnstFlag}[0] = (\text{lfnst\_idx} > 0) ? 1 : 0 \quad (173)$$

— The following applies for `cIdx = 1, 2`:

$$\text{ApplyLfnstFlag}[cIdx] = (\text{lfnst\_idx} > 0 \ \&\& \ \text{treeType} \neq \text{DUAL\_TREE\_CHROMA}) ? 1 : 0 \quad (174)$$

**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`:

$$\text{IbcVirBuf}[0][ (x + ( \text{IbcBufWidthY} >> 1 ) ) \% \text{IbcBufWidthY} ][ y \% \text{CtbSizeY} ] = -1 \quad (176)$$

#### 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 subclause 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:

$$\text{CurrentPaletteSize[ startComp ]} = \text{NumPredictedPaletteEntries} + \text{num\_signalled\_palette\_entries} \quad (177)$$

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:

$$\text{LocalDualTreeFlag} = ( \text{treeType} \neq \text{SINGLE\_TREE} \ \&\& \ ( \text{sh\_slice\_type} \neq 1 \ || \ ( \text{sh\_slice\_type} = 1 \ \&\& \ \text{sps\_qtbtt\_dual\_tree\_intra\_flag} = 0 ) ) ) ? 1 : 0 \quad (178)$$

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++ )
    for( i = 0; i < num_signalled_palette_entries; i++ )
        CurrentPaletteEntries[ cIdx ][ numPredictedPaletteEntries + i ] =
            new_palette_entries[ cIdx ][ i ]
    (179)

```

**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 ]
    else {
        if( !palette_transpose_flag )
            adjustedRefPaletteIndex = PaletteIndexMap[ xC ][ yC - 1 ]
        else
            adjustedRefPaletteIndex = PaletteIndexMap[ xC - 1 ][ yC ]
    }
}
```

(180)

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

$\text{mmvd\_direction\_idx}[x0][y0]$ . 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  $\text{mmvd\_cand\_flag}[x0][y0]$  is not present, it is inferred to be equal to 0.

$\text{mmvd\_distance\_idx}[x0][y0]$  specifies the index used to derive  $\text{MmvdDistance}[x0][y0]$  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  $\text{MmvdDistance}[x0][y0]$  based on  $\text{mmvd\_distance\_idx}[x0][y0]$**

| $\text{mmvd\_distance\_idx}[x0][y0]$ | $\text{MmvdDistance}[x0][y0]$              |                                            |
|--------------------------------------|--------------------------------------------|--------------------------------------------|
|                                      | $\text{ph\_mmvd\_fullpel\_only\_flag} = 0$ | $\text{ph\_mmvd\_fullpel\_only\_flag} = 1$ |
| 0                                    | 1                                          | 4                                          |
| 1                                    | 2                                          | 8                                          |
| 2                                    | 4                                          | 16                                         |
| 3                                    | 8                                          | 32                                         |
| 4                                    | 16                                         | 64                                         |
| 5                                    | 32                                         | 128                                        |
| 6                                    | 64                                         | 256                                        |
| 7                                    | 128                                        | 512                                        |

$\text{mmvd\_direction\_idx}[x0][y0]$  specifies index used to derive  $\text{MmvdSign}[x0][y0]$  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  $\text{MmvdSign}[x0][y0]$  based on  $\text{mmvd\_direction\_idx}[x0][y0]$**

| $\text{mmvd\_direction\_idx}[x0][y0]$ | $\text{MmvdSign}[x0][y0][0]$ | $\text{MmvdSign}[x0][y0][1]$ |
|---------------------------------------|------------------------------|------------------------------|
| 0                                     | +1                           | 0                            |
| 1                                     | -1                           | 0                            |
| 2                                     | 0                            | +1                           |
| 3                                     | 0                            | -1                           |

Both components of the merge plus MVD offset  $\text{MmvdOffset}[x0][y0]$  are derived as follows:

$$\text{MmvdOffset}[x0][y0][0] = (\text{MmvdDistance}[x0][y0] \ll 2) * \text{MmvdSign}[x0][y0][0] \quad (182)$$

$$\text{MmvdOffset}[x0][y0][1] = (\text{MmvdDistance}[x0][y0] \ll 2) * \text{MmvdSign}[x0][y0][1] \quad (183)$$

$\text{ciip\_flag}[x0][y0]$  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  $\text{ciip\_flag}[x0][y0]$  is not present, it is inferred as follows:

— If all the following conditions are true,  $\text{ciip\_flag}[x0][y0]$  is inferred to be equal to 1:

- $\text{sps\_ciip\_enabled\_flag}$  is equal to 1.
- $\text{general\_merge\_flag}[x0][y0]$  is equal to 1.
- $\text{merge\_subblock\_flag}[x0][y0]$  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:

$$\text{lMvd}[\text{compIdx}] = \text{abs\_mvd\_greater0\_flag}[\text{compIdx}] * (\text{abs\_mvd\_minus2}[\text{compIdx}] + 2) * (1 - 2 * \text{mvd\_sign\_flag}[\text{compIdx}]) \quad (184)$$

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][cldx][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 `cldx` 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][cldx][xC][yC]` is not specified in subclause 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_cbr_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_cbr_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_cbr_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_cbr_residual_flag[x0][y0]` is not present, it is inferred to be equal to 0.

Depending on `tu_joint_cbr_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_cbr_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  $\text{tu\_cb\_coded\_flag}[x0][y0]$  is equal to 1, the variable  $\text{TuCResMode}[x0][y0]$  is set equal to 2.
- Otherwise, the variable  $\text{TuCResMode}[x0][y0]$  is set equal to 3.

**cu\_qp\_delta\_abs** specifies the absolute value of the difference  $\text{CuQpDeltaVal}$  between the quantization parameter of the current coding unit and its prediction.

**cu\_qp\_delta\_sign\_flag** specifies the sign of  $\text{CuQpDeltaVal}$  as follows:

- If  $\text{cu\_qp\_delta\_sign\_flag}$  is equal to 0, the corresponding  $\text{CuQpDeltaVal}$  has a positive value.
- Otherwise ( $\text{cu\_qp\_delta\_sign\_flag}$  is equal to 1), the corresponding  $\text{CuQpDeltaVal}$  has a negative value.

When  $\text{cu\_qp\_delta\_sign\_flag}$  is not present, it is inferred to be equal to 0.

When  $\text{cu\_qp\_delta\_abs}$  is present, the variables  $\text{IsCuQpDeltaCoded}$  and  $\text{CuQpDeltaVal}$  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  $\text{CuQpDeltaVal}$  shall be in the range of  $-(32 + \text{QpBdOffset} / 2)$  to  $+(31 + \text{QpBdOffset} / 2)$ , inclusive.

**cu\_chroma\_qp\_offset\_flag** when present and equal to 1, specifies that an entry in the  $\text{pps\_cb\_qp\_offset\_list}[]$  is used to determine the value of  $\text{CuQpOffset}_{\text{Cb}}$ , a corresponding entry in the  $\text{pps\_cr\_qp\_offset\_list}[]$  is used to determine the value of  $\text{CuQpOffset}_{\text{Cr}}$ , and a corresponding entry in the  $\text{pps\_joint\_cbcr\_qp\_offset\_list}[]$  is used to determine the value of  $\text{CuQpOffset}_{\text{CbCr}}$ .  $\text{cu\_chroma\_qp\_offset\_flag}$  equal to 0 specifies that these lists are not used to determine the values of  $\text{CuQpOffset}_{\text{Cb}}$ ,  $\text{CuQpOffset}_{\text{Cr}}$ , and  $\text{CuQpOffset}_{\text{CbCr}}$ .

**cu\_chroma\_qp\_offset\_idx**, when present, specifies the index into the  $\text{pps\_cb\_qp\_offset\_list}[]$ ,  $\text{pps\_cr\_qp\_offset\_list}[]$ , and  $\text{pps\_joint\_cbcr\_qp\_offset\_list}[]$  that is used to determine the value of  $\text{CuQpOffset}_{\text{Cb}}$ ,  $\text{CuQpOffset}_{\text{Cr}}$ , and  $\text{CuQpOffset}_{\text{CbCr}}$ . When present, the value of  $\text{cu\_chroma\_qp\_offset\_idx}$  shall be in the range of 0 to  $\text{pps\_chroma\_qp\_offset\_list\_len\_minus1}$ , inclusive. When not present, the value of  $\text{cu\_chroma\_qp\_offset\_idx}$  is inferred to be equal to 0.

When  $\text{cu\_chroma\_qp\_offset\_flag}$  is present, the following applies:

- The variable  $\text{IsCuChromaQpOffsetCoded}$  is set equal to 1.
- The variables  $\text{CuQpOffset}_{\text{Cb}}$ ,  $\text{CuQpOffset}_{\text{Cr}}$ , and  $\text{CuQpOffset}_{\text{CbCr}}$  are derived as follows:
  - If  $\text{cu\_chroma\_qp\_offset\_flag}$  is equal to 1, the following applies:

$$\text{CuQpOffset}_{\text{Cb}} = \text{pps\_cb\_qp\_offset\_list}[\text{cu\_chroma\_qp\_offset\_idx}] \quad (187)$$

$$\text{CuQpOffset}_{\text{Cr}} = \text{pps\_cr\_qp\_offset\_list}[\text{cu\_chroma\_qp\_offset\_idx}] \quad (188)$$

$$\text{CuQpOffset}_{\text{CbCr}} = \text{pps\_joint\_cbcr\_qp\_offset\_list}[\text{cu\_chroma\_qp\_offset\_idx}] \quad (189)$$

- Otherwise ( $\text{cu\_chroma\_qp\_offset\_flag}$  is equal to 0),  $\text{CuQpOffset}_{\text{Cb}}$ ,  $\text{CuQpOffset}_{\text{Cr}}$ , and  $\text{CuQpOffset}_{\text{CbCr}}$  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.  $\text{transform\_skip\_flag}[x0][y0][cIdx]$  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 subclause 7.3.11.11, it is inferred to be equal to 0. When the value of AbsLevelPass1[ xC ][ yC ] is not specified in subclause 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:

$$\text{CoeffMin} = -(1 \ll (\text{sps\_extended\_precision\_flag} ? \text{Max}(15, \text{Min}(20, \text{BitDepth} + 6)) : 15)) \quad (190)$$

$$\text{CoeffMax} = (1 \ll (\text{sps\_extended\_precision\_flag} ? \text{Max}(15, \text{Min}(20, \text{BitDepth} + 6)) : 15)) - 1 \quad (191)$$

The array QStateTransTable[ ][ ] is specified as follows:

$$\text{QStateTransTable}[ ][ ] = \{ \{ 0, 2 \}, \{ 2, 0 \}, \{ 1, 3 \}, \{ 3, 1 \} \} \quad (192)$$

**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:

$$\text{LastSignificantCoeffX} = \text{last\_sig\_coeff\_x\_prefix} \quad (193)$$

- Otherwise (last\_sig\_coeff\_x\_suffix is present), the following applies:

$$\text{LastSignificantCoeffX} = (1 \ll ((\text{last\_sig\_coeff\_x\_prefix} \gg 1) - 1)) * (2 + (\text{last\_sig\_coeff\_x\_prefix} \& 1)) + \text{last\_sig\_coeff\_x\_suffix} \quad (194)$$

When `sh_reverse_last_sig_coeff_flag` is equal to 1, the value of `LastSignificantCoeffX` is modified as follows:

$$\text{LastSignificantCoeffX} = (1 \ll \text{Log2ZoTbWidth}) - 1 - \text{LastSignificantCoeffX} \quad (195)$$

**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  $(1 \ll ((\text{last\_sig\_coeff\_y\_prefix} \gg 1) - 1)) - 1$ , 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:

$$\text{LastSignificantCoeffY} = \text{last\_sig\_coeff\_y\_prefix} \quad (196)$$

— Otherwise (`last_sig_coeff_y_suffix` is present), the following applies:

$$\text{LastSignificantCoeffY} = (1 \ll ((\text{last\_sig\_coeff\_y\_prefix} \gg 1) - 1)) * (2 + (\text{last\_sig\_coeff\_y\_prefix} \& 1)) + \text{last\_sig\_coeff\_y\_suffix} \quad (197)$$

When `sh_reverse_last_sig_coeff_flag` is equal to 1, the value of `LastSignificantCoeffY` is modified as follows:

$$\text{LastSignificantCoeffY} = (1 \ll \text{Log2ZoTbHeight}) - 1 - \text{LastSignificantCoeffY} \quad (198)$$

**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 transform coefficient levels:

When `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.

When `sb_coded_flag`[`xS`][`yS`] is not present, it is inferred to be equal to 1.

**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`][`cldx`] 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:

—  $(xC \& ((1 \ll \log2SbW) - 1), yC \& ((1 \ll \log2SbH) - 1))$  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 subclause 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 ( $\text{CoeffSignLevel}[xC][yC]$  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 document.

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 document 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 document 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 subclause, for both IRAP pictures and GDR pictures, enable identification of the CLVSS pictures and CLVSSs of each layer and consequently enable identification of the CVSS AUs and the boundaries of the CVSSs 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 document 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 document 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 document 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  $((\text{opi\_htid\_plus1} > 0) ? \text{opi\_htid\_plus1} - 1 : 0)$ .
  - Otherwise, Htid is set equal to  $\text{vps\_ptl\_max\_tid}[\text{vps\_ols\_ptl\_idx}[\text{TargetOlsIdx}]]$ .

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 document 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  $((\text{opi\_htid\_plus1} > 0) ? \text{opi\_htid\_plus1} - 1 : 0)$ .

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 subclause C.1, DecodingUnitHrdFlag is set as specified in subclause C.1.
- Otherwise, DecodingUnitHrdFlag is set equal to  $(\text{DuHrdPreferredFlag} \ \&\& \ \text{general\_du\_hrd\_params\_present\_flag})$ , 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.

Subclause 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 subclause 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 subclause 8.2.
2. The processes in subclause 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 subclause 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 subclause 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 subclause 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 subclause 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 subclause 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 subclause 8.3.6 is invoked for derivation of the variables `ColPic` and `NoBackwardPredFlag`.
3. The processes in subclauses 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 processes of subclause 8.8 have 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 processes of subclause 8.8 have 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 subclause.

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  $\text{MaxPicOrderCntLsb} / 2$ . However, on the other hand, the bitstream conformance constraint specified by item 8 in subclause C.4 disallows the value of POC difference between the current picture and the prevTid0Pic to be greater than or equal to  $\text{MaxPicOrderCntLsb} / 2$ . 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  $\text{sps\_log2\_max\_pic\_order\_cnt\_lsb\_minus4}$  to be large enough such that the POC difference between the current picture and the prevTid0Pic is less than  $\text{MaxPicOrderCntLsb} / 2$ .

NOTE 2 When  $\text{vps\_max\_tid\_il\_ref\_pics\_plus1}[i][j]$  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  $\text{ph\_recovery\_poc\_cnt}$  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 subclause C.6, encoders are expected to set the value of  $\text{sps\_log2\_max\_pic\_order\_cnt\_lsb\_minus4}$  to be large enough such that the POC difference between the current picture and the prevTid0Pic is less than  $\text{MaxPicOrderCntLsb} / 2$ . Due to that the bitstream conformance constraint specified by item 8 in subclause C.4 disallows the value of POC difference between the current picture and the prevTid0Pic to be greater than or equal to  $\text{MaxPicOrderCntLsb} / 2$ , the other option that ensures correct POC derivation by including  $\text{ph\_poc\_msb\_cycle\_val}$  for each IRAP picture and each GDR picture with  $\text{ph\_recovery\_poc\_cnt}$  equal to 0 is precluded.

— The variable prevPicOrderCntLsb is set equal to  $\text{ph\_pic\_order\_cnt\_lsb}$  of prevTid0Pic.

— The variable prevPicOrderCntMsb is set equal to  $\text{PicOrderCntMsb}$  of prevTid0Pic.

The variable  $\text{PicOrderCntMsb}$  of the current picture is derived as follows:

- If  $\text{ph\_poc\_msb\_cycle\_val}$  is present,  $\text{PicOrderCntMsb}$  is set equal to  $\text{ph\_poc\_msb\_cycle\_val} * \text{MaxPicOrderCntLsb}$ .
- Otherwise ( $\text{ph\_poc\_msb\_cycle\_val}$  is not present), if the current picture is a CLVSS picture,  $\text{PicOrderCntMsb}$  is set equal to 0.
- Otherwise,  $\text{PicOrderCntMsb}$  is derived as follows:

```

if( ( ph_pic_order_cnt_lsb < prevPicOrderCntLsb ) &&
    ( ( prevPicOrderCntLsb - ph_pic_order_cnt_lsb ) >= ( MaxPicOrderCntLsb / 2 ) ) )
    PicOrderCntMsb = prevPicOrderCntMsb + MaxPicOrderCntLsb
else if( ( ph_pic_order_cnt_lsb > prevPicOrderCntLsb ) &&
    ( ( ph_pic_order_cnt_lsb - prevPicOrderCntLsb ) > ( MaxPicOrderCntLsb / 2 ) ) )
    PicOrderCntMsb = prevPicOrderCntMsb - MaxPicOrderCntLsb
else
    PicOrderCntMsb = prevPicOrderCntMsb

```

(199)

$\text{PicOrderCntVal}$  is derived as follows:

$$\text{PicOrderCntVal} = \text{PicOrderCntMsb} + \text{ph\_pic\_order\_cnt\_lsb} \quad (200)$$

NOTE 3 All CLVSS pictures for which  $\text{ph\_poc\_msb\_cycle\_val}$  is not present have  $\text{PicOrderCntVal}$  equal to  $\text{ph\_pic\_order\_cnt\_lsb}$  since for those pictures  $\text{PicOrderCntMsb}$  is set equal to 0.

The value of  $\text{PicOrderCntVal}$  shall be in the range of  $-2^{31}$  to  $2^{31} - 1$ , inclusive.

In one CVS, the  $\text{PicOrderCntVal}$  values for any two coded pictures with the same value of  $\text{nuh\_layer\_id}$  shall not be the same.

All pictures in any particular AU shall have the same value of  $\text{PicOrderCntVal}$ .

The function  $\text{PicOrderCnt}(\text{picX})$  is specified as follows:

$$\text{PicOrderCnt}(\text{picX}) = \text{PicOrderCntVal of the picture picX} \quad (201)$$

The function  $\text{DiffPicOrderCnt}(\text{picA}, \text{picB})$  is specified as follows:

$$\text{DiffPicOrderCnt}(\text{picA}, \text{picB}) = \text{PicOrderCnt}(\text{picA}) - \text{PicOrderCnt}(\text{picB}) \quad (202)$$

The bitstream shall not contain data that result in values of  $\text{DiffPicOrderCnt}(\text{picA}, \text{picB})$  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  $\text{DiffPicOrderCnt}(X, Y)$  and  $\text{DiffPicOrderCnt}(X, Z)$  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.,  $\text{RefPicList}[0]$ ), is used in decoding of the slice data. When decoding a B slice, both RPL 0 and RPL 1 (i.e.,  $\text{RefPicList}[1]$ ) are used in decoding of the slice data.

At the beginning of the decoding process for each slice of a picture, the RPLs  $\text{RefPicList}[0]$  and  $\text{RefPicList}[1]$  are derived. The RPLs are used in marking of reference pictures as specified in subclause 8.3.3 or in decoding of the slice data.

NOTE 1 For an I slice of a picture,  $\text{RefPicList}[0]$  and  $\text{RefPicList}[1]$  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,  $\text{RefPicList}[1]$  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  $\text{sps\_idr\_rpl\_present\_flag}$  is equal to 0,  $\text{pps\_rpl\_info\_in\_ph\_flag}$  is equal to 0, and  $\text{nal\_unit\_type}$  is equal to  $\text{IDR\_W\_RADL}$  or  $\text{IDR\_N\_LP}$ , the RPLs  $\text{RefPicList}[0]$  and  $\text{RefPicList}[1]$  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  $\text{RplsIdx}[i]$  is inferred to be equal to  $\text{sps\_num\_ref\_pic\_lists}[i]$ .
- The value of  $\text{num\_ref\_entries}[i][\text{RplsIdx}[i]]$  is inferred to be equal to 0.
- The value of  $\text{NumRefIdxActive}[i]$  is inferred to be equal to 0.

Otherwise, the RPLs  $\text{RefPicList}[0]$  and  $\text{RefPicList}[1]$ , the reference picture scaling ratios  $\text{RefPicScale}[i][j][0]$  and  $\text{RefPicScale}[i][j][1]$ , and the reference picture scaled flags  $\text{RprConstraintsActiveFlag}[0][j]$  and  $\text{RprConstraintsActiveFlag}[1][j]$  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"
            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 {

```

(203)

```

        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 ][ RplIdx[ 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 subclause 8.3.4.

- For each  $i$  equal to 0 or 1,  $\text{num\_ref\_entries}[i][\text{RplIdx}[i]]$  shall not be less than  $\text{NumRefIdxActive}[i]$ .
- 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  $\text{ph\_non\_ref\_pic\_flag}$  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  $\text{setOfRefPics}$  be the set of unique pictures referred to by all entries in RefPicList[ 0 ] that have the same  $\text{nuh\_layer\_id}$  as the current picture and all entries in RefPicList[ 1 ] that have the same  $\text{nuh\_layer\_id}$  as the current picture. The number of pictures in  $\text{setOfRefPics}$  shall be less than or equal to  $\text{MaxDpbSize} - 1$ , inclusive, where  $\text{MaxDpbSize}$  is as specified in subclause A.4.2, and  $\text{setOfRefPics}$  shall be the same for all slices of a picture.
- When the current slice has  $\text{nal\_unit\_type}$  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  $\text{nuh\_layer\_id}$  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  $\text{nuh\_layer\_id}$  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  $\text{nuh\_layer\_id}$  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  $\text{tId}$ ,  $\text{nuh\_layer\_id}$  equal to a particular value  $\text{layerId}$ , and subpicture index equal to a particular value  $\text{subpicIdx}$ , is a subpicture that follows, in decoding order, an STSA subpicture with TemporalId equal to  $\text{tId}$ ,  $\text{nuh\_layer\_id}$  equal to  $\text{layerId}$ , and subpicture index equal to  $\text{subpicIdx}$ , there shall be no picture with TemporalId equal to  $\text{tId}$  and  $\text{nuh\_layer\_id}$  equal to  $\text{layerId}$  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:
  - 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
  - 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
  - A leading picture associated with a CRA picture with `NoOutputBeforeRecoveryFlag` equal to 1
  - 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:
  - The picture shall be in the same AU as the current picture.
  - The picture shall be present in the DPB.
  - The picture shall have nuh\_layer\_id refPicLayerId less than the nuh\_layer\_id of the current picture.
  - Either of the following conditions shall apply:
    - The picture is a GDR picture with ph\_recovery\_poc\_cnt equal to 0 or an IRAP picture.

- The picture has TemporalId less than  $\text{vps\_max\_tid\_il\_ref\_pics\_plus1}[\text{currLayerIdx}][\text{refLayerIdx}]$ , where  $\text{currLayerIdx}$  and  $\text{refLayerIdx}$  are equal to  $\text{GeneralLayerIdx}[\text{nuh\_layer\_id}]$  and  $\text{GeneralLayerIdx}[\text{refpicLayerId}]$ , respectively.
- Each ILRP entry, when present, in  $\text{RefPicList}[0]$  or  $\text{RefPicList}[1]$  of a slice shall be an active entry.
- When  $\text{sps\_num\_subpics\_minus1}$  is greater than 0 and the current subpicture with subpicture index  $\text{subpicIdx}$  has  $\text{sps\_subpic\_treated\_as\_pic\_flag}[\text{subpicIdx}]$  equal to 1, it is a requirement of bitstream conformance that both of the following conditions shall be true:
  - Exactly one and not both of the following two conditions shall be true:
    - The picture referred to by each active entry in  $\text{RefPicList}[0]$  or  $\text{RefPicList}[1]$  and the current picture have the same value for each of the following:
      - $\text{pps\_pic\_width\_in\_luma\_samples}$
      - $\text{pps\_pic\_height\_in\_luma\_samples}$
      - $\text{sps\_num\_subpics\_minus1}$
      - $\text{sps\_log2\_ctu\_size\_minus5}$
      - $\text{sps\_subpic\_ctu\_top\_left\_x}[i]$ ,  $\text{sps\_subpic\_ctu\_top\_left\_y}[i]$ ,  $\text{sps\_subpic\_width\_minus1}[i]$ ,  $\text{sps\_subpic\_height\_minus1}[i]$ , and  $\text{sps\_subpic\_treated\_as\_pic\_flag}[i]$ , respectively, for each value of  $i$  in the range of 0 to  $\text{sps\_num\_subpics\_minus1}$ , inclusive
      - $\text{SubpicIdVal}[\text{subpicIdx}]$
    - The picture referred to by each active entry in  $\text{RefPicList}[0]$  or  $\text{RefPicList}[1]$  is an ILRP entry for which the value of  $\text{sps\_num\_subpics\_minus1}$  is equal to 0.
  - 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  $\text{RefPicList}[0]$  or  $\text{RefPicList}[1]$  have the same value for each of the following:
    - $\text{pps\_pic\_width\_in\_luma\_samples}$
    - $\text{pps\_pic\_height\_in\_luma\_samples}$
    - $\text{sps\_num\_subpics\_minus1}$
    - $\text{sps\_log2\_ctu\_size\_minus5}$
    - $\text{sps\_subpic\_ctu\_top\_left\_x}[i]$ ,  $\text{sps\_subpic\_ctu\_top\_left\_y}[i]$ ,  $\text{sps\_subpic\_width\_minus1}[i]$ ,  $\text{sps\_subpic\_height\_minus1}[i]$ ,  $\text{sps\_subpic\_treated\_as\_pic\_flag}[i]$ ,  $\text{SubpicIdVal}[i]$ , respectively, for each value of  $i$  in the range of 0 to  $\text{sps\_num\_subpics\_minus1}$ , 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 subclause 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  $\text{Log}_2(\text{MaxPicOrderCntLsb})$  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][RplIdx[i]] - 1`, inclusive, that is equal to "no reference picture", a picture is generated as specified in subclause 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][RplIdx[i]][j]` is equal to 1 and `inter_layer_ref_pic_flag[i][RplIdx[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][RplIdx[i]][j]` is equal to 0 and `inter_layer_ref_pic_flag[i][RplIdx[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  $(\text{RefPicLtPocList}[i][j] \& (\text{MaxPicOrderCntLsb} - 1))$ , 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  $1 \ll (\text{BitDepth} - 1)$ .
- When  $\text{sps\_chroma\_format\_idc}$  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  $1 \ll (\text{BitDepth} - 1)$ .
- The prediction mode  $\text{CuPredMode}[0][x][y]$  is set equal to  $\text{MODE\_INTRA}$  for  $x$  ranging from 0 to  $\text{pps\_pic\_width\_in\_luma\_samples} - 1$ , inclusive, and  $y$  ranging from 0 to  $\text{pps\_pic\_height\_in\_luma\_samples} - 1$ , inclusive.

NOTE The output of the recovery point picture of a GDR picture with  $\text{NoOutputBeforeRecoveryFlag}$  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  $\text{CuPredMode}[0][x][y]$ .

### 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 subclause 8.3.2, but prior to the parsing and decoding of any coding unit.

Outputs of this process are  $\text{RefIdxSymL0}$  and  $\text{RefIdxSymL1}$  specifying the list 0 and list 1 reference picture indices for symmetric motion vector differences, i.e., when  $\text{sym\_mvd\_flag}$  is equal to 1 for a coding unit.

The variable  $\text{RefIdxSymLX}$ , with  $X = 0..1$ , is derived as follows:

- The variable  $\text{currPic}$  specifies the current picture.
- $\text{RefIdxSymL0}$  is set equal to -1.
- For each index  $i$  with  $i = 0..\text{NumRefIdxActive}[0] - 1$ , the following applies:
  - When all of the following conditions are true,  $\text{RefIdxSymL0}$  is set equal to  $i$ :
    - $\text{RefPicList}[0][i]$  is a short-term-reference picture,
    - $\text{DiffPicOrderCnt}(\text{currPic}, \text{RefPicList}[0][i]) > 0$ ,
    - $\text{DiffPicOrderCnt}(\text{currPic}, \text{RefPicList}[0][i]) < \text{DiffPicOrderCnt}(\text{currPic}, \text{RefPicList}[0][\text{RefIdxSymL0}])$  or  $\text{RefIdxSymL0}$  is equal to -1.
- $\text{RefIdxSymL1}$  is set equal to -1.
- For each index  $i$  with  $i = 0..\text{NumRefIdxActive}[1] - 1$ , the following applies:
  - When all of the following conditions are true,  $\text{RefIdxSymL1}$  is set equal to  $i$ :
    - $\text{RefPicList}[1][i]$  is a short-term-reference picture,
    - $\text{DiffPicOrderCnt}(\text{currPic}, \text{RefPicList}[1][i]) < 0$ ,
    - $\text{DiffPicOrderCnt}(\text{currPic}, \text{RefPicList}[1][i]) > \text{DiffPicOrderCnt}(\text{currPic}, \text{RefPicList}[1][\text{RefIdxSymL1}])$  or  $\text{RefIdxSymL1}$  is equal to -1.
- When  $\text{RefIdxSymL0}$  is equal to -1 or  $\text{RefIdxSymL1}$  is equal to -1, the following applies:
  - $\text{RefIdxSymL0}$  is set equal to -1 and  $\text{RefIdxSymL1}$  is set equal to -1.
  - For each index  $i$  with  $i = 0..\text{NumRefIdxActive}[0] - 1$ , the following applies:
    - When all of the following conditions are true,  $\text{RefIdxSymL0}$  is set equal to  $i$ :
      - $\text{RefPicList}[0][i]$  is a short-term-reference picture,

- $\text{DiffPicOrderCnt}(\text{currPic}, \text{RefPicList}[0][i]) < 0$ ,
- $\text{DiffPicOrderCnt}(\text{currPic}, \text{RefPicList}[0][i]) > \text{DiffPicOrderCnt}(\text{currPic}, \text{RefPicList}[0][\text{RefIdxSymL0}])$  or  $\text{RefIdxSymL0}$  is equal to  $-1$ .
- For each index  $i$  with  $i = 0.. \text{NumRefIdxActive}[1] - 1$ , the following applies:
  - When all of the following conditions are true,  $\text{RefIdxSymL1}$  is set equal to  $i$ :
    - $\text{RefPicList}[1][i]$  is a short-term-reference picture,
    - $\text{DiffPicOrderCnt}(\text{currPic}, \text{RefPicList}[1][i]) > 0$ ,
    - $\text{DiffPicOrderCnt}(\text{currPic}, \text{RefPicList}[1][i]) < \text{DiffPicOrderCnt}(\text{currPic}, \text{RefPicList}[1][\text{RefIdxSymL1}])$  or  $\text{RefIdxSymL1}$  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 subclause 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  $\text{DiffPicOrderCnt}(\text{aPic}, \text{currPic})$  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  $(x_{Cb}, y_{Cb})$  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 subclause 8.7.1 is invoked with the luma location  $(x_{Cb}, y_{Cb})$ , 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 subclause 8.4.5.3 is invoked with  $(x_{CbComp}, y_{CbComp})$  set equal to the luma location  $(x_{Cb}, y_{Cb})$ , 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  $x = x_{Cb} \cdot x_{Cb} + cbWidth - 1$  and  $y = y_{Cb} \cdot y_{Cb} + cbHeight - 1$  is set to be equal to `intra_mip_mode[ xCb ][ yCb ]`.
    - Otherwise, the derivation process for the luma intra prediction mode as specified in subclause 8.4.2 is invoked with the luma location  $(x_{Cb}, y_{Cb})$ , 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 `predModelIntra` 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 subclause 8.4.5.1 is invoked with the sample location  $(x_{Tb0}, y_{Tb0})$  set equal to the luma location  $(x_{Cb}, y_{Cb})$ , 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`, `predModelIntra`, and the variable `cIdx` set equal to 0 and `controlPara` set equal to 1 as inputs, and the output is a residual sample array `resSamplesL`.
    - The general decoding process for intra blocks as specified in subclause 8.4.5.1 is invoked with the sample location  $(x_{Tb0}, y_{Tb0})$  set equal to the luma location  $(x_{Cb}, y_{Cb})$ , 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`, `predModelIntra`, and the variable `cIdx` set equal to 1 and `controlPara` set equal to 1 as inputs, and the output is a residual sample array `resSamplesCb`.
    - The general decoding process for intra blocks as specified in subclause 8.4.5.1 is invoked with the sample location  $(x_{Tb0}, y_{Tb0})$  set equal to the luma location  $(x_{Cb}, y_{Cb})$ , 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`, `predModelIntra`, and the variable `cIdx` set equal to 2 and `controlPara` set equal to 1 as inputs, and the output is a residual sample array `resSamplesCr`.
    - The residual modification process for residual blocks using colour space conversion as specified in subclause 8.7.4.6 is invoked with the variable `nTbW` set equal to `cbWidth`, the variable `nTbH` set equal to `cbHeight`, the array `rY` set equal to `resSamplesL`, the array `rCb` set equal to `resSamplesCb`, and the array `rCr` set equal to `resSamplesCr` as inputs, and the output are modified versions of the arrays `resSamplesL`, `resSamplesCb` and `resSamplesCr`.
  4. The general decoding process for intra blocks as specified in subclause 8.4.5.1 is invoked with the sample location  $(x_{Tb0}, y_{Tb0})$  set equal to the luma location  $(x_{Cb}, y_{Cb})$ , 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`, `predModelIntra`, 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 `resSamplesL` 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 subclause 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 subclause 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 subclause 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 subclause 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 `resSamplesCb` 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 subclause 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 `resSamplesCr` 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 subclause 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 cldx 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:

$$\text{candModeList}[ 0 ] = \text{candIntraPredModeA} \quad (204)$$

$$\text{candModeList}[ 1 ] = 2 + ( ( \text{candIntraPredModeA} + 61 ) \% 64 ) \quad (205)$$

$$\text{candModeList}[ 2 ] = 2 + ( ( \text{candIntraPredModeA} - 1 ) \% 64 ) \quad (206)$$

$$\text{candModeList}[ 3 ] = 2 + ( ( \text{candIntraPredModeA} + 60 ) \% 64 ) \quad (207)$$

$$\text{candModeList}[ 4 ] = 2 + ( \text{candIntraPredModeA} \% 64 ) \quad (208)$$

- Otherwise, if  $\text{candIntraPredModeB}$  is not equal to  $\text{candIntraPredModeA}$  and  $\text{candIntraPredModeA}$  or  $\text{candIntraPredModeB}$  is greater than  $\text{INTRA\_DC}$ , the following applies:

- The variables  $\text{minAB}$  and  $\text{maxAB}$  are derived as follows:

$$\text{minAB} = \text{Min}(\text{candIntraPredModeA}, \text{candIntraPredModeB}) \quad (209)$$

$$\text{maxAB} = \text{Max}(\text{candIntraPredModeA}, \text{candIntraPredModeB}) \quad (210)$$

- If  $\text{candIntraPredModeA}$  and  $\text{candIntraPredModeB}$  are both greater than  $\text{INTRA\_DC}$ ,  $\text{candModeList}[x]$  with  $x = 0..4$  is derived as follows:

$$\text{candModeList}[0] = \text{candIntraPredModeA} \quad (211)$$

$$\text{candModeList}[1] = \text{candIntraPredModeB} \quad (212)$$

- If  $\text{maxAB} - \text{minAB}$  is equal to 1, the following applies:

$$\text{candModeList}[2] = 2 + ((\text{minAB} + 61) \% 64) \quad (213)$$

$$\text{candModeList}[3] = 2 + ((\text{maxAB} - 1) \% 64) \quad (214)$$

$$\text{candModeList}[4] = 2 + ((\text{minAB} + 60) \% 64) \quad (215)$$

- Otherwise, if  $\text{maxAB} - \text{minAB}$  is greater than or equal to 62, the following applies:

$$\text{candModeList}[2] = 2 + ((\text{minAB} - 1) \% 64) \quad (216)$$

$$\text{candModeList}[3] = 2 + ((\text{maxAB} + 61) \% 64) \quad (217)$$

$$\text{candModeList}[4] = 2 + (\text{minAB} \% 64) \quad (218)$$

- Otherwise, if  $\text{maxAB} - \text{minAB}$  is equal to 2, the following applies:

$$\text{candModeList}[2] = 2 + ((\text{minAB} - 1) \% 64) \quad (219)$$

$$\text{candModeList}[3] = 2 + ((\text{minAB} + 61) \% 64) \quad (220)$$

$$\text{candModeList}[4] = 2 + ((\text{maxAB} - 1) \% 64) \quad (221)$$

- Otherwise, the following applies:

$$\text{candModeList}[2] = 2 + ((\text{minAB} + 61) \% 64) \quad (222)$$

$$\text{candModeList}[3] = 2 + ((\text{minAB} - 1) \% 64) \quad (223)$$

$$\text{candModeList}[4] = 2 + ((\text{maxAB} + 61) \% 64) \quad (224)$$

- Otherwise ( $\text{candIntraPredModeA}$  or  $\text{candIntraPredModeB}$  is greater than  $\text{INTRA\_DC}$ ),  $\text{candModeList}[x]$  with  $x = 0..4$  is derived as follows:

$$\text{candModeList}[0] = \text{maxAB} \quad (225)$$

$$\text{candModeList}[1] = 2 + ((\text{maxAB} + 61) \% 64) \quad (226)$$

$$\text{candModeList}[2] = 2 + ((\text{maxAB} - 1) \% 64) \quad (227)$$

$$\text{candModeList}[3] = 2 + ( (\text{maxAB} + 60) \% 64 ) \quad (228)$$

$$\text{candModeList}[4] = 2 + ( \text{maxAB} \% 64 ) \quad (229)$$

— Otherwise, the following applies:

$$\text{candModeList}[0] = \text{INTRA\_DC} \quad (230)$$

$$\text{candModeList}[1] = \text{INTRA\_ANGULAR50} \quad (231)$$

$$\text{candModeList}[2] = \text{INTRA\_ANGULAR18} \quad (232)$$

$$\text{candModeList}[3] = \text{INTRA\_ANGULAR46} \quad (233)$$

$$\text{candModeList}[4] = \text{INTRA\_ANGULAR54} \quad (234)$$

4.  $\text{IntraPredModeY}[\text{xCb}][\text{yCb}]$  is derived by applying the following procedure:

- If  $\text{intra\_luma\_mpm\_flag}[\text{xCb}][\text{yCb}]$  is equal to 1, the  $\text{IntraPredModeY}[\text{xCb}][\text{yCb}]$  is set equal to  $\text{candModeList}[\text{intra\_luma\_mpm\_idx}[\text{xCb}][\text{yCb}]]$ .
- Otherwise,  $\text{IntraPredModeY}[\text{xCb}][\text{yCb}]$  is derived by applying the following ordered steps:

1. When  $\text{candModeList}[i]$  is greater than  $\text{candModeList}[j]$  for  $i = 0..3$  and for each  $i$ ,  $j = (i + 1)..4$ , both values are swapped as follows:

$$(\text{candModeList}[i], \text{candModeList}[j]) = \text{Swap}(\text{candModeList}[i], \text{candModeList}[j]) \quad (235)$$

2.  $\text{IntraPredModeY}[\text{xCb}][\text{yCb}]$  is derived by the following ordered steps:

- i.  $\text{IntraPredModeY}[\text{xCb}][\text{yCb}]$  is set equal to  $\text{intra\_luma\_mpm\_remainder}[\text{xCb}][\text{yCb}]$ .
- ii. The value of  $\text{IntraPredModeY}[\text{xCb}][\text{yCb}]$  is incremented by one.
- iii. For  $i$  equal to 0 to 4, inclusive, when  $\text{IntraPredModeY}[\text{xCb}][\text{yCb}]$  is greater than or equal to  $\text{candModeList}[i]$ , the value of  $\text{IntraPredModeY}[\text{xCb}][\text{yCb}]$  is incremented by one.

The variable  $\text{IntraPredModeY}[x][y]$  with  $x = \text{xCb}.. \text{xCb} + \text{cbWidth} - 1$  and  $y = \text{yCb}.. \text{yCb} + \text{cbHeight} - 1$  is set to be equal to  $\text{IntraPredModeY}[\text{xCb}][\text{yCb}]$ .

### 8.4.3 Derivation process for chroma intra prediction mode

Input to this process are:

- a luma location (  $\text{xCb}, \text{yCb}$  ) specifying the top-left sample of the current chroma coding block relative to the top-left luma sample of the current picture,
- a variable  $\text{cbWidth}$  specifying the width of the current coding block in luma samples,
- a variable  $\text{cbHeight}$  specifying the height of the current coding block in luma samples,
- a variable  $\text{treeType}$  specifying whether a single or a dual tree is used.

In this process, the chroma intra prediction mode  $\text{IntraPredModeC}[\text{xCb}][\text{yCb}]$  and the MIP chroma direct mode flag  $\text{MipChromaDirectFlag}[\text{xCb}][\text{yCb}]$  are derived.

If  $\text{treeType}$  is equal to  $\text{SINGLE\_TREE}$ ,  $\text{sps\_chroma\_format\_idc}$  is equal to 3,  $\text{intra\_chroma\_pred\_mode}$  is equal to 4, and  $\text{IntraMipFlag}[\text{xCb}][\text{yCb}]$  is equal to 1, the following applies:

- The MIP chroma direct mode flag  $\text{MipChromaDirectFlag}[\text{xCb}][\text{yCb}]$  is set equal to 1.

— The chroma intra prediction mode  $\text{IntraPredModeC}[\text{xCb}][\text{yCb}]$  is set equal to  $\text{IntraPredModeY}[\text{xCb}][\text{yCb}]$ .

Otherwise, the following applies:

- The MIP chroma direct mode flag  $\text{MipChromaDirectFlag}[\text{xCb}][\text{yCb}]$  is set equal to 0.
- The corresponding luma intra prediction mode  $\text{lumaIntraPredMode}$  is derived as follows:
  - If  $\text{IntraMipFlag}[\text{xCb} + \text{cbWidth} / 2][\text{yCb} + \text{cbHeight} / 2]$  is equal to 1,  $\text{lumaIntraPredMode}$  is set equal to INTRA\_PLANAR.
  - Otherwise, if  $\text{CuPredMode}[0][\text{xCb} + \text{cbWidth} / 2][\text{yCb} + \text{cbHeight} / 2]$  is equal to MODE\_IBC or MODE\_PLT,  $\text{lumaIntraPredMode}$  is set equal to INTRA\_DC.
  - Otherwise,  $\text{lumaIntraPredMode}$  is set equal to  $\text{IntraPredModeY}[\text{xCb} + \text{cbWidth} / 2][\text{yCb} + \text{cbHeight} / 2]$ .
- The chroma intra prediction mode  $\text{IntraPredModeC}[\text{xCb}][\text{yCb}]$  is derived as follows:
  - If  $\text{cu\_act\_enabled\_flag}[\text{xCb}][\text{yCb}]$  is equal to 1, the chroma intra prediction mode  $\text{IntraPredModeC}[\text{xCb}][\text{yCb}]$  is set equal to  $\text{lumaIntraPredMode}$ .
  - Otherwise, if  $\text{BdpcmFlag}[\text{xCb}][\text{yCb}][1]$  is equal to 1,  $\text{IntraPredModeC}[\text{xCb}][\text{yCb}]$  is set equal to  $\text{BdpcmDir}[\text{xCb}][\text{yCb}][1] ? \text{INTRA\_ANGULAR50} : \text{INTRA\_ANGULAR18}$ .
  - Otherwise (  $\text{cu\_act\_enabled\_flag}[\text{xCb}][\text{yCb}]$  is equal to 0 and  $\text{BdpcmFlag}[\text{xCb}][\text{yCb}][1]$  is equal to 0 ), the chroma intra prediction mode  $\text{IntraPredModeC}[\text{xCb}][\text{yCb}]$  is derived using  $\text{cclm\_mode\_flag}$ ,  $\text{cclm\_mode\_idx}$ ,  $\text{intra\_chroma\_pred\_mode}$  and  $\text{lumaIntraPredMode}$  as specified in Table 20.

**Table 20 — Specification of  $\text{IntraPredModeC}[\text{xCb}][\text{yCb}]$  depending on  $\text{cclm\_mode\_flag}$ ,  $\text{cclm\_mode\_idx}$ ,  $\text{intra\_chroma\_pred\_mode}$  and  $\text{lumaIntraPredMode}$**

| cclm_mode_flag | cclm_mode_idx | intra_chroma_pred_mode | lumaIntraPredMode |    |    |    |                       |
|----------------|---------------|------------------------|-------------------|----|----|----|-----------------------|
|                |               |                        | 0                 | 50 | 18 | 1  | X<br>( 0 <= 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  $\text{sps\_chroma\_format\_idc}$  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**

|               |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|---------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| <b>mode X</b> | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| <b>mode Y</b> | 0  | 1  | 61 | 62 | 63 | 64 | 65 | 66 | 2  | 3  | 5  | 6  | 8  | 10 | 12 | 13 | 14 | 16 |
| <b>mode X</b> | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 |
| <b>mode Y</b> | 18 | 20 | 22 | 23 | 24 | 26 | 28 | 30 | 31 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 |
| <b>mode X</b> | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 |
| <b>mode Y</b> | 41 | 42 | 43 | 43 | 44 | 44 | 45 | 45 | 46 | 47 | 48 | 48 | 49 | 49 | 50 | 51 | 51 | 52 |
| <b>mode X</b> | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 |    |    |    |    |    |
| <b>mode Y</b> | 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\_qtbt\_dual\_tree\_intra\_flag is equal to 0.
  - sh\_slice\_type is not equal to 1.
  - CtbLog2SizeY is less than 6.
- Otherwise, the following applies:
  - The variables xCb64, yCb64, yCb32 are derived as follows:

$$xCb64 = (xCb \gg 6) \ll 6 \quad (236)$$

$$yCb64 = (yCb \gg 6) \ll 6 \quad (237)$$

$$yCb32 = (yCb \gg 5) \ll 5 \quad (238)$$

- 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.

- $\text{CqtDepth}[1][\text{xCb64}][\text{yCb64}]$  is greater than  $\text{CtbLog2SizeY} - 6$ .
- $\text{CqtDepth}[1][\text{xCb64}][\text{yCb64}]$  is equal to  $\text{CtbLog2SizeY} - 6$ ,  $\text{MttSplitMode}[\text{xCb64}][\text{yCb64}][0]$  is equal to  $\text{SPLIT\_BT\_HOR}$ , and  $\text{MttSplitMode}[\text{xCb64}][\text{yCb32}][1]$  is equal to  $\text{SPLIT\_BT\_VER}$ .
- Otherwise, the variable  $\text{CclmEnabled}$  is set equal to 0.
- When  $\text{CclmEnabled}$  is equal to 1 and one of the following conditions is true,  $\text{CclmEnabled}$  is set equal to 0:
  - $\text{CbWidth}[0][\text{xCb64}][\text{yCb64}]$  and  $\text{CbHeight}[0][\text{xCb64}][\text{yCb64}]$  are both equal to 64, and  $\text{IntraSubPartitionsModeFlag}[\text{xCb64}][\text{yCb64}]$  is equal to 1.
  - $\text{CbWidth}[0][\text{xCb64}][\text{yCb64}]$  or  $\text{CbHeight}[0][\text{xCb64}][\text{yCb64}]$  is less than 64, and  $\text{CqtDepth}[0][\text{xCb64}][\text{yCb64}]$  is equal to  $\text{CtbLog2SizeY} - 6$ .

## 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  $(\text{xTb0}, \text{yTb0})$  specifying the top-left sample of the current transform block relative to the top-left sample of the current picture,
- a variable  $\text{nCbW}$  specifying the width of the current coding block,
- a variable  $\text{nCbH}$  specifying the height of the current coding block,
- a variable  $\text{nTbW}$  specifying the width of the current transform block,
- a variable  $\text{nTbH}$  specifying the height of the current transform block,
- a variable  $\text{predModeIntra}$  specifying the intra prediction mode,
- a variable  $\text{cIdx}$  specifying the colour component of the current block,
- a variable  $\text{controlPara}$  specifying the output of the process,
- when  $\text{controlPara}$  is equal to 2, an array of residual samples  $\text{resSamplesRec}$  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  $\text{controlPara}$  is not equal to 1 or a residual sample array when  $\text{controlPara}$  is equal to 1.

The maximum transform block width  $\text{maxTbWidth}$  and height  $\text{maxTbHeight}$  are derived as follows:

$$\text{maxTbWidth} = (\text{cIdx} == 0) ? \text{MaxTbSizeY} : \text{MaxTbSizeY} / \text{SubWidthC} \quad (239)$$

$$\text{maxTbHeight} = (\text{cIdx} == 0) ? \text{MaxTbSizeY} : \text{MaxTbSizeY} / \text{SubHeightC} \quad (240)$$

The luma sample location is derived as follows:

$$(\text{xTbY}, \text{yTbY}) = (\text{cIdx} == 0) ? (\text{xTb0}, \text{yTb0}) : (\text{xTb0} * \text{SubWidthC}, \text{yTb0} * \text{SubHeightC}) \quad (241)$$

Depending on  $\text{maxTbSize}$ , the following applies:

- If  $\text{IntraSubPartitionsSplitType}$  is equal to  $\text{ISP\_NO\_SPLIT}$  and  $\text{nTbW}$  is greater than  $\text{maxTbWidth}$  or  $\text{nTbH}$  is greater than  $\text{maxTbHeight}$ , the following ordered steps apply.

1. The variables `verSplitFirst`, `newTbW`, and `newTbH` are derived as follows:

$$\text{verSplitFirst} = ( \text{nTbW} * ( \text{cIdx} == 0 ? 1 : \text{SubWidthC} ) > \text{nTbH} * ( \text{cIdx} == 0 ? 1 : \text{SubHeightC} ) ) \&\& ( \text{nTbW} > \text{maxTbWidth} ) \quad (242)$$

$$\text{newTbW} = \text{verSplitFirst} ? ( \text{nTbW} / 2 ) : \text{nTbW} \quad (243)$$

$$\text{newTbH} = !\text{verSplitFirst} ? ( \text{nTbH} / 2 ) : \text{nTbH} \quad (244)$$

2. The general decoding process for intra blocks as specified in this subclause 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 subclause is invoked with the location  $(xTb0, yTb0)$  set equal to  $(xTb0 + \text{newTbW}, yTb0)$ , 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 subclause is invoked with the location  $(xTb0, yTb0)$  set equal to  $(xTb0, yTb0 + \text{newTbH})$ , 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:

$$\text{nW} = ( \text{cIdx} == 0 \&\& \text{IntraSubPartitionsSplitType} == \text{ISP\_VER\_SPLIT} ) ? \text{nTbW} / \text{NumIntraSubPartitions} : \text{nTbW} \quad (245)$$

$$\text{nH} = ( \text{cIdx} == 0 \&\& \text{IntraSubPartitionsSplitType} == \text{ISP\_HOR\_SPLIT} ) ? \text{nTbH} / \text{NumIntraSubPartitions} : \text{nTbH} \quad (246)$$

$$\text{xPartInc} = ( \text{cIdx} == 0 \&\& \text{IntraSubPartitionsSplitType} == \text{ISP\_VER\_SPLIT} ) ? 1 : 0 \quad (247)$$

$$\text{yPartInc} = ( \text{cIdx} == 0 \&\& \text{IntraSubPartitionsSplitType} == \text{ISP\_HOR\_SPLIT} ) ? 1 : 0 \quad (248)$$

$$\text{nPbW} = \text{Max}( 4, \text{nW} ) \quad (249)$$

$$\text{pbFactor} = \text{nPbW} / \text{nW} \quad (250)$$

$$\text{numPartitions} = ( \text{cIdx} == 0 ) ? \text{NumIntraSubPartitions} : 1 \quad (251)$$

— For  $i = 0.. \text{numPartitions} - 1$ , the following applies:

1. The variables `xPartIdx`, `yPartIdx`, and `xPartPbIdx` are derived as follows:

$$\text{xPartIdx} = i * \text{xPartInc} \quad (252)$$

$$yPartIdx = i * yPartInc \quad (253)$$

$$xPartPbIdx = xPartIdx \% pbFactor \quad (254)$$

2. When controlPara is not equal to 1 and xPartPbIdx is equal to 0, the intra sample prediction process as specified in subclause 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 predModelIntra, 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 cldx 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 subclause 8.7.2 is invoked with the luma location ( xTbY, yTbY ) set equal to ( xTbY + nW \* xPartIdx, yTbY + nH \* yPartIdx ), the variable cldx, 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 subclause 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 cldx, 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 predModelIntra 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 cldx 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 cldx is equal to 0, or if MipChromaDirectFlag[ xTbCmp ][ yTbCmp ] is equal to 1 and cldx is not equal to 0, the matrix-based intra sample prediction process as specified in subclause 8.4.5.2.2 is invoked with the location

( $x_{TbCmp}$ ,  $y_{TbCmp}$ ), the intra prediction mode  $predModelIntra$ , the transform block width  $nTbW$  and height  $nTbH$ , and the variable  $cldx$  as inputs, and the output is  $predSamples$ .

- Otherwise, the general intra sample prediction process as specified in subclause 8.4.5.2.6 is invoked with the location ( $x_{TbCmp}$ ,  $y_{TbCmp}$ ), the intra prediction mode  $predModelIntra$ , the transform block width  $nTbW$  and height  $nTbH$ , the coding block width  $nCbW$  and height  $nCbH$ , and the variable  $cldx$  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 ( $x_{TbCmp}$ ,  $y_{TbCmp}$ ) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture,
- a variable  $predModelIntra$  specifying the intra prediction mode,
- a variable  $nTbW$  specifying the transform block width,
- a variable  $nTbH$  specifying the transform block height,
- a variable  $cldx$  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:

$$isTransposed = intra\_mip\_transposed\_flag[x_{TbCmp}][y_{TbCmp}] \quad (255)$$

The variables  $inSize$ , variables  $refW$  and  $refH$  are derived as follows:

$$inSize = (2 * boundarySize) - ((mipSizeId == 2) ? 1 : 0) \quad (256)$$

$$refW = nTbW + 1 \quad (257)$$

$$refH = nTbH + 1 \quad (258)$$

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 subclause 8.4.5.2.8 is invoked with the sample location (  $x_{TbCmp}$ ,  $y_{TbCmp}$  ), reference line index equal to 0, the reference sample width  $refW$ , the reference sample height  $refH$ , and the colour component index  $cldx$  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 subclause 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  $cldx$  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:

$$refT[x] = refUnfilt[x][-1] \quad (259)$$

$$refL[y] = refUnfilt[-1][y] \quad (260)$$

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 subclause 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 subclause 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:

$$p[x] = pTemp[x + 1] - pTemp[0] \quad (261)$$

- Otherwise ( $mipSizeId$  is less than 2), the following applies:

$$\begin{aligned} p[0] &= (1 \ll (BitDepth - 1)) - pTemp[0] \\ p[x] &= pTemp[x] - pTemp[0] \text{ for } x = 1..inSize - 1 \end{aligned} \quad (262)$$

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  $modelId$  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 subclause 8.4.5.2.4 with  $mipSizeId$  and  $modelId$  as inputs.



- The matrix-based intra prediction samples  $\text{predMip}[x][y]$ , with  $x = 0..\text{predSize} - 1$ ,  $y = 0..\text{predSize} - 1$  are derived as follows:

$$\text{oW} = 32 - 32 * (\sum_{i=0}^{\text{inSize}-1} p[i]) \quad (263)$$

$$\text{predMip}[x][y] = ((\sum_{i=0}^{\text{inSize}-1} \text{mWeight}[i][y * \text{predSize} + x] * p[i]) + \text{oW}) >> 6) + \text{pTemp}[0] \quad (264)$$

2. The matrix-based intra prediction samples  $\text{predMip}[x][y]$ , with  $x = 0..\text{predSize} - 1$ ,  $y = 0..\text{predSize} - 1$  are clipped as follows:

$$\text{predMip}[x][y] = \text{Clip1}(\text{predMip}[x][y]) \quad (265)$$

3. When  $\text{isTransposed}$  is equal to TRUE, the  $\text{predSize} \times \text{predSize}$  array  $\text{predMip}[x][y]$  with  $x = 0..\text{predSize} - 1$ ,  $y = 0..\text{predSize} - 1$  is transposed as follows:

$$\text{predTemp}[y][x] = \text{predMip}[x][y] \quad (266)$$

$$\text{predMip} = \text{predTemp} \quad (267)$$

4. The predicted samples  $\text{predSamples}[x][y]$ , with  $x = 0..\text{nTbW} - 1$ ,  $y = 0..\text{nTbH} - 1$  are derived as follows:

- If  $\text{nTbW}$  is greater than  $\text{predSize}$  or  $\text{nTbH}$  is greater than  $\text{predSize}$ , the MIP prediction upsampling process as specified in subclause 8.4.5.2.5 is invoked with the input block size  $\text{predSize}$ , matrix-based intra prediction samples  $\text{predMip}[x][y]$  with  $x = 0..\text{predSize} - 1$ ,  $y = 0..\text{predSize} - 1$ , the transform block width  $\text{nTbW}$ , the transform block height  $\text{nTbH}$ , the top reference samples  $\text{refT}[x]$  with  $x = 0..\text{nTbW} - 1$ , and the left reference samples  $\text{refL}[y]$  with  $y = 0..\text{nTbH} - 1$  as inputs, and the output is the predicted sample array  $\text{predSamples}$ .

- Otherwise,  $\text{predSamples}[x][y]$ , with  $x = 0..\text{nTbW} - 1$ ,  $y = 0..\text{nTbH} - 1$  is set equal to  $\text{predMip}[x][y]$ .

#### 8.4.5.2.3 MIP boundary sample downsampling process

Inputs to this process are:

- a variable  $\text{nTbS}$  specifying the transform block size,
- reference samples  $\text{refS}[x]$  with  $x = 0..\text{nTbS} - 1$ ,
- a variable  $\text{boundarySize}$  specifying the downsampled boundary size.

Outputs of this process are the reduced boundary samples  $\text{redS}[x]$  with  $x = 0..\text{boundarySize} - 1$ .

The reduced boundary samples  $\text{redS}[x]$  with  $x = 0..\text{boundarySize} - 1$  are derived as follows:

- If  $\text{boundarySize}$  is less than  $\text{nTbS}$ , the following applies:

$$\text{bDwn} = \text{nTbS} / \text{boundarySize} \quad (268)$$

$$\text{redS}[x] = (\sum_{i=0}^{\text{bDwn}-1} \text{refS}[x * \text{bDwn} + i] + (1 \ll (\text{Log2}(\text{bDwn}) - 1))) \gg \text{Log2}(\text{bDwn}) \quad (269)$$

- Otherwise ( $\text{boundarySize}$  is equal to  $\text{nTbS}$ ),  $\text{redS}[x]$  is set equal to  $\text{refS}[x]$ .

#### 8.4.5.2.4 MIP weight matrix derivation process

Inputs to this process are:

- a variable  $\text{mipSizeId}$ ,
- a variable  $\text{modelId}$ .

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  $modelId$  as follows:

— If  $mipSizeId$  is equal to 0 and  $modelId$  is equal to 0, the following applies:

$$mWeight[x][y] = \begin{Bmatrix} \{ 32, 30, 90, 28 \}, \{ 32, 32, 72, 28 \}, \{ 34, 77, 53, 30 \}, \{ 51, 124, 36, 37 \}, \\ \{ 31, 31, 95, 37 \}, \{ 33, 31, 70, 50 \}, \{ 52, 80, 25, 60 \}, \{ 78, 107, 1, 65 \}, \\ \{ 31, 29, 37, 95 \}, \{ 38, 34, 19, 101 \}, \{ 73, 85, 0, 81 \}, \{ 92, 99, 0, 65 \}, \\ \{ 34, 29, 14, 111 \}, \{ 48, 48, 7, 100 \}, \{ 80, 91, 0, 74 \}, \{ 89, 97, 0, 64 \} \end{Bmatrix}, \quad (270)$$

— Otherwise, if  $mipSizeId$  is equal to 0 and  $modelId$  is equal to 1, the following applies:

$$mWeight[x][y] = \begin{Bmatrix} \{ 31, 23, 34, 29 \}, \{ 31, 43, 34, 31 \}, \{ 30, 95, 34, 32 \}, \{ 29, 100, 35, 33 \}, \\ \{ 31, 23, 34, 29 \}, \{ 31, 43, 34, 31 \}, \{ 30, 95, 34, 32 \}, \{ 29, 99, 35, 33 \}, \\ \{ 31, 24, 35, 29 \}, \{ 31, 44, 34, 31 \}, \{ 30, 95, 35, 32 \}, \{ 29, 99, 35, 33 \}, \\ \{ 31, 24, 35, 30 \}, \{ 31, 44, 35, 31 \}, \{ 30, 95, 35, 32 \}, \{ 29, 99, 35, 33 \} \end{Bmatrix}, \quad (271)$$

— Otherwise, if  $mipSizeId$  is equal to 0 and  $modelId$  is equal to 2, the following applies:

$$mWeight[x][y] = \begin{Bmatrix} \{ 32, 32, 36, 58 \}, \{ 32, 29, 26, 66 \}, \{ 36, 87, 23, 61 \}, \{ 79, 84, 3, 37 \}, \\ \{ 32, 32, 30, 69 \}, \{ 33, 29, 24, 71 \}, \{ 44, 16, 21, 70 \}, \{ 96, 18, 0, 57 \}, \\ \{ 32, 31, 24, 74 \}, \{ 33, 30, 23, 71 \}, \{ 36, 24, 24, 71 \}, \{ 59, 9, 16, 68 \}, \\ \{ 32, 32, 23, 75 \}, \{ 33, 30, 24, 70 \}, \{ 32, 30, 25, 71 \}, \{ 36, 26, 25, 70 \} \end{Bmatrix}, \quad (272)$$

— Otherwise, if  $mipSizeId$  is equal to 0 and  $modelId$  is equal to 3, the following applies:

$$mWeight[x][y] = \begin{Bmatrix} \{ 32, 33, 34, 32 \}, \{ 32, 30, 22, 38 \}, \{ 29, 46, 25, 38 \}, \{ 53, 123, 28, 22 \}, \\ \{ 32, 33, 30, 37 \}, \{ 32, 30, 21, 38 \}, \{ 32, 40, 24, 38 \}, \{ 64, 116, 26, 17 \}, \\ \{ 32, 32, 23, 49 \}, \{ 32, 30, 21, 39 \}, \{ 34, 39, 24, 37 \}, \{ 72, 109, 23, 16 \}, \\ \{ 33, 31, 17, 60 \}, \{ 32, 31, 21, 39 \}, \{ 35, 41, 24, 37 \}, \{ 72, 106, 22, 18 \} \end{Bmatrix}, \quad (273)$$

— Otherwise, if  $mipSizeId$  is equal to 0 and  $modelId$  is equal to 4, the following applies:

$$mWeight[x][y] = \begin{Bmatrix} \{ 34, 25, 89, 20 \}, \{ 38, 32, 47, 24 \}, \{ 40, 86, 29, 27 \}, \{ 38, 98, 32, 29 \}, \\ \{ 34, 31, 94, 40 \}, \{ 44, 25, 83, 27 \}, \{ 54, 72, 43, 16 \}, \{ 47, 94, 33, 22 \}, \\ \{ 33, 31, 36, 94 \}, \{ 43, 23, 51, 76 \}, \{ 62, 55, 64, 25 \}, \{ 57, 89, 38, 15 \}, \\ \{ 32, 32, 28, 101 \}, \{ 38, 26, 33, 94 \}, \{ 55, 38, 68, 47 \}, \{ 59, 80, 52, 16 \} \end{Bmatrix}, \quad (274)$$

— Otherwise, if  $mipSizeId$  is equal to 0 and  $modelId$  is equal to 5, the following applies:

$$mWeight[x][y] = \begin{Bmatrix} \{ 28, 30, 68, 29 \}, \{ 23, 48, 23, 48 \}, \{ 39, 98, 16, 42 \}, \{ 84, 86, 20, 17 \}, \\ \{ 25, 31, 52, 74 \}, \{ 38, 68, 5, 70 \}, \{ 95, 78, 7, 21 \}, \{ 127, 54, 12, 0 \}, \\ \{ 30, 47, 14, 107 \}, \{ 79, 76, 0, 53 \}, \{ 127, 59, 7, 1 \}, \{ 127, 51, 9, 0 \}, \\ \{ 50, 71, 1, 96 \}, \{ 109, 69, 7, 25 \}, \{ 127, 56, 9, 0 \}, \{ 123, 53, 13, 0 \} \end{Bmatrix}, \quad (275)$$

- Otherwise, if mipSizeId is equal to 0 and modeId is equal to 6, the following applies:

$$\text{mWeight}[x][y] = \begin{Bmatrix} \{ 40, 20, 72, 18 \}, \{ 48, 29, 44, 18 \}, \{ 53, 81, 35, 18 \}, \{ 48, 96, 33, 22 \}, \\ \{ 45, 23, 79, 49 \}, \{ 61, 21, 56, 49 \}, \{ 72, 52, 32, 48 \}, \{ 65, 69, 20, 50 \}, \\ \{ 41, 27, 29, 96 \}, \{ 49, 22, 28, 94 \}, \{ 52, 22, 28, 93 \}, \{ 49, 27, 27, 92 \}, \\ \{ 37, 29, 26, 98 \}, \{ 39, 28, 28, 97 \}, \{ 38, 28, 30, 97 \}, \{ 38, 29, 30, 95 \} \end{Bmatrix}, \quad (276)$$

- Otherwise, if mipSizeId is equal to 0 and modeId is equal to 7, the following applies:

$$\text{mWeight}[x][y] = \begin{Bmatrix} \{ 33, 27, 43, 27 \}, \{ 32, 29, 31, 31 \}, \{ 31, 73, 33, 31 \}, \{ 35, 104, 34, 28 \}, \\ \{ 32, 30, 63, 22 \}, \{ 33, 26, 33, 29 \}, \{ 33, 57, 33, 30 \}, \{ 37, 100, 35, 27 \}, \\ \{ 32, 31, 85, 25 \}, \{ 34, 25, 39, 25 \}, \{ 35, 39, 32, 28 \}, \{ 40, 91, 35, 25 \}, \\ \{ 32, 30, 77, 50 \}, \{ 34, 26, 54, 22 \}, \{ 37, 31, 34, 27 \}, \{ 45, 75, 34, 23 \} \end{Bmatrix}, \quad (277)$$

- Otherwise, if mipSizeId is equal to 0 and modeId is equal to 8, the following applies:

$$\text{mWeight}[x][y] = \begin{Bmatrix} \{ 34, 25, 77, 19 \}, \{ 36, 34, 56, 24 \}, \{ 41, 83, 39, 30 \}, \{ 47, 96, 28, 35 \}, \\ \{ 34, 31, 70, 65 \}, \{ 38, 29, 53, 77 \}, \{ 43, 36, 37, 83 \}, \{ 48, 39, 28, 83 \}, \\ \{ 33, 31, 31, 98 \}, \{ 33, 31, 30, 99 \}, \{ 34, 30, 31, 98 \}, \{ 36, 29, 31, 96 \}, \\ \{ 32, 32, 30, 97 \}, \{ 32, 32, 31, 96 \}, \{ 31, 33, 33, 96 \}, \{ 32, 33, 34, 94 \} \end{Bmatrix}, \quad (278)$$

- Otherwise, if mipSizeId is equal to 0 and modeId is equal to 9, the following applies:

$$\text{mWeight}[x][y] = \begin{Bmatrix} \{ 30, 30, 93, 19 \}, \{ 31, 59, 67, 34 \}, \{ 31, 79, 36, 59 \}, \{ 30, 67, 17, 79 \}, \\ \{ 30, 38, 68, 69 \}, \{ 29, 40, 43, 91 \}, \{ 26, 35, 32, 101 \}, \{ 23, 32, 30, 101 \}, \\ \{ 26, 34, 30, 101 \}, \{ 23, 36, 30, 102 \}, \{ 20, 32, 31, 102 \}, \{ 18, 33, 32, 102 \}, \\ \{ 23, 33, 31, 100 \}, \{ 20, 34, 32, 100 \}, \{ 18, 35, 33, 100 \}, \{ 18, 35, 33, 100 \} \end{Bmatrix}, \quad (279)$$

- Otherwise, if mipSizeId is equal to 0 and modeId is equal to 10, the following applies:

$$\text{mWeight}[x][y] = \begin{Bmatrix} \{ 31, 54, 90, 26 \}, \{ 32, 60, 53, 61 \}, \{ 34, 49, 37, 84 \}, \{ 34, 39, 35, 89 \}, \\ \{ 35, 38, 41, 88 \}, \{ 35, 35, 32, 96 \}, \{ 35, 31, 33, 96 \}, \{ 35, 32, 35, 94 \}, \\ \{ 34, 34, 30, 97 \}, \{ 35, 32, 33, 95 \}, \{ 35, 32, 34, 94 \}, \{ 35, 34, 34, 93 \}, \\ \{ 34, 34, 34, 93 \}, \{ 35, 34, 34, 93 \}, \{ 35, 34, 34, 92 \}, \{ 36, 34, 35, 91 \} \end{Bmatrix}, \quad (280)$$

- Otherwise, if mipSizeId is equal to 0 and modeId is equal to 11, the following applies:

$$\text{mWeight}[x][y] = \begin{Bmatrix} \{ 32, 29, 54, 24 \}, \{ 31, 32, 34, 29 \}, \{ 31, 43, 34, 29 \}, \{ 32, 67, 36, 28 \}, \\ \{ 31, 34, 69, 37 \}, \{ 31, 35, 46, 33 \}, \{ 30, 35, 39, 33 \}, \{ 30, 42, 39, 36 \}, \\ \{ 31, 35, 39, 88 \}, \{ 30, 38, 41, 84 \}, \{ 30, 39, 40, 81 \}, \{ 39, 46, 38, 78 \}, \\ \{ 31, 36, 34, 96 \}, \{ 34, 38, 37, 93 \}, \{ 55, 42, 38, 82 \}, \{ 89, 53, 38, 65 \} \end{Bmatrix}, \quad (281)$$

- Otherwise, if mipSizeId is equal to 0 and modeId is equal to 12, the following applies:

$$\text{mWeight}[x][y] = \begin{Bmatrix} \{ 32, 33, 43, 29 \}, \{ 32, 30, 29, 33 \}, \{ 31, 47, 31, 33 \}, \{ 33, 100, 31, 31 \}, \\ \{ 32, 33, 74, 25 \}, \{ 32, 32, 34, 31 \}, \{ 32, 33, 30, 33 \}, \{ 32, 68, 30, 32 \}, \\ \{ 32, 31, 91, 40 \}, \{ 32, 32, 58, 26 \}, \{ 31, 31, 30, 32 \}, \{ 31, 42, 30, 33 \}, \\ \{ 32, 31, 49, 85 \}, \{ 32, 31, 83, 35 \}, \{ 31, 33, 48, 29 \}, \{ 31, 36, 32, 33 \} \end{Bmatrix}, \quad (282)$$

— Otherwise, if mipSizeId is equal to 0 and modelId is equal to 13, the following applies:

$$\text{mWeight}[x][y] = \begin{Bmatrix} \{ 31, 29, 81, 35 \}, \{ 32, 28, 34, 50 \}, \{ 31, 75, 16, 43 \}, \{ 34, 103, 29, 32 \}, \\ \{ 32, 32, 53, 78 \}, \{ 31, 28, 36, 88 \}, \{ 30, 52, 18, 73 \}, \{ 52, 88, 17, 35 \}, \\ \{ 32, 32, 35, 94 \}, \{ 30, 31, 35, 95 \}, \{ 36, 29, 31, 92 \}, \{ 100, 43, 16, 40 \}, \\ \{ 32, 32, 35, 93 \}, \{ 30, 32, 38, 93 \}, \{ 55, 18, 37, 83 \}, \{ 127, 0, 30, 40 \} \end{Bmatrix}, \quad (283)$$

— Otherwise, if mipSizeId is equal to 0 and modelId is equal to 14, the following applies:

$$\text{mWeight}[x][y] = \begin{Bmatrix} \{ 31, 22, 47, 30 \}, \{ 31, 48, 25, 34 \}, \{ 30, 95, 31, 32 \}, \{ 32, 103, 33, 32 \}, \\ \{ 30, 24, 57, 31 \}, \{ 30, 47, 26, 34 \}, \{ 31, 95, 31, 32 \}, \{ 43, 97, 35, 25 \}, \\ \{ 29, 26, 44, 63 \}, \{ 37, 38, 24, 47 \}, \{ 74, 63, 28, 20 \}, \{ 110, 58, 34, 3 \}, \\ \{ 46, 22, 5, 108 \}, \{ 93, 5, 9, 77 \}, \{ 127, 0, 17, 52 \}, \{ 127, 0, 15, 50 \} \end{Bmatrix}, \quad (284)$$

— Otherwise, if mipSizeId is equal to 0 and modelId is equal to 15, the following applies:

$$\text{mWeight}[x][y] = \begin{Bmatrix} \{ 32, 27, 68, 24 \}, \{ 35, 23, 35, 28 \}, \{ 35, 64, 29, 29 \}, \{ 37, 104, 33, 28 \}, \\ \{ 32, 32, 91, 40 \}, \{ 36, 23, 67, 36 \}, \{ 49, 23, 39, 28 \}, \{ 60, 67, 30, 20 \}, \\ \{ 32, 32, 36, 95 \}, \{ 35, 29, 38, 93 \}, \{ 50, 16, 30, 84 \}, \{ 72, 16, 15, 65 \}, \\ \{ 32, 32, 27, 100 \}, \{ 33, 32, 29, 100 \}, \{ 37, 29, 30, 98 \}, \{ 48, 21, 29, 90 \} \end{Bmatrix}, \quad (285)$$

— Otherwise, if mipSizeId is equal to 1 and modelId is equal to 0, the following applies:

$$\text{mWeight}[x][y] = \begin{Bmatrix} \{ 30, 63, 46, 37, 25, 33, 33, 34 \}, \{ 30, 60, 66, 38, 32, 31, 32, 33 \}, \\ \{ 29, 45, 74, 42, 32, 32, 32, 33 \}, \{ 30, 39, 62, 58, 32, 33, 32, 33 \}, \\ \{ 30, 66, 55, 39, 32, 30, 30, 66 \}, \{ 29, 54, 69, 40, 33, 31, 31, 33 \}, \\ \{ 28, 48, 71, 43, 32, 33, 32, 33 \}, \{ 28, 41, 72, 46, 32, 34, 32, 33 \}, \\ \{ 30, 66, 56, 40, 32, 33, 28, 33 \}, \{ 29, 55, 69, 39, 33, 33, 30, 32 \}, \\ \{ 27, 46, 72, 43, 33, 33, 32, 33 \}, \{ 27, 42, 69, 48, 32, 34, 32, 33 \}, \\ \{ 30, 63, 55, 40, 32, 33, 35, 30 \}, \{ 29, 56, 66, 40, 33, 33, 33, 30 \}, \\ \{ 27, 47, 69, 44, 33, 33, 33, 32 \}, \{ 27, 42, 65, 50, 32, 34, 32, 33 \} \end{Bmatrix}, \quad (286)$$

— Otherwise, if mipSizeId is equal to 1 and modelId is equal to 1, the following applies:

$$\text{mWeight}[x][y] = \begin{Bmatrix} \{ 32, 33, 30, 31, 74, 30, 31, 32 \}, \{ 33, 56, 28, 30, 41, 29, 32, 32 \}, \\ \{ 33, 77, 52, 26, 29, 34, 30, 32 \}, \{ 33, 37, 80, 41, 31, 34, 30, 32 \}, \\ \{ 32, 32, 33, 31, 59, 76, 28, 31 \}, \{ 33, 31, 31, 30, 78, 40, 28, 32 \}, \\ \{ 33, 47, 28, 29, 53, 27, 31, 31 \}, \{ 33, 61, 44, 28, 34, 32, 31, 31 \}, \\ \{ 32, 31, 34, 30, 26, 64, 76, 27 \}, \{ 32, 31, 34, 29, 45, 86, 36, 29 \}, \\ \{ 33, 27, 34, 29, 73, 55, 25, 32 \}, \{ 33, 33, 34, 30, 62, 33, 30, 31 \}, \\ \{ 32, 31, 34, 30, 30, 29, 58, 74 \}, \{ 32, 31, 35, 29, 27, 53, 77, 35 \}, \\ \{ 32, 30, 36, 29, 40, 80, 44, 31 \}, \{ 33, 28, 37, 30, 58, 60, 31, 33 \} \end{Bmatrix}, \quad (287)$$

— Otherwise, if mipSizeId is equal to 1 and modelId is equal to 2, the following applies:

$$\text{mWeight}[x][y] = \begin{Bmatrix} \{ 32, 51, 27, 32, 27, 50, 29, 32 \}, \{ 32, 95, 42, 29, 29, 42, 30, 32 \}, \\ \{ 32, 27, 99, 34, 31, 41, 29, 32 \}, \{ 32, 34, 21, 104, 31, 42, 30, 32 \}, \\ \{ 32, 45, 30, 32, 9, 88, 40, 30 \}, \{ 32, 77, 38, 30, 9, 76, 38, 30 \}, \\ \{ 32, 38, 78, 33, 14, 67, 37, 30 \}, \{ 32, 30, 87, 20, 59, 38, 31 \}, \\ \{ 33, 37, 32, 32, 27, 18, 106, 34 \}, \{ 34, 44, 34, 31, 25, 17, 108, 31 \}, \\ \{ 36, 39, 45, 31, 24, 15, 108, 30 \}, \{ 37, 31, 31, 54, 25, 14, 101, 32 \}, \\ \{ 36, 33, 32, 30, 29, 37, 13, 110 \}, \{ 39, 32, 32, 29, 27, 37, 15, 108 \}, \\ \{ 44, 33, 31, 27, 25, 37, 16, 106 \}, \{ 47, 30, 31, 32, 25, 34, 19, 102 \} \end{Bmatrix}, \quad (288)$$

— Otherwise, if mipSizeId is equal to 1 and modeId is equal to 3, the following applies:

$$\text{mWeight}[x][y] = \quad (289)$$

```
{
  { 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:

$$\text{mWeight}[x][y] = \quad (290)$$

```
{
  { 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, 86, 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:

$$\text{mWeight}[x][y] = \quad (291)$$

```
{
  { 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:

$$\text{mWeight}[x][y] = \quad (292)$$

```
{
  { 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:

$$\text{mWeight}[x][y] = \quad (293)$$

```
{
  { 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:

$$\text{mWeight}[x][y] = \quad (294)$$

```
{
  { 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 modelId 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 modelId 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 modelId 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 modelId 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 modelId 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, 33, 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:

$$upHor = nTbW / predSize \quad (300)$$

$$upVer = nTbH / predSize \quad (301)$$

$$predSamples[(x + 1) * upHor - 1][(y + 1) * upVer - 1] = predMip[x][y] \quad (302)$$

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:

$$sum = (upHor - dX) * predSamples[xHor][yHor] + dX * predSamples[xHor + upHor][yHor] \quad (303)$$

$$predSamples[xHor + dX][yHor] = (sum + upHor / 2) / upHor \quad (304)$$

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:

$$sum = (upVer - dY) * predSamples[xVer][yVer] + dY * predSamples[xVer][yVer + upVer] \quad (305)$$

$$predSamples[xVer][yVer + dY] = (sum + upVer / 2) / upVer \quad (306)$$

#### 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  $predModelIntra$  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  $\text{predSamples}[x][y]$ , with  $x = 0..nTbW - 1$ ,  $y = 0..nTbH - 1$ .

The variables  $\text{refW}$  and  $\text{refH}$  are derived as follows:

— If  $\text{IntraSubPartitionsSplitType}$  is equal to  $\text{ISP\_NO\_SPLIT}$  or  $\text{cldx}$  is not equal to 0, the following applies:

$$\text{refW} = nTbW * 2 \quad (307)$$

$$\text{refH} = nTbH * 2 \quad (308)$$

— Otherwise (  $\text{IntraSubPartitionsSplitType}$  is not equal to  $\text{ISP\_NO\_SPLIT}$  and  $\text{cldx}$  is equal to 0 ), the following applies:

$$\text{refW} = nCbW + nTbW \quad (309)$$

$$\text{refH} = nCbH + nTbH \quad (310)$$

The variable  $\text{refIdx}$  specifying the intra prediction reference line index is derived as follows:

$$\text{refIdx} = (\text{cldx} == 0) ? \text{IntraLumaRefLineIdx}[\text{xTbCmp}][\text{yTbCmp}] : 0 \quad (311)$$

The wide angle intra prediction mode mapping process as specified in subclause 8.4.5.2.7 is invoked with  $\text{predModeIntra}$ ,  $nCbW$ ,  $nCbH$ ,  $nTbW$ ,  $nTbH$  and  $\text{cldx}$  as inputs, and the modified  $\text{predModeIntra}$  as output.

The variable  $\text{refFilterFlag}$  is derived as follows:

— If  $\text{predModeIntra}$  is equal to 0, -14, -12, -10, -6, 2, 34, 66, 72, 76, 78, or 80,  $\text{refFilterFlag}$  is set equal to 1.

— Otherwise,  $\text{refFilterFlag}$  is set equal to 0.

For the generation of 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 following ordered steps apply:

1. The reference sample availability marking process as specified in subclause 8.4.5.2.8 is invoked with the sample location (  $\text{xTbCmp}$ ,  $\text{yTbCmp}$  ), the intra prediction reference line index  $\text{refIdx}$ , the reference sample width  $\text{refW}$ , the reference sample height  $\text{refH}$ , the colour component index  $\text{cldx}$  as inputs, and the reference samples  $\text{refUnfilt}[x][y]$  with  $x = -1 - \text{refIdx}$ ,  $y = -1 - \text{refIdx}..refH - 1$  and  $x = -\text{refIdx}..refW - 1$ ,  $y = -1 - \text{refIdx}$  as output.
2. When at least one sample  $\text{refUnfilt}[x][y]$  with  $x = -1 - \text{refIdx}$ ,  $y = -1 - \text{refIdx}..refH - 1$  and  $x = -\text{refIdx}..refW - 1$ ,  $y = -1 - \text{refIdx}$  is marked as "not available for intra prediction", the reference sample substitution process as specified in subclause 8.4.5.2.9 is invoked with the intra prediction reference line index  $\text{refIdx}$ , the reference sample width  $\text{refW}$ , the reference sample height  $\text{refH}$ , the reference samples  $\text{refUnfilt}[x][y]$  with  $x = -1 - \text{refIdx}$ ,  $y = -1 - \text{refIdx}..refH - 1$  and  $x = -\text{refIdx}..refW - 1$ ,  $y = -1 - \text{refIdx}$ , and the colour component index  $\text{cldx}$  as inputs, and the modified reference samples  $\text{refUnfilt}[x][y]$  with  $x = -1 - \text{refIdx}$ ,  $y = -1 - \text{refIdx}..refH - 1$  and  $x = -\text{refIdx}..refW - 1$ ,  $y = -1 - \text{refIdx}$  as output.
3. The reference sample filtering process as specified in subclause 8.4.5.2.10 is invoked with the intra prediction reference line index  $\text{refIdx}$ , the transform block width  $nTbW$  and height  $nTbH$ , the reference sample width  $\text{refW}$ , the reference sample height  $\text{refH}$ , the reference filter flag  $\text{refFilterFlag}$ , the unfiltered samples  $\text{refUnfilt}[x][y]$  with  $x = -1 - \text{refIdx}$ ,  $y = -1 - \text{refIdx}..refH - 1$  and  $x = -\text{refIdx}..refW - 1$ ,  $y = -1 - \text{refIdx}$ , and the colour component index  $\text{cldx}$  as inputs, and the reference samples  $p[x][y]$  with  $x = -1 - \text{refIdx}$ ,  $y = -1 - \text{refIdx}..refH - 1$  and  $x = -\text{refIdx}..refW - 1$ ,  $y = -1 - \text{refIdx}$  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 subclause 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 subclause 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 subclause 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 subclause 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 subclause 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 *cldx* 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 *cldx* 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 *cldx* is equal to 0 ), the following applies:

$$nW = nCbW \quad (314)$$

$$nH = nCbH \quad (315)$$

The variable *whRatio* is set equal to  $\text{Abs}(\text{Log2}(nW) - \text{Log2}(nH))$ .

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 ( *xBtCmp*, *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 *cldx* specifying the colour component of the current block.

Outputs of this process are the reference samples *refUnfilt[x][y]* with  $x = -1 - \text{refIdx}$ ,  $y = -1 - \text{refIdx} \cdot \text{refH} - 1$  and  $x = -\text{refIdx} \cdot \text{refW} - 1$ ,  $y = -1 - \text{refIdx}$  for intra sample prediction.

The  $\text{refW} + \text{refH} + 1 + (2 * \text{refIdx})$  neighbouring samples *refUnfilt[x][y]* that are constructed samples prior to the in-loop filter process, with  $x = -1 - \text{refIdx}$ ,  $y = -1 - \text{refIdx} \cdot \text{refH} - 1$  and  $x = -\text{refIdx} \cdot \text{refW} - 1$ ,  $y = -1 - \text{refIdx}$ , are derived as follows:



— The neighbouring location (  $x_{NbCmp}$ ,  $y_{NbCmp}$  ) is specified by:

$$(x_{NbCmp}, y_{NbCmp}) = (x_{TbCmp} + x, y_{TbCmp} + y) \quad (316)$$

— The current luma location (  $x_{TbY}$ ,  $y_{TbY}$  ) and the neighbouring luma location (  $x_{NbY}$ ,  $y_{NbY}$  ) are derived as follows:

$$(x_{TbY}, y_{TbY}) = (cIdx == 0) ? (x_{TbCmp}, y_{TbCmp}) : (x_{TbCmp} * SubWidthC, y_{TbCmp} * SubHeightC) \quad (317)$$

$$(x_{NbY}, y_{NbY}) = (cIdx == 0) ? (x_{NbCmp}, y_{NbCmp}) : (x_{NbCmp} * SubWidthC, y_{NbCmp} * SubHeightC) \quad (318)$$

— The derivation process for neighbouring block availability as specified in subclause 6.4.4 is invoked with the current luma location (  $x_{Curr}$ ,  $y_{Curr}$  ) set equal to (  $x_{TbY}$ ,  $y_{TbY}$  ), the neighbouring luma location (  $x_{NbY}$ ,  $y_{NbY}$  ),  $checkPredModeY$  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 (  $x_{NbCmp}$ ,  $y_{NbCmp}$  ) 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  $1 \ll (BitDepth - 1)$ .
- 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  $x = -1 - \text{refIdx}$ ,  $y = -(\text{refH} - 2) \dots 1 + \text{refIdx}$ , when  $\text{refUnfilt}[x][y]$  is marked as "not available for intra prediction", the value of  $\text{refUnfilt}[x][y]$  is set equal to the value of  $\text{refUnfilt}[x][y + 1]$ .
3. For  $x = -\text{refIdx} \dots \text{refW} - 1$ ,  $y = -1 - \text{refIdx}$ , when  $\text{refUnfilt}[x][y]$  is marked as "not available for intra prediction", the value of  $\text{refUnfilt}[x][y]$  is set equal to the value of  $\text{refUnfilt}[x - 1][y]$ .

All samples  $\text{refUnfilt}[x][y]$  with  $x = -1 - \text{refIdx}$ ,  $y = -1 - \text{refIdx} \dots \text{refH} - 1$  and  $x = -\text{refIdx} \dots \text{refW} - 1$ ,  $y = -1 - \text{refIdx}$  are marked as "available for intra prediction".

#### 8.4.5.2.10 Reference sample filtering process

Inputs to this process are:

- a variable  $\text{refIdx}$  specifying the intra prediction reference line index,
- a variable  $\text{nTbW}$  specifying the transform block width,
- a variable  $\text{nTbH}$  specifying the transform block height,
- a variable  $\text{refW}$  specifying the reference samples width,
- a variable  $\text{refH}$  specifying the reference samples height,
- a variable  $\text{refFilterFlag}$  specifying the value of reference filter flag,
- the (unfiltered) neighbouring samples  $\text{refUnfilt}[x][y]$ , with  $x = -1 - \text{refIdx}$ ,  $y = -1 - \text{refIdx} \dots \text{refH} - 1$  and  $x = -\text{refIdx} \dots \text{refW} - 1$ ,  $y = -1 - \text{refIdx}$ ,
- a variable  $\text{cIdx}$  specifying the colour component of the current block.

Outputs of this process are the reference samples  $p[x][y]$ , with  $x = -1 - \text{refIdx}$ ,  $y = -1 - \text{refIdx} \dots \text{refH} - 1$  and  $x = -\text{refIdx} \dots \text{refW} - 1$ ,  $y = -1 - \text{refIdx}$ .

The variable  $\text{filterFlag}$  is derived as follows:

- If all of the following conditions are true,  $\text{filterFlag}$  is set equal to 1:
  - $\text{refIdx}$  is equal to 0;
  - $\text{nTbW} * \text{nTbH}$  is greater than 32;
  - $\text{cIdx}$  is equal to 0;
  - $\text{IntraSubPartitionsSplitType}$  is equal to  $\text{ISP\_NO\_SPLIT}$ ;
  - $\text{refFilterFlag}$  is equal to 1;
- Otherwise,  $\text{filterFlag}$  is set equal to 0.

For the derivation of the reference samples  $p[x][y]$  the following applies:

- If  $\text{filterFlag}$  is equal to 1, the filtered sample values  $p[x][y]$  with  $x = -1$ ,  $y = -1 \dots \text{refH} - 1$  and  $x = 0 \dots \text{refW} - 1$ ,  $y = -1$  are derived as follows:

$$p[-1][-1] = (\text{refUnfilt}[-1][0] + 2 * \text{refUnfilt}[-1][-1] + \text{refUnfilt}[0][-1] + 2) \gg 2 \quad (319)$$

$$p[-1][y] = (\text{refUnfilt}[-1][y + 1] + 2 * \text{refUnfilt}[-1][y] + \text{refUnfilt}[-1][y - 1] + 2) \gg 2 \quad \text{for } y = 0 \dots \text{refH} - 2 \quad (320)$$

$$p[-1][\text{refH} - 1] = \text{refUnfilt}[-1][\text{refH} - 1] \quad (321)$$

$$p[x][-1] = (\text{refUnfilt}[x - 1][-1] + 2 * \text{refUnfilt}[x][-1] + \text{refUnfilt}[x + 1][-1] + 2) \gg 2 \quad \text{for } x = 0 \dots \text{refW} - 2 \quad (322)$$

$$p[\text{refW} - 1][ -1 ] = \text{refUnfilt}[\text{refW} - 1][ -1 ] \quad (323)$$

- Otherwise, the reference samples values  $p[x][y]$  are set equal to the unfiltered sample values  $\text{refUnfilt}[x][y]$  with  $x = -1 - \text{refldx}$ ,  $y = -1 - \text{refldx} \cdot \text{refH} - 1$  and  $x = -\text{refldx} \cdot \text{refW} - 1$ ,  $y = -1 - \text{refldx}$ .

#### 8.4.5.2.11 Specification of INTRA\_PLANAR intra prediction mode

Inputs to this process are:

- a variable  $\text{nTbW}$  specifying the transform block width,
- a variable  $\text{nTbH}$  specifying the transform block height,
- the neighbouring samples  $p[x][y]$ , with  $x = -1$ ,  $y = -1 \dots \text{nTbH}$  and  $x = 0 \dots \text{nTbW}$ ,  $y = -1$ .

Outputs of this process are the predicted samples  $\text{predSamples}[x][y]$ , with  $x = 0 \dots \text{nTbW} - 1$ ,  $y = 0 \dots \text{nTbH} - 1$ .

The values of the prediction samples  $\text{predSamples}[x][y]$ , with  $x = 0 \dots \text{nTbW} - 1$  and  $y = 0 \dots \text{nTbH} - 1$ , are derived as follows:

$$\text{predV}[x][y] = ((\text{nTbH} - 1 - y) * p[x][ -1 ] + (y + 1) * p[ -1 ][\text{nTbH}]) \ll \text{Log2}(\text{nTbW}) \quad (324)$$

$$\text{predH}[x][y] = ((\text{nTbW} - 1 - x) * p[ -1 ][y] + (x + 1) * p[\text{nTbW}][ -1 ]) \ll \text{Log2}(\text{nTbH}) \quad (325)$$

$$\text{predSamples}[x][y] = (\text{predV}[x][y] + \text{predH}[x][y] + \text{nTbW} * \text{nTbH}) \gg (\text{Log2}(\text{nTbW}) + \text{Log2}(\text{nTbH}) + 1) \quad (326)$$

#### 8.4.5.2.12 Specification of INTRA\_DC intra prediction mode

Inputs to this process are:

- a variable  $\text{nTbW}$  specifying the transform block width,
- a variable  $\text{nTbH}$  specifying the transform block height,
- a variable  $\text{refldx}$  specifying the intra prediction reference line index,
- the neighbouring samples  $p[x][y]$ , with  $x = -1 - \text{refldx}$ ,  $y = 0 \dots \text{nTbH} - 1$  and  $x = 0 \dots \text{nTbW} - 1$ ,  $y = -1 - \text{refldx}$ .

Outputs of this process are the predicted samples  $\text{predSamples}[x][y]$ , with  $x = 0 \dots \text{nTbW} - 1$ ,  $y = 0 \dots \text{nTbH} - 1$ .

The values of the prediction samples  $\text{predSamples}[x][y]$ , with  $x = 0 \dots \text{nTbW} - 1$ ,  $y = 0 \dots \text{nTbH} - 1$ , are derived by the following ordered steps:

1. A variable  $\text{dcVal}$  is derived as follows:

- When  $\text{nTbW}$  is equal to  $\text{nTbH}$ :

$$\text{dcVal} = (\sum_{x'=0}^{\text{nTbW}-1} p[x'][-1 - \text{refldx}] + \quad (327)$$

$$\sum_{y'=0}^{\text{nTbH}-1} p[-1 - \text{refldx}][y'] + \text{nTbW}) \gg (\text{Log2}(\text{nTbW}) + 1)$$

- When  $\text{nTbW}$  is greater than  $\text{nTbH}$ :

$$\text{dcVal} = (\sum_{x'=0}^{\text{nTbW}-1} p[x'][-1 - \text{refldx}] + (\text{nTbW} \gg 1)) \gg \text{Log2}(\text{nTbW}) \quad (328)$$

- When  $\text{nTbW}$  is less than  $\text{nTbH}$ :

$$dcVal = ( \sum_{y'=0}^{nTbH-1} p[-1 - refIdx][y'] + ( nTbH >> 1 ) ) >> \text{Log2}( nTbH ) \quad (329)$$

2. The prediction samples  $predSamples[x][y]$  are derived as follows:

$$predSamples[x][y] = dcVal, \text{ with } x = 0..nTbW - 1, y = 0..nTbH - 1 \quad (330)$$

#### 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  $( \text{Log2}( nTbW ) + \text{Log2}( nTbH ) ) >> 1$ .

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  $\text{Min}( \text{Abs}( predModeIntra - 50 ), \text{Abs}( predModeIntra - 18 ) )$ .
  - 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        |

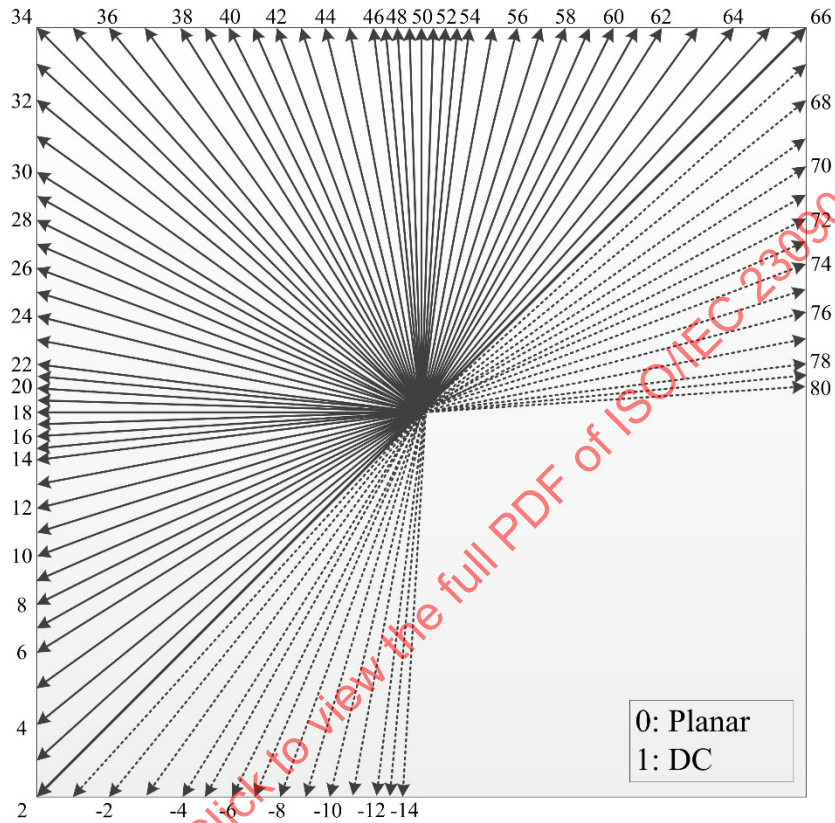
**Figure 9 — Intra prediction directions**

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 predModelIntra and the angle parameter intraPredAngle.

**Table 24 — Specification of intraPredAngle**

|                |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|----------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| predModelIntra | -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  |
| predModelIntra | 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  |
| predModelIntra | 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 |
| predModelIntra | 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   |

|                |    |    |     |     |     |     |     |     |    |    |    |    |    |    |    |    |    |
|----------------|----|----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|
| predModelIntra | 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 |
| predModelIntra | 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:

$$\text{invAngle} = \text{Round}\left(\frac{512 \times 32}{\text{intraPredAngle}}\right) \quad (331)$$

The interpolation filter coefficients  $\text{fC}[\text{phase}][j]$  and  $\text{fG}[\text{phase}][j]$  with  $\text{phase} = 0..31$  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 |                   |                   |                   |
|------------------------------|--------------------------------------|-------------------|-------------------|-------------------|--------------------------------------|-------------------|-------------------|-------------------|
|                              | $\text{fC}[p][0]$                    | $\text{fC}[p][1]$ | $\text{fC}[p][2]$ | $\text{fC}[p][3]$ | $\text{fG}[p][0]$                    | $\text{fG}[p][1]$ | $\text{fG}[p][2]$ | $\text{fG}[p][3]$ |
| 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  $\text{predSamples}[x][y]$ , with  $x = 0..nTbW - 1$ ,  $y = 0..nTbH - 1$  are derived as follows:

— If  $\text{predModeIntra}$  is greater than or equal to 34, the following ordered steps apply:

1. The reference sample array  $\text{ref}[x]$  is specified as follows:

— The following applies:

$$\text{ref}[x] = p[-1 - \text{refIdx} + x][ -1 - \text{refIdx} ], \text{ with } x = 0..nTbW + \text{refIdx} + 1 \quad (332)$$

— If  $\text{intraPredAngle}$  is less than 0, the main reference sample array is extended as follows:

$$\text{ref}[x] = p[-1 - \text{refIdx}][ -1 - \text{refIdx} + \text{Min}( (x * \text{invAngle} + 256) \gg 9, nTbH ) ],$$

with  $x = -nTbH..-1$

(333)

— Otherwise, the following applies:

$$\text{ref}[x] = p[-1 - \text{refIdx} + x][ -1 - \text{refIdx} ], \text{ with } x = nTbW + 2 + \text{refIdx}.. \text{refW} + \text{refIdx} \quad (334)$$

— The additional samples  $\text{ref}[\text{refW} + \text{refIdx} + x]$  with  $x = 1..(\text{Max}(1, nTbW / nTbH) * \text{refIdx} + 1)$  are derived as follows:

$$\text{ref}[\text{refW} + \text{refIdx} + x] = p[-1 + \text{refW}][ -1 - \text{refIdx} ] \quad (335)$$

2. The values of the prediction samples  $\text{predSamples}[x][y]$ , 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:

$$iIdx = ( ( (y + 1 + \text{refIdx}) * \text{intraPredAngle} ) \gg 5 ) + \text{refIdx} \quad (336)$$

$$iFact = ( (y + 1 + \text{refIdx}) * \text{intraPredAngle} ) \& 31 \quad (337)$$

— If  $cIdx$  is equal to 0, the following applies:

— The interpolation filter coefficients  $fT[j]$  with  $j = 0..3$  are derived as follows:

$$fT[j] = \text{filterFlag} ? fG[iFact][j] : fC[iFact][j] \quad (338)$$

— The value of the prediction samples  $\text{predSamples}[x][y]$  is derived as follows:

$$\text{predSamples}[x][y] = \text{Clip1}( ( ( \sum_{i=0}^3 fT[i] * \text{ref}[x + iIdx + i] ) + 32 ) \gg 6 ) \quad (339)$$

— 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  $\text{predSamples}[x][y]$  is derived as follows:

$$\text{predSamples}[x][y] = ( ( 32 - iFact ) * \text{ref}[x + iIdx + 1] + iFact * \text{ref}[x + iIdx + 2] + 16 ) \gg 5 \quad (340)$$

— Otherwise, the value of the prediction samples  $\text{predSamples}[x][y]$  is derived as follows:

$$\text{predSamples}[x][y] = \text{ref}[x + iIdx + 1] \quad (341)$$

— Otherwise ( $\text{predModeIntra}$  is less than 34), the following ordered steps apply:

1. The reference sample array  $\text{ref}[x]$  is specified as follows:

— The following applies:

$$\text{ref}[x] = p[-1 - \text{refIdx}][ -1 - \text{refIdx} + x ], \text{ with } x = 0..nTbH + \text{refIdx} + 1 \quad (342)$$

— If  $\text{intraPredAngle}$  is less than 0, the main reference sample array is extended as follows:

$$\text{ref}[x] = p[-1 - \text{refIdx} + \text{Min}( (x * \text{invAngle} + 256) \gg 9, nTbW ) ][ -1 - \text{refIdx} ], \\ \text{with } x = -nTbW..-1 \quad (343)$$

— Otherwise, the following applies:

$$\text{ref}[x] = p[-1 - \text{refIdx}][ -1 - \text{refIdx} + x ], \text{ with } x = nTbH + 2 + \text{refIdx}.. \text{refH} + \text{refIdx} \quad (344)$$

— The additional samples  $\text{ref}[\text{refH} + \text{refIdx} + x]$  with  $x = 1..( \text{Max}( 1, nTbH / nTbW ) * \text{refIdx} + 1 )$  are derived as follows:

$$\text{ref}[\text{refH} + \text{refIdx} + x] = p[-1 - \text{refIdx}][ -1 + \text{refH} ] \quad (345)$$

2. The values of the prediction samples  $\text{predSamples}[x][y]$ , with  $x = 0..nTbW - 1$ ,  $y = 0..nTbH - 1$  are derived as follows:

— The index variable  $\text{iIdx}$  and the multiplication factor  $\text{iFact}$  are derived as follows:

$$\text{iIdx} = ( ( (x + 1 + \text{refIdx}) * \text{intraPredAngle} ) \gg 5 ) + \text{refIdx} \quad (346)$$

$$\text{iFact} = ( (x + 1 + \text{refIdx}) * \text{intraPredAngle} ) \& 31 \quad (347)$$

— If  $\text{cIdx}$  is equal to 0, the following applies:

— The interpolation filter coefficients  $\text{fT}[j]$  with  $j = 0..3$  are derived as follows:

$$\text{fT}[j] = \text{filterFlag} ? \text{fG}[\text{iFact}][j] : \text{fC}[\text{iFact}][j] \quad (348)$$

— The value of the prediction samples  $\text{predSamples}[x][y]$  is derived as follows:

$$\text{predSamples}[x][y] = \text{Clip1}( ( ( \sum_{i=0}^3 \text{fT}[i] * \text{ref}[y + \text{iIdx} + i] ) + 32 ) \gg 6 ) \quad (349)$$

— Otherwise ( $\text{cIdx}$  is not equal to 0), depending on the value of  $\text{iFact}$ , the following applies:

— If  $\text{iFact}$  is not equal to 0, the value of the prediction samples  $\text{predSamples}[x][y]$  is derived as follows:

$$\text{predSamples}[x][y] = \\ ( ( (32 - \text{iFact}) * \text{ref}[y + \text{iIdx} + 1] + \text{iFact} * \text{ref}[y + \text{iIdx} + 2] + 16 ) \gg 5 ) \quad (350)$$

— Otherwise, the value of the prediction samples  $\text{predSamples}[x][y]$  is derived as follows:

$$\text{predSamples}[x][y] = \text{ref}[y + \text{iIdx} + 1] \quad (351)$$

#### 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  $\text{predModeIntra}$ ,
- 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:

$$(xTbY, yTbY) = (xTbC \ll (SubWidthC - 1), yTbC \ll (SubHeightC - 1)) \quad (352)$$

The variables  $availL$  and  $availT$  are derived as follows:

- The derivation process for neighbouring block availability as specified in subclause 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 subclause 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 subclause 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 subclause 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:

$$numSampT = availT ? nTbW : 0 \quad (353)$$

$$numSampL = availL ? nTbH : 0 \quad (354)$$

- Otherwise, the following applies:

$$\text{numSampT} = ( \text{availT} \ \&\& \ \text{predModeIntra} == \text{INTRA\_T\_CCLM} ) ? ( \text{nTbW} + \text{Min}( \text{numTopRight}, \text{nTbH} ) ) : 0 \quad (355)$$

$$\text{numSampL} = ( \text{availL} \ \&\& \ \text{predModeIntra} == \text{INTRA\_L\_CCLM} ) ? ( \text{nTbH} + \text{Min}( \text{numLeftBelow}, \text{nTbW} ) ) : 0 \quad (356)$$

The variable bCTUboundary is derived as follows:

$$\text{bCTUboundary} = ( ( \text{yTbY} \ \& \ ( \text{CtbSizeY} - 1 ) ) == 0 ) ? \text{TRUE} : \text{FALSE} \quad (357)$$

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:

$$\text{numIs4N} = ( ( \text{availT} \ \&\& \ \text{availL} \ \&\& \ \text{predModeIntra} == \text{INTRA\_LT\_CCLM} ) ? 0 : 1 ) \quad (358)$$

— The variable startPosN is set equal to  $\text{numSampN} \gg ( 2 + \text{numIs4N} )$ .

— The variable pickStepN is set equal to  $\text{Max}( 1, \text{numSampN} \gg ( 1 + \text{numIs4N} ) )$ .

— 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  $\text{Min}( \text{numSampN}, ( 1 + \text{numIs4N} ) \ll 1 )$ .

— pickPosN[ pos ] is set equal to  $( \text{startPosN} + \text{pos} * \text{pickStepN} )$ , with  $\text{pos} = 0.. \text{cntN} - 1$ .

— Otherwise, cntN is set equal to 0.

The prediction samples  $\text{predSamples}[x][y]$  with  $x = 0.. \text{nTbW} - 1, y = 0.. \text{nTbH} - 1$  are derived as follows:

— If both numSampL and numSampT are equal to 0, the following applies:

$$\text{predSamples}[x][y] = 1 \ll ( \text{BitDepth} - 1 ) \quad (359)$$

— Otherwise, the following ordered steps apply:

1. The collocated luma samples  $\text{pY}[x][y]$  with  $x = 0.. \text{nTbW} * \text{SubWidthC} - 1, y = 0.. \text{nTbH} * \text{SubHeightC} - 1$  are set equal to the reconstructed luma samples prior to the deblocking filter process at the locations  $( x\text{TbY} + x, y\text{TbY} + y )$ .

2. The neighbouring luma samples  $\text{pY}[x][y]$  are derived as follows:

— When availL is equal to TRUE, the neighbouring luma samples  $\text{pY}[x][y]$  with  $x = -3..-1, y = ( \text{availT} ? -1 : 0 ).. \text{SubHeightC} * \text{Max}( \text{nTbH}, \text{numSampL} ) - 1$ , are set equal to the reconstructed luma samples prior to the deblocking filter process at the locations  $( x\text{TbY} + x, y\text{TbY} + y )$ .

— When availT is equal to FALSE, the neighbouring luma samples  $\text{pY}[x][y]$  with  $x = -2.. \text{SubWidthC} * \text{nTbW} - 1, y = -2..-1$ , are set equal to the luma samples  $\text{pY}[x][0]$ .

— When availT is equal to TRUE, the neighbouring luma samples  $\text{pY}[x][y]$  with  $x = ( \text{availL} ? -1 : 0 ).. \text{SubWidthC} * \text{Max}( \text{nTbW}, \text{numSampT} ) - 1, y = -3..-1$ , are set equal to the reconstructed luma samples prior to the deblocking filter process at the locations  $( x\text{TbY} + x, y\text{TbY} + y )$ .

— When availL is equal to FALSE, the neighbouring luma samples  $\text{pY}[x][y]$  with  $x = -1, y = -2.. \text{SubHeightC} * \text{nTbH} - 1$ , are set equal to the reconstructed luma samples  $\text{pY}[0][y]$ .

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:

$$pDsY[x][y] = pY[x][y] \quad (360)$$

— Otherwise, if  $SubHeightC$  is equal to 1, the following applies:

$$pDsY[x][y] = (pY[SubWidthC * x - 1][y] + 2 * pY[SubWidthC * x][y] + pY[SubWidthC * x + 1][y] + 2) >> 2 \quad (361)$$

— Otherwise ( $SubHeightC$  is not equal to 1), the following applies:

— If  $sps\_chroma\_vertical\_collocated\_flag$  is equal to 1, the following applies:

$$pDsY[x][y] = (pY[SubWidthC * x][SubHeightC * y - 1] + pY[SubWidthC * x - 1][SubHeightC * y] + 4 * pY[SubWidthC * x][SubHeightC * y] + pY[SubWidthC * x + 1][SubHeightC * y] + pY[SubWidthC * x][SubHeightC * y + 1] + 4) >> 3 \quad (362)$$

— Otherwise ( $sps\_chroma\_vertical\_collocated\_flag$  is equal to 0), the following applies:

$$pDsY[x][y] = (pY[SubWidthC * x - 1][SubHeightC * y] + pY[SubWidthC * x - 1][SubHeightC * y + 1] + 2 * pY[SubWidthC * x][SubHeightC * y] + 2 * pY[SubWidthC * x][SubHeightC * y + 1] + pY[SubWidthC * x + 1][SubHeightC * y] + pY[SubWidthC * x + 1][SubHeightC * y + 1] + 4) >> 3 \quad (363)$$

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:

$$pSelDsY[idx] = pY[x][-1] \quad (364)$$

— 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:

$$pSelDsY[idx] = (pY[SubWidthC * x][-3] + pY[SubWidthC * x - 1][-2] + 4 * pY[SubWidthC * x][-2] + pY[SubWidthC * x + 1][-2] + pY[SubWidthC * x][-1] + 4) >> 3 \quad (365)$$

— Otherwise ( $sps\_chroma\_vertical\_collocated\_flag$  is equal to 0), the following applies:

$$\begin{aligned}
pSelDsY[idx] = & ( pY[ SubWidthC * x - 1 ][ -1 ] + \\
& pY[ SubWidthC * x - 1 ][ -2 ] + \\
& 2 * pY[ SubWidthC * x ][ -1 ] + \\
& 2 * pY[ SubWidthC * x ][ -2 ] + \\
& pY[ SubWidthC * x + 1 ][ -1 ] + \\
& pY[ SubWidthC * x + 1 ][ -2 ] + 4 ) >> 3
\end{aligned}
\tag{366}$$

—Otherwise (SubHeightC is equal to 1 or bCTUboundary is equal to TRUE), the following applies:

$$\begin{aligned}
pSelDsY[idx] = & ( pY[ SubWidthC * x - 1 ][ -1 ] + \\
& 2 * pY[ SubWidthC * x ][ -1 ] + \\
& pY[ SubWidthC * x + 1 ][ -1 ] + 2 ) >> 2
\end{aligned}
\tag{367}$$

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:

$$pSelDsY[idx] = pY[-1][y] \tag{368}$$

- Otherwise, if SubHeightC is equal to 1, the following applies:

$$\begin{aligned}
pSelDsY[idx] = & ( pY[-1 - SubWidthC][y] + \\
& 2 * pY[-SubWidthC][y] + \\
& pY[1 - SubWidthC][y + 2] ) >> 2
\end{aligned}
\tag{369}$$

- Otherwise, the following applies:

—If `sps_chroma_vertical_collocated_flag` is equal to 1, the following applies:

$$\begin{aligned}
pSelDsY[idx] = & ( pY[-SubWidthC][ SubHeightC * y - 1 ] + \\
& pY[-1 - SubWidthC][ SubHeightC * y ] + \\
& 4 * pY[-SubWidthC][ SubHeightC * y ] + \\
& pY[1 - SubWidthC][ SubHeightC * y ] + \\
& pY[-SubWidthC][ SubHeightC * y + 1 ] + 4 ) >> 3
\end{aligned}
\tag{370}$$

—Otherwise (`sps_chroma_vertical_collocated_flag` is equal to 0), the following applies:

$$\begin{aligned}
pSelDsY[idx] = & ( pY[-1 - SubWidthC][ SubHeightC * y ] + \\
& pY[-1 - SubWidthC][ SubHeightC * y + 1 ] + \\
& 2 * pY[-SubWidthC][ SubHeightC * y ] + \\
& 2 * pY[-SubWidthC][ SubHeightC * y + 1 ] + \\
& pY[1 - SubWidthC][ SubHeightC * y ] + \\
& pY[1 - SubWidthC][ SubHeightC * y + 1 ] + 4 ) >> 3
\end{aligned}
\tag{371}$$

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:

$$\text{minGrpIdx}[0] = 0 \quad (372)$$

$$\text{minGrpIdx}[1] = 2 \quad (373)$$

$$\text{maxGrpIdx}[0] = 1 \quad (374)$$

$$\text{maxGrpIdx}[1] = 3 \quad (375)$$

- When pSelDsY[ minGrpIdx[ 0 ] ] is greater than pSelDsY[ minGrpIdx[ 1 ] ], minGrpIdx[ 0 ] and minGrpIdx[ 1 ] are swapped as follows:

$$(\text{minGrpIdx}[0], \text{minGrpIdx}[1]) = \text{Swap}(\text{minGrpIdx}[0], \text{minGrpIdx}[1]) \quad (376)$$

- 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:

$$\text{diffC} = \text{maxC} - \text{minC} \quad (385)$$

$$x = \text{Floor}(\text{Log2}(\text{diff})) \quad (386)$$

$$\text{normDiff} = ((\text{diff} \ll 4) \gg x) \& 15 \quad (387)$$

$$x += (\text{normDiff} \neq 0) ? 1 : 0 \quad (388)$$

$$y = \text{Abs}(\text{diffC}) > 0 ? \text{Floor}(\text{Log2}(\text{Abs}(\text{diffC}))) + 1 : 0 \quad (389)$$

$$a = ( \text{diffC} * ( \text{divSigTable}[ \text{normDiff} ] | 8 ) + 2^{y-1} ) \gg y \quad (390)$$

$$k = ( ( 3 + x - y ) < 1 ) ? 1 : 3 + x - y \quad (391)$$

$$a = ( ( 3 + x - y ) < 1 ) ? \text{Sign}( a ) * 15 : a \quad (392)$$

$$b = \text{minC} - ( ( a * \text{minY} ) \gg k ) \quad (393)$$

where  $\text{divSigTable}[ ]$  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  $\text{predSamples}[x][y]$  with  $x = 0..nTbW - 1, y = 0..nTbH - 1$  are derived as follows:

$$\text{predSamples}[x][y] = \text{Clip1}( ( ( \text{pDsY}[x][y] * a ) \gg k ) + b ) \quad (398)$$

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  $\text{predSamples}[x][y]$ , 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  $\text{predSamples}[x][y]$  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  $\text{Min}( 2, \text{Log2}( nTbH ) - \text{Floor}( \text{Log2}( 3 * \text{invAngle} - 2 ) ) + 8 )$ , using `invAngle` as specified in subclause 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  $\text{Min}( 2, \text{Log2}( nTbW ) - \text{Floor}( \text{Log2}( 3 * \text{invAngle} - 2 ) ) + 8 )$ , using `invAngle` as specified in subclause 8.4.5.2.13.
- Otherwise, `nScale` is set equal to  $( ( \text{Log2}( nTbW ) + \text{Log2}( nTbH ) - 2 ) \gg 2 )$ .

The reference sample arrays  $\text{mainRef}[x]$  and  $\text{sideRef}[y]$ , with  $x = 0..\text{refW} - 1$  and  $y = 0..\text{refH} - 1$  are derived as follows:

$$\begin{aligned}\text{mainRef}[x] &= p[x][-1] \\ \text{sideRef}[y] &= p[-1][y]\end{aligned}\quad (399)$$

The variables  $\text{refL}[x][y]$ ,  $\text{refT}[x][y]$ ,  $\text{wT}[y]$ , and  $\text{wL}[x]$  with  $x = 0..\text{nTbW} - 1$ ,  $y = 0..\text{nTbH} - 1$  are derived as follows:

— If  $\text{predModeIntra}$  is equal to  $\text{INTRA\_PLANAR}$  or  $\text{INTRA\_DC}$ , the following applies:

$$\text{refL}[x][y] = p[-1][y] \quad (400)$$

$$\text{refT}[x][y] = p[x][-1] \quad (401)$$

$$\text{wT}[y] = 32 \gg ((y \ll 1) \gg \text{nScale}) \quad (402)$$

$$\text{wL}[x] = 32 \gg ((x \ll 1) \gg \text{nScale}) \quad (403)$$

— Otherwise, if  $\text{predModeIntra}$  is equal to  $\text{INTRA\_ANGULAR18}$  or  $\text{INTRA\_ANGULAR50}$ , the following applies:

$$\text{refL}[x][y] = p[-1][y] - p[-1][-1] + \text{predSamples}[x][y] \quad (404)$$

$$\text{refT}[x][y] = p[x][-1] - p[-1][-1] + \text{predSamples}[x][y] \quad (405)$$

$$\text{wT}[y] = (\text{predModeIntra} == \text{INTRA\_ANGULAR18}) ? 32 \gg ((y \ll 1) \gg \text{nScale}) : 0 \quad (406)$$

$$\text{wL}[x] = (\text{predModeIntra} == \text{INTRA\_ANGULAR50}) ? 32 \gg ((x \ll 1) \gg \text{nScale}) : 0 \quad (407)$$

— Otherwise, if  $\text{predModeIntra}$  is less than  $\text{INTRA\_ANGULAR18}$  and  $\text{nScale}$  is equal to or greater than 0, the following ordered steps apply:

1. The variables  $\text{dXInt}[y]$  and  $\text{dX}[x][y]$  are derived as follows using  $\text{invAngle}$  as specified in subclause 8.4.5.2.13 depending on  $\text{intraPredMode}$ :

$$\begin{aligned}\text{dXInt}[y] &= ((y + 1) * \text{invAngle} + 256) \gg 9 \\ \text{dX}[x][y] &= x + \text{dXInt}[y]\end{aligned}\quad (408)$$

2. The variables  $\text{refL}[x][y]$ ,  $\text{refT}[x][y]$ ,  $\text{wT}[y]$ , and  $\text{wL}[x]$  are derived as follows:

$$\text{refL}[x][y] = 0 \quad (409)$$

$$\text{refT}[x][y] = (y < (3 \ll \text{nScale})) ? \text{mainRef}[\text{dX}[x][y]] : 0 \quad (410)$$

$$\text{wT}[y] = 32 \gg ((y \ll 1) \gg \text{nScale}) \quad (411)$$

$$\text{wL}[x] = 0 \quad (412)$$

— Otherwise, if  $\text{predModeIntra}$  is greater than  $\text{INTRA\_ANGULAR50}$  and  $\text{nScale}$  is equal to or greater than 0, the following ordered steps apply:

1. The variables  $\text{dYInt}[x]$  and  $\text{dY}[x][y]$  are derived as follows using  $\text{invAngle}$  as specified in subclause 8.4.5.2.13 depending on  $\text{intraPredMode}$ :

$$\begin{aligned}\text{dYInt}[x] &= ((x + 1) * \text{invAngle} + 256) \gg 9 \\ \text{dY}[x][y] &= y + \text{dYInt}[x]\end{aligned}\quad (413)$$

2. The variables  $\text{refL}[x][y]$ ,  $\text{refT}[x][y]$ ,  $\text{wT}[y]$ , and  $\text{wL}[x]$  are derived as follows:

$$\text{refL}[x][y] = (x < (3 \ll \text{nScale})) ? \text{sideRef}[dY[x][y]] : 0 \quad (414)$$

$$\text{refT}[x][y] = 0 \quad (415)$$

$$\text{wT}[y] = 0 \quad (416)$$

$$\text{wL}[x] = 32 \gg ((x \ll 1) \gg \text{nScale}) \quad (417)$$

— Otherwise,  $\text{refL}[x][y]$ ,  $\text{refT}[x][y]$ ,  $\text{wT}[y]$ , and  $\text{wL}[x]$  are all set equal to 0.

The values of the modified predicted samples  $\text{predSamples}[x][y]$ , with  $x = 0..nTbW - 1$ ,  $y = 0..nTbH - 1$  are derived as follows:

$$\text{predSamples}[x][y] = \text{Clip1}((\text{refL}[x][y] * \text{wL}[x] + \text{refT}[x][y] * \text{wT}[y] + (64 - \text{wL}[x] - \text{wT}[y]) * \text{predSamples}[x][y] + 32) \gg 6) \quad (418)$$

#### 8.4.5.3 Decoding process for palette mode

Inputs to this process are:

- a location  $(x_{CbComp}, y_{CbComp})$  specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,
- a variable  $\text{treeType}$  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  $\text{cldx}$  specifying the colour component of the current block,
- two variables  $\text{nCbW}$  and  $\text{nCbH}$  specifying the width and height of the current coding block, respectively.

Output of this process is an array  $\text{recSamples}[x][y]$ , with  $x = 0..nCbW - 1$ ,  $y = 0..nCbH - 1$  specifying reconstructed sample values for the block.

Depending on the value of  $\text{treeType}$ , the variables  $\text{startComp}$ ,  $\text{numComps}$  and  $\text{maxNumPalettePredictorSize}$  are derived as follows:

- If  $\text{treeType}$  is equal to  $\text{SINGLE\_TREE}$ , the following applies:

$$\text{startComp} = 0 \quad (419)$$

$$\text{numComps} = \text{sps\_chroma\_format\_idc} == 0 ? 1 : 3 \quad (420)$$

$$\text{maxNumPalettePredictorSize} = 63 \quad (421)$$

- Otherwise, if  $\text{treeType}$  is equal to  $\text{DUAL\_TREE\_LUMA}$ , the following applies:

$$\text{startComp} = 0 \quad (422)$$

$$\text{numComps} = 1 \quad (423)$$

$$\text{maxNumPalettePredictorSize} = 31 \quad (424)$$

- Otherwise ( $\text{treeType}$  is equal to  $\text{DUAL\_TREE\_CHROMA}$ ), the following applies:

$$\text{startComp} = 1 \quad (425)$$

$$\text{numComps} = 2 \quad (426)$$

$$\text{maxNumPalettePredictorSize} = 31 \quad (427)$$

Depending on the value of  $\text{cIdx}$ , the variables  $\text{nSubWidth}$  and  $\text{nSubHeight}$  are derived as follows:

- If  $\text{cIdx}$  is greater than 0 and  $\text{startComp}$  is equal to 0,  $\text{nSubWidth}$  is set equal to  $\text{SubWidthC}$  and  $\text{nSubHeight}$  is set equal to  $\text{SubHeightC}$ .
- Otherwise,  $\text{nSubWidth}$  is set equal to 1 and  $\text{nSubHeight}$  is set equal to 1.

The (  $\text{nCbW} \times \text{nCbH}$  ) block of the reconstructed sample array  $\text{recSamples}$  at location (  $\text{xCbComp}, \text{yCbComp}$  ) is represented by  $\text{recSamples}[x][y]$  with  $x = 0.. \text{nCbW} - 1$  and  $y = 0.. \text{nCbH} - 1$ , and the value of  $\text{recSamples}[x][y]$  for each  $x$  in the range of 0 to  $\text{nCbW} - 1$ , inclusive, and each  $y$  in the range of 0 to  $\text{nCbH} - 1$ , inclusive, is derived as follows:

- The variables  $\text{xL}$ ,  $\text{yL}$ ,  $\text{xCbL}$ , and  $\text{yCbL}$  are derived as follows:

$$\text{xL} = x * \text{nSubWidth} \quad (428)$$

$$\text{yL} = y * \text{nSubHeight} \quad (429)$$

$$\text{xCbL} = \text{xCbComp} * \text{nSubWidth} \quad (430)$$

$$\text{yCbL} = \text{yCbComp} * \text{nSubHeight} \quad (431)$$

- The variable  $\text{blsEscapeSample}$  is derived as follows:

- If  $\text{PaletteIndexMap}[\text{xCbL} + \text{xL}][\text{yCbL} + \text{yL}]$  is equal to  $\text{MaxPaletteIndex}$  and  $\text{palette\_escape\_val\_present\_flag}$  is equal to 1,  $\text{blsEscapeSample}$  is set equal to 1.

- Otherwise,  $\text{blsEscapeSample}$  is set equal to 0.

- If  $\text{blsEscapeSample}$  is equal to 0, the following applies:

$$\text{recSamples}[x][y] = \text{CurrentPaletteEntries}[\text{cIdx}][\text{PaletteIndexMap}[\text{xCbL} + \text{xL}][\text{yCbL} + \text{yL}]] \quad (432)$$

- Otherwise ( $\text{blsEscapeSample}$  is equal to 1), the following ordered steps apply:

1. The quantization parameter  $qP$  is derived as follows:

- If  $\text{cIdx}$  is equal to 0, the following applies:

$$qP = \text{Max}(\text{QpPrimeTsMin}, \text{Qp}'Y) \quad (433)$$

- Otherwise, if  $\text{cIdx}$  is equal to 1, the following applies:

$$qP = \text{Max}(\text{QpPrimeTsMin}, \text{Qp}'Cb) \quad (434)$$

- Otherwise ( $\text{cIdx}$  is equal to 2), the following applies:

$$qP = \text{Max}(\text{QpPrimeTsMin}, \text{Qp}'Cr) \quad (435)$$

2. The list  $\text{levelScale}[]$  is specified as  $\text{levelScale}[k] = \{ 40, 45, 51, 57, 64, 72 \}$  with  $k = 0..5$ .

3. The following applies:

$$\text{tmpVal} = ( ( \text{PaletteEscapeVal}[\text{cIdx}][\text{xCbL} + \text{xL}][\text{yCbL} + \text{yL}] * \text{levelScale}[qP \% 6] ) << ( qP / 6 ) + 32 ) >> 6 \quad (436)$$

$$\text{recSamples}[x][y] = \text{Clip3}(0, (1 << \text{BitDepth}) - 1, \text{tmpVal}) \quad (437)$$

When LocalDualTreeFlag is equal to 1, the following applies:

- When treeType is equal to DUAL\_TREE\_LUMA, the following applies for  $i = 0..num\_signalled\_palette\_entries - 1$ :

$$CurrentPaletteEntries[1][NumPredictedPaletteEntries + i] = 1 \ll (BitDepth - 1) \quad (438)$$

$$CurrentPaletteEntries[2][NumPredictedPaletteEntries + i] = 1 \ll (BitDepth - 1) \quad (439)$$

- The variables CurrentPaletteSize[0], startComp, numComps and maxNumPalettePredictorSize are derived as follows:

$$CurrentPaletteSize[0] = CurrentPaletteSize[startComp] \quad (440)$$

$$startComp = 0 \quad (441)$$

$$numComps = 3 \quad (442)$$

$$maxNumPalettePredictorSize = 63 \quad (443)$$

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++ )
            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
```

(444)

When sps\_qtbt\_dual\_tree\_intra\_flag is equal to 0 or sh\_slice\_type is not equal to 1, the following applies:

$$PredictorPaletteSize[1] = newPredictorPaletteSize \quad (445)$$

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 (  $x_{Cb}$ ,  $y_{Cb}$  ) 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 subclause 8.7.1 is invoked with the luma location (  $x_{Cb}$ ,  $y_{Cb}$  ), 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  $hpelIdx$  is set equal to 0.
2. The motion vector components and reference indices of the current coding unit are derived as follows:
  - If  $MergeGpmFlag[x_{Cb}][y_{Cb}]$ ,  $inter\_affine\_flag[x_{Cb}][y_{Cb}]$  and  $merge\_subblock\_flag[x_{Cb}][y_{Cb}]$  are all equal to 0, the following applies:
    - The derivation process for motion vector components and reference indices as specified in subclause 8.5.2.1 is invoked with the luma coding block location (  $x_{Cb}$ ,  $y_{Cb}$  ), 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  $hpelIdx$ , 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.
      - $general\_merge\_flag[x_{Cb}][y_{Cb}]$  is equal to 1.
      - both  $predFlagL0[0][0]$  and  $predFlagL1[0][0]$  are equal to 1.
      - $mmvd\_merge\_flag[x_{Cb}][y_{Cb}]$  is equal to 0.
      - $ciip\_flag[x_{Cb}][y_{Cb}]$  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 `refPicLXL` of luma samples and two ordered two-dimensional arrays `refPicLXCb` and `refPicLXCr` of chroma samples is derived by invoking the process specified in subclause 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:
 
$$\text{numSbX} = (\text{cbWidth} > 16) ? (\text{cbWidth} \gg 4) : 1 \quad (446)$$

$$\text{numSbY} = (\text{cbHeight} > 16) ? (\text{cbHeight} \gg 4) : 1 \quad (447)$$

$$\text{sbWidth} = (\text{cbWidth} > 16) ? 16 : \text{cbWidth} \quad (448)$$

$$\text{sbHeight} = (\text{cbHeight} > 16) ? 16 : \text{cbHeight} \quad (449)$$
  - 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:
 
$$\text{mvLX}[xSbIdx][ySbIdx] = \text{mvLX}[0][0] \quad (450)$$

$$\text{predFlagLX}[xSbIdx][ySbIdx] = \text{predFlagLX}[0][0] \quad (451)$$

$$\text{xSb}[xSbIdx][ySbIdx] = \text{xCb} + xSbIdx * \text{sbWidth} \quad (452)$$

$$\text{ySb}[xSbIdx][ySbIdx] = \text{yCb} + ySbIdx * \text{sbHeight} \quad (453)$$
    - The decoder-side motion vector refinement process specified in subclause 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 `refPicLXL` as inputs and delta motion vectors `dMvLX[xSbIdx][ySbIdx]` with  $X = 0..1$ , and the minimum 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 subclause 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 subclause 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 subclause 8.5.4.1 is invoked with the luma coding block location  $(x_{Cb}, y_{Cb})$ , 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 subclause 8.5.5.1 is invoked with the luma coding block location  $(x_{Cb}, y_{Cb})$ , 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  $x_{SbIdx} = 0..numSbX - 1$ , and  $y_{SbIdx} = 0..numSbY - 1$ , 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  $x_{Idx} = 0..cbWidth/numSbX - 1$ ,  $y_{Idx} = 0..cbHeight/numSbY - 1$  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  $x_{SbIdx} = 0..numSbX - 1$ ,  $y_{SbIdx} = 0..numSbY - 1$ :
- If `dmvrFlag` is equal to 1, the derivation process for chroma motion vectors in subclause 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:
 
$$\text{refMvLX}[x_{SbIdx}][y_{SbIdx}] = \text{mvLX}[x_{SbIdx}][y_{SbIdx}] + \text{dMvLX}[x_{SbIdx}][y_{SbIdx}] \quad (454)$$

$$\text{refMvLX}[x_{SbIdx}][y_{SbIdx}][0] = \text{Clip3}(-2^{17}, 2^{17} - 1, \text{refMvLX}[x_{SbIdx}][y_{SbIdx}][0]) \quad (455)$$

$$\text{refMvLX}[x_{SbIdx}][y_{SbIdx}][1] = \text{Clip3}(-2^{17}, 2^{17} - 1, \text{refMvLX}[x_{SbIdx}][y_{SbIdx}][1]) \quad (456)$$
  - Otherwise (`dmvrFlag` is equal to 0), the following applies:
 
$$\text{refMvLX}[x_{SbIdx}][y_{SbIdx}] = \text{mvLX}[x_{SbIdx}][y_{SbIdx}] \quad (457)$$

$$\text{refMvCLX}[x_{SbIdx}][y_{SbIdx}] = \text{mvCLX}[x_{SbIdx}][y_{SbIdx}] \quad (458)$$
- NOTE The array `refMvLX` is stored in `MvDmvrLX` and used in the derivation process for collocated motion vectors in subclause 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 subclause 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 refldxL0 and refldxL1, the prediction list utilization flags predFlagL0[ xSbIdx ][ ySbIdx ] and predFlagL1[ xSbIdx ][ ySbIdx ], the half sample interpolation filter index hpellfIdx, the bi-prediction weight index bcwIdx, the minimum 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<sub>L</sub> 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 subclause 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 refldxL0 and refldxL1, the prediction list utilization flags predFlagL0[ xSbIdx ][ ySbIdx ] and predFlagL1[ xSbIdx ][ ySbIdx ], the half sample interpolation filter index hpellfIdx, the bi-prediction weight index bcwIdx, the minimum 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<sub>Cb</sub> 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 subclause 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 refldxL0 and refldxL1, the prediction list utilization flags predFlagL0[ xSbIdx ][ ySbIdx ] and predFlagL1[ xSbIdx ][ ySbIdx ], the half sample interpolation filter index hpellfIdx, the bi-prediction weight index bcwIdx, the minimum sum of absolute difference values in

decoder-side motion vector refinement process  $\text{dmvrSad}[x\text{SbIdx}][y\text{SbIdx}]$ , the decoder-side motion vector refinement flag  $\text{dmvrFlag}$ , the variable  $\text{cldx}$  set equal to 2, the prediction refinement utility flags  $\text{cbProffFlagL0}$  and  $\text{cbProffFlagL1}$ , and motion vector difference arrays  $\text{diffMvL0}[x\text{Idx}][y\text{Idx}]$  and  $\text{diffMvL1}[x\text{Idx}][y\text{Idx}]$  with  $x\text{Idx} = 0..\text{cbWidth} / \text{numSbX} - 1$ , and  $y\text{Idx} = 0..\text{cbHeight} / \text{numSbY} - 1$  as inputs, and the inter prediction samples ( $\text{predSamples}$ ) that are an  $(\text{cbWidth} / \text{SubWidthC}) \times (\text{cbHeight} / \text{SubHeightC})$  array  $\text{predSamples}_{\text{Cr}}$  of prediction chroma samples for the chroma components  $\text{Cr}$  as outputs.

- Otherwise ( $\text{MergeGpmFlag}[x\text{Cb}][y\text{Cb}]$  is equal to 1), the decoding process for geometric partitioning mode inter blocks as specified in subclause 8.5.7.1 is invoked with the luma coding block location  $(x\text{Cb}, y\text{Cb})$ , the luma coding block width  $\text{cbWidth}$  and the luma coding block height  $\text{cbHeight}$ , the luma motion vectors  $\text{mvA}$  and  $\text{mvB}$ , the chroma motion vectors  $\text{mvCA}$  and  $\text{mvCB}$ , the reference indices  $\text{refIdxA}$  and  $\text{refIdxB}$ , and the prediction list flags  $\text{predListFlagA}$  and  $\text{predListFlagB}$  as inputs, and the inter prediction samples ( $\text{predSamples}$ ) that are an  $(\text{cbWidth}) \times (\text{cbHeight})$  array  $\text{predSamples}_{\text{L}}$  of prediction luma samples and, when  $\text{sps\_chroma\_format\_idc}$  is not equal to 0, two  $(\text{cbWidth} / \text{SubWidthC}) \times (\text{cbHeight} / \text{SubHeightC})$  arrays  $\text{predSamples}_{\text{Cb}}$  and  $\text{predSamples}_{\text{Cr}}$  of prediction chroma samples, one for each of the chroma components  $\text{Cb}$  and  $\text{Cr}$ , as outputs.
- 5. The variables  $\text{NumSbX}[x\text{Cb}][y\text{Cb}]$  and  $\text{NumSbY}[x\text{Cb}][y\text{Cb}]$  are set equal to  $\text{numSbX}$  and  $\text{numSbY}$ , 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 subclause 8.5.8 is invoked with the location  $(x\text{Tb0}, y\text{Tb0})$  set equal to the luma location  $(x\text{Cb}, y\text{Cb})$ , the width  $\text{nCbW}$  set equal to the luma coding block width  $\text{cbWidth}$ , the height  $\text{nCbH}$  set equal to the luma coding block height  $\text{cbHeight}$ , the width  $\text{nTbW}$  set equal to the luma coding block width  $\text{cbWidth}$ , the height  $\text{nTbH}$  set equal to the luma coding block height  $\text{cbHeight}$  and the variable  $\text{cldx}$  set equal to 0 as inputs, and the array  $\text{resSamples}_{\text{L}}$  as output.
  - When  $\text{sps\_chroma\_format\_idc}$  is not equal to 0, the decoding process for the residual signal of coding blocks coded in inter prediction mode as specified in subclause 8.5.8 is invoked with the location  $(x\text{Tb0}, y\text{Tb0})$  set equal to the chroma location  $(x\text{Cb} / \text{SubWidthC}, y\text{Cb} / \text{SubHeightC})$ , the width  $\text{nCbW}$  set equal to the chroma coding block width  $\text{cbWidth} / \text{SubWidthC}$ , the height  $\text{nCbH}$  set equal to the chroma coding block height  $\text{cbHeight} / \text{SubHeightC}$ , the width  $\text{nTbW}$  set equal to the chroma coding block width  $\text{cbWidth} / \text{SubWidthC}$ , the height  $\text{nTbH}$  set equal to the chroma coding block height  $\text{cbHeight} / \text{SubHeightC}$  and the variable  $\text{cldx}$  set equal to 1 as inputs, and the array  $\text{resSamples}_{\text{Cb}}$  as output.
  - When  $\text{sps\_chroma\_format\_idc}$  is not equal to 0, the decoding process for the residual signal of coding blocks coded in inter prediction mode as specified in subclause 8.5.8 is invoked with the location  $(x\text{Tb0}, y\text{Tb0})$  set equal to the chroma location  $(x\text{Cb} / \text{SubWidthC}, y\text{Cb} / \text{SubHeightC})$ , the width  $\text{nCbW}$  set equal to the chroma coding block width  $\text{cbWidth} / \text{SubWidthC}$ , the height  $\text{nCbH}$  set equal to the chroma coding block height  $\text{cbHeight} / \text{SubHeightC}$ , the width  $\text{nTbW}$  set equal to the chroma coding block width  $\text{cbWidth} / \text{SubWidthC}$ , the height  $\text{nTbH}$  set equal to the chroma coding block height  $\text{cbHeight} / \text{SubHeightC}$  and the variable  $\text{cldx}$  set equal to 2 as inputs, and the array  $\text{resSamples}_{\text{Cr}}$  as output.
  - When  $\text{cu\_act\_enabled\_flag}[x\text{Cb}][y\text{Cb}]$  is equal to 1, the residual modification process for residual blocks using colour space conversion as specified in subclause 8.7.4.6 is invoked with the variable  $\text{nTbW}$  set equal to  $\text{cbWidth}$ , the variable  $\text{nTbH}$  set equal to  $\text{cbHeight}$ , the array  $r_{\text{Y}}$  set equal to  $\text{resSamples}_{\text{L}}$ , the array  $r_{\text{Cb}}$  set equal to  $\text{resSamples}_{\text{Cb}}$ , and the array  $r_{\text{Cr}}$



set equal to  $\text{resSamples}_{Cr}$  as inputs, and the output are modified versions of the arrays  $\text{resSamples}_L$ ,  $\text{resSamples}_{Cb}$  and  $\text{resSamples}_{Cr}$ .

7. The reconstructed samples of the current coding unit are derived as follows:

- The picture reconstruction process for a colour component as specified in subclause 8.7.5.1 is invoked with the block location  $(x_{Curr}, y_{Curr})$  set equal to  $(x_{Cb}, y_{Cb})$ , the block width  $n_{CurrSw}$  set equal to  $cbWidth$ , the block height  $n_{CurrSh}$  set equal to  $cbHeight$ , the variable  $cldx$  set equal to 0, the  $(cbWidth) \times (cbHeight)$  array  $\text{predSamples}$  set equal to  $\text{predSamples}_L$  and the  $(cbWidth) \times (cbHeight)$  array  $\text{resSamples}$  set equal to  $\text{resSamples}_L$  as inputs, and the output is a modified reconstructed picture before in-loop filtering.
- When  $\text{sps\_chroma\_format\_idc}$  is not equal to 0, the decoding process for the reconstructed signal of chroma coding blocks coded in inter prediction mode as specified in subclause 8.5.9 is invoked with the transform block location  $(x_{Tb0}, y_{Tb0})$  set equal to  $(x_{Cb} / \text{SubWidthC}, y_{Cb} / \text{SubHeightC})$ , the transform block width  $n_{TbW}$  set equal to  $cbWidth / \text{SubWidthC}$  and the height  $n_{TbH}$  set equal to  $cbHeight / \text{SubHeightC}$ , the variable  $cldx$  set equal to 1, the  $(cbWidth / \text{SubWidthC}) \times (cbHeight / \text{SubHeightC})$  array  $\text{predSamples}$  set equal to  $\text{predSamples}_{Cb}$  and the  $(cbWidth / \text{SubWidthC}) \times (cbHeight / \text{SubHeightC})$  array  $\text{resSamples}$  set equal to  $\text{resSamples}_{Cb}$  as inputs, and the output is a modified reconstructed picture before in-loop filtering.
- When  $\text{sps\_chroma\_format\_idc}$  is not equal to 0, the decoding process for the reconstructed signal of chroma coding blocks coded in inter prediction mode as specified in subclause 8.5.9 is invoked with the transform block location  $(x_{Tb0}, y_{Tb0})$  set equal to  $(x_{Cb} / \text{SubWidthC}, y_{Cb} / \text{SubHeightC})$ , the transform block width  $n_{TbW}$  set equal to  $cbWidth / \text{SubWidthC}$  and the height  $n_{TbH}$  set equal to  $cbHeight / \text{SubHeightC}$ , the variable  $cldx$  set equal to 2, the  $(cbWidth / \text{SubWidthC}) \times (cbHeight / \text{SubHeightC})$  array  $\text{predSamples}$  set equal to  $\text{predSamples}_{Cr}$  and the  $(cbWidth / \text{SubWidthC}) \times (cbHeight / \text{SubHeightC})$  array  $\text{resSamples}$  set equal to  $\text{resSamples}_{Cr}$  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  $(x_{Cb}, y_{Cb})$  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  $\text{mvL0}[0][0]$  and  $\text{mvL1}[0][0]$ ,
- the reference indices  $\text{refIdxL0}$  and  $\text{refIdxL1}$ ,
- the prediction list utilization flags  $\text{predFlagL0}[0][0]$  and  $\text{predFlagL1}[0][0]$ ,
- the half sample interpolation filter index  $\text{hpelIdx}$ ,
- the bi-prediction weight index  $\text{bcwIdx}$ .

For the derivation of the variables  $\text{mvL0}[0][0]$  and  $\text{mvL1}[0][0]$ ,  $\text{refIdxL0}$  and  $\text{refIdxL1}$ , as well as  $\text{predFlagL0}[0][0]$  and  $\text{predFlagL1}[0][0]$ , the following applies:

- If  $\text{general\_merge\_flag}[x_{Cb}][y_{Cb}]$  is equal to 1, the derivation process for luma motion vectors for merge mode as specified in subclause 8.5.2.2 is invoked with the luma location  $(x_{Cb}, y_{Cb})$  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  $hpelIdx$ , 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  $predFlagLX[0][0]$ ,  $mvLX[0][0]$ ,  $refIdxLX$ ,  $PRED\_LX$ ,  $ref\_idx\_IX$  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:

$$refIdxLX = ref\_idx\_IX[xCb][yCb] \quad (459)$$

$$predFlagLX[0][0] = 1 \quad (460)$$

— Otherwise, the variables  $refIdxLX$  and  $predFlagLX[0][0]$  are specified by:

$$refIdxLX = -1 \quad (461)$$

$$predFlagLX[0][0] = 0 \quad (462)$$

2. The variable  $mvdLX$  is derived as follows:

$$mvdLX[0] = MvdLX[xCb][yCb][0] \quad (463)$$

$$mvdLX[1] = MvdLX[xCb][yCb][1] \quad (464)$$

3. When  $predFlagLX[0][0]$  is equal to 1, the derivation process for luma motion vector prediction in subclause 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:

$$uLX[0] = (mvpLX[0] + mvdLX[0]) \& (2^{18} - 1) \quad (465)$$

$$mvLX[0][0][0] = (uLX[0] \geq 2^{17}) ? (uLX[0] - 2^{18}) : uLX[0] \quad (466)$$

$$uLX[1] = (mvpLX[1] + mvdLX[1]) \& (2^{18} - 1) \quad (467)$$

$$mvLX[0][0][1] = (uLX[1] \geq 2^{17}) ? (uLX[1] - 2^{18}) : uLX[1] \quad (468)$$

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  $hpelIdx$  is derived as follows:

$$hpelIdx = AmvrShift = 3 ? 1 : 0 \quad (469)$$

— 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  $(xCb + cbWidth) \gg \text{Log2ParMrgLevel}$  is greater than  $xCb \gg \text{Log2ParMrgLevel}$  and  $(yCb + cbHeight) \gg \text{Log2ParMrgLevel}$  is greater than  $yCb \gg \text{Log2ParMrgLevel}$ , the updating process for the history-based motion vector predictor list as specified in subclause 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  $hpellfIdx$ .

### 8.5.2.2 Derivation process for luma motion vectors for merge mode

This process is only invoked when  $\text{general\_merge\_flag}[xCb][yCb]$  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  $hpellfIdx$ ,
- the bi-prediction weight index  $bcwIdx$ ,
- the merging candidate list  $\text{mergeCandList}$ .

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 subclause 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  $\text{availableFlagA}_0$ ,  $\text{availableFlagA}_1$ ,  $\text{availableFlagB}_0$ ,  $\text{availableFlagB}_1$  and  $\text{availableFlagB}_2$ , 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  $hpellfIdxA_0$ ,  $hpellfIdxA_1$ ,  $hpellfIdxB_0$ ,  $hpellfIdxB_1$ ,  $hpellfIdxB_2$ , 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  $hpellfIdxCol$  is set equal to 0.
3. The derivation process for temporal luma motion vector prediction as specified in subclause 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:

$$\text{availableFlagCol} = \text{availableFlagL0Col} \quad (470)$$

$$\text{predFlagL0Col} = \text{availableFlagL0Col} \quad (471)$$

$$\text{predFlagL1Col} = 0 \quad (472)$$

4. When `sh_slice_type` is equal to B, the derivation process for temporal luma motion vector prediction as specified in subclause 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:

$$\text{availableFlagCol} = \text{availableFlagL0Col} \mid \mid \text{availableFlagL1Col} \quad (473)$$

$$\text{predFlagL1Col} = \text{availableFlagL1Col} \quad (474)$$

5. The merging candidate list, `mergeCandList`, is constructed as follows:

$$\begin{aligned} & i = 0 \\ & \text{if( availableFlagB}_1 \text{ )} \\ & \quad \text{mergeCandList[ i++ ]} = B_1 \\ & \text{if( availableFlagA}_1 \text{ )} \\ & \quad \text{mergeCandList[ i++ ]} = A_1 \\ & \text{if( availableFlagB}_0 \text{ )} \\ & \quad \text{mergeCandList[ i++ ]} = B_0 \\ & \text{if( availableFlagA}_0 \text{ )} \\ & \quad \text{mergeCandList[ i++ ]} = A_0 \\ & \text{if( availableFlagB}_2 \text{ )} \\ & \quad \text{mergeCandList[ i++ ]} = B_2 \\ & \text{if( availableFlagCol )} \\ & \quad \text{mergeCandList[ i++ ]} = \text{Col} \end{aligned} \quad (475)$$

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 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 subclause 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 `hpelIdxN` 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 hpellfldxavgCand 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 subclause 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<sub>m</sub> and refIdxL1zeroCand<sub>m</sub>, the prediction list utilization flags predFlagL0zeroCand<sub>m</sub> and predFlagL1zeroCand<sub>m</sub> and the motion vectors mvL0zeroCand<sub>m</sub> and mvL1zeroCand<sub>m</sub> of every new candidate zeroCand<sub>m</sub> being added into mergeCandList. The half sample interpolation filter index hpellfldx of every new candidate zeroCand<sub>m</sub> being added into mergeCandList is set equal to 0. The bi-prediction weight index bcwIdx of every new candidate zeroCand<sub>m</sub> 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 ] ] ):
 

$$\text{hpellfldx} = \text{hpellfldxN} \quad (476)$$

$$\text{bcwIdx} = \text{bcwIdxN} \quad (477)$$

— For X = 0..1, the following applies:

$$\text{refIdxLX} = \text{refIdxLXN} \quad (478)$$

$$\text{predFlagLX}[ 0 ][ 0 ] = \text{predFlagLXN} \quad (479)$$

$$\text{mvLX}[ 0 ][ 0 ][ 0 ] = \text{mvLXN}[ 0 ] \quad (480)$$

$$\text{mvLX}[ 0 ][ 0 ][ 1 ] = \text{mvLXN}[ 1 ] \quad (481)$$

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 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:

$$\text{mvLX}[ 0 ][ 0 ][ 0 ] += \text{mMvdLX}[ 0 ] \quad (482)$$

$$\text{mvLX}[ 0 ][ 0 ][ 1 ] += \text{mMvdLX}[ 1 ] \quad (483)$$

$$\text{mvLX}[ 0 ][ 0 ][ 0 ] = \text{Clip3}( -2^{17}, 2^{17} - 1, \text{mvLX}[ 0 ][ 0 ][ 0 ] ) \quad (484)$$

$$\text{mvLX}[ 0 ][ 0 ][ 1 ] = \text{Clip3}( -2^{17}, 2^{17} - 1, \text{mvLX}[ 0 ][ 0 ][ 1 ] ) \quad (485)$$

### 8.5.2.3 Derivation process for spatial merging candidates

Inputs to this process are:

- a luma location (  $x_{Cb}, y_{Cb}$  ) 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  $hpelIdxA_0$ ,  $hpelIdxA_1$ ,  $hpelIdxB_0$ ,  $hpelIdxB_1$ , and  $hpelIdxB_2$ ,
- the bi-prediction weight indices  $bcwIdxA_0$ ,  $bcwIdxA_1$ ,  $bcwIdxB_0$ ,  $bcwIdxB_1$ , and  $bcwIdxB_2$ .

For the derivation of  $availableFlagB_1$ ,  $refIdxLXB_1$ ,  $predFlagLXB_1$ ,  $mvLXB_1$ ,  $hpelIdxB_1$  and  $bcwIdxB_1$  the following applies:

- The luma location (  $x_{NbB_1}, y_{NbB_1}$  ) inside the neighbouring luma coding block is set equal to (  $x_{Cb} + cbWidth - 1, y_{Cb} - 1$  ).
- The derivation process for neighbouring block availability as specified in subclause 6.4.4 is invoked with the current luma location (  $x_{Curr}, y_{Curr}$  ) set equal to (  $x_{Cb}, y_{Cb}$  ), the neighbouring luma location (  $x_{NbB_1}, y_{NbB_1}$  ),  $checkPredModeY$  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  $x_{Cb} \gg \log_2 ParMrgLevel$  is equal to  $x_{NbB_1} \gg \log_2 ParMrgLevel$  and  $y_{Cb} \gg \log_2 ParMrgLevel$  is equal to  $y_{NbB_1} \gg \log_2 ParMrgLevel$ ,  $availableB_1$  is set equal to FALSE.
- The variables  $availableFlagB_1$ ,  $refIdxLXB_1$ ,  $predFlagLXB_1$ ,  $mvLXB_1$ ,  $hpelIdxB_1$  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  $X = 0..1$ ,  $hpelIdxB_1$  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:

$$mvLXB_1 = MvLX[ x_{NbB_1} ][ y_{NbB_1} ] \quad (486)$$

$$refIdxLXB_1 = RefIdxLX[ x_{NbB_1} ][ y_{NbB_1} ] \quad (487)$$

$$predFlagLXB_1 = PredFlagLX[ x_{NbB_1} ][ y_{NbB_1} ] \quad (488)$$

$$hpelIdxB_1 = HpelIdx[ x_{NbB_1} ][ y_{NbB_1} ] \quad (489)$$

$$bcwIdxB_1 = BcwIdx[ x_{NbB_1} ][ y_{NbB_1} ] \quad (490)$$

For the derivation of availableFlagA<sub>1</sub>, refIdxLXA<sub>1</sub>, predFlagLXA<sub>1</sub>, mvLXA<sub>1</sub>, hpellIdxA<sub>1</sub> and bcwIdxA<sub>1</sub> the following applies:

- The luma location ( xNbA<sub>1</sub>, yNbA<sub>1</sub> ) 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 subclause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring luma location ( xNbA<sub>1</sub>, yNbA<sub>1</sub> ), checkPredModeY set equal to TRUE, and cldx set equal to 0 as inputs, and the output is assigned to the block availability flag availableA<sub>1</sub>.
- When xCb >> Log2ParMrgLevel is equal to xNbA<sub>1</sub> >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to yNbA<sub>1</sub> >> Log2ParMrgLevel, availableA<sub>1</sub> is set equal to FALSE.
- The variables availableFlagA<sub>1</sub>, refIdxLXA<sub>1</sub>, predFlagLXA<sub>1</sub>, mvLXA<sub>1</sub>, hpellIdxA<sub>1</sub> and bcwIdxA<sub>1</sub> are derived as follows:
  - If one or more of the following conditions are true, availableFlagA<sub>1</sub> is set equal to 0, both components of mvLXA<sub>1</sub> are set equal to 0, refIdxLXA<sub>1</sub> is set equal to -1 and predFlagLXA<sub>1</sub> is set equal to 0, with X = 0..1, hpellIdxA<sub>1</sub> is set equal to 0, and bcwIdxA<sub>1</sub> is set equal to 0:
    - availableA<sub>1</sub> is equal to FALSE.
    - availableB<sub>1</sub> is equal to TRUE and the luma locations ( xNbA<sub>1</sub>, yNbA<sub>1</sub> ) and ( xNbB<sub>1</sub>, yNbB<sub>1</sub> ) have the same motion vectors and the same reference indices.
  - Otherwise, availableFlagA<sub>1</sub> is set equal to 1 and the following assignments are made:
 
$$\text{mvLXA}_1 = \text{MvLX}[ \text{xNbA}_1 ][ \text{yNbA}_1 ] \quad (491)$$

$$\text{refIdxLXA}_1 = \text{RefIdxLX}[ \text{xNbA}_1 ][ \text{yNbA}_1 ] \quad (492)$$

$$\text{predFlagLXA}_1 = \text{PredFlagLX}[ \text{xNbA}_1 ][ \text{yNbA}_1 ] \quad (493)$$

$$\text{hpellIdxA}_1 = \text{HpellIdx}[ \text{xNbA}_1 ][ \text{yNbA}_1 ] \quad (494)$$

$$\text{bcwIdxA}_1 = \text{BcwIdx}[ \text{xNbA}_1 ][ \text{yNbA}_1 ] \quad (495)$$

For the derivation of availableFlagB<sub>0</sub>, refIdxLXB<sub>0</sub>, predFlagLXB<sub>0</sub>, mvLXB<sub>0</sub>, hpellIdxB<sub>0</sub> and bcwIdxB<sub>0</sub> the following applies:

- The luma location ( xNbB<sub>0</sub>, yNbB<sub>0</sub> ) inside the neighbouring luma coding block is set equal to ( xCb + cbWidth, yCb - 1 ).
- The derivation process for neighbouring block availability as specified in subclause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring luma location ( xNbB<sub>0</sub>, yNbB<sub>0</sub> ), checkPredModeY set equal to TRUE, and cldx set equal to 0 as inputs, and the output is assigned to the block availability flag availableB<sub>0</sub>.
- When xCb >> Log2ParMrgLevel is equal to xNbB<sub>0</sub> >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to yNbB<sub>0</sub> >> Log2ParMrgLevel, availableB<sub>0</sub> is set equal to FALSE.
- The variables availableFlagB<sub>0</sub>, refIdxLXB<sub>0</sub>, predFlagLXB<sub>0</sub>, mvLXB<sub>0</sub>, hpellIdxB<sub>0</sub> and bcwIdxB<sub>0</sub> are derived as follows:
  - If one or more of the following conditions are true, availableFlagB<sub>0</sub> is set equal to 0, both components of mvLXB<sub>0</sub> are set equal to 0, refIdxLXB<sub>0</sub> is set equal to -1 and predFlagLXB<sub>0</sub> is set equal to 0, with X = 0..1, hpellIdxB<sub>0</sub> is set equal to 0, and bcwIdxB<sub>0</sub> is set equal to 0:
    - availableB<sub>0</sub> is equal to FALSE.



- availableB<sub>1</sub> is equal to TRUE and the luma locations ( xNbB<sub>1</sub>, yNbB<sub>1</sub> ) and ( xNbB<sub>0</sub>, yNbB<sub>0</sub> ) have the same motion vectors and the same reference indices.
- Otherwise, availableFlagB<sub>0</sub> is set equal to 1 and the following assignments are made:

$$mvLXB_0 = MvLX[ xNbB_0 ][ yNbB_0 ] \quad (496)$$

$$refIdxLXB_0 = RefIdxLX[ xNbB_0 ][ yNbB_0 ] \quad (497)$$

$$predFlagLXB_0 = PredFlagLX[ xNbB_0 ][ yNbB_0 ] \quad (498)$$

$$hpelIdxB_0 = HpelIdx[ xNbB_0 ][ yNbB_0 ] \quad (499)$$

$$bcwIdxB_0 = BcwIdx[ xNbB_0 ][ yNbB_0 ] \quad (500)$$

For the derivation of availableFlagA<sub>0</sub>, refIdxLXA<sub>0</sub>, predFlagLXA<sub>0</sub>, mvLXA<sub>0</sub>, hpelIdxA<sub>0</sub> and bcwIdxA<sub>0</sub> the following applies:

- The luma location ( xNbA<sub>0</sub>, yNbA<sub>0</sub> ) inside the neighbouring luma coding block is set equal to ( xCb - 1, yCb + cbHeight ).
- The derivation process for neighbouring block availability as specified in subclause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring luma location ( xNbA<sub>0</sub>, yNbA<sub>0</sub> ), checkPredModeY set equal to TRUE, and cldx set equal to 0 as inputs, and the output is assigned to the block availability flag availableA<sub>0</sub>.
- When xCb >> Log2ParMrgLevel is equal to xNbA<sub>0</sub> >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to yNbA<sub>0</sub> >> Log2ParMrgLevel, availableA<sub>0</sub> is set equal to FALSE.
- The variables availableFlagA<sub>0</sub>, refIdxLXA<sub>0</sub>, predFlagLXA<sub>0</sub>, mvLXA<sub>0</sub>, hpelIdxA<sub>0</sub> and bcwIdxA<sub>0</sub> are derived as follows:
  - If one or more of the following conditions are true, availableFlagA<sub>0</sub> is set equal to 0, both components of mvLXA<sub>0</sub> are set equal to 0, refIdxLXA<sub>0</sub> is set equal to -1 and predFlagLXA<sub>0</sub> is set equal to 0, with X = 0..1, hpelIdxA<sub>0</sub> is set equal to 0, and bcwIdxA<sub>0</sub> is set equal to 0:
    - availableA<sub>0</sub> is equal to FALSE.
    - availableA<sub>1</sub> is equal to TRUE and the luma locations ( xNbA<sub>1</sub>, yNbA<sub>1</sub> ) and ( xNbA<sub>0</sub>, yNbA<sub>0</sub> ) have the same motion vectors and the same reference indices.
  - Otherwise, availableFlagA<sub>0</sub> is set equal to 1 and the following assignments are made:

$$mvLXA_0 = MvLX[ xNbA_0 ][ yNbA_0 ] \quad (501)$$

$$refIdxLXA_0 = RefIdxLX[ xNbA_0 ][ yNbA_0 ] \quad (502)$$

$$predFlagLXA_0 = PredFlagLX[ xNbA_0 ][ yNbA_0 ] \quad (503)$$

$$hpelIdxA_0 = HpelIdx[ xNbA_0 ][ yNbA_0 ] \quad (504)$$

$$bcwIdxA_0 = BcwIdx[ xNbA_0 ][ yNbA_0 ] \quad (505)$$

For the derivation of availableFlagB<sub>2</sub>, refIdxLXB<sub>2</sub>, predFlagLXB<sub>2</sub>, mvLXB<sub>2</sub>, hpelIdxB<sub>2</sub> and bcwIdxB<sub>2</sub> the following applies:

- The luma location ( xNbB<sub>2</sub>, yNbB<sub>2</sub> ) inside the neighbouring luma coding block is set equal to ( xCb - 1, yCb - 1 ).

- The derivation process for neighbouring block availability as specified in subclause 6.4.4 is invoked with the current luma location (  $x_{Curr}, y_{Curr}$  ) set equal to (  $x_{Cb}, y_{Cb}$  ), the neighbouring luma location (  $x_{NbB_2}, y_{NbB_2}$  ), checkPredModeY set equal to TRUE, and cldx set equal to 0 as inputs, and the output is assigned to the block availability flag availableB<sub>2</sub>.
- When  $x_{Cb} \gg \text{Log2ParMrgLevel}$  is equal to  $x_{NbB_2} \gg \text{Log2ParMrgLevel}$  and  $y_{Cb} \gg \text{Log2ParMrgLevel}$  is equal to  $y_{NbB_2} \gg \text{Log2ParMrgLevel}$ , availableB<sub>2</sub> is set equal to FALSE.
- The variables availableFlagB<sub>2</sub>, refldxLXB<sub>2</sub>, predFlagLXB<sub>2</sub>, mvLXB<sub>2</sub>, hpellfldxB<sub>2</sub> and bcwIdxB<sub>2</sub> are derived as follows:
  - If one or more of the following conditions are true, availableFlagB<sub>2</sub> is set equal to 0, both components of mvLXB<sub>2</sub> are set equal to 0, refldxLXB<sub>2</sub> is set equal to -1 and predFlagLXB<sub>2</sub> is set equal to 0, with X = 0..1, hpellfldxB<sub>2</sub> is set equal to 0, and bcwIdxB<sub>2</sub> is set equal to 0:
    - availableB<sub>2</sub> is equal to FALSE.
    - availableA<sub>1</sub> is equal to TRUE and the luma locations (  $x_{NbA_1}, y_{NbA_1}$  ) and (  $x_{NbB_2}, y_{NbB_2}$  ) have the same motion vectors and the same reference indices.
    - availableB<sub>1</sub> is equal to TRUE and the luma locations (  $x_{NbB_1}, y_{NbB_1}$  ) and (  $x_{NbB_2}, y_{NbB_2}$  ) have the same motion vectors and the same reference indices.
    - availableFlagA<sub>0</sub> + availableFlagA<sub>1</sub> + availableFlagB<sub>0</sub> + availableFlagB<sub>1</sub> is equal to 4.
  - Otherwise, availableFlagB<sub>2</sub> is set equal to 1 and the following assignments are made:
 
$$mvLXB_2 = MvLX[ x_{NbB_2} ][ y_{NbB_2} ] \quad (506)$$

$$refldxLXB_2 = RefldxLX[ x_{NbB_2} ][ y_{NbB_2} ] \quad (507)$$

$$predFlagLXB_2 = PredFlagLX[ x_{NbB_2} ][ y_{NbB_2} ] \quad (508)$$

$$hpellfldxB_2 = Hpellfldx[ x_{NbB_2} ][ y_{NbB_2} ] \quad (509)$$

$$bcwIdxB_2 = BcwIdx[ x_{NbB_2} ][ y_{NbB_2} ] \quad (510)$$

#### 8.5.2.4 Derivation process for pairwise average merging candidate

Inputs to this process are:

- a merging candidate list mergeCandList,
- the reference indices refldxL0N and refldxL1N 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 hpellfldxN 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 refldxL0avgCand and refldxL1avgCand 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 `hpellIdxavgCand` of every candidate `avgCand` added into `mergeCandList` during the invocation of this process.

The variable `numRefLists` is derived as follows:

$$\text{numRefLists} = (\text{sh\_slice\_type} == B) ? 2 : 1 \quad (511)$$

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`:

$$\text{p0Cand} = \text{mergeCandList}[0] \quad (512)$$

$$\text{p1Cand} = \text{mergeCandList}[1] \quad (513)$$

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:

$$\text{refIdxLXavgCand} = \text{refIdxLXp0Cand} \quad (514)$$

$$\text{predFlagLXavgCand} = 1 \quad (515)$$

- The rounding process for motion vectors as specified in subclause 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:

$$\text{refIdxLXavgCand} = \text{refIdxLXp0Cand} \quad (516)$$

$$\text{predFlagLXavgCand} = 1 \quad (517)$$

$$\text{mvLXavgCand}[0] = \text{mvLXp0Cand}[0] \quad (518)$$

$$\text{mvLXavgCand}[1] = \text{mvLXp0Cand}[1] \quad (519)$$

- 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:

$$\text{refIdxLXavgCand} = \text{refIdxLXp1Cand} \quad (520)$$

$$\text{predFlagLXavgCand} = 1 \quad (521)$$

$$\text{mvLXavgCand}[0] = \text{mvLXp1Cand}[0] \quad (522)$$

$$\text{mvLXavgCand}[1] = \text{mvLXp1Cand}[1] \quad (523)$$

- Otherwise, if  $\text{predFlagLXp0Cand}$  is equal to 0 and  $\text{predFlagLXp1Cand}$  is equal to 0, the variables  $\text{refIdxLXavgCand}$ ,  $\text{predFlagLXavgCand}$ ,  $\text{mvLXavgCand}[0]$ ,  $\text{mvLXavgCand}[1]$  are derived as follows:

$$\text{refIdxLXavgCand} = -1 \quad (524)$$

$$\text{predFlagLXavgCand} = 0 \quad (525)$$

$$\text{mvLXavgCand}[0] = 0 \quad (526)$$

$$\text{mvLXavgCand}[1] = 0 \quad (527)$$

- When  $\text{numRefLists}$  is equal to 1, the following applies:

$$\text{refIdxL1avgCand} = -1 \quad (528)$$

$$\text{predFlagL1avgCand} = 0 \quad (529)$$

- The half sample interpolation filter index  $\text{hpellfIdxavgCand}$  is derived as follows:

- If  $\text{hpellfIdxp0Cand}$  is equal to  $\text{hpellfIdxp1Cand}$ ,  $\text{hpellfIdxavgCand}$  is set equal to  $\text{hpellfIdxp0Cand}$ .

- Otherwise,  $\text{hpellfIdxavgCand}$  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  $\text{mergeCandList}$ ,
- the reference indices  $\text{refIdxL0N}$  and  $\text{refIdxL1N}$  of every candidate  $N$  in  $\text{mergeCandList}$ ,
- the prediction list utilization flags  $\text{predFlagL0N}$  and  $\text{predFlagL1N}$  of every candidate  $N$  in  $\text{mergeCandList}$ ,
- the motion vectors  $\text{mvL0N}$  and  $\text{mvL1N}$  of every candidate  $N$  in  $\text{mergeCandList}$ ,
- the number of elements  $\text{numCurrMergeCand}$  within  $\text{mergeCandList}$ .

Outputs of this process are:

- the merging candidate list  $\text{mergeCandList}$ ,
- the number of elements  $\text{numCurrMergeCand}$  within  $\text{mergeCandList}$ ,
- the reference indices  $\text{refIdxL0zeroCand}_m$  and  $\text{refIdxL1zeroCand}_m$  of every new candidate  $\text{zeroCand}_m$  added into  $\text{mergeCandList}$  during the invocation of this process,
- the prediction list utilization flags  $\text{predFlagL0zeroCand}_m$  and  $\text{predFlagL1zeroCand}_m$  of every new candidate  $\text{zeroCand}_m$  added into  $\text{mergeCandList}$  during the invocation of this process,
- the motion vectors  $\text{mvL0zeroCand}_m$  and  $\text{mvL1zeroCand}_m$  of every new candidate  $\text{zeroCand}_m$  added into  $\text{mergeCandList}$  during the invocation of this process.

The variable  $\text{numRefIdx}$  is derived as follows:

- If  $\text{sh\_slice\_type}$  is equal to  $P$ ,  $\text{numRefIdx}$  is set equal to  $\text{NumRefIdxActive}[0]$ .

- 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<sub>m</sub> with m equal to ( numCurrMergeCand – numInputMergeCand ) is added at the end of mergeCandList, i.e., mergeCandList[ numCurrMergeCand ] is set equal to zeroCand<sub>m</sub>, and the reference indices, the prediction list utilization flags and the motion vectors of zeroCand<sub>m</sub> are derived as follows and numCurrMergeCand is incremented by 1:

$$\text{refIdxL0zeroCand}_m = ( \text{zeroIdx} < \text{numRefIdx} ) ? \text{zeroIdx} : 0 \quad (530)$$

$$\text{refIdxL1zeroCand}_m = -1 \quad (531)$$

$$\text{predFlagL0zeroCand}_m = 1 \quad (532)$$

$$\text{predFlagL1zeroCand}_m = 0 \quad (533)$$

$$\text{mvL0zeroCand}_m[ 0 ] = 0 \quad (534)$$

$$\text{mvL0zeroCand}_m[ 1 ] = 0 \quad (535)$$

$$\text{mvL1zeroCand}_m[ 0 ] = 0 \quad (536)$$

$$\text{mvL1zeroCand}_m[ 1 ] = 0 \quad (537)$$

$$\text{numCurrMergeCand} = \text{numCurrMergeCand} + 1 \quad (538)$$

- Otherwise (sh\_slice\_type is equal to B), the candidate zeroCand<sub>m</sub> with m equal to ( numCurrMergeCand – numInputMergeCand ) is added at the end of mergeCandList, i.e., mergeCandList[ numCurrMergeCand ] is set equal to zeroCand<sub>m</sub>, and the reference indices, the prediction list utilization flags and the motion vectors of zeroCand<sub>m</sub> are derived as follows and numCurrMergeCand is incremented by 1:

$$\text{refIdxL0zeroCand}_m = ( \text{zeroIdx} < \text{numRefIdx} ) ? \text{zeroIdx} : 0 \quad (539)$$

$$\text{refIdxL1zeroCand}_m = ( \text{zeroIdx} < \text{numRefIdx} ) ? \text{zeroIdx} : 0 \quad (540)$$

$$\text{predFlagL0zeroCand}_m = 1 \quad (541)$$

$$\text{predFlagL1zeroCand}_m = 1 \quad (542)$$

$$\text{mvL0zeroCand}_m[ 0 ] = 0 \quad (543)$$

$$\text{mvL0zeroCand}_m[ 1 ] = 0 \quad (544)$$

$$\text{mvL1zeroCand}_m[ 0 ] = 0 \quad (545)$$

$$\text{mvL1zeroCand}_m[ 1 ] = 0 \quad (546)$$

$$\text{numCurrMergeCand} = \text{numCurrMergeCand} + 1 \quad (547)$$

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<sub>1</sub> or B<sub>1</sub>, 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:

$$\text{currPocDiffL0} = \text{DiffPicOrderCnt}(\text{currPic}, \text{RefPicList}[0][\text{refIdxL0}]) \quad (549)$$

$$\text{currPocDiffL1} = \text{DiffPicOrderCnt}(\text{currPic}, \text{RefPicList}[1][\text{refIdxL1}]) \quad (550)$$

- If currPocDiffL0 is equal to currPocDiffL1, the following applies:

$$\text{mMvdL0}[0] = \text{MmvdOffset}[\text{xCb}][\text{yCb}][0] \quad (551)$$



$$mMvdL0[1] = MmvdOffset[xCb][yCb][1] \quad (552)$$

$$mMvdL1[0] = MmvdOffset[xCb][yCb][0] \quad (553)$$

$$mMvdL1[1] = MmvdOffset[xCb][yCb][1] \quad (554)$$

— Otherwise, if  $Abs(currPocDiffL0)$  is greater than or equal to  $Abs(currPocDiffL1)$ , the following applies:

$$mMvdL0[0] = MmvdOffset[xCb][yCb][0] \quad (555)$$

$$mMvdL0[1] = MmvdOffset[xCb][yCb][1] \quad (556)$$

— 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:

$$td = Clip3(-128, 127, currPocDiffL0) \quad (557)$$

$$tb = Clip3(-128, 127, currPocDiffL1) \quad (558)$$

$$tx = (16384 + (Abs(td) >> 1)) / td \quad (559)$$

$$distScaleFactor = Clip3(-4096, 4095, (tb * tx + 32) >> 6) \quad (560)$$

$$mMvdL1[0] = Clip3(-2^{17}, 2^{17} - 1, (distScaleFactor * mMvdL0[0] + 128 - (distScaleFactor * mMvdL0[0] >= 0)) >> 8) \quad (561)$$

$$mMvdL1[1] = Clip3(-2^{17}, 2^{17} - 1, (distScaleFactor * mMvdL0[1] + 128 - (distScaleFactor * mMvdL0[1] >= 0)) >> 8) \quad (562)$$

— Otherwise, the following applies:

$$mMvdL1[0] = Sign(currPocDiffL0) == Sign(currPocDiffL1) ? mMvdL0[0] : -mMvdL0[0] \quad (563)$$

$$mMvdL1[1] = Sign(currPocDiffL0) == Sign(currPocDiffL1) ? mMvdL0[1] : -mMvdL0[1] \quad (564)$$

— Otherwise ( $Abs(currPocDiffL0)$  is less than  $Abs(currPocDiffL1)$ ), the following applies:

$$mMvdL1[0] = MmvdOffset[xCb][yCb][0] \quad (565)$$

$$mMvdL1[1] = MmvdOffset[xCb][yCb][1] \quad (566)$$

— 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:

$$td = Clip3(-128, 127, currPocDiffL1) \quad (567)$$

$$tb = Clip3(-128, 127, currPocDiffL0) \quad (568)$$

$$tx = (16384 + (Abs(td) >> 1)) / td \quad (569)$$

$$distScaleFactor = Clip3(-4096, 4095, (tb * tx + 32) >> 6) \quad (570)$$

$$mMvdL0[0] = \text{Clip3}(-2^{17}, 2^{17} - 1, (\text{distScaleFactor} * mMvdL1[0] + 128 - (\text{distScaleFactor} * mMvdL1[0] \geq 0)) \gg 8) \quad (571)$$

$$mMvdL0[1] = \text{Clip3}(-2^{17}, 2^{17} - 1, (\text{distScaleFactor} * mMvdL1[1] + 128 - (\text{distScaleFactor} * mMvdL1[1] \geq 0)) \gg 8) \quad (572)$$

— Otherwise, the following applies:

$$mMvdL0[0] = \text{Sign}(\text{currPocDiffL0}) == \text{Sign}(\text{currPocDiffL1}) ? mMvdL1[0] : -mMvdL1[0] \quad (573)$$

$$mMvdL0[1] = \text{Sign}(\text{currPocDiffL0}) == \text{Sign}(\text{currPocDiffL1}) ? mMvdL1[1] : -mMvdL1[1] \quad (574)$$

— Otherwise (predFlagL0 or predFlagL1 is equal to 1), the following applies for X = 0, 1:

$$mMvdLX[0] = (\text{predFlagLX} == 1) ? MmvdOffset[xCb][yCb][0] : 0 \quad (575)$$

$$mMvdLX[1] = (\text{predFlagLX} == 1) ? MmvdOffset[xCb][yCb][1] : 0 \quad (576)$$

#### 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 subclause 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, mvPLstLX, as output.
2. The motion vector predictor mvPLX is derived as follows:

$$mvPLX = mvPLstLX[mvp\_lX\_flag[xCb][yCb]] \quad (577)$$

#### 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 mvPLstLX in 1/16 fractional-sample accuracy with X being 0 or 1.

The motion vector predictor candidate list  $mvListLX$  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 subclause 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 subclause 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 subclause 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 subclause 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,  $mvListLX$ , is constructed as follows:
 

```

numCurrMvpCand = 0
if( availableFlagLXA ) {
    mvListLX[ numCurrMvpCand++ ] = mvLXA
    if( availableFlagLXB && ( mvLXA != mvLXB ) )
        mvListLX[ numCurrMvpCand++ ] = mvLXB
} else if( availableFlagLXB )
    mvListLX[ numCurrMvpCand++ ] = mvLXB
if( numCurrMvpCand < 2 && availableFlagLXCol )
    mvListLX[ numCurrMvpCand++ ] = mvLXCol
      
```

(578)
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 subclause 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  $mvListLX[ numCurrMvpCand++ ]$ .
6. When  $numCurrMvpCand$  is less than 2, the following applies until  $numCurrMvpCand$  is equal to 2:

$\text{mvpListLX}[\text{numCurrMvpCand}][0] = 0$  (579)

$\text{mvpListLX}[\text{numCurrMvpCand}][1] = 0$  (580)

$\text{numCurrMvpCand}++$  (581)

#### 8.5.2.10 Derivation process for spatial motion vector predictor candidates

Inputs to this process are:

- a luma location  $(x_{Cb}, y_{Cb})$  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  $\text{refIdxLX}$ , with  $X$  being 0 or 1.

Outputs of this process are (with  $N$  being replaced by  $A$  or  $B$ ):

- the motion vectors  $\text{mvLXN}$  in 1/16 fractional-sample accuracy of the neighbouring coding units,
- the availability flags  $\text{availableFlagLXN}$  of the neighbouring coding units.

Figure 10 provides an overview of spatial motion vector neighbours.

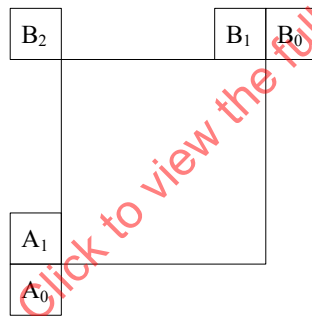


**Figure 10 — Spatial motion vector neighbours**

The motion vector  $\text{mvLXA}$  and the availability flag  $\text{availableFlagLXA}$  are derived in the following ordered steps:

1. The sample location  $(x_{NbA_0}, y_{NbA_0})$  is set equal to  $(x_{Cb} - 1, y_{Cb} + cbHeight)$  and the sample location  $(x_{NbA_1}, y_{NbA_1})$  is set equal to  $(x_{NbA_0}, y_{NbA_0} - 1)$ .
2. The availability flag  $\text{availableFlagLXA}$  is set equal to 0 and both components of  $\text{mvLXA}$  are set equal to 0.
3. The derivation process for neighbouring block availability as specified in subclause 6.4.4 is invoked with the current luma location  $(x_{Curr}, y_{Curr})$  set equal to  $(x_{Cb}, y_{Cb})$ , the neighbouring luma location  $(x_{NbA_0}, y_{NbA_0})$ ,  $\text{checkPredModeY}$  set equal to TRUE, and  $\text{cldx}$  set equal to 0 as inputs, and the output is assigned to the block availability flag  $\text{availableA}_0$ .
4. The derivation process for neighbouring block availability as specified in subclause 6.4.4 is invoked with the current luma location  $(x_{Curr}, y_{Curr})$  set equal to  $(x_{Cb}, y_{Cb})$ , the neighbouring luma location  $(x_{NbA_1}, y_{NbA_1})$ ,  $\text{checkPredModeY}$  set equal to TRUE, and  $\text{cldx}$  set equal to 0 as inputs, and the output is assigned to the block availability flag  $\text{availableA}_1$ .

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:

$$mvLXA = MvLX[xNbA_k][yNbA_k] \quad (582)$$

— 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:

$$mvLXA = MvLY[xNbA_k][yNbA_k] \quad (583)$$

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 subclause 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  $cldx$  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:

$$mvLXB = MvLX[xNbB_k][yNbB_k] \quad (584)$$

— 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:

$$mvLXB = MvLY[xNbB_k][yNbB_k] \quad (585)$$

### 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:

$$xColBr = xCb + cbWidth \quad (586)$$

$$yColBr = yCb + cbHeight \quad (587)$$

$$\begin{aligned} rightBoundaryPos &= sps\_subpic\_treated\_as\_pic\_flag[ CurrSubpicIdx ] ? \\ &\quad SubpicRightBoundaryPos : pps\_pic\_width\_in\_luma\_samples - 1 \end{aligned} \quad (588)$$

$$\begin{aligned} botBoundaryPos &= sps\_subpic\_treated\_as\_pic\_flag[ CurrSubpicIdx ] ? \\ &\quad SubpicBotBoundaryPos : pps\_pic\_height\_in\_luma\_samples - 1 \end{aligned} \quad (589)$$

- 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 subclause 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:

$$xColCtr = xCb + ( cbWidth >> 1 ) \quad (590)$$

$$yColCtr = yCb + ( cbHeight >> 1 ) \quad (591)$$

- 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 subclause 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:

$$\text{colPocDiff} = \text{DiffPicOrderCnt}(\text{ColPic}, \text{refPicList}[\text{listCol}][\text{refIdxCol}]) \quad (592)$$

$$\text{currPocDiff} = \text{DiffPicOrderCnt}(\text{currPic}, \text{RefPicList}[X][\text{refIdxLX}]) \quad (593)$$

- The temporal motion buffer compression process for collocated motion vectors as specified in subclause 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:

$$\text{mvLXCol} = \text{Clip3}(-131072, 131071, \text{mvCol}) \quad (594)$$

- Otherwise, mvLXCol is derived as a scaled version of the motion vector mvCol as follows:

$$\text{tx} = (16384 + (\text{Abs}(\text{td}) \gg 1)) / \text{td} \quad (595)$$

$$\text{distScaleFactor} = \text{Clip3}(-4096, 4095, (\text{tb} * \text{tx} + 32) \gg 6) \quad (596)$$

$$\text{mvLXCol} = \text{Clip3}(-131072, 131071, (\text{distScaleFactor} * \text{mvCol} + 128 - (\text{distScaleFactor} * \text{mvCol} \geq 0)) \gg 8) \quad (597)$$

where td and tb are derived as follows:

$$\text{td} = \text{Clip3}(-128, 127, \text{colPocDiff}) \quad (598)$$

$$\text{tb} = \text{Clip3}(-128, 127, \text{currPocDiff}) \quad (599)$$

### 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:

$$\text{mvCLX}[0] = \text{mvLX}[0] * 2 / \text{SubWidthC} \quad (600)$$

$$\text{mvCLX}[1] = \text{mvLX}[1] * 2 / \text{SubHeightC} \quad (601)$$

### 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:

$$\text{offset} = (\text{rightShift} == 0) ? 0 : ( (1 \ll (\text{rightShift} - 1)) - 1 ) \quad (602)$$

$$\text{mvX}[0] = \text{Sign}(\text{mvX}[0]) * ((\text{Abs}(\text{mvX}[0]) + \text{offset}) \gg \text{rightShift}) \ll \text{leftShift} \quad (603)$$

$$\text{mvX}[1] = \text{Sign}(\text{mvX}[1]) * ((\text{Abs}(\text{mvX}[1]) + \text{offset}) \gg \text{rightShift}) \ll \text{leftShift} \quad (604)$$

### 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:

$$s = \text{mv}[\text{compIdx}] \gg 17 \quad (605)$$

$$f = \text{Floor}(\text{Log2}((\text{mv}[\text{compIdx}] \wedge s) | 31)) - 4 \quad (606)$$

$$\text{mask} = (-1 \ll f) \gg 1 \quad (607)$$

$$\text{round} = (1 \ll f) \gg 2 \quad (608)$$

$$\text{mv}[\text{compIdx}] = (\text{mv}[\text{compIdx}] + \text{round}) \& \text{mask} \quad (609)$$

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 hpelIdx.

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  $hpelIdx$ .

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 minimum 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  $(sbWidth + 2 * srRange) \times (sbHeight + 2 * srRange)$  array  $predSamplesLX_L$  of prediction luma sample values is derived by invoking the fractional sample bilinear interpolation process specified in 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 subclause 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 subclause 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 subclause 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:
 
$$dMvL0[0] += 16 * intOffX \quad (610)$$

$$dMvL0[1] += 16 * intOffY \quad (611)$$
  - When  $subPelFlag$  is equal to 1, the parametric motion vector refinement process specified in subclause 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:
 
$$dMvL1[0] = -dMvL0[0] \quad (612)$$

$$dMvL1[1] = -dMvL0[1] \quad (613)$$

### 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  $(predWidth) \times (predHeight)$  array  $predSamplesLX_L$  of luma prediction sample values.

Let  $(x_{\text{Int}_L}, y_{\text{Int}_L})$  be a luma location given in full-sample units and  $(x_{\text{Frac}_L}, y_{\text{Frac}_L})$  be an offset given in 1/16-sample units. These variables are used only in this subclause for specifying fractional-sample locations inside the reference sample array  $\text{refPicLXL}$ .

For each luma sample location  $(x_L = 0..\text{predWidth} - 1, y_L = 0..\text{predHeight} - 1)$  inside the luma prediction sample array  $\text{predSamplesLXL}$ , the corresponding luma prediction sample value  $\text{predSamplesLXL}[x_L][y_L]$  is derived as follows:

— The variables  $x_{\text{Int}_L}, y_{\text{Int}_L}, x_{\text{Frac}_L}$  and  $y_{\text{Frac}_L}$  are derived as follows:

$$x_{\text{Int}_L} = x_{\text{Sb}} + (\text{mvLX}[0] \gg 4) + x_L - \text{srRange} \quad (614)$$

$$y_{\text{Int}_L} = y_{\text{Sb}} + (\text{mvLX}[1] \gg 4) + y_L - \text{srRange} \quad (615)$$

$$x_{\text{Frac}_L} = \text{mvLX}[0] \& 15 \quad (616)$$

$$y_{\text{Frac}_L} = \text{mvLX}[1] \& 15 \quad (617)$$

— The luma prediction sample value  $\text{predSamplesLXL}[x_L][y_L]$  is derived by invoking the luma sample bilinear interpolation process specified in subclause 8.5.3.2.2 with  $(x_{\text{Int}_L}, y_{\text{Int}_L}), (x_{\text{Frac}_L}, y_{\text{Frac}_L})$ , and  $\text{refPicLXL}$  as inputs.

#### 8.5.3.2.2 Luma sample bilinear interpolation process

Inputs to this process are:

- a luma location in full-sample units  $(x_{\text{Int}_L}, y_{\text{Int}_L})$ ,
- a luma location in fractional-sample units  $(x_{\text{Frac}_L}, y_{\text{Frac}_L})$ ,
- the luma reference sample array  $\text{refPicLXL}$ .

Output of this process is a predicted luma sample value  $\text{predSampleLXL}$ .

The variables  $\text{shift1}, \text{shift2}, \text{shift3}, \text{shift4}, \text{offset1}, \text{offset2}$  and  $\text{offset4}$  are derived as follows:

$$\text{shift1} = \text{BitDepth} - 6 \quad (618)$$

$$\text{offset1} = 1 \ll (\text{shift1} - 1) \quad (619)$$

$$\text{shift2} = 4 \quad (620)$$

$$\text{offset2} = 1 \ll (\text{shift2} - 1) \quad (621)$$

$$\text{shift3} = 10 - \text{BitDepth} \quad (622)$$

$$\text{shift4} = \text{BitDepth} - 10 \quad (623)$$

$$\text{offset4} = 1 \ll (\text{shift4} - 1) \quad (624)$$

The variable  $\text{picW}$  is set equal to  $\text{pps\_pic\_width\_in\_luma\_samples}$  of the reference picture  $\text{refPicLX}$  and the variable  $\text{picH}$  is set equal to  $\text{pps\_pic\_height\_in\_luma\_samples}$  of the reference picture  $\text{refPicLX}$ .

The luma interpolation filter coefficients  $\text{fb}_L[p]$  for each 1/16 fractional sample position  $p$  equal to  $x_{\text{Frac}_L}$  or  $y_{\text{Frac}_L}$  are specified in Table 26.

The luma locations in full-sample units  $(x_{\text{Int}_i}, y_{\text{Int}_i})$  are derived as follows for  $i = 0..1$ :

- If  $\text{sps\_subpic\_treated\_as\_pic\_flag}[\text{CurrSubpicIdx}]$  is equal to 1 and  $\text{sps\_num\_subpics\_minus1}$  for the reference picture  $\text{refPicLX}$  is greater than 0, the following applies:



$$\begin{aligned} xInt_i &= \text{Clip3}(\text{SubpicLeftBoundaryPos}, \text{SubpicRightBoundaryPos}, \\ &\quad \text{pps\_ref\_wraparound\_enabled\_flag?} \\ &\quad \text{ClipH}((\text{PpsRefWraparoundOffset}) * \text{MinCbSizeY}, \text{picW}, (xInt_L + i)) : xInt_L + i) \end{aligned} \quad (625)$$

$$yInt_i = \text{Clip3}(\text{SubpicTopBoundaryPos}, \text{SubpicBotBoundaryPos}, yInt_L + i) \quad (626)$$

- Otherwise ( $\text{sps\_subpic\_treated\_as\_pic\_flag}[\text{CurrSubpicIdx}]$  is equal to 0 or  $\text{sps\_num\_subpics\_minus1}$  for the reference picture  $\text{refPicLX}$  is equal to 0), the following applies:

$$\begin{aligned} xInt_i &= \text{Clip3}(0, \text{picW} - 1, \text{pps\_ref\_wraparound\_enabled\_flag?} \\ &\quad \text{ClipH}((\text{PpsRefWraparoundOffset}) * \text{MinCbSizeY}, \text{picW}, (xInt_L + i)) : xInt_L + i) \end{aligned} \quad (627)$$

$$yInt_i = \text{Clip3}(0, \text{picH} - 1, yInt_L + i) \quad (628)$$

The predicted luma sample value  $\text{predSampleLX}_L$  is derived as follows:

If both  $x\text{Frac}_L$  and  $y\text{Frac}_L$  are equal to 0, the value of  $\text{predSampleLX}_L$  is derived as follows:

$$\begin{aligned} \text{predSampleLX}_L &= \text{BitDepth} \leq 10 ? (\text{refPicLX}_L[xInt_0][yInt_0] \ll \text{shift3}) : \\ &\quad ((\text{refPicLX}_L[xInt_0][yInt_0] + \text{offset4}) \gg \text{shift4}) \end{aligned} \quad (629)$$

- Otherwise, if  $x\text{Frac}_L$  is not equal to 0 and  $y\text{Frac}_L$  is equal to 0, the value of  $\text{predSampleLXL}$  is derived as follows:

$$\text{predSampleLXL} = ((\sum_{i=0}^1 \text{fb}_L[x\text{Frac}_L][i] * \text{refPicLXL}[xInt_i][yInt_0]) + \text{offset1}) \gg \text{shift1} \quad (630)$$

- Otherwise, if  $x\text{Frac}_L$  is equal to 0 and  $y\text{Frac}_L$  is not equal to 0, the value of  $\text{predSampleLXL}$  is derived as follows:

$$\text{predSampleLXL} = ((\sum_{i=0}^1 \text{fb}_L[y\text{Frac}_L][i] * \text{refPicLXL}[xInt_0][yInt_i]) + \text{offset1}) \gg \text{shift1} \quad (631)$$

- Otherwise, if  $x\text{Frac}_L$  is not equal to 0 and  $y\text{Frac}_L$  is not equal to 0, the value of  $\text{predSampleLXL}$  is derived as follows:

- The sample array  $\text{temp}[n]$  with  $n = 0..1$ , is derived as follows:

$$\text{temp}[n] = ((\sum_{i=0}^1 \text{fb}_L[x\text{Frac}_L][i] * \text{refPicLXL}[xInt_i][yInt_n]) + \text{offset1}) \gg \text{shift1} \quad (632)$$

- The predicted luma sample value  $\text{predSampleLX}_L$  is derived as follows:

$$\text{predSampleLX}_L = ((\sum_{i=0}^1 \text{fb}_L[y\text{Frac}_L][i] * \text{temp}[i]) + \text{offset2}) \gg \text{shift2} \quad (633)$$

**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  $(nSbW + 4) \times (nSbH + 4)$  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:

$$sad = \sum_{x=0}^{nSbW-1} \sum_{y=0}^{nSbH/2-1} Abs(pL0[x+2+dX][2*y+2+dY] - pL1[x+2-dX][2*y+2-dY]) \quad (634)$$

When both  $dX$  and  $dY$  are equal to 0, the value of  $sad$  is modified as follows:

$$sad = sad - (sad >> 2) \quad (635)$$

#### 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 ]
            intOffX = dX
            intOffY = dY
        }
    }
}

```

(636)

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 subclause 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 subclause 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:

$$\text{dMvL0}[0] += \text{dMvX} \quad (637)$$

$$\text{dMvL0}[1] += \text{dMvY} \quad (638)$$

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
    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  $\text{num} = (\text{sadMinus} - \text{sadPlus}) \ll 3$  by  $\text{denom} = (\text{sadMinus} + \text{sadPlus} - (\text{sadCenter} \ll 1))$ . Given the fact that  $\text{sadMinus}$ ,  $\text{sadCenter}$ , and  $\text{sadPlus}$  are all positive, and  $\text{sadCenter}$  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 (  $x_{Cb}, y_{Cb}$  ) 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 subclause 8.5.4.2 is invoked with the luma location (  $x_{Cb}, y_{Cb}$  ), 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 subclause 8.5.2.13 is invoked with mvA as input, and the output being mvCA.

The derivation process for chroma motion vectors in subclause 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 subclause 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:
 
$$m = \text{merge\_gpm\_idx0}[ \text{xCb} ][ \text{yCb} ] \quad (640)$$

$$n = \text{merge\_gpm\_idx1}[ \text{xCb} ][ \text{yCb} ] + ( ( \text{merge\_gpm\_idx1}[ \text{xCb} ][ \text{yCb} ] \geq m ) ? 1 : 0 ) \quad (641)$$
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:

$$\text{mvA}[ 0 ] = \text{mvLXM}[ 0 ] \quad (642)$$

$$\text{mvA}[ 1 ] = \text{mvLXM}[ 1 ] \quad (643)$$

$$\text{refIdxA} = \text{refIdxLXM} \quad (644)$$

$$\text{predListFlagA} = X \quad (645)$$

7. Let  $\text{refIdxL0N}$  and  $\text{refIdxL1N}$ ,  $\text{predFlagL0N}$  and  $\text{predFlagL1N}$ , and  $\text{mvL0N}$  and  $\text{mvL1N}$  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  $\text{mergeCandList}$  ( $N = \text{mergeCandList}[n]$ ).
8. The variable  $X$  is set equal to  $(n \& 0x01)$ .
9. When  $\text{predFlagLXN}$  is equal to 0,  $X$  is set equal to  $(1 - X)$ .
10. The following applies:

$\text{mvB}[0] = \text{mvLXN}[0]$  (646)

$\text{mvB}[1] = \text{mvLXN}[1]$  (647)

$\text{refIdxB} = \text{refIdxLXN}$  (648)

$\text{predListFlagB} = X$  (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  $(x_{Cb}, y_{Cb})$  of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
- a variable  $\text{cbWidth}$  specifying the width of the current coding block in luma samples,
- a variable  $\text{cbHeight}$  specifying the height of the current coding block in luma samples.

Outputs of this process are:

- the reference indices  $\text{refIdxL0}$  and  $\text{refIdxL1}$ ,
- the number of luma subblocks in horizontal direction  $\text{numSbX}$  and in vertical direction  $\text{numSbY}$ ,
- the prediction list utilization flag arrays  $\text{predFlagL0}[x_{SbIdx}][y_{SbIdx}]$  and  $\text{predFlagL1}[x_{SbIdx}][y_{SbIdx}]$  with  $x_{SbIdx} = 0..\text{numSbX} - 1$ ,  $y_{SbIdx} = 0..\text{numSbY} - 1$ ,
- the luma subblock motion vector arrays in 1/16 fractional-sample accuracy  $\text{mvL0}[x_{SbIdx}][y_{SbIdx}]$  and  $\text{mvL1}[x_{SbIdx}][y_{SbIdx}]$  with  $x_{SbIdx} = 0..\text{numSbX} - 1$ ,  $y_{SbIdx} = 0..\text{numSbY} - 1$ ,
- the chroma subblock motion vector arrays in 1/32 fractional-sample accuracy  $\text{mvCL0}[x_{SbIdx}][y_{SbIdx}]$  and  $\text{mvCL1}[x_{SbIdx}][y_{SbIdx}]$  with  $x_{SbIdx} = 0..\text{numSbX} - 1$ ,  $y_{SbIdx} = 0..\text{numSbY} - 1$ ,
- the bi-prediction weight index  $\text{bcwIdx}$ ,
- the prediction refinement utilization flags  $\text{cbProfFlagL0}$ ,  $\text{cbProfFlagL1}$ ,
- the motion vector difference arrays  $\text{diffMvL0}[x_{Idx}][y_{Idx}]$  and  $\text{diffMvL1}[x_{Idx}][y_{Idx}]$  with  $x_{Idx} = 0..\text{cbWidth}/\text{numSbX} - 1$ ,  $y_{Idx} = 0..\text{cbHeight}/\text{numSbY} - 1$ .

The variables  $\text{cbProfFlagL0}$  and  $\text{cbProfFlagL1}$  are initialized to be equal to zero.

For the derivation of the variables  $\text{mvL0}[x_{SbIdx}][y_{SbIdx}]$ ,  $\text{mvL1}[x_{SbIdx}][y_{SbIdx}]$ ,  $\text{mvCL0}[x_{SbIdx}][y_{SbIdx}]$  and  $\text{mvCL1}[x_{SbIdx}][y_{SbIdx}]$ ,  $\text{refIdxL0}$ ,  $\text{refIdxL1}$ ,  $\text{numSbX}$ ,  $\text{numSbY}$ ,  $\text{predFlagL0}[x_{SbIdx}][y_{SbIdx}]$  and  $\text{predFlagL1}[x_{SbIdx}][y_{SbIdx}]$ , the following applies:

- If  $\text{merge\_subblock\_flag}[x_{Cb}][y_{Cb}]$  is equal to 1, the derivation process for motion vectors and reference indices in subblock merge mode as specified in 8.5.5.2 is invoked with the luma coding block location  $(x_{Cb}, y_{Cb})$ , the luma coding block width  $\text{cbWidth}$  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:
 
$$\text{refIdxLX} = \text{ref\_idx\_lX}[ \text{xCb} ][ \text{yCb} ] \quad (650)$$

$$\text{predFlagLX}[ 0 ][ 0 ] = 1 \quad (651)$$
      - Otherwise, the variables refIdxLX and predFlagLX are specified as follows:
 
$$\text{refIdxLX} = -1 \quad (652)$$

$$\text{predFlagLX}[ 0 ][ 0 ] = 0 \quad (653)$$
    2. The variable mvdCpLX[ 0 ] is derived as follows:
 
$$\text{mvdCpLX}[ 0 ][ 0 ] = \text{MvdCpLX}[ \text{xCb} ][ \text{yCb} ][ 0 ][ 0 ] \quad (654)$$

$$\text{mvdCpLX}[ 0 ][ 1 ] = \text{MvdCpLX}[ \text{xCb} ][ \text{yCb} ][ 0 ][ 1 ] \quad (655)$$
    3. The variable mvdCpLX[ cpIdx ] with cpIdx ranging from 1 to numCpMv - 1, is derived as follows:
 
$$\text{mvdCpLX}[ \text{cpIdx} ][ 0 ] = \text{MvdCpLX}[ \text{xCb} ][ \text{yCb} ][ \text{cpIdx} ][ 0 ] + \text{mvdCpLX}[ 0 ][ 0 ] \quad (656)$$

$$\text{mvdCpLX}[ \text{cpIdx} ][ 1 ] = \text{MvdCpLX}[ \text{xCb} ][ \text{yCb} ][ \text{cpIdx} ][ 1 ] + \text{mvdCpLX}[ 0 ][ 1 ] \quad (657)$$
    4. When predFlagLX[ 0 ][ 0 ] is equal to 1, the derivation process for luma affine control point motion vector predictors as specified in subclause 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:
 
$$\text{uLX}[ \text{cpIdx} ][ 0 ] = ( \text{mvpCpLX}[ \text{cpIdx} ][ 0 ] + \text{mvdCpLX}[ \text{cpIdx} ][ 0 ] ) \& ( 2^{18} - 1 ) \quad (658)$$

$$\text{cpMvLX}[ \text{cpIdx} ][ 0 ] = ( \text{uLX}[ \text{cpIdx} ][ 0 ] \geq 2^{17} ) ? ( \text{uLX}[ \text{cpIdx} ][ 0 ] - 2^{18} ) : \text{uLX}[ \text{cpIdx} ][ 0 ] \quad (659)$$

$$\text{uLX}[ \text{cpIdx} ][ 1 ] = ( \text{mvpCpLX}[ \text{cpIdx} ][ 1 ] + \text{mvdCpLX}[ \text{cpIdx} ][ 1 ] ) \& ( 2^{18} - 1 ) \quad (660)$$

$$\text{cpMvLX}[ \text{cpIdx} ][ 1 ] = ( \text{uLX}[ \text{cpIdx} ][ 1 ] \geq 2^{17} ) ? ( \text{uLX}[ \text{cpIdx} ][ 1 ] - 2^{18} ) : \text{uLX}[ \text{cpIdx} ][ 1 ] \quad (661)$$
- The variables numSbX and numSbY are derived as follows:

$$\text{numSbX} = (\text{cbWidth} \gg 2) \quad (662)$$

$$\text{numSbY} = (\text{cbHeight} \gg 2) \quad (663)$$

— For  $\text{xSbIdx} = 0..\text{numSbX} - 1$ ,  $\text{ySbIdx} = 0..\text{numSbY} - 1$ , the following applies:

$$\text{predFlagLX}[\text{xSbIdx}][\text{ySbIdx}] = \text{predFlagLX}[0][0] \quad (664)$$

— For  $X = 0..1$ , the following applies:

— When  $\text{predFlagLX}[0][0]$  is equal to 1, the derivation process for motion vector arrays from affine control point motion vectors as specified in subclause 8.5.5.9 is invoked with the luma coding block location  $(\text{xCb}, \text{yCb})$ , the luma coding block width  $\text{cbWidth}$ , the luma coding block height  $\text{cbHeight}$ , the number of control point motion vectors  $\text{numCpMv}$ , the control point motion vectors  $\text{cpMvLX}[\text{cpIdx}]$  with  $\text{cpIdx}$  being  $0..\text{numCpMv} - 1$ , the prediction list utilization flags  $\text{predFlagL0}[0][0]$  and  $\text{predFlagL1}[0][0]$ , the reference index  $\text{refIdxLX}$  and the number of luma subblocks in horizontal direction  $\text{numSbX}$  and in vertical direction  $\text{numSbY}$  as inputs, the luma motion vector array  $\text{mvLX}[\text{xSbIdx}][\text{ySbIdx}]$ , the chroma motion vector array  $\text{mvCLX}[\text{xSbIdx}][\text{ySbIdx}]$  with  $\text{xSbIdx} = 0..\text{numSbX} - 1$ ,  $\text{ySbIdx} = 0..\text{numSbY} - 1$ , the prediction refinement utility flag  $\text{cbProfFlagLX}$ , and motion vector difference array  $\text{diffMvLX}[\text{xIdx}][\text{yIdx}]$  with  $\text{xIdx} = 0..\text{cbWidth} / \text{numSbX} - 1$ ,  $\text{yIdx} = 0..\text{cbHeight} / \text{numSbY} - 1$  as outputs.

— The bi-prediction weight index  $\text{bcwIdx}$  is set equal to  $\text{bcw\_idx}[\text{xCb}][\text{yCb}]$ .

#### 8.5.5.2 Derivation process for motion vectors and reference indices in subblock merge mode

Inputs to this process are:

- a luma location  $(\text{xCb}, \text{yCb})$  of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
- two variables  $\text{cbWidth}$  and  $\text{cbHeight}$  specifying the width and the height of the luma coding block.

Outputs of this process are:

- the number of luma subblocks in horizontal direction  $\text{numSbX}$  and in vertical direction  $\text{numSbY}$ ,
- the reference indices  $\text{refIdxL0}$  and  $\text{refIdxL1}$ ,
- the prediction list utilization flag arrays  $\text{predFlagL0}[\text{xSbIdx}][\text{ySbIdx}]$  and  $\text{predFlagL1}[\text{xSbIdx}][\text{ySbIdx}]$  with  $\text{xSbIdx} = 0..\text{numSbX} - 1$ ,  $\text{ySbIdx} = 0..\text{numSbY} - 1$ ,
- the luma subblock motion vector arrays in 1/16 fractional-sample accuracy  $\text{mvL0}[\text{xSbIdx}][\text{ySbIdx}]$  and  $\text{mvL1}[\text{xSbIdx}][\text{ySbIdx}]$  with  $\text{xSbIdx} = 0..\text{numSbX} - 1$ ,  $\text{ySbIdx} = 0..\text{numSbY} - 1$ ,
- the chroma subblock motion vector arrays in 1/32 fractional-sample accuracy  $\text{mvCL0}[\text{xSbIdx}][\text{ySbIdx}]$  and  $\text{mvCL1}[\text{xSbIdx}][\text{ySbIdx}]$  with  $\text{xSbIdx} = 0..\text{numSbX} - 1$ ,  $\text{ySbIdx} = 0..\text{numSbY} - 1$ ,
- the prediction refinement utilization flags  $\text{cbProfFlagL0}$  and  $\text{cbProfFlagL1}$ ,
- the motion vector difference arrays  $\text{diffMvL0}[\text{xIdx}][\text{yIdx}]$  and  $\text{diffMvL1}[\text{xIdx}][\text{yIdx}]$  with  $\text{xIdx} = 0..\text{cbWidth} / \text{numSbX} - 1$ ,  $\text{yIdx} = 0..\text{cbHeight} / \text{numSbY} - 1$ ,
- the bi-prediction weight index  $\text{bcwIdx}$ .

The variables  $\text{numSbColX}$ ,  $\text{numSbColY}$  and the subblock merging candidate list,  $\text{subblockMergeCandList}$  are derived by the following ordered steps:

1. The variables  $\text{availableFlagSbCol}$ ,  $\text{availableFlagA}$ ,  $\text{availableFlagB}$ , and  $\text{availableFlagConstK}$  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 `availableFlagA1`, `refIdxLXA1`, `predFlagLXA1` and `mvLXA1` the following applies:
  - The luma location ( `xNbA1`, `yNbA1` ) 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 subclause 6.4.4 is invoked with the current luma location ( `xCurr`, `yCurr` ) set equal to ( `xCb`, `yCb` ), the neighbouring luma location ( `xNbA1`, `yNbA1` ), `checkPredModeY` set equal to TRUE, and `cIdx` set equal to 0 as inputs, and the output is assigned to the block availability flag `availableA1`.
  - When `xCb >> Log2ParMrgLevel` is equal to `xNbA1 >> Log2ParMrgLevel` and `yCb >> Log2ParMrgLevel` is equal to `yNbA1 >> Log2ParMrgLevel`, `availableA1` is set equal to FALSE.
  - The variables `availableFlagA1`, `refIdxLXA1`, `predFlagLXA1` and `mvLXA1` are derived as follows:
    - If `availableA1` is equal to FALSE, `availableFlagA1` is set equal to 0, both components of `mvLXA1` are set equal to 0, `refIdxLXA1` is set equal to -1 and `predFlagLXA1` is set equal to 0, with `X = 0..1`, and `bcwIdxA1` is set equal to 0.
    - Otherwise, `availableFlagA1` is set equal to 1 and the following assignments are made:
 
$$\text{mvLXA}_1 = \text{MvLX}[ \text{xNbA}_1 ][ \text{yNbA}_1 ] \quad (665)$$

$$\text{refIdxLXA}_1 = \text{RefIdxLX}[ \text{xNbA}_1 ][ \text{yNbA}_1 ] \quad (666)$$

$$\text{predFlagLXA}_1 = \text{PredFlagLX}[ \text{xNbA}_1 ][ \text{yNbA}_1 ] \quad (667)$$
- The derivation process for subblock-based temporal merging candidates as specified in subclause 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 `availableFlagA1`, the reference index `refIdxLXA1`, the prediction list utilization flag `predFlagLXA1`, and the motion vector `mvLXA1` 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 ( `xNbA0`, `yNbA0` ), ( `xNbA1`, `yNbA1` ), ( `xNbA2`, `yNbA2` ), ( `xNbB0`, `yNbB0` ), ( `xNbB1`, `yNbB1` ), ( `xNbB2`, `yNbB2` ), and ( `xNbB3`, `yNbB3` ) are derived as follows:

$$( \text{xNbA}_0, \text{yNbA}_0 ) = ( \text{xCb} - 1, \text{yCb} + \text{cbHeight} ) \quad (668)$$

$$( \text{xNbA}_1, \text{yNbA}_1 ) = ( \text{xCb} - 1, \text{yCb} + \text{cbHeight} - 1 ) \quad (669)$$

$$( \text{xNbA}_2, \text{yNbA}_2 ) = ( \text{xCb} - 1, \text{yCb} ) \quad (670)$$

$$( \text{xNbB}_0, \text{yNbB}_0 ) = ( \text{xCb} + \text{cbWidth}, \text{yCb} - 1 ) \quad (671)$$

$$( \text{xNbB}_1, \text{yNbB}_1 ) = ( \text{xCb} + \text{cbWidth} - 1, \text{yCb} - 1 ) \quad (672)$$

$$( \text{xNbB}_2, \text{yNbB}_2 ) = ( \text{xCb} - 1, \text{yCb} - 1 ) \quad (673)$$

$$( \text{xNbB}_3, \text{yNbB}_3 ) = ( \text{xCb}, \text{yCb} - 1 ) \quad (674)$$

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 subclause 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 `cldx` set equal to 0 as inputs, and the output is assigned to the block availability flag `availableAk`.
- When  $xCb \gg \text{Log2ParMrgLevel}$  is equal to  $xNbA_k \gg \text{Log2ParMrgLevel}$  and  $yCb \gg \text{Log2ParMrgLevel}$  is equal to  $yNbA_k \gg \text{Log2ParMrgLevel}$ , `availableAk` is set equal to FALSE.
- When `availableAk` is equal to TRUE and `MotionModelIdc[xNbAk][yNbAk]` 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[xNbAk][yNbAk]`,  $(xNb, yNb)$  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[xNbAk][yNbAk]` is equal to 1, the derivation process for luma affine control point motion vectors from a neighbouring block as specified in subclause 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:

$$\text{predFlagLXA} = \text{PredFlagLX}[xNbA_k][yNbA_k] \quad (675)$$

$$\text{refIdxLXA} = \text{RefIdxLX}[xNbA_k][yNbA_k] \quad (676)$$

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 subclause 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 `cldx` set equal to 0 as inputs, and the output is assigned to the block availability flag `availableBk`.
- When  $xCb \gg \text{Log2ParMrgLevel}$  is equal to  $xNbB_k \gg \text{Log2ParMrgLevel}$  and  $yCb \gg \text{Log2ParMrgLevel}$  is equal to  $yNbB_k \gg \text{Log2ParMrgLevel}$ , `availableBk` is set equal to FALSE.
- When `availableBk` 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  $(CbPosX[0][xNbB_k][yNbB_k], CbPosY[0][xNbB_k][yNbB_k])$ , `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  $\text{PredFlagLX}[xNbB_k][yNbB_k]$  is equal to TRUE, the derivation process for luma affine control point motion vectors from a neighbouring block as specified in subclause 8.5.5.5 is invoked with the luma coding block location  $(x_{Cb}, y_{Cb})$ , the luma coding block width and height  $(cbWidth, cbHeight)$ , the neighbouring luma coding block location  $(x_{Nb}, y_{Nb})$ , 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  $\text{sps\_affine\_enabled\_flag}$  is equal to 1, the following applies:

— The derivation process for neighbouring block availability as specified in subclause 6.4.4 is invoked with the current luma location  $(x_{Curr}, y_{Curr})$  set equal to  $(x_{Cb}, y_{Cb})$ , the neighbouring luma location  $(x_{NbA_2}, y_{NbA_2})$ ,  $\text{checkPredModeY}$  set equal to TRUE, and  $clIdx$  set equal to 0 as inputs, and the output is assigned to the block availability flag  $\text{availableA}_2$ .

— When  $x_{Cb} \gg \text{Log2ParMrgLevel}$  is equal to  $x_{NbA_2} \gg \text{Log2ParMrgLevel}$  and  $y_{Cb} \gg \text{Log2ParMrgLevel}$  is equal to  $y_{NbA_2} \gg \text{Log2ParMrgLevel}$ ,  $\text{availableA}_2$  is set equal to FALSE.

— The derivation process for neighbouring block availability as specified in subclause 6.4.4 is invoked with the current luma location  $(x_{Curr}, y_{Curr})$  set equal to  $(x_{Cb}, y_{Cb})$ , the neighbouring luma location  $(x_{NbB_3}, y_{NbB_3})$ ,  $\text{checkPredModeY}$  set equal to TRUE, and  $clIdx$  set equal to 0 as inputs, and the output is assigned to the block availability flag  $\text{availableB}_3$ .

— When  $x_{Cb} \gg \text{Log2ParMrgLevel}$  is equal to  $x_{NbB_3} \gg \text{Log2ParMrgLevel}$  and  $y_{Cb} \gg \text{Log2ParMrgLevel}$  is equal to  $y_{NbB_3} \gg \text{Log2ParMrgLevel}$ ,  $\text{availableB}_3$  is set equal to FALSE.

— The derivation process for constructed affine control point motion vector merging candidates as specified in subclause 8.5.5.6 is invoked with the luma coding block location  $(x_{Cb}, y_{Cb})$ , the luma coding block width and height  $(cbWidth, cbHeight)$ , the availability flags  $\text{availableA}_0$ ,  $\text{availableA}_1$ ,  $\text{availableA}_2$ ,  $\text{availableB}_0$ ,  $\text{availableB}_1$ ,  $\text{availableB}_2$  and  $\text{availableB}_3$ , and the sample locations  $(x_{NbA_0}, y_{NbA_0})$ ,  $(x_{NbA_1}, y_{NbA_1})$ ,  $(x_{NbA_2}, y_{NbA_2})$ ,  $(x_{NbB_0}, y_{NbB_0})$ ,  $(x_{NbB_1}, y_{NbB_1})$ ,  $(x_{NbB_2}, y_{NbB_2})$  and  $(x_{NbB_3}, y_{NbB_3})$  as inputs, and the availability flags  $\text{availableFlagConstK}$ , the reference indices  $\text{refIdxLXConstK}$ , prediction list utilization flags  $\text{predFlagLXConstK}$ , motion model indices  $\text{motionModelIdxConstK}$ , bi-prediction weight indices  $\text{bcwIdxConstK}$  and  $\text{cpMvLXConstK}[cpIdx]$  with  $X = 0..1$ ,  $K = 1..6$  and  $cpIdx = 0..2$  as outputs.

7. The initial subblock merging candidate list,  $\text{subblockMergeCandList}$ , 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
if( availableFlagConst2 && i < MaxNumSubblockMergeCand )
    subblockMergeCandList[ i++ ] = Const2

```

(679)



```

        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<sub>m</sub> with m equal to ( numCurrMergeCand – numOrigMergeCand ) are derived as follows:

$$\text{refIdxL0ZeroCand}_m = 0 \quad (680)$$

$$\text{predFlagL0ZeroCand}_m = 1 \quad (681)$$

$$\text{cpMvL0ZeroCand}_m[ 0 ] = \text{mvZero} \quad (682)$$

$$\text{cpMvL0ZeroCand}_m[ 1 ] = \text{mvZero} \quad (683)$$

$$\text{refIdxL1ZeroCand}_m = ( \text{sh\_slice\_type} == B ) ? 0 : -1 \quad (684)$$

$$\text{predFlagL1ZeroCand}_m = ( \text{sh\_slice\_type} == B ) ? 1 : 0 \quad (685)$$

$$\text{cpMvL1ZeroCand}_m[ 0 ] = \text{mvZero} \quad (686)$$

$$\text{cpMvL1ZeroCand}_m[ 1 ] = \text{mvZero} \quad (687)$$

$$\text{motionModelIdxZeroCand}_m = 1 \quad (688)$$

$$\text{bcwIdxZeroCand}_m = 0 \quad (689)$$

- The candidate zeroCand<sub>m</sub> with m equal to ( numCurrMergeCand – numOrigMergeCand ) is added at the end of subblockMergeCandList and numCurrMergeCand is incremented by 1 as follows:

$$\text{subblockMergeCandList}[ \text{numCurrMergeCand}++ ] = \text{zeroCand}_m \quad (690)$$

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:

$$\text{numSbX} = \text{cbWidth} >> 2 \quad (691)$$

$$\text{numSbY} = \text{cbHeight} >> 2 \quad (692)$$

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  $\text{subblockMergeCandList}[\text{merge\_subblock\_idx}[\text{xCb}][\text{yCb}]]$  is equal to  $\text{SbCol}$ , the bi-prediction weight index  $\text{bcwIdx}$  is set equal to 0 and the following applies for  $X = 0..1$ :

$$\text{refIdxLX} = \text{refIdxLXSbCol} \quad (693)$$

- For  $\text{xSbIdx} = 0..\text{numSbX} - 1$ ,  $\text{ySbIdx} = 0..\text{numSbY} - 1$ , the following applies:

$$\text{predFlagLX}[\text{xSbIdx}][\text{ySbIdx}] = \text{predFlagLXSbCol}[\text{xSbIdx}][\text{ySbIdx}] \quad (694)$$

$$\text{mvLX}[\text{xSbIdx}][\text{ySbIdx}][0] = \text{mvLXSbCol}[\text{xSbIdx}][\text{ySbIdx}][0] \quad (695)$$

$$\text{mvLX}[\text{xSbIdx}][\text{ySbIdx}][1] = \text{mvLXSbCol}[\text{xSbIdx}][\text{ySbIdx}][1] \quad (696)$$

- When  $\text{predFlagLX}[\text{xSbIdx}][\text{ySbIdx}]$  is equal to 1, the derivation process for chroma motion vectors in subclause 8.5.2.13 is invoked with  $\text{mvLX}[\text{xSbIdx}][\text{ySbIdx}]$  as input, and the output being  $\text{mvCLX}[\text{xSbIdx}][\text{ySbIdx}]$ .

- The following assignment is made for  $x = \text{xCb}..\text{xCb} + \text{cbWidth} - 1$  and  $y = \text{yCb}..\text{yCb} + \text{cbHeight} - 1$ :

$$\text{MotionModelIdx}[x][y] = 0 \quad (697)$$

- Otherwise ( $\text{subblockMergeCandList}[\text{merge\_subblock\_idx}[\text{xCb}][\text{yCb}]]$  is not equal to  $\text{SbCol}$ ), the following applies:

- For  $X = 0..1$ , the following assignments are made with  $N$  being the candidate at position  $\text{merge\_subblock\_idx}[\text{xCb}][\text{yCb}]$  in the subblock merging candidate list  $\text{subblockMergeCandList} ( N = \text{subblockMergeCandList}[\text{merge\_subblock\_idx}[\text{xCb}][\text{yCb}]] )$ :

$$\text{refIdxLX} = \text{refIdxLXN} \quad (698)$$

$$\text{predFlagLX}[0][0] = \text{predFlagLXN} \quad (699)$$

$$\text{numCpMv} = \text{motionModelIdxN} + 1 \quad (700)$$

$$\text{bcwIdx} = \text{bcwIdxN} \quad (701)$$

- For  $\text{cpIdx} = 0..\text{numCpMv} - 1$ , the following applies:

$$\text{cpMvLX}[\text{cpIdx}] = \text{cpMvLXN}[\text{cpIdx}] \quad (702)$$

- For  $\text{xSbIdx} = 0..\text{numSbX} - 1$ ,  $\text{ySbIdx} = 0..\text{numSbY} - 1$  and  $X = 0..1$ , the following applies:

$$\text{predFlagLX}[\text{xSbIdx}][\text{ySbIdx}] = \text{predFlagLX}[0][0] \quad (703)$$

- For  $X = 0..1$ , the following applies:

When  $\text{predFlagLX}[0][0]$  is equal to 1, the derivation process for motion vector arrays from affine control point motion vectors as specified in subclause 8.5.5.9 is invoked with the luma coding block location  $(\text{xCb}, \text{yCb})$ , the luma coding block width  $\text{cbWidth}$ , the luma coding block height  $\text{cbHeight}$ , the number of control point motion vectors  $\text{numCpMv}$ , the control point motion vectors  $\text{cpMvLX}[\text{cpIdx}]$  with  $\text{cpIdx}$  being  $0..\text{numCpMv} - 1$ , the prediction list utilization flags  $\text{predFlagL0}[0][0]$  and  $\text{predFlagL1}[0][0]$ , the reference index  $\text{refIdxLX}$ , and the number of luma subblocks in horizontal direction  $\text{numSbX}$  and in vertical direction  $\text{numSbY}$  as inputs, the luma subblock motion vector array  $\text{mvLX}[\text{xSbIdx}][\text{ySbIdx}]$  and the chroma subblock motion vector array  $\text{mvCLX}[\text{xSbIdx}][\text{ySbIdx}]$  with  $\text{xSbIdx} = 0..\text{numSbX} - 1$ ,  $\text{ySbIdx} = 0..\text{numSbY} - 1$ , the prediction refinement utility flag  $\text{cbProfFlagLX}$ , and motion vector difference array  $\text{diffMvLX}[\text{xIdx}][\text{yIdx}]$  with  $\text{xIdx} = 0..\text{cbWidth} / \text{numSbX} - 1$ ,  $\text{yIdx} = 0..\text{cbHeight} / \text{numSbY} - 1$  as outputs.

- The following assignment is made for  $x = \text{xCb}..\text{xCb} + \text{cbWidth} - 1$  and  $y = \text{yCb}..\text{yCb} + \text{cbHeight} - 1$ :

$$\text{MotionModelIdc}[x][y] = \text{numCpMv} - 1 \quad (704)$$

### 8.5.5.3 Derivation process for subblock-based temporal merging candidates

Inputs to this process are:

- a luma location (  $x_{Cb}, y_{Cb}$  ) 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  $\text{availableFlagA}_1$  of the neighbouring coding unit,
- the reference indices  $\text{refIdxLXA}_1$  of the neighbouring coding unit with  $X = 0..1$ ,
- the prediction list utilization flags  $\text{predFlagLXA}_1$  of the neighbouring coding unit with  $X = 0..1$ ,
- the motion vector in 1/16 fractional-sample accuracy  $\text{mvLXA}_1$  of the neighbouring coding unit with  $X = 0..1$ .

Outputs of this process are:

- the availability flag  $\text{availableFlagSbCol}$ ,
- the number of luma subblocks in horizontal direction  $\text{numSbX}$  and in vertical direction  $\text{numSbY}$ ,
- the reference indices  $\text{refIdxL0SbCol}$  and  $\text{refIdxL1SbCol}$ ,
- the luma motion vectors in 1/16 fractional-sample accuracy  $\text{mvL0SbCol}[x_{SbIdx}][y_{SbIdx}]$  and  $\text{mvL1SbCol}[x_{SbIdx}][y_{SbIdx}]$  with  $x_{SbIdx} = 0..\text{numSbX} - 1$ ,  $y_{SbIdx} = 0..\text{numSbY} - 1$ ,
- the prediction list utilization flags  $\text{predFlagL0SbCol}[x_{SbIdx}][y_{SbIdx}]$  and  $\text{predFlagL1SbCol}[x_{SbIdx}][y_{SbIdx}]$  with  $x_{SbIdx} = 0..\text{numSbX} - 1$ ,  $y_{SbIdx} = 0..\text{numSbY} - 1$ .

The availability flag  $\text{availableFlagSbCol}$  is derived as follows:

- If one or more of the following conditions are true,  $\text{availableFlagSbCol}$  is set equal to 0:
  - $\text{ph\_temporal\_mvp\_enabled\_flag}$  is equal to 0.
  - $\text{sps\_sbtmp\_enabled\_flag}$  is equal to 0.
  - $cbWidth$  is less than 8.
  - $cbHeight$  is less than 8.
- Otherwise, the following ordered steps apply:
  1. The location (  $x_{Ctb}, y_{Ctb}$  ) of the top-left sample of the luma coding tree block that contains the current coding block and the location (  $x_{CtrCb}, y_{CtrCb}$  ) of the below-right center sample of the current luma coding block are derived as follows:
 
$$x_{Ctb} = (x_{Cb} \gg CtbLog2SizeY) \ll CtbLog2SizeY \quad (705)$$

$$y_{Ctb} = (y_{Cb} \gg CtbLog2SizeY) \ll CtbLog2SizeY \quad (706)$$

$$x_{CtrCb} = x_{Cb} + (cbWidth / 2) \quad (707)$$

$$y_{CtrCb} = y_{Cb} + (cbHeight / 2) \quad (708)$$
  2. The derivation process for subblock-based temporal merging base motion data as specified in subclause 8.5.5.4 is invoked with the location (  $x_{Ctb}, y_{Ctb}$  ), the location (  $x_{CtrCb}, y_{CtrCb}$  ), the availability flag  $\text{availableFlagA}_1$ , and the prediction list utilization flag  $\text{predFlagLXA}_1$ , and the

reference index  $\text{refIdxLXA}_1$ , and the motion vector  $\text{mvLXA}_1$ , with  $X = 0..1$  as inputs and the motion vectors  $\text{ctrMvLX}$ , and the prediction list utilization flags  $\text{ctrPredFlagLX}$  of the collocated block, with  $X = 0..1$ , and the temporal motion vector  $\text{tempMv}$  as outputs.

3. The variable  $\text{availableFlagSbCol}$  is derived as follows:

- If both  $\text{ctrPredFlagL0}$  and  $\text{ctrPredFlagL1}$  are equal to 0,  $\text{availableFlagSbCol}$  is set equal to 0.
- Otherwise,  $\text{availableFlagSbCol}$  is set equal to 1.

When  $\text{availableFlagSbCol}$  is equal to 1, the following applies:

— The variables  $\text{numSbX}$ ,  $\text{numSbY}$ ,  $\text{sbWidth}$ ,  $\text{sbHeight}$  and  $\text{refIdxLXSbCol}$  are derived as follows:

$$\text{numSbX} = \text{cbWidth} \gg 3 \quad (709)$$

$$\text{numSbY} = \text{cbHeight} \gg 3 \quad (710)$$

$$\text{sbWidth} = \text{cbWidth} / \text{numSbX} \quad (711)$$

$$\text{sbHeight} = \text{cbHeight} / \text{numSbY} \quad (712)$$

$$\text{refIdxLXSbCol} = 0 \quad (713)$$

— For  $\text{xSbIdx} = 0..\text{numSbX} - 1$  and  $\text{ySbIdx} = 0..\text{numSbY} - 1$ , the motion vectors  $\text{mvLXSbCol}[\text{xSbIdx}][\text{ySbIdx}]$  and prediction list utilization flags  $\text{predFlagLXSbCol}[\text{xSbIdx}][\text{ySbIdx}]$  are derived as follows:

— The luma location (  $\text{xSb}$ ,  $\text{ySb}$  ) 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:

$$\text{xSb} = \text{xCb} + \text{xSbIdx} * \text{sbWidth} + \text{sbWidth} / 2 \quad (714)$$

$$\text{ySb} = \text{yCb} + \text{ySbIdx} * \text{sbHeight} + \text{sbHeight} / 2 \quad (715)$$

— The location (  $\text{xColSb}$ ,  $\text{yColSb}$  ) of the collocated subblock inside  $\text{ColPic}$  is derived as follows:

— The following applies:

$$\begin{aligned} \text{yColSb} = & \text{Clip3}( \text{yCtb}, \\ & \text{Min}( \text{pps\_pic\_height\_in\_luma\_samples} - 1, \text{yCtb} + ( 1 \ll \text{CtbLog2SizeY} ) - 1 ), \\ & \text{ySb} + \text{tempMv}[ 1 ] ) \end{aligned} \quad (716)$$

— If  $\text{sps\_subpic\_treated\_as\_pic\_flag}[\text{CurrSubpicIdx}]$  is equal to 1, the following applies:

$$\begin{aligned} \text{xColSb} = & \text{Clip3}( \text{xCtb}, \\ & \text{Min}( \text{SubpicRightBoundaryPos}, \text{xCtb} + ( 1 \ll \text{CtbLog2SizeY} ) + 3 ), \\ & \text{xSb} + \text{tempMv}[ 0 ] ) \end{aligned} \quad (717)$$

— Otherwise (  $\text{sps\_subpic\_treated\_as\_pic\_flag}[\text{CurrSubpicIdx}]$  is equal to 0), the following applies:

$$\begin{aligned} \text{xColSb} = & \text{Clip3}( \text{xCtb}, \\ & \text{Min}( \text{pps\_pic\_width\_in\_luma\_samples} - 1, \text{xCtb} + ( 1 \ll \text{CtbLog2SizeY} ) + 3 ), \\ & \text{xSb} + \text{tempMv}[ 0 ] ) \end{aligned} \quad (718)$$

— The variable  $\text{currCb}$  specifies the luma coding block covering the current subblock inside the current picture.

— The luma location (  $\text{xColCb}$ ,  $\text{yColCb}$  ) is set equal to ( (  $\text{xColSb} \gg 3$  )  $\ll 3$ , (  $\text{yColSb} \gg 3$  )  $\ll 3$  ).

- 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 subclause 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 subclause 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 `availableFlagA1` of the neighbouring coding unit,
- the reference indices `refIdxLXA1` of the neighbouring coding unit with  $X = 0..1$ ,
- the prediction list utilization flags `predFlagLXA1` of the neighbouring coding unit with  $X = 0..1$ ,
- the motion vectors in 1/16 fractional-sample accuracy `mvLXA1` 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:

$$tempMv[0] = 0 \quad (721)$$

$$tempMv[1] = 0 \quad (722)$$

The variable `currPic` specifies the current picture.

When `availableFlagA1` is equal to TRUE, the following applies:

- If all of the following conditions are true, `tempMv` is set equal to `mvL0A1`:
  - `predFlagL0A1` is equal to 1,

- $\text{DiffPicOrderCnt}(\text{ColPic}, \text{RefPicList}[0][\text{refIdxL0A}_1])$  is equal to 0,
- Otherwise, if all of the following conditions are true,  $\text{tempMv}$  is set equal to  $\text{mvL1A}_1$ :
  - $\text{sh\_slice\_type}$  is equal to B,
  - $\text{predFlagL1A}_1$  is equal to 1,
  - $\text{DiffPicOrderCnt}(\text{ColPic}, \text{RefPicList}[1][\text{refIdxL1A}_1])$  is equal to 0.

The rounding process for motion vectors as specified in subclause 8.5.2.14 is invoked with  $\text{mvX}$  set equal to  $\text{tempMv}$ ,  $\text{rightShift}$  set equal to 4, and  $\text{leftShift}$  set equal to 0 as inputs and the rounded  $\text{tempMv}$  as output.

The location  $(\text{xColCb}, \text{yColCb})$  of the collocated block inside  $\text{ColPic}$  is derived as follows:

- The following applies:

$$\begin{aligned} \text{yColCb} = & \text{Clip3}(\text{yCtb}, \\ & \text{Min}(\text{pps\_pic\_height\_in\_luma\_samples} - 1, \text{yCtb} + (1 \ll \text{CtbLog2SizeY}) - 1), (723) \\ & \text{yCtb} + \text{tempMv}[1]) \end{aligned}$$

- If  $\text{sps\_subpic\_treated\_as\_pic\_flag}[\text{CurrSubpicIdx}]$  is equal to 1, the following applies:

$$\begin{aligned} \text{xColCb} = & \text{Clip3}(\text{xCtb}, \\ & \text{Min}(\text{SubpicRightBoundaryPos}, \text{xCtb} + (1 \ll \text{CtbLog2SizeY}) + 3), (724) \\ & \text{xCtb} + \text{tempMv}[0]) \end{aligned}$$

- Otherwise ( $\text{sps\_subpic\_treated\_as\_pic\_flag}[\text{CurrSubpicIdx}]$  is equal to 0), the following applies:

$$\begin{aligned} \text{xColCb} = & \text{Clip3}(\text{xCtb}, \\ & \text{Min}(\text{pps\_pic\_width\_in\_luma\_samples} - 1, \text{xCtb} + (1 \ll \text{CtbLog2SizeY}) + 3), (725) \\ & \text{xCtb} + \text{tempMv}[0]) \end{aligned}$$

The array  $\text{colPredMode}$  is set equal to the prediction mode array  $\text{CuPredMode}[0]$  of the collocated picture specified by  $\text{ColPic}$ .

The motion vectors  $\text{ctrMvL0}$  and  $\text{ctrMvL1}$ , and the prediction list utilization flags  $\text{ctrPredFlagL0}$  and  $\text{ctrPredFlagL1}$  are derived as follows:

- If  $\text{colPredMode}[(\text{xColCb} \gg 3) \ll 3][(\text{yColCb} \gg 3) \ll 3]$  is equal to  $\text{MODE\_INTER}$ , the following applies:
  - The variable  $\text{currCb}$  specifies the luma coding block covering  $(\text{xCtb}, \text{yCtb})$  inside the current picture.
  - The luma location  $(\text{xColCb}, \text{yColCb})$  is set equal to  $((\text{xColCb} \gg 3) \ll 3, (\text{yColCb} \gg 3) \ll 3)$ .
  - The variable  $\text{colCb}$  specifies the luma coding block covering the location  $(\text{xColCb}, \text{yColCb})$  inside the collocated picture specified by  $\text{ColPic}$ .
  - The prediction list utilization flags  $\text{ctrPredFlagL0}$  and  $\text{ctrPredFlagL1}$  are initialized to be equal to FALSE.
  - The derivation process for collocated motion vectors specified in subclause 8.5.2.12 is invoked with  $\text{currCb}$ ,  $\text{colCb}$ ,  $(\text{xColCb}, \text{yColCb})$ ,  $\text{refIdxL0}$  set equal to 0, and  $\text{sbFlag}$  set equal to 1 as inputs and the output being assigned to  $\text{ctrMvL0}$  and  $\text{ctrPredFlagL0}$ .
  - When  $\text{sh\_slice\_type}$  is equal to B, the derivation process for collocated motion vectors specified in subclause 8.5.2.12 is invoked with  $\text{currCb}$ ,  $\text{colCb}$ ,  $(\text{xColCb}, \text{yColCb})$ ,  $\text{refIdxL1}$  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:

$$\text{ctrPredFlagL0} = 0 \quad (726)$$

$$\text{ctrPredFlagL1} = 0 \quad (727)$$

#### 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:

$$\text{log2NbW} = \text{Log2}( \text{nNbW} ) \quad (728)$$

$$\text{log2NbH} = \text{Log2}( \text{nNbH} ) \quad (729)$$

The variables mvScaleHor, mvScaleVer, dHorX and dVerX are derived as follows:

- If isCTUboundary is equal to TRUE, the following applies:

$$\text{mvScaleHor} = \text{MvLX}[ \text{xNb} ][ \text{yNb} + \text{nNbH} - 1 ][ 0 ] \ll 7 \quad (730)$$

$$\text{mvScaleVer} = \text{MvLX}[ \text{xNb} ][ \text{yNb} + \text{nNbH} - 1 ][ 1 ] \ll 7 \quad (731)$$

$$\text{dHorX} = ( \text{MvLX}[ \text{xNb} + \text{nNbW} - 1 ][ \text{yNb} + \text{nNbH} - 1 ][ 0 ] - \text{MvLX}[ \text{xNb} ][ \text{yNb} + \text{nNbH} - 1 ][ 0 ] ) \ll ( 7 - \text{log2NbW} ) \quad (732)$$

$$\text{dVerX} = ( \text{MvLX}[ \text{xNb} + \text{nNbW} - 1 ][ \text{yNb} + \text{nNbH} - 1 ][ 1 ] - \text{MvLX}[ \text{xNb} ][ \text{yNb} + \text{nNbH} - 1 ][ 1 ] ) \ll ( 7 - \text{log2NbW} ) \quad (733)$$

- Otherwise (isCTUboundary is equal to FALSE), the following applies:

$$\text{mvScaleHor} = \text{CpMvLX}[ \text{xNb} ][ \text{yNb} ][ 0 ][ 0 ] \ll 7 \quad (734)$$

$$\text{mvScaleVer} = \text{CpMvLX}[ \text{xNb} ][ \text{yNb} ][ 0 ][ 1 ] \ll 7 \quad (735)$$



$$\begin{aligned} dHorX = & ( CpMvLX[ xNb + nNbW - 1 ][ yNb ][ 1 ][ 0 ] - CpMvLX[ xNb ][ yNb ][ 0 ][ 0 ] ) \\ & << ( 7 - \log2NbW ) \end{aligned} \quad (736)$$

$$\begin{aligned} dVerX = & ( CpMvLX[ xNb + nNbW - 1 ][ yNb ][ 1 ][ 1 ] - CpMvLX[ xNb ][ yNb ][ 0 ][ 1 ] ) \\ & << ( 7 - \log2NbW ) \end{aligned} \quad (737)$$

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:

$$\begin{aligned} dHorY = & ( CpMvLX[ xNb ][ yNb + nNbH - 1 ][ 2 ][ 0 ] - CpMvLX[ xNb ][ yNb ][ 0 ][ 0 ] ) \\ & << ( 7 - \log2NbH ) \end{aligned} \quad (738)$$

$$\begin{aligned} dVerY = & ( CpMvLX[ xNb ][ yNb + nNbH - 1 ][ 2 ][ 1 ] - CpMvLX[ xNb ][ yNb ][ 0 ][ 1 ] ) \\ & << ( 7 - \log2NbH ) \end{aligned} \quad (739)$$

- 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 subclause 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  $\text{availableA}_0$ ,  $\text{availableA}_1$ ,  $\text{availableA}_2$ ,  $\text{availableB}_0$ ,  $\text{availableB}_1$ ,  $\text{availableB}_2$ ,  $\text{availableB}_3$ ,
- the sample locations  $(x_{NbA_0}, y_{NbA_0})$ ,  $(x_{NbA_1}, y_{NbA_1})$ ,  $(x_{NbA_2}, y_{NbA_2})$ ,  $(x_{NbB_0}, y_{NbB_0})$ ,  $(x_{NbB_1}, y_{NbB_1})$ ,  $(x_{NbB_2}, y_{NbB_2})$  and  $(x_{NbB_3}, y_{NbB_3})$ .

Output of this process are:

- the availability flags of the constructed affine control point motion vector merging candidates  $\text{availableFlagConstK}$ , with  $K = 1..6$ ,
- the reference indices  $\text{refIdxLXConstK}$ , with  $K = 1..6$ ,  $X = 0..1$ ,
- the prediction list utilization flags  $\text{predFlagLXConstK}$ , with  $K = 1..6$ ,  $X = 0..1$ ,
- the affine motion model indices  $\text{motionModelIdxConstK}$ , with  $K = 1..6$ ,
- the bi-prediction weight indices  $\text{bcwIdxConstK}$ , with  $K = 1..6$ ,
- the constructed affine control point motion vectors  $\text{cpMvLXConstK}[\text{cpIdx}]$  with  $\text{cpIdx} = 0..2$ ,  $K = 1..6$  and  $X = 0..1$ .

The availability flags of the constructed affine control point motion vector merging candidates  $\text{availableFlagConstK}$ , with  $K = 1..6$ , are initialized to be equal to FALSE.

The prediction list utilization flags  $\text{predFlagLXConstK}$ , with  $K = 1..6$ ,  $X = 0..1$ , are initialized to be equal to FALSE.

The first (top-left) control point motion vector  $\text{cpMvLXCorner}[0]$ , reference index  $\text{refIdxLXCorner}[0]$ , prediction list utilization flag  $\text{predFlagLXCorner}[0]$ , bi-prediction weight index  $\text{bcwIdxCorner}[0]$  and the availability flag  $\text{availableFlagCorner}[0]$  with  $X = 0..1$  are derived as follows:

- The availability flag  $\text{availableFlagCorner}[0]$  is set equal to FALSE.
- The following applies for  $(x_{NbTL}, y_{NbTL})$  with TL being replaced by  $B_2$ ,  $B_3$ , and  $A_2$ :
  - When  $\text{availableTL}$  is equal to TRUE and  $\text{availableFlagCorner}[0]$  is equal to FALSE, the following applies for  $X = 0..1$ :

$$\text{refIdxLXCorner}[0] = \text{RefIdxLX}[x_{NbTL}][y_{NbTL}] \quad (750)$$

$$\text{predFlagLXCorner}[0] = \text{PredFlagLX}[x_{NbTL}][y_{NbTL}] \quad (751)$$

$$\text{cpMvLXCorner}[0] = \text{MvLX}[x_{NbTL}][y_{NbTL}] \quad (752)$$

$$\text{bcwIdxCorner}[0] = \text{BcwIdx}[x_{NbTL}][y_{NbTL}] \quad (753)$$

$$\text{availableFlagCorner}[0] = \text{TRUE} \quad (754)$$

The second (top-right) control point motion vector  $\text{cpMvLXCorner}[1]$ , reference index  $\text{refIdxLXCorner}[1]$ , prediction list utilization flag  $\text{predFlagLXCorner}[1]$ , bi-prediction weight index  $\text{bcwIdxCorner}[1]$  and the availability flag  $\text{availableFlagCorner}[1]$  with  $X = 0..1$  are derived as follows:

- The availability flag  $\text{availableFlagCorner}[1]$  is set equal to FALSE.
- The following applies for  $(x_{NbTR}, y_{NbTR})$  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:

$$\text{refIdxLXCorner}[ 1 ] = \text{RefIdxLX}[ \text{xNbTR} ][ \text{yNbTR} ] \quad (755)$$

$$\text{predFlagLXCorner}[ 1 ] = \text{PredFlagLX}[ \text{xNbTR} ][ \text{yNbTR} ] \quad (756)$$

$$\text{cpMvLXCorner}[ 1 ] = \text{MvLX}[ \text{xNbTR} ][ \text{yNbTR} ] \quad (757)$$

$$\text{bcwIdxCorner}[ 1 ] = \text{BcwIdx}[ \text{xNbTR} ][ \text{yNbTR} ] \quad (758)$$

$$\text{availableFlagCorner}[ 1 ] = \text{TRUE} \quad (759)$$

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<sub>1</sub> and A<sub>0</sub>:

- When availableBL is equal to TRUE and availableFlagCorner[ 2 ] is equal to FALSE, the following applies for X = 0..1:

$$\text{refIdxLXCorner}[ 2 ] = \text{RefIdxLX}[ \text{xNbBL} ][ \text{yNbBL} ] \quad (760)$$

$$\text{predFlagLXCorner}[ 2 ] = \text{PredFlagLX}[ \text{xNbBL} ][ \text{yNbBL} ] \quad (761)$$

$$\text{cpMvLXCorner}[ 2 ] = \text{MvLX}[ \text{xNbBL} ][ \text{yNbBL} ] \quad (762)$$

$$\text{availableFlagCorner}[ 2 ] = \text{TRUE} \quad (763)$$

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 X = 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:

$$\text{xColBr} = \text{xCb} + \text{cbWidth} \quad (764)$$

$$\text{yColBr} = \text{yCb} + \text{cbHeight} \quad (765)$$

$$\text{rightBoundaryPos} = \text{sps\_subpic\_treated\_as\_pic\_flag}[ \text{CurrSubpicIdx} ] ?$$

$$\text{SubpicRightBoundaryPos} :$$

$$\text{pps\_pic\_width\_in\_luma\_samples} - 1 \quad (766)$$

$$\text{botBoundaryPos} = \text{sps\_subpic\_treated\_as\_pic\_flag}[ \text{CurrSubpicIdx} ] ?$$

$$\text{SubpicBotBoundaryPos} :$$

$$\text{pps\_pic\_height\_in\_luma\_samples} - 1 \quad (767)$$

- 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 subclause 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)
      - cpMvLXConst1[ 1 ] = cpMvLXCorner[ 1 ] (778)
      - cpMvLXConst1[ 2 ] = cpMvLXCorner[ 2 ] (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:
 
$$predFlagLXConst2 = 1 \quad (780)$$

$$refIdxLXConst2 = refIdxLXCorner[0] \quad (781)$$

$$cpMvLXConst2[0] = cpMvLXCorner[0] \quad (782)$$

$$cpMvLXConst2[1] = cpMvLXCorner[1] \quad (783)$$

$$cpMvLXConst2[2] = cpMvLXCorner[3] + cpMvLXCorner[0] - cpMvLXCorner[1] \quad (784)$$

$$cpMvLXConst2[2][0] = Clip3(-2^{17}, 2^{17} - 1, cpMvLXConst2[2][0]) \quad (785)$$

$$cpMvLXConst2[2][1] = Clip3(-2^{17}, 2^{17} - 1, cpMvLXConst2[2][1]) \quad (786)$$
- 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 (787)

refIdxLXConst3 = refIdxLXCorner[ 0 ] (788)

cpMvLXConst3[ 0 ] = cpMvLXCorner[ 0 ] (789)

cpMvLXConst3[ 1 ] = cpMvLXCorner[ 3 ] + cpMvLXCorner[ 0 ] - cpMvLXCorner[ 2 ] (790)

cpMvLXConst3[ 1 ][ 0 ] = Clip3(  $-2^{17}$ ,  $2^{17} - 1$ , cpMvLXConst3[ 1 ][ 0 ] ) (791)

cpMvLXConst3[ 1 ][ 1 ] = Clip3(  $-2^{17}$ ,  $2^{17} - 1$ , cpMvLXConst3[ 1 ][ 1 ] ) (792)

cpMvLXConst3[ 2 ] = cpMvLXCorner[ 2 ] (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 ];



- $\text{refIdxLXCorner}[1]$  is equal to  $\text{refIdxLXCorner}[3]$ ;
- Otherwise,  $\text{availableFlagLX}$  is set equal to FALSE.
- When  $\text{availableFlagLX}$  is equal to TRUE, the following assignments are made:
  - $\text{predFlagLXConst4} = 1$  (794)
  - $\text{refIdxLXConst4} = \text{refIdxLXCorner}[1]$  (795)
  - $\text{cpMvLXConst4}[0] = \text{cpMvLXCorner}[1] + \text{cpMvLXCorner}[2] - \text{cpMvLXCorner}[3]$  (796)
  - $\text{cpMvLXConst4}[0][0] = \text{Clip3}(-2^{17}, 2^{17} - 1, \text{cpMvLXConst4}[0][0])$  (797)
  - $\text{cpMvLXConst4}[0][1] = \text{Clip3}(-2^{17}, 2^{17} - 1, \text{cpMvLXConst4}[0][1])$  (798)
  - $\text{cpMvLXConst4}[1] = \text{cpMvLXCorner}[1]$  (799)
  - $\text{cpMvLXConst4}[2] = \text{cpMvLXCorner}[2]$  (800)
- The bi-prediction weight index  $\text{bcwIdxConst4}$  is derived as follows:
  - If both  $\text{availableFlagL0}$  and  $\text{availableFlagL1}$  are equal to 1,  $\text{bcwIdxConst4}$  is set equal to  $\text{bcwIdxCorner}[1]$ .
  - Otherwise,  $\text{bcwIdxConst4}$  is set equal to 0.
- The variables  $\text{availableFlagConst4}$  and  $\text{motionModelIdcConst4}$  are derived as follows:
  - If  $\text{availableFlagL0}$  or  $\text{availableFlagL1}$  is equal to 1,  $\text{availableFlagConst4}$  is set equal to TRUE and  $\text{motionModelIdcConst4}$  is set equal to 2.
  - Otherwise,  $\text{availableFlagConst4}$  is set equal to FALSE and  $\text{motionModelIdcConst4}$  is set equal to 0.

The last two constructed affine control point motion vector merging candidates  $\text{ConstK}$  with  $K = 5..6$  including the availability flags  $\text{availableFlagConstK}$ , the reference indices  $\text{refIdxLXConstK}$ , the prediction list utilization flags  $\text{predFlagLXConstK}$ , the affine motion model indices  $\text{motionModelIdcConstK}$ , and the constructed affine control point motion vectors  $\text{cpMvLXConstK}[\text{cpIdx}]$  with  $\text{cpIdx} = 0..2$  and  $X = 0..1$  are derived as follows:

5. When  $\text{availableFlagCorner}[0]$  is equal to TRUE and  $\text{availableFlagCorner}[1]$  is equal to TRUE, the following applies:
  - For  $X = 0..1$ , the following applies:
    - The variable  $\text{availableFlagLX}$  is derived as follows:
      - If all of the following conditions are TRUE,  $\text{availableFlagLX}$  is set equal to TRUE:
        - $\text{predFlagLXCorner}[0]$  is equal to 1;
        - $\text{predFlagLXCorner}[1]$  is equal to 1;
        - $\text{refIdxLXCorner}[0]$  is equal to  $\text{refIdxLXCorner}[1]$ ;
      - Otherwise,  $\text{availableFlagLX}$  is set equal to FALSE.
    - When  $\text{availableFlagLX}$  is equal to TRUE, the following assignments are made:
      - $\text{predFlagLXConst5} = 1$  (801)
      - $\text{refIdxLXConst5} = \text{refIdxLXCorner}[0]$  (802)
      - $\text{cpMvLXConst5}[0] = \text{cpMvLXCorner}[0]$  (803)
      - $\text{cpMvLXConst5}[1] = \text{cpMvLXCorner}[1]$  (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:
 
$$cpMvLXCorner[1][0] = (cpMvLXCorner[0][0] \ll 7) + ((cpMvLXCorner[2][1] - cpMvLXCorner[0][1]) \ll (7 + \log_2(cbWidth) - \log_2(cbHeight))) \quad (805)$$

$$cpMvLXCorner[1][1] = (cpMvLXCorner[0][1] \ll 7) - ((cpMvLXCorner[2][0] - cpMvLXCorner[0][0]) \ll (7 + \log_2(cbWidth) - \log_2(cbHeight))) \quad (806)$$
      - The rounding process for motion vectors as specified in subclause 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:
 
$$predFlagLXConst6 = 1 \quad (807)$$

$$refIdxLXConst6 = refIdxLXCorner[0] \quad (808)$$

$$cpMvLXConst6[0] = cpMvLXCorner[0] \quad (809)$$

$$cpMvLXConst6[1] = cpMvLXCorner[1] \quad (810)$$

$$cpMvLXConst6[1][0] = Clip3(-2^{17}, 2^{17} - 1, cpMvLXConst6[1][0]) \quad (811)$$

$$cpMvLXConst6[1][1] = Clip3(-2^{17}, 2^{17} - 1, cpMvLXConst6[1][1]) \quad (812)$$
  - 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<sub>0</sub>, yNbA<sub>0</sub> ), ( xNbA<sub>1</sub>, yNbA<sub>1</sub> ), ( xNbB<sub>0</sub>, yNbB<sub>0</sub> ), ( xNbB<sub>1</sub>, yNbB<sub>1</sub> ), and ( xNbB<sub>2</sub>, yNbB<sub>2</sub> ) 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<sub>k</sub>, yNbA<sub>k</sub> ) from ( xNbA<sub>0</sub>, yNbA<sub>0</sub> ) to ( xNbA<sub>1</sub>, yNbA<sub>1</sub> ):

- The derivation process for neighbouring block availability as specified in subclause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring luma location ( xNbA<sub>k</sub>, yNbA<sub>k</sub> ), checkPredModeY set equal to TRUE, and cldx set equal to 0 as inputs, and the output is assigned to the block availability flag availableA<sub>k</sub>.

- When availableA<sub>k</sub> is equal to TRUE and MotionModelIdc[ xNbA<sub>k</sub> ][ yNbA<sub>k</sub> ] is greater than 0 and availableFlagA is equal to FALSE, the following applies:

- The variable ( xNb, yNb ) is set equal to ( CbPosX[ 0 ][ xNbA<sub>k</sub> ][ yNbA<sub>k</sub> ], CbPosY[ 0 ][ xNbA<sub>k</sub> ][ yNbA<sub>k</sub> ] ), nbW is set equal to CbWidth[ 0 ][ xNbA<sub>k</sub> ][ yNbA<sub>k</sub> ], and nbH is set equal to CbHeight[ 0 ][ xNbA<sub>k</sub> ][ yNbA<sub>k</sub> ].

- If PredFlagLX[ xNbA<sub>k</sub> ][ yNbA<sub>k</sub> ] is equal to 1 and DiffPicOrderCnt( RefPicList[ X ][ RefIdxLX[ xNbA<sub>k</sub> ][ yNbA<sub>k</sub> ] ], 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 subclause 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 subclause 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:

$$\text{cpMvpListLX}[\text{numCpMvpCandLX}][\text{cpIdx}] = \text{cpMvpLX}[\text{cpIdx}] \quad (818)$$

- The following applies:

$$\text{numCpMvpCandLX} = \text{numCpMvpCandLX} + 1 \quad (819)$$

- Otherwise, if PredFlagLY[ xNbA<sub>k</sub> ][ yNbA<sub>k</sub> ] (with Y = !X) is equal to 1 and DiffPicOrderCnt( RefPicList[ Y ][ RefIdxLY[ xNbA<sub>k</sub> ][ yNbA<sub>k</sub> ] ], 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 subclause 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 subclause 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:

$$\text{cpMvpListLX}[\text{numCpMvpCandLX}][\text{cpIdx}] = \text{cpMvpLY}[\text{cpIdx}] \quad (820)$$

- The following applies:

$$\text{numCpMvpCandLX} = \text{numCpMvpCandLX} + 1 \quad (821)$$

5. The following applies for ( xNbB<sub>k</sub>, yNbB<sub>k</sub> ) from ( xNbB<sub>0</sub>, yNbB<sub>0</sub> ) to ( xNbB<sub>2</sub>, yNbB<sub>2</sub> ):

- The derivation process for neighbouring block availability as specified in subclause 6.4.4 is invoked with the current luma location (xCurr,yCurr) set equal to (xCb,yCb), the neighbouring luma location (xNbB<sub>k</sub>,yNbB<sub>k</sub>), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag availableB<sub>k</sub>.
- When availableB<sub>k</sub> is equal to TRUE and MotionModelIdx[ xNbB<sub>k</sub> ][ yNbB<sub>k</sub> ] is greater than 0 and availableFlagB is equal to FALSE, the following applies:
  - The variable ( xNb, yNb ) is set equal to ( CbPosX[ 0 ][ xNbB<sub>k</sub> ][ yNbB<sub>k</sub> ], CbPosY[ 0 ][ xNbB<sub>k</sub> ][ yNbB<sub>k</sub> ] ), nbW is set equal to CbWidth[ 0 ][ xNbB<sub>k</sub> ][ yNbB<sub>k</sub> ], and nbH is set equal to CbHeight[ 0 ][ xNbB<sub>k</sub> ][ yNbB<sub>k</sub> ].

- If  $\text{PredFlagLX}[xNbB_k][yNbB_k]$  is equal to 1 and  $\text{DiffPicOrderCnt}(\text{RefPicList}[X][\text{RefIdxLX}[xNbB_k][yNbB_k]], \text{RefPicList}[X][\text{refIdxLX}])$  is equal to 0, the following applies:
    - The variable  $\text{availableFlagB}$  is set equal to TRUE.
    - The derivation process for luma affine control point motion vectors from a neighbouring block as specified in subclause 8.5.5.5 is invoked with the luma coding block location  $(x_{Cb}, y_{Cb})$ , the luma coding block width and height  $(cbWidth, cbHeight)$ , the neighbouring luma coding block location  $(x_{Nb}, y_{Nb})$ , the neighbouring luma coding block width and height  $(nbW, nbH)$ , and the number of control point motion vectors  $\text{numCpMv}$  as input, the control point motion vector predictor candidates  $\text{cpMvpLX}[cpIdx]$  with  $cpIdx = 0..\text{numCpMv} - 1$  as output.
    - The rounding process for motion vectors as specified in subclause 8.5.2.14 is invoked with  $mvX$  set equal to  $\text{cpMvpLX}[cpIdx]$ ,  $\text{rightShift}$  set equal to  $\text{AmvrShift}$ , and  $\text{leftShift}$  set equal to  $\text{AmvrShift}$  as inputs and the rounded  $\text{cpMvpLX}[cpIdx]$  with  $cpIdx = 0..\text{numCpMv} - 1$  as output.
    - For  $cpIdx = 0..\text{numCpMv} - 1$ , the following applies:
 
$$\text{cpMvpListLX}[\text{numCpMvpCandLX}][cpIdx] = \text{cpMvpLX}[cpIdx] \quad (822)$$
    - The following applies:
 
$$\text{numCpMvpCandLX} = \text{numCpMvpCandLX} + 1 \quad (823)$$
  - Otherwise, if  $\text{PredFlagLY}[xNbB_k][yNbB_k]$  (with  $Y \neq X$ ) is equal to 1 and  $\text{DiffPicOrderCnt}(\text{RefPicList}[Y][\text{RefIdxLY}[xNbB_k][yNbB_k]], \text{RefPicList}[X][\text{refIdxLX}])$  is equal to 0, the following applies:
    - The variable  $\text{availableFlagB}$  is set equal to TRUE.
    - The derivation process for luma affine control point motion vectors from a neighbouring block as specified in subclause 8.5.5.5 is invoked with the luma coding block location  $(x_{Cb}, y_{Cb})$ , the luma coding block width and height  $(cbWidth, cbHeight)$ , the neighbouring luma coding block location  $(x_{Nb}, y_{Nb})$ , the neighbouring luma coding block width and height  $(nbW, nbH)$ , and the number of control point motion vectors  $\text{numCpMv}$  as input, the control point motion vector predictor candidates  $\text{cpMvpLY}[cpIdx]$  with  $cpIdx = 0..\text{numCpMv} - 1$  as output.
    - The rounding process for motion vectors as specified in subclause 8.5.2.14 is invoked with  $mvX$  set equal to  $\text{cpMvpLY}[cpIdx]$ ,  $\text{rightShift}$  set equal to  $\text{AmvrShift}$ , and  $\text{leftShift}$  set equal to  $\text{AmvrShift}$  as inputs and the rounded  $\text{cpMvpLY}[cpIdx]$  with  $cpIdx = 0..\text{numCpMv} - 1$  as output.
    - For  $cpIdx = 0..\text{numCpMv} - 1$ , the following applies:
 
$$\text{cpMvpListLX}[\text{numCpMvpCandLX}][cpIdx] = \text{cpMvpLY}[cpIdx] \quad (824)$$
    - The following applies:
 
$$\text{numCpMvpCandLX} = \text{numCpMvpCandLX} + 1 \quad (825)$$
6. When  $\text{numCpMvpCandLX}$  is less than 2, the following applies:
- The derivation process for constructed affine control point motion vector prediction candidate as specified in subclause 8.5.5.8 is invoked with the luma coding block location  $(x_{Cb}, y_{Cb})$ , the luma coding block width  $cbWidth$ , the luma coding block height  $cbHeight$ , and the reference index of the current coding unit  $\text{refIdxLX}$  as inputs, and the availability flag  $\text{availableConsFlagLX}$ , the availability flags  $\text{availableFlagLX}[cpIdx]$  and  $\text{cpMvpLX}[cpIdx]$  with  $cpIdx = 0..2$  as outputs.

— When availableConsFlagLX is equal to 1, the following applies:

— For cpIdx = 0..numCpMv – 1, the following applies:

$$\text{cpMvpListLX}[\text{numCpMvpCandLX}][\text{cpIdx}] = \text{cpMvpLX}[\text{cpIdx}] \quad (826)$$

— The following applies:

$$\text{numCpMvpCandLX} = \text{numCpMvpCandLX} + 1 \quad (827)$$

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:

$$\text{cpMvpListLX}[\text{numCpMvpCandLX}][\text{cpIdx}] = \text{cpMvpLX}[\text{nbCpIdx}] \quad (828)$$

— The following applies:

$$\text{numCpMvpCandLX} = \text{numCpMvpCandLX} + 1 \quad (829)$$

8. When numCpMvpCandLX is less than 2, the following applies:

— The derivation process for temporal luma motion vector prediction as specified in subclause 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 subclause 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:

$$\text{cpMvpListLX}[\text{numCpMvpCandLX}][\text{cpIdx}] = \text{mvLXCol} \quad (830)$$

— The following applies:

$$\text{numCpMvpCandLX} = \text{numCpMvpCandLX} + 1 \quad (831)$$

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:

$$\text{cpMvpListLX}[\text{numCpMvpCandLX}][\text{cpIdx}] = \text{mvZero} \quad (832)$$

— The following applies:

$$\text{numCpMvpCandLX} = \text{numCpMvpCandLX} + 1 \quad (833)$$

The affine control point motion vector predictors cpMvpLX[cpIdx] with cpIdx = 0..numCpMv – 1 are derived as follows:

$$\text{cpMvpLX}[\text{cpIdx}] = \text{cpMvpListLX}[\text{mvp\_IX\_flag}[\text{xCb}][\text{yCb}]][\text{cpIdx}] \quad (834)$$

#### 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 candidates  $availableConsFlagLX$  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 candidates  $cpMvLX[cpIdx]$  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 subclause 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:
 
$$cpMvLX[0] = MvLX[xNbTL][yNbTL] \quad (835)$$
    - 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:
 
$$cpMvLX[0] = MvLY[xNbTL][yNbTL] \quad (836)$$
    - When  $availableFlagLX[0]$  is equal to 1, the rounding process for motion vectors as specified in subclause 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 subclause 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 cldx 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  $\text{PredFlagLX}[xNbTR][yNbTR]$  is equal to 1, and  $\text{DiffPicOrderCnt}(\text{RefPicList}[X][\text{RefIdxLX}[xNbTR][yNbTR]], \text{RefPicList}[X][\text{refIdxLX}])$  is equal to 0, availableFlagLX[ 1 ] is set equal to 1 and the following assignments are made:

$$\text{cpMvLX}[1] = \text{MvLX}[xNbTR][yNbTR] \quad (837)$$

— Otherwise, when  $\text{PredFlagLY}[xNbTR][yNbTR]$  (with  $Y \neq X$ ) is equal to 1 and  $\text{DiffPicOrderCnt}(\text{RefPicList}[Y][\text{RefIdxLY}[xNbTR][yNbTR]], \text{RefPicList}[X][\text{refIdxLX}])$  is equal to 0, availableFlagLX[ 1 ] is set equal to 1 and the following assignments are made:

$$\text{cpMvLX}[1] = \text{MvLY}[xNbTR][yNbTR] \quad (838)$$

— When availableFlagLX[ 1 ] is equal to 1, the rounding process for motion vectors as specified in subclause 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<sub>1</sub>, yNbA<sub>1</sub>) and (xNbA<sub>0</sub>, yNbA<sub>0</sub>) 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<sub>1</sub> and A<sub>0</sub>:

— The derivation process for neighbouring block availability as specified in subclause 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 cldx 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  $\text{PredFlagLX}[xNbBL][yNbBL]$  is equal to 1, and  $\text{DiffPicOrderCnt}(\text{RefPicList}[X][\text{RefIdxLX}[xNbBL][yNbBL]], \text{RefPicList}[X][\text{refIdxLX}])$  is equal to 0, availableFlagLX[ 2 ] is set equal to 1 and the following assignments are made:

$$\text{cpMvLX}[2] = \text{MvLX}[xNbBL][yNbBL] \quad (839)$$

— Otherwise, when  $\text{PredFlagLY}[xNbBL][yNbBL]$  (with  $Y \neq X$ ) is equal to 1 and  $\text{DiffPicOrderCnt}(\text{RefPicList}[Y][\text{RefIdxLY}[xNbBL][yNbBL]], \text{RefPicList}[X][\text{refIdxLX}])$  is equal to 0, availableFlagLX[ 2 ] is set equal to 1 and the following assignments are made:

$$\text{cpMvLX}[2] = \text{MvLY}[xNbBL][yNbBL] \quad (840)$$

— When availableFlagLX[ 2 ] is equal to 1, the rounding process for motion vectors as specified in subclause 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  $\text{MotionModelIdx}[xCb][yCb]$  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:

$$\text{CpMvLX}[x][y][\text{cpIdx}] = \text{cpMvLX}[\text{cpIdx}] \quad (841)$$

The variables log2CbW and log2CbH are derived as follows:

$$\text{log2CbW} = \text{Log2}(\text{cbWidth}) \quad (842)$$

$$\text{log2CbH} = \text{Log2}(\text{cbHeight}) \quad (843)$$

The variables mvScaleHor, mvScaleVer, dHorX and dVerX are derived as follows:

$$\text{mvScaleHor} = \text{cpMvLX}[0][0] \ll 7 \quad (844)$$

$$\text{mvScaleVer} = \text{cpMvLX}[0][1] \ll 7 \quad (845)$$

$$\text{dHorX} = (\text{cpMvLX}[1][0] - \text{cpMvLX}[0][0]) \ll (7 - \text{log2CbW}) \quad (846)$$

$$\text{dVerX} = (\text{cpMvLX}[1][1] - \text{cpMvLX}[0][1]) \ll (7 - \text{log2CbW}) \quad (847)$$

The variables dHorY and dVerY are derived as follows:

- If numCpMv is equal to 3, the following applies:

$$\text{dHorY} = (\text{cpMvLX}[2][0] - \text{cpMvLX}[0][0]) \ll (7 - \text{log2CbH}) \quad (848)$$

$$\text{dVerY} = (\text{cpMvLX}[2][1] - \text{cpMvLX}[0][1]) \ll (7 - \text{log2CbH}) \quad (849)$$

- Otherwise ( numCpMv is equal to 2), the following applies:

$$\text{dHorY} = -\text{dVerX} \quad (850)$$

$$\text{dVerY} = \text{dHorX} \quad (851)$$

The variable fallbackModeTriggered is set equal to 1 and modified as follows:

- The variables bxWX<sub>4</sub>, bxHX<sub>4</sub>, bxWX<sub>h</sub>, bxHX<sub>h</sub>, bxWX<sub>v</sub> and bxHX<sub>v</sub> are derived as follows:

$$\text{maxW}_4 = \text{Max}(0, \text{Max}(4 * (2048 + \text{dHorX}), \text{Max}(4 * \text{dHorY}, 4 * (2048 + \text{dHorX}) + 4 * \text{dHorY}))) \quad (852)$$

$$\text{minW}_4 = \text{Min}(0, \text{Min}(4 * (2048 + \text{dHorX}), \text{Min}(4 * \text{dHorY}, 4 * (2048 + \text{dHorX}) + 4 * \text{dHorY}))) \quad (853)$$

$$\text{maxH}_4 = \text{Max}(0, \text{Max}(4 * \text{dVerX}, \text{Max}(4 * (2048 + \text{dVerY}), 4 * \text{dVerX} + 4 * (2048 + \text{dVerY})))) \quad (854)$$

$$\text{minH}_4 = \text{Min}(0, \text{Min}(4 * \text{dVerX}, \text{Min}(4 * (2048 + \text{dVerY}), 4 * \text{dVerX} + 4 * (2048 + \text{dVerY})))) \quad (855)$$

$$\text{bxWX}_4 = ((\text{maxW}_4 - \text{minW}_4) >> 11) + 9 \quad (856)$$

$$\text{bxHX}_4 = ((\text{maxH}_4 - \text{minH}_4) >> 11) + 9 \quad (857)$$

$$\text{bxWX}_h = ((\text{Max}(0, 4 * (2048 + \text{dHorX})) - \text{Min}(0, 4 * (2048 + \text{dHorX}))) >> 11) + 9 \quad (858)$$

$$\text{bxHX}_h = ((\text{Max}(0, 4 * \text{dVerX}) - \text{Min}(0, 4 * \text{dVerX})) >> 11) + 9 \quad (859)$$

$$\text{bxWX}_v = ((\text{Max}(0, 4 * \text{dHorY}) - \text{Min}(0, 4 * \text{dHorY})) >> 11) + 9 \quad (860)$$

$$\text{bxHX}_v = ((\text{Max}(0, 4 * (2048 + \text{dVerY})) - \text{Min}(0, 4 * (2048 + \text{dVerY}))) >> 11) + 9 \quad (861)$$

- If both predFlagL0 and predFlagL1 are equal to 1, and bxWX<sub>4</sub> \* bxHX<sub>4</sub> is less than or equal to 225, fallbackModeTriggered is set equal to 0.
- Otherwise, if either predFlagL0 or predFlagL1 is equal to 0, bxWX<sub>h</sub> \* bxHX<sub>h</sub> is less than or equal to 165, and bxWX<sub>v</sub> \* bxHX<sub>v</sub> 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:

$$\text{xPosCb} = (\text{cbWidth} >> 1) \quad (862)$$

$$\text{yPosCb} = (\text{cbHeight} >> 1) \quad (863)$$

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

$$\text{xPosCb} = 2 + (\text{xSbIdx} << 2) \quad (864)$$

$$\text{yPosCb} = 2 + (\text{ySbIdx} << 2) \quad (865)$$

- The luma motion vector mvLX[ xSbIdx ][ ySbIdx ] is derived as follows :

$$\text{mvLX}[ \text{xSbIdx} ][ \text{ySbIdx} ][ 0 ] = (\text{mvScaleHor} + \text{dHorX} * \text{xPosCb} + \text{dHorY} * \text{yPosCb}) \quad (866)$$

$$\text{mvLX}[ \text{xSbIdx} ][ \text{ySbIdx} ][ 1 ] = (\text{mvScaleVer} + \text{dVerX} * \text{xPosCb} + \text{dVerY} * \text{yPosCb}) \quad (867)$$

- The rounding process for motion vectors as specified in subclause 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:

$$\text{mvLX}[ \text{xSbIdx} ][ \text{ySbIdx} ][ 0 ] = \text{Clip3}(-2^{17}, 2^{17} - 1, \text{mvLX}[ \text{xSbIdx} ][ \text{ySbIdx} ][ 0 ]) \quad (868)$$

$$\text{mvLX}[ \text{xSbIdx} ][ \text{ySbIdx} ][ 1 ] = \text{Clip3}(-2^{17}, 2^{17} - 1, \text{mvLX}[ \text{xSbIdx} ][ \text{ySbIdx} ][ 1 ]) \quad (869)$$

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:

$$\begin{aligned} mvAvgLX = & mvLX[(xSbIdx \gg (SubWidthC - 1) \ll (SubWidthC - 1))] \\ & [(ySbIdx \gg (SubHeightC - 1) \ll (SubHeightC - 1))] + \\ & mvLX[(xSbIdx \gg (SubWidthC - 1) \ll (SubWidthC - 1)) + (SubWidthC - 1)] \\ & [(ySbIdx \gg (SubHeightC - 1) \ll (SubHeightC - 1)) + (SubHeightC - 1)] \end{aligned} \quad (871)$$

$$mvAvgLX[0] = (mvAvgLX[0] + 1 - (mvAvgLX[0] \geq 0)) \gg 1 \quad (872)$$

$$mvAvgLX[1] = (mvAvgLX[1] + 1 - (mvAvgLX[1] \geq 0)) \gg 1 \quad (873)$$

— The derivation process for chroma motion vectors in subclause 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:

$$sbWidth = cbWidth / numSbX \quad (874)$$

$$sbHeight = cbHeight / numSbY \quad (875)$$

$$dmvLimit = 1 \ll 5 \quad (876)$$

$$posOffsetX = 6 * dHorX + 6 * dHorY \quad (877)$$

$$posOffsetY = 6 * dVerX + 6 * dVerY \quad (878)$$

— For  $x = 0..sbWidth - 1$  and  $y = 0..sbHeight - 1$ , the following applies:

$$diffMvLX[x][y][0] = x * (dHorX \ll 2) + y * (dHorY \ll 2) - posOffsetX \quad (879)$$

$$diffMvLX[x][y][1] = x * (dVerX \ll 2) + y * (dVerY \ll 2) - posOffsetY \quad (880)$$

— The rounding process for motion vectors as specified in subclause 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 (  $x_{Cb}$ ,  $y_{Cb}$  ) 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  $hpellIdx$ ,
- the bi-prediction weight index  $bcwIdx$ ,
- the minimum 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  $cldx$  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  $(cbWidth) \times (cbHeight)$  arrays of predicted luma sample values and,  $predSamplesL0_{Cb}$ ,  $predSamplesL1_{Cb}$ ,  $predSamplesL0_{Cr}$  and  $predSamplesL1_{Cr}$ ,  $predSamplesIntra_{Cb}$ , and  $predSamplesIntra_{Cr}$  be  $(cbWidth / SubWidthC) \times (cbHeight / SubHeightC)$  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:
 
$$\text{numSbX} = ( \text{cbWidth} > 16 ) ? ( \text{cbWidth} \gg 4 ) : 1 \quad (882)$$

$$\text{numSbY} = ( \text{cbHeight} > 16 ) ? ( \text{cbHeight} \gg 4 ) : 1 \quad (883)$$
  - 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 subblock sbWidth, sbHeight in luma samples are derived as follows:

$$\text{sbWidth} = \text{cbWidth} / \text{numSbX} \quad (884)$$

$$\text{sbHeight} = \text{cbHeight} / \text{numSbY} \quad (885)$$

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:
 
$$( \text{xSb}, \text{ySb} ) = ( \text{xCb} + \text{xSbIdx} * \text{sbWidth}, \text{yCb} + \text{ySbIdx} * \text{sbHeight} ) \quad (886)$$
- The variable sbBdofFlag is set equal to FALSE.
- When bdofFlag is equal to TRUE, the variable sbBdofFlag is further modified 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 `refPicLXL` of luma samples and two ordered two-dimensional arrays `refPicLXCb` and `refPicLXCr` of chroma samples is derived by invoking the process specified in subclause 8.5.6.2 with  $X$  and `refIdxLX` as inputs.
  - The motion vector offset `mvOffset` is set equal to `refMvLX[ xSbIdx ][ ySbIdx ] - mvLX[ xSbIdx ][ ySbIdx ]`.
  - If `cIdx` is equal to 0, the following applies:
    - The array `predSamplesLXL` is derived by invoking the fractional sample interpolation process specified in subclause 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 ][ ySbIdx ]`, the reference array `refPicLXL`, the variable `bdofFlag` set equal to `sbBdofFlag`, `cbProfFlagLX`, `dmvrFlag`, `hpellfIdx`, `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 subclause 8.5.6.4 is invoked with `sbWidth`, `sbHeight`, the  $(sbWidth + 2) \times (sbHeight + 2)$  array `predSamplesLXL` 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  $(sbWidth) \times (sbHeight)$  array `predSamplesLXL` as output.
  - Otherwise, if `MotionModelIdx[ 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 `predSamplesLXCb` is derived by invoking the fractional sample interpolation process specified in subclause 8.5.6.3 with the luma location  $(xSb, ySb)$ , the subblock width `sbWidth` set equal to  $(MotionModelIdx[ xCb ][ yCb ] ? sbWidth : sbWidth / SubWidthC)$ , the subblock height `sbHeight` set equal to  $(MotionModelIdx[ xCb ][ yCb ] ? sbHeight : sbHeight / SubHeightC)$ , the chroma motion vector offset `mvOffset`, the refined chroma motion vector `refMvLX[ xSbIdx ][ ySbIdx ]`, the reference array `refPicLXCb`, `bdofFlag`, `cbProfFlagLX`, `dmvrFlag`, `hpellfIdx`, `cIdx`, `RprConstraintsActiveFlag[ X ][ refIdxLX ]`, and `RefPicScale[ X ][ refIdxLX ]` as inputs.
    - Otherwise (`cIdx` is equal to 2), the following applies:
      - The array `predSamplesLXCr` is derived by invoking the fractional sample interpolation process specified in subclause 8.5.6.3 with the luma location  $(xSb, ySb)$ , the subblock width `sbWidth` set equal to  $(MotionModelIdx[ xCb ][ yCb ] ? sbWidth : sbWidth / SubWidthC)$ , the subblock height `sbHeight` set equal to  $(MotionModelIdx[ xCb ][ yCb ] ? sbHeight : sbHeight / SubHeightC)$ , the chroma motion vector offset `mvOffset`, the refined chroma motion vector `refMvLX[ xSbIdx ][ ySbIdx ]`, the reference array `refPicLXCr`, `bdofFlag`, `cbProfFlagLX`, `dmvrFlag`, `hpellfIdx`, `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 subclause 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 subclause 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 subclause 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 subclause 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  $y = 0..sbHeight - 1$ :

$$MvL0[xSb + x][ySb + y] = mvL0[xSbIdx][ySbIdx] \quad (887)$$

$$MvL1[xSb + x][ySb + y] = mvL1[xSbIdx][ySbIdx] \quad (888)$$

$$MvDmvrL0[xSb + x][ySb + y] = refMvL0[xSbIdx][ySbIdx] \quad (889)$$

$$MvDmvrL1[xSb + x][ySb + y] = refMvL1[xSbIdx][ySbIdx] \quad (890)$$

$$RefIdxL0[xSb + x][ySb + y] = refIdxL0 \quad (891)$$

$$RefIdxL1[xSb + x][ySb + y] = refIdxL1 \quad (892)$$

$$PredFlagL0[xSb + x][ySb + y] = predFlagL0[xSbIdx][ySbIdx] \quad (893)$$

$$\text{PredFlagL1}[xSb + x][ySb + y] = \text{predFlagL1}[xSbIdx][ySbIdx] \quad (894)$$

$$\text{HpelIfIdx}[xSb + x][ySb + y] = \text{hpelIfIdx} \quad (895)$$

$$\text{BcwIdx}[xSb + x][ySb + y] = \text{bcwIdx} \quad (896)$$

When  $\text{ciip\_flag}[xCb][yCb]$  is equal to 1, the array  $\text{predSamples}$  of prediction samples is modified as follows:

- If  $\text{cldx}$  is equal to 0, the following applies:
  - The general intra sample prediction process as specified in subclause 8.4.5.2.6 is invoked with the location  $(xTbCmp, yTbCmp)$  set equal to  $(xCb, yCb)$ , the intra prediction mode  $\text{predModeIntra}$  set equal to  $\text{INTRA\_PLANAR}$ , the transform block width  $nTbW$  and height  $nTbH$  set equal to  $\text{cbWidth}$  and  $\text{cbHeight}$ , the coding block width  $nCbW$  and height  $nCbH$  set equal to  $\text{cbWidth}$  and  $\text{cbHeight}$ , and the variable  $\text{cldx}$  as inputs, and the output is assigned to the  $(\text{cbWidth})x(\text{cbHeight})$  array  $\text{predSamplesIntra}_L$ .
  - The weighted sample prediction process for combined merge and intra prediction as specified in subclause 8.5.6.7 is invoked with the location  $(xTbCmp, yTbCmp)$  set equal to  $(xCb, yCb)$ , the coding block width  $\text{cbWidth}$ , the coding block height  $\text{cbHeight}$ , the sample arrays  $\text{predSamplesInter}$  and  $\text{predSamplesIntra}$  set equal to  $\text{predSamples}$  and  $\text{predSamplesIntra}_L$ , respectively, and the colour component index  $\text{cldx}$  as inputs, and the output is assigned to the  $(\text{cbWidth})x(\text{cbHeight})$  array  $\text{predSamples}$ .
- Otherwise, if  $\text{cldx}$  is equal to 1 and  $\text{cbWidth} / \text{SubWidthC}$  is greater than or equal to 4, the following applies:
  - The general intra sample prediction process as specified in subclause 8.4.5.2.6 is invoked with the location  $(xTbCmp, yTbCmp)$  set equal to  $(xCb / \text{SubWidthC}, yCb / \text{SubHeightC})$ , the intra prediction mode  $\text{predModeIntra}$  set equal to  $\text{INTRA\_PLANAR}$ , the transform block width  $nTbW$  and height  $nTbH$  set equal to  $\text{cbWidth} / \text{SubWidthC}$  and  $\text{cbHeight} / \text{SubHeightC}$ , the coding block width  $nCbW$  and height  $nCbH$  set equal to  $\text{cbWidth} / \text{SubWidthC}$  and  $\text{cbHeight} / \text{SubHeightC}$ , and the variable  $\text{cldx}$  as inputs, and the output is assigned to the  $(\text{cbWidth} / \text{SubWidthC})x(\text{cbHeight} / \text{SubHeightC})$  array  $\text{predSamplesIntra}_{Cb}$ .
  - The weighted sample prediction process for combined merge and intra prediction as specified in subclause 8.5.6.7 is invoked with the location  $(xTbCmp, yTbCmp)$  set equal to  $(xCb, yCb)$ , the coding block width  $\text{cbWidth} / \text{SubWidthC}$ , the coding block height  $\text{cbHeight} / \text{SubHeightC}$ , the sample arrays  $\text{predSamplesInter}$  and  $\text{predSamplesIntra}$  set equal to  $\text{predSamples}_{Cb}$  and  $\text{predSamplesIntra}_{Cb}$ , respectively, and the colour component index  $\text{cldx}$  as inputs, and the output is assigned to the  $(\text{cbWidth} / \text{SubWidthC})x(\text{cbHeight} / \text{SubHeightC})$  array  $\text{predSamples}$ .
- Otherwise, if  $\text{cldx}$  is equal to 2 and  $\text{cbWidth} / \text{SubWidthC}$  is greater than or equal to 4, the following applies:
  - The general intra sample prediction process as specified in subclause 8.4.5.2.6 is invoked with the location  $(xTbCmp, yTbCmp)$  set equal to  $(xCb / \text{SubWidthC}, yCb / \text{SubHeightC})$ , the intra prediction mode  $\text{predModeIntra}$  set equal to  $\text{INTRA\_PLANAR}$ , the transform block width  $nTbW$  and height  $nTbH$  set equal to  $\text{cbWidth} / \text{SubWidthC}$  and  $\text{cbHeight} / \text{SubHeightC}$ , the coding block width  $nCbW$  and height  $nCbH$  set equal to  $\text{cbWidth} / \text{SubWidthC}$  and  $\text{cbHeight} / \text{SubHeightC}$ , and the variable  $\text{cldx}$  as inputs, and the output is assigned to the  $(\text{cbWidth} / \text{SubWidthC})x(\text{cbHeight} / \text{SubHeightC})$  array  $\text{predSamplesIntra}_{Cr}$ .
  - The weighted sample prediction process for combined merge and intra prediction as specified in subclause 8.5.6.7 is invoked with the location  $(xTbCmp, yTbCmp)$  set equal to  $(xCb, yCb)$ , the coding block width  $\text{cbWidth} / \text{SubWidthC}$ , the coding block height  $\text{cbHeight} / \text{SubHeightC}$ , the sample arrays  $\text{predSamplesInter}$  and  $\text{predSamplesIntra}$  set equal to  $\text{predSamples}_{Cr}$  and

$\text{predSamplesIntra}_{Cr}$ , respectively, and the colour component index  $\text{cIdx}$  as inputs, and the output is assigned to the  $(\text{cbWidth} / \text{SubWidthC}) \times (\text{cbHeight} / \text{SubHeightC})$  array  $\text{predSamples}$ .

### 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  $\text{refIdxLX}$ .

Output of this process is a reference picture consisting of a two-dimensional array of luma samples  $\text{refPicLX}_L$  and two two-dimensional arrays of chroma samples  $\text{refPicLX}_{Cb}$  and  $\text{refPicLX}_{Cr}$ .

The output reference picture  $\text{RefPicList}[X][\text{refIdxLX}]$ , where  $X$  is the value of  $X$  that this process is invoked for, consists of a  $\text{pps\_pic\_width\_in\_luma\_samples}$  by  $\text{pps\_pic\_height\_in\_luma\_samples}$  array of luma samples  $\text{refPicLX}_L$  and two  $\text{PicWidthInSamplesC}$  by  $\text{PicHeightInSamplesC}$  arrays of chroma samples  $\text{refPicLX}_{Cb}$  and  $\text{refPicLX}_{Cr}$ .

The reference picture sample arrays  $\text{refPicLX}_L$ ,  $\text{refPicLX}_{Cb}$  and  $\text{refPicLX}_{Cr}$  correspond to decoded sample arrays  $S_L$ ,  $S_{Cb}$  and  $S_{Cr}$  derived in subclause 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  $\text{sbWidth}$  specifying the width of the current subblock,
- a variable  $\text{sbHeight}$  specifying the height of the current subblock,
- a motion vector offset  $\text{mvOffset}$ ,
- a refined motion vector  $\text{refMvLX}$ ,
- the selected reference picture sample array  $\text{refPicLX}$ ,
- the bi-directional optical flow flag  $\text{bdofFlag}$ ,
- the prediction refinement utilization flag  $\text{cbProfFlagLX}$ ,
- the decoder-side motion vector refinement flag  $\text{dmvrFlag}$ ,
- the half sample interpolation filter index  $\text{hpelIdx}$ ,
- a variable  $\text{cIdx}$  specifying the colour component index of the current block,
- a variable  $\text{refPicIsScaled}$  indicating whether the selected reference picture requires scaling,
- a list of two scaling ratios, horizontal and vertical,  $\text{scalingRatio}$ .

Outputs of this process are:

- an  $(\text{sbWidth} + \text{brdExtSize}) \times (\text{sbHeight} + \text{brdExtSize})$  array  $\text{predSamplesLX}$  of prediction sample values.

The border extension size  $\text{brdExtSize}$  is derived as follows:

$$\text{brdExtSize} = (\text{bdofFlag} \mid \mid \text{cbProfFlagLX}) ? 2 : 0 \quad (897)$$

The variable  $\text{refWraparoundEnabledFlag}$  is set equal to  $(\text{pps\_ref\_wraparound\_enabled\_flag} \ \&\& \ !\text{refPicIsScaled})$ .



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  $(sbWidth + brdExtSize) \times (sbHeight + brdExtSize)$  array  $predSamplesLX$  of prediction sample values is derived as follows:

- The motion vector  $mvLX$  is set equal to  $(refMvLX - mvOffset)$ .
- If  $cldx$  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 subclause 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 subclause 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 subclause 8.5.6.3.2 with  $(xInt_L - (brdExtSize > 0 ? 1 : 0), yInt_L - (brdExtSize > 0 ? 1 : 0))$ ,  $(xFrac_L, yFrac_L)$ ,  $(xSbInt_L, ySbInt_L)$ ,  $refPicLX$ ,  $hpelIdx$ ,  $sbWidth$ ,  $sbHeight$ ,  $dmvrFlag$ ,  $refWraparoundEnabledFlag$ ,  $scalingRatio[0]$ ,  $scalingRatio[1]$ , 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 subclause 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:
 
$$addX = sps\_chroma\_horizontal\_collocated\_flag ? 0 : 8 * (scalingRatio[0] - (1 << 14)) \quad (906)$$

$$addY = sps\_chroma\_vertical\_collocated\_flag ? 0 : 8 * (scalingRatio[1] - (1 << 14)) \quad (907)$$

$$refxSb_C = ((xSb - (SubWidthC * pps\_scaling\_win\_left\_offset)) / SubWidthC << 5) + refMvLX[0] * scalingRatio[0] + addX \quad (908)$$

$$refx_C = ((Sign(refxSb_C) * (Abs(refxSb_C) + 256) >> 9) + xC * ((scalingRatio[0] + 8) >> 4)) + fRefLeftOffset / SubWidthC + 16 >> 5 \quad (909)$$

$$refySb_C = ((ySb - (SubHeightC * pps\_scaling\_win\_top\_offset)) / SubHeightC << 5) + refMvLX[1] * scalingRatio[1] + addY \quad (910)$$

$$refy_C = ((Sign(refySb_C) * (Abs(refySb_C) + 256) >> 9) + yC * ((scalingRatio[1] + 8) >> 4)) + fRefTopOffset / SubHeightC + 16 >> 5 \quad (911)$$
  - The variables  $xInt_C$ ,  $yInt_C$ ,  $xFrac_C$  and  $yFrac_C$  are derived as follows:
 
$$xInt_C = refx_C >> 5 \quad (912)$$

$$yInt_C = refy_C >> 5 \quad (913)$$

$$xFrac_C = refx_C \& 31 \quad (914)$$

$$yFrac_C = refy_C \& 31 \quad (915)$$

- The prediction sample value  $\text{predSamplesLX}[x_C][y_C]$  is derived by invoking the process specified in subclause 8.5.6.3.4 with  $(x_{\text{Int}_C}, y_{\text{Int}_C})$ ,  $(x_{\text{Frac}_C}, y_{\text{Frac}_C})$ ,  $(x_{\text{SbInt}_C}, y_{\text{SbInt}_C})$ ,  $\text{sbWidth}$ ,  $\text{sbHeight}$ ,  $\text{refPicLX}$ ,  $\text{dmvrFlag}$ ,  $\text{refWraparoundEnabledFlag}$ ,  $\text{scalingRatio}[0]$ , and  $\text{scalingRatio}[1]$  as inputs.

NOTE Unlike the process specified in subclause 8.4.5.2.14, this process uses both  $\text{sps\_chroma\_vertical\_collocated\_flag}$  and  $\text{sps\_chroma\_horizontal\_collocated\_flag}$ .

#### 8.5.6.3.2 Luma sample interpolation filtering process

Inputs to this process are:

- a luma location in full-sample units  $(x_{\text{Int}_L}, y_{\text{Int}_L})$ ,
- a luma location in fractional-sample units  $(x_{\text{Frac}_L}, y_{\text{Frac}_L})$ ,
- a luma location in full-sample units  $(x_{\text{SbInt}_L}, y_{\text{SbInt}_L})$  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  $\text{refPicLX}_L$ ,
- the half sample interpolation filter index  $\text{hpelIfldx}$ ,
- a variable  $\text{sbWidth}$  specifying the width of the current subblock,
- a variable  $\text{sbHeight}$  specifying the height of the current subblock,
- the decoder-side motion vector refinement flag  $\text{dmvrFlag}$ ,
- a variable  $\text{refWraparoundEnabledFlag}$  indicating whether horizontal wrap-around motion compensation is enabled,
- a fixedpoint representation of the horizontal scaling factor  $\text{scalingRatio}[0]$ ,
- a fixedpoint representation of the vertical scaling factor  $\text{scalingRatio}[1]$ ,
- a luma location  $(x_{\text{Sb}}, y_{\text{Sb}})$  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  $\text{predSampleLX}_L$ .

The variables  $\text{shift1}$ ,  $\text{shift2}$  and  $\text{shift3}$  are derived as follows:

- The variable  $\text{shift1}$  is set equal to  $\text{Min}(4, \text{BitDepth} - 8)$ , the variable  $\text{shift2}$  is set equal to 6 and the variable  $\text{shift3}$  is set equal to  $\text{Max}(2, 14 - \text{BitDepth})$ .
- The variable  $\text{picW}$  is set equal to  $\text{pps\_pic\_width\_in\_luma\_samples}$  of the reference picture  $\text{refPicLX}$  and the variable  $\text{picH}$  is set equal to  $\text{pps\_pic\_height\_in\_luma\_samples}$  of the reference picture  $\text{refPicLX}$ .

The horizontal and vertical half sample interpolation filter indices  $\text{hpelHorIfldx}$  and  $\text{hpelVerIfldx}$  are derived as follows:

$$\text{hpelHorIfldx} = (\text{scalingRatio}[0] == 16384) ? \text{hpelIfldx} : 0 \quad (916)$$

$$\text{hpelVerIfldx} = (\text{scalingRatio}[1] == 16384) ? \text{hpelIfldx} : 0 \quad (917)$$

The horizontal luma interpolation filter coefficients  $f_{LH}[p]$  for each  $1/16$  fractional sample position  $p$  equal to  $x_{\text{Frac}_L}$  or  $y_{\text{Frac}_L}$  are derived as follows:

- If  $\text{MotionModelIdx}[x_{\text{Sb}}][y_{\text{Sb}}]$  is greater than 0, and  $\text{sbWidth}$  and  $\text{sbHeight}$  are both equal to 4, and  $\text{scalingRatio}[0]$  is greater than 28 672, luma interpolation filter coefficients  $f_{LH}[p]$  are specified in Table 32.

- Otherwise, if  $\text{MotionModelIdc}[xSb][ySb]$  is greater than 0, and  $sbWidth$  and  $sbHeight$  are both equal to 4, and  $\text{scalingRatio}[0]$  is greater than 20 480, luma interpolation filter coefficients  $f_{LH}[p]$  are specified in Table 31.
- Otherwise, if  $\text{MotionModelIdc}[xSb][ySb]$  is greater than 0, and  $sbWidth$  and  $sbHeight$  are both equal to 4, the luma interpolation filter coefficients  $f_{LH}[p]$  are specified in Table 30.
- Otherwise, if  $\text{scalingRatio}[0]$  is greater than 28 672, luma interpolation filter coefficients  $f_{LH}[p]$  are specified in Table 29.
- Otherwise, if  $\text{scalingRatio}[0]$  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  $hpelHfIdx$  set equal to  $hpelHfIdx$ .

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  $\text{MotionModelIdc}[xSb][ySb]$  is greater than 0, and  $sbWidth$  and  $sbHeight$  are both equal to 4, and  $\text{scalingRatio}[1]$  is greater than 28 672, the luma interpolation filter coefficients  $f_{LV}[p]$  are specified in Table 32.
- Otherwise, if  $\text{MotionModelIdc}[xSb][ySb]$  is greater than 0, and  $sbWidth$  and  $sbHeight$  are both equal to 4, and  $\text{scalingRatio}[1]$  is greater than 20 480, the luma interpolation filter coefficients  $f_{LV}[p]$  are specified in Table 31.
- Otherwise, if  $\text{MotionModelIdc}[xSb][ySb]$  is greater than 0, and  $sbWidth$  and  $sbHeight$  are both equal to 4, the luma interpolation filter coefficients  $f_{LV}[p]$  are specified in Table 30.
- Otherwise, if  $\text{scalingRatio}[1]$  is greater than 28 672, luma interpolation filter coefficients  $f_{LV}[p]$  are specified in Table 29.
- Otherwise, if  $\text{scalingRatio}[1]$  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  $hpelVfIdx$  set equal to  $hpelVfIdx$ .

The luma locations in full-sample units ( $xInt_i, yInt_i$ ) are derived as follows for  $i = 0..7$ :

$$xInt_i = xInt_L + i - 3 \quad (918)$$

$$yInt_i = yInt_L + i - 3 \quad (919)$$

- When  $dmvrFlag$  is equal to 1, the following applies:

$$xInt_i = \text{Clip3}(xSbInt_L - 3, xSbInt_L + sbWidth - 1 + 4, xInt_i) \quad (920)$$

$$yInt_i = \text{Clip3}(ySbInt_L - 3, ySbInt_L + sbHeight - 1 + 4, yInt_i) \quad (921)$$

- If  $\text{sps\_subpic\_treated\_as\_pic\_flag}[\text{CurrSubpicIdx}]$  is equal to 1 and  $\text{sps\_num\_subpics\_minus1}$  for the reference picture  $\text{refPicLX}$  is greater than 0, the following applies:

$$xInt_i = \text{Clip3}(\text{SubpicLeftBoundaryPos}, \text{SubpicRightBoundaryPos}, \text{refWraparoundEnabledFlag} ? \text{ClipH}((\text{PpsRefWraparoundOffset}) * \text{MinCbSizeY}, \text{picW}, xInt_i) : xInt_i) \quad (922)$$

$$yInt_i = \text{Clip3}(\text{SubpicTopBoundaryPos}, \text{SubpicBotBoundaryPos}, yInt_i) \quad (923)$$

- Otherwise ( $\text{sps\_subpic\_treated\_as\_pic\_flag}[\text{CurrSubpicIdx}]$  is equal to 0 or  $\text{sps\_num\_subpics\_minus1}$  for the reference picture  $\text{refPicLX}$  is equal to 0), the following applies:

$$xInt_i = \text{Clip3}(0, \text{picW} - 1, \text{refWraparoundEnabledFlag} ? \text{ClipH}((\text{PpsRefWraparoundOffset}) * \text{MinCbSizeY}, \text{picW}, xInt_i) : xInt_i) \quad (924)$$

$$yInt_i = \text{Clip3}(0, \text{picH} - 1, yInt_i) \quad (925)$$

The predicted luma sample value  $\text{predSampleLX}_L$  is derived as follows:

- If both  $x\text{Frac}_L$  and  $y\text{Frac}_L$  are equal to 0, and both  $\text{scalingRatio}[0]$  and  $\text{scalingRatio}[1]$  are less than 20481, the value of  $\text{predSampleLX}_L$  is derived as follows:

$$\text{predSampleLX}_L = \text{refPicLX}_L[xInt_3][yInt_3] \ll \text{shift3} \quad (926)$$

- Otherwise, if  $y\text{Frac}_L$  is equal to 0 and  $\text{scalingRatio}[1]$  is less than 20481, the value of  $\text{predSampleLX}_L$  is derived as follows:

$$\text{predSampleLX}_L = \left( \sum_{i=0}^7 f_{LH}[x\text{Frac}_L][i] * \text{refPicLX}_L[xInt_i][yInt_3] \right) \gg \text{shift1} \quad (927)$$

- Otherwise, if  $x\text{Frac}_L$  is equal to 0 and  $\text{scalingRatio}[0]$  is less than 20481, the value of  $\text{predSampleLX}_L$  is derived as follows:

$$\text{predSampleLX}_L = \left( \sum_{i=0}^7 f_{LV}[y\text{Frac}_L][i] * \text{refPicLX}_L[xInt_3][yInt_i] \right) \gg \text{shift1} \quad (928)$$

- Otherwise, the value of  $\text{predSampleLX}_L$  is derived as follows:

- The sample array  $\text{temp}[n]$  with  $n = 0..7$ , is derived as follows:

$$\text{temp}[n] = \left( \sum_{i=0}^7 f_{LH}[x\text{Frac}_L][i] * \text{refPicLX}_L[xInt_i][yInt_n] \right) \gg \text{shift1} \quad (929)$$

- The predicted luma sample value  $\text{predSampleLX}_L$  is derived as follows:

$$\text{predSampleLX}_L = \left( \sum_{i=0}^7 f_{LV}[y\text{Frac}_L][i] * \text{temp}[i] \right) \gg \text{shift2} \quad (930)$$

**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<br>( $\text{hpellfdx} = 0$ ) | -1                                | 4           | -11         | 40          | 40          | -11         | 4           | -1          |
| 8<br>( $\text{hpellfdx} = 1$ ) | 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 ( $x_{Int_L}$ ,  $y_{Int_L}$ ),
- 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  $\text{Max}(2, 14 - \text{BitDepth})$ .

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 ( $x_{Int}$ ,  $y_{Int}$ ) 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:

$$x_{Int} = \text{Clip3}(\text{SubpicLeftBoundaryPos}, \text{SubpicRightBoundaryPos}, \text{refWraparoundEnabledFlag} ? \text{ClipH}((\text{PpsRefWraparoundOffset}) * \text{MinCbSizeY}, \text{picW}, x_{Int_L}) : x_{Int_L}) \quad (931)$$

$$y_{Int} = \text{Clip3}(\text{SubpicTopBoundaryPos}, \text{SubpicBotBoundaryPos}, y_{Int_L}) \quad (932)$$

- 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:

$$x_{Int} = \text{Clip3}(0, \text{picW} - 1, \text{refWraparoundEnabledFlag} ? \text{ClipH}((\text{PpsRefWraparoundOffset}) * \text{MinCbSizeY}, \text{picW}, x_{Int_L}) : x_{Int_L}) \quad (933)$$

$$yInt = Clip3( 0, picH - 1, yInt_L ) \quad (934)$$

The predicted luma sample value  $predSampleLX_L$  is derived as follows:

$$predSampleLX_L = refPicLX_L[ xInt ][ yInt ] \ll shift \quad (935)$$

#### 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  $\text{Min}( 4, \text{BitDepth} - 8 )$ , the variable  $shift2$  is set equal to 6 and the variable  $shift3$  is set equal to  $\text{Max}( 2, 14 - \text{BitDepth} )$ .
- The variable  $picW_C$  is set equal to  $\text{pps\_pic\_width\_in\_luma\_samples} / \text{SubWidthC}$  of the reference picture  $refPicLX$  and the variable  $picH_C$  is set equal to  $\text{pps\_pic\_height\_in\_luma\_samples} / \text{SubHeightC}$  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:

$$xInt_i = Clip3( xSbInt_C - 1, xSbInt_C + sbWidth - 1 + 2, xInt_i ) \quad (938)$$

$$yInt_i = Clip3( ySbInt_C - 1, ySbInt_C + sbHeight - 1 + 2, yInt_i ) \quad (939)$$

— 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:

$$xInt_i = Clip3( SubpicLeftBoundaryPos / SubWidthC, SubpicRightBoundaryPos / SubWidthC, \\ refWraparoundEnabledFlag ? ClipH( xOffset, picWC, xInt_i ) : xInt_i ) \quad (940)$$

$$yInt_i = Clip3( SubpicTopBoundaryPos / SubHeightC, SubpicBotBoundaryPos / SubHeightC, yInt_i ) \quad (941)$$

— 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:

$$xInt_i = Clip3( 0, picWC - 1, refWraparoundEnabledFlag ? ClipH( xOffset, picWC, xInt_i ) : xInt_i ) \quad (942)$$

$$yInt_i = Clip3( 0, picHC - 1, yInt_i ) \quad (943)$$

The predicted chroma sample value  $predSampleLXC$  is derived as follows:

— If both  $xFracC$  and  $yFracC$  are equal to 0, and both  $scalingRatio[ 0 ]$  and  $scalingRatio[ 1 ]$  are less than 20481, the value of  $predSampleLXC$  is derived as follows:

$$predSampleLXC = refPicLXC[ xInt1 ][ yInt1 ] << shift3 \quad (944)$$

— Otherwise, if  $yFracC$  is equal to 0 and  $scalingRatio[ 1 ]$  is less than 20481, the value of  $predSampleLXC$  is derived as follows:

$$predSampleLXC = ( \sum_{i=0}^3 fCH[ xFracC ][ i ] * refPicLXC[ xInt_i ][ yInt1 ] ) >> shift1 \quad (945)$$

— Otherwise, if  $xFracC$  is equal to 0 and  $scalingRatio[ 0 ]$  is less than 20481, the value of  $predSampleLXC$  is derived as follows:

$$predSampleLXC = ( \sum_{i=0}^3 fCV[ yFracC ][ i ] * refPicLXC[ xInt1 ][ yInt_i ] ) >> shift1 \quad (946)$$

— Otherwise, the value of  $predSampleLXC$  is derived as follows:

— The sample array  $temp[ n ]$  with  $n = 0..3$ , is derived as follows:

$$temp[ n ] = ( \sum_{i=0}^3 fCH[ xFracC ][ i ] * refPicLXC[ xInt_i ][ yIntn ] ) >> shift1 \quad (947)$$

— The predicted chroma sample value  $predSampleLXC$  is derived as follows:

$$predSampleLXC = fCV[ yFracC ][ 0 ] * temp[ 0 ] + \\ fCV[ yFracC ][ 1 ] * temp[ 1 ] + \\ fCV[ yFracC ][ 2 ] * temp[ 2 ] + \quad (948)$$

$$f_{CV}[yFracC][3] * temp[3]) >> shift2$$

**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          |

| Fractional<br>sample<br>position $p$ | interpolation filter coefficients |            |            |            |
|--------------------------------------|-----------------------------------|------------|------------|------------|
|                                      | $fc[p][0]$                        | $fc[p][1]$ | $fc[p][2]$ | $fc[p][3]$ |
| 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  $fc[p]$  for each  $1/32$  fractional sample position  $p$  for scaling factors of around  $2x$

| Fractional<br>sample<br>position $p$ | interpolation filter coefficients |            |            |            |
|--------------------------------------|-----------------------------------|------------|------------|------------|
|                                      | $fc[p][0]$                        | $fc[p][1]$ | $fc[p][2]$ | $fc[p][3]$ |
| 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          |

| Fractional<br>sample<br>position $p$ | interpolation filter coefficients |            |            |            |
|--------------------------------------|-----------------------------------|------------|------------|------------|
|                                      | $fc[p][0]$                        | $fc[p][1]$ | $fc[p][2]$ | $fc[p][3]$ |
| 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  $(sbWidth + 2) \times (sbHeight + 2)$  prediction sample array  $predSamplesLXL$ ,
- one  $(sbWidth) \times (sbHeight)$  motion vector difference array  $diffMvLX$ .

Output of this process is the  $(sbWidth) \times (sbHeight)$  array  $sbSamplesLXL$  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:

$$gradientH[x][y] = (predSamplesLXL[x + 2][y + 1] \gg shift1) - (predSamplesLXL[x][y + 1] \gg shift1) \quad (949)$$

$$gradientV[x][y] = (predSamplesLXL[x + 1][y + 2] \gg shift1) - (predSamplesLXL[x + 1][y] \gg shift1) \quad (950)$$

- The variable  $dI$  is derived as follows:

$$dI = gradientH[x][y] * diffMvLX[x][y][0] + gradientV[x][y] * diffMvLX[x][y][1] \quad (951)$$

- Prediction sample value at location  $(x, y)$  in the subblock is derived as follows:

$$dILimit = (1 \ll \text{Max}(13, \text{BitDepth} + 1)) \quad (952)$$

$$sbSamplesLXL[x][y] = predSamplesLXL[x + 1][y + 1] + \text{Clip3}(-dILimit, dILimit - 1, dI) \quad (953)$$



### 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  $(nCbW + 2) \times (nCbH + 2)$  luma prediction sample arrays predSamplesL0 and predSamplesL1.

Output of this process is the  $(nCbW) \times (nCbH)$  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  $\text{Max}(3, 15 - \text{BitDepth})$  and the variable offset4 is set equal to  $1 \ll (\text{shift4} - 1)$ .
- The variable mvRefineThres is set equal to  $1 \ll 4$ .

For xIdx = 0..(nCbW >> 2) - 1 and yIdx = 0..(nCbH >> 2) - 1, the following applies:

- The variable xSb is set equal to  $(xIdx \ll 2) + 1$  and ySb is set equal to  $(yIdx \ll 2) + 1$ .
- 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:

$$h_x = \text{Clip3}(1, nCbW, x) \quad (954)$$

$$v_y = \text{Clip3}(1, nCbH, y) \quad (955)$$

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:

$$\text{diff}[x][y] = (\text{predSamplesL0}[h_x][v_y] \gg \text{shift2}) - (\text{predSamplesL1}[h_x][v_y] \gg \text{shift2}) \quad (960)$$

$$\text{tempH}[x][y] = (\text{gradientHL0}[x][y] + \text{gradientHL1}[x][y]) \gg \text{shift3} \quad (961)$$

$$\text{tempV}[x][y] = (\text{gradientVL0}[x][y] + \text{gradientVL1}[x][y]) \gg \text{shift3} \quad (962)$$

— The variables sGx2, sGy2, sGxGy, sGxdI and sGydI are derived as follows:

$$sGx2 = \sum_i \sum_j \text{Abs}( \text{tempH}[ xSb + i ][ ySb + j ] ) \text{ with } i, j = -1..4 \quad (963)$$

$$sGy2 = \sum_i \sum_j \text{Abs}( \text{tempV}[ xSb + i ][ ySb + j ] ) \text{ with } i, j = -1..4 \quad (964)$$

$$sGxGy = \sum_i \sum_j ( \text{Sign}( \text{tempV}[ xSb + i ][ ySb + j ] ) * \text{tempH}[ xSb + i ][ ySb + j ] ) \text{ with } i, j = -1..4 \quad (965)$$

$$sGxdI = \sum_i \sum_j ( -\text{Sign}( \text{tempH}[ xSb + i ][ ySb + j ] ) * \text{diff}[ xSb + i ][ ySb + j ] ) \text{ with } i, j = -1..4 \quad (966)$$

$$sGydI = \sum_i \sum_j ( -\text{Sign}( \text{tempV}[ xSb + i ][ ySb + j ] ) * \text{diff}[ xSb + i ][ ySb + j ] ) \text{ with } i, j = -1..4 \quad (967)$$

— The horizontal and vertical motion offset of the current subblock are derived as:

$$v_x = sGx2 > 0 ? \text{Clip3}( -\text{mvRefineThres} + 1, \text{mvRefineThres} - 1, ( sGxdI < < 2 ) >> \text{Floor}( \text{Log2}( sGx2 ) ) ) : 0 \quad (968)$$

$$v_y = sGy2 > 0 ? \text{Clip3}( -\text{mvRefineThres} + 1, \text{mvRefineThres} - 1, ( ( sGydI < < 2 ) - ( v_x * sGxGy ) >> 1 ) ) >> \text{Floor}( \text{Log2}( sGy2 ) ) ) : 0 \quad (969)$$

— For  $x = xSb - 1..xSb + 2$ ,  $y = ySb - 1..ySb + 2$ , the prediction sample values of the current subblock are derived as follows:

$$\text{bdofOffset} = v_x * ( \text{gradientHL0}[ x + 1 ][ y + 1 ] - \text{gradientHL1}[ x + 1 ][ y + 1 ] ) + v_y * ( \text{gradientVL0}[ x + 1 ][ y + 1 ] - \text{gradientVL1}[ x + 1 ][ y + 1 ] ) \quad (970)$$

$$\text{pbSamples}[ x ][ y ] = \text{Clip3}( 0, ( 2^{\text{BitDepth}} ) - 1, ( \text{predSamplesL0}[ x + 1 ][ y + 1 ] + \text{offset4} + \text{predSamplesL1}[ x + 1 ][ y + 1 ] + \text{bdofOffset} ) >> \text{shift4} ) \quad (971)$$

### 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 subclause 8.5.6.6.2 with the luma location ( `xCb`, `yCb` ), the coding block width `nCbW`, the coding block height `nCbH`, two ( `nCbW` ) $\times$ ( `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 subclause 8.5.6.6.3 with the coding block width `nCbW`, the coding block height `nCbH`, two ( `nCbW` ) $\times$ ( `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` ) $\times$ ( `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` ) $\times$ ( `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  $\text{Max}( 2, 14 - \text{bitDepth} )$  and the variable `shift2` is set equal to  $\text{Max}( 3, 15 - \text{bitDepth} )$ .
- The variable `offset1` is set equal to  $1 \ll ( \text{shift1} - 1 )$ .
- The variable `offset2` is set equal to  $1 \ll ( \text{shift2} - 1 )$ .
- The variable `offset3` is set equal to  $1 \ll ( \text{shift1} + 2 )$ .

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:

$$\text{pbSamples}[ x ][ y ] = \text{Clip3}( 0, ( 1 \ll \text{bitDepth} ) - 1, ( \text{predSamplesL0}[ x ][ y ] + \text{offset1} ) \gg \text{shift1} ) \quad (972)$$

- Otherwise, if `predFlagL0` is equal to 0 and `predFlagL1` is equal to 1, the prediction sample values are derived as follows:

$$\text{pbSamples}[ x ][ y ] = \text{Clip3}( 0, ( 1 \ll \text{bitDepth} ) - 1, ( \text{predSamplesL1}[ x ][ y ] + \text{offset1} ) \gg \text{shift1} ) \quad (973)$$

- 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:

$$\text{pbSamples}[x][y] = \text{Clip3}(0, (1 \ll \text{bitDepth}) - 1, (\text{predSamplesL0}[x][y] + \text{predSamplesL1}[x][y] + \text{offset2}) \gg \text{shift2}) \quad (974)$$

— Otherwise ( $\text{bcwIdx}$  is not equal to 0 and  $\text{ciip\_flag}[x_{Cb}][y_{Cb}]$  is equal to 0), the following applies:

- The variable  $w1$  is set equal to  $\text{bcwWLut}[\text{bcwIdx}]$  with  $\text{bcwWLut}[k] = \{4, 5, 3, 10, -2\}$ .
- The variable  $w0$  is set equal to  $(8 - w1)$ .
- The prediction sample values are derived as follows:

$$\text{pbSamples}[x][y] = \text{Clip3}(0, (1 \ll \text{bitDepth}) - 1, (w0 * \text{predSamplesL0}[x][y] + w1 * \text{predSamplesL1}[x][y] + \text{offset3}) \gg (\text{shift1} + 3)) \quad (975)$$

### 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  $(nCbW) \times (nCbH)$  arrays  $\text{predSamplesL0}$  and  $\text{predSamplesL1}$ ,
- the prediction list utilization flags,  $\text{predFlagL0}$  and  $\text{predFlagL1}$ ,
- the reference indices,  $\text{refIdxL0}$  and  $\text{refIdxL1}$ ,
- the variable  $cIdx$  specifying the colour component index,
- the sample bit depth,  $\text{bitDepth}$ .

Output of this process is the  $(nCbW) \times (nCbH)$  array  $\text{pbSamples}$  of prediction sample values.

The variable  $\text{shift1}$  is set equal to  $\text{Max}(2, 14 - \text{bitDepth})$ .

The variables  $\log2Wd$ ,  $o0$ ,  $o1$ ,  $w0$  and  $w1$  are derived as follows:

- If  $cIdx$  is equal to 0 for luma samples, the following applies:

$$\log2Wd = \text{luma\_log2\_weight\_denom} + \text{shift1} \quad (976)$$

- When  $\text{predFlagL0}$  is equal to 1, the variables  $w0$  and  $o0$  are derived as follows:

$$w0 = \text{LumaWeightL0}[\text{refIdxL0}] \quad (977)$$

$$o0 = \text{luma\_offset\_l0}[\text{refIdxL0}] \ll (\text{bitDepth} - 8) \quad (978)$$

- When  $\text{predFlagL1}$  is equal to 1, the variables  $w1$  and  $o1$  are derived as follows:

$$w1 = \text{LumaWeightL1}[\text{refIdxL1}] \quad (979)$$

$$o1 = \text{luma\_offset\_l1}[\text{refIdxL1}] \ll (\text{bitDepth} - 8) \quad (980)$$

- Otherwise ( $cIdx$  is not equal to 0 for chroma samples), the following applies:

$$\log2Wd = \text{ChromaLog2WeightDenom} + \text{shift1} \quad (981)$$

- When  $\text{predFlagL0}$  is equal to 1, the variables  $w0$  and  $o0$  are derived as follows:

$$w0 = \text{ChromaWeightL0}[\text{refIdxL0}][cIdx - 1] \quad (982)$$

$$o0 = \text{ChromaOffsetL0}[\text{refIdxL0}][\text{cIdx} - 1] \ll (\text{bitDepth} - 8) \quad (983)$$

— When  $\text{predFlagL1}$  is equal to 1, the variables  $w1$  and  $o1$  are derived as follows:

$$w1 = \text{ChromaWeightL1}[\text{refIdxL1}][\text{cIdx} - 1] \quad (984)$$

$$o1 = \text{ChromaOffsetL1}[\text{refIdxL1}][\text{cIdx} - 1] \ll (\text{bitDepth} - 8) \quad (985)$$

The prediction sample  $\text{pbSamples}[x][y]$  with  $x = 0..nCbW - 1$  and  $y = 0..nCbH - 1$  are derived as follows:

— If  $\text{predFlagL0}$  is equal to 1 and  $\text{predFlagL1}$  is equal to 0, the prediction sample values are derived as follows:

$$\text{pbSamples}[x][y] = \text{Clip3}(0, (1 \ll \text{bitDepth}) - 1, ((\text{predSamplesL0}[x][y] * w0 + 2^{\log_2 Wd - 1}) \gg \log_2 Wd) + o0) \quad (986)$$

— Otherwise, if  $\text{predFlagL0}$  is equal to 0 and  $\text{predFlagL1}$  is equal to 1, the prediction sample values are derived as follows:

$$\text{pbSamples}[x][y] = \text{Clip3}(0, (1 \ll \text{bitDepth}) - 1, ((\text{predSamplesL1}[x][y] * w1 + 2^{\log_2 Wd - 1}) \gg \log_2 Wd) + o1) \quad (987)$$

— Otherwise ( $\text{predFlagL0}$  is equal to 1 and  $\text{predFlagL1}$  is equal to 1), the prediction sample values are derived as follows:

$$\text{pbSamples}[x][y] = \text{Clip3}(0, (1 \ll \text{bitDepth}) - 1, ((\text{predSamplesL0}[x][y] * w0 + \text{predSamplesL1}[x][y] * w1 + ((o0 + o1 + 1) \ll \log_2 Wd)) \gg (\log_2 Wd + 1))) \quad (988)$$

#### 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  $(cbWidth) \times (cbHeight)$  arrays  $\text{predSamplesInter}$  and  $\text{predSamplesIntra}$ ,
- a variable  $cIdx$  specifying the colour component index.

Output of this process is the  $(cbWidth) \times (cbHeight)$  array  $\text{predSamplesComb}$  of prediction sample values.

The variables  $\text{scaleFactX}$  and  $\text{scaleFactY}$  are derived as follows:

$$\text{scaleFactX} = (cIdx == 0 \mid \mid \text{SubWidthC} == 1) ? 0 : 1 \quad (989)$$

$$\text{scaleFactY} = (cIdx == 0 \mid \mid \text{SubHeightC} == 1) ? 0 : 1 \quad (990)$$

The neighbouring luma locations  $(xNbA, yNbA)$  and  $(xNbB, yNbB)$  are set equal to  $(xCb - 1, yCb - 1 + (cbHeight \ll \text{scaleFactY}))$  and  $(xCb - 1 + (cbWidth \ll \text{scaleFactX}), yCb - 1)$ , respectively.

For X being replaced by either A or B, the variables  $\text{availableX}$  and  $\text{isIntraCodedNeighbourX}$  are derived as follows:

- The derivation process for neighbouring block availability as specified in subclause 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 cldx 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 FALSE, w is set equal to 1.
- Otherwise, w is set equal to 2.

When cldx 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:

$$\begin{aligned} idxY &= predSamplesInter[ x ][ y ] \gg \text{Log2}( OrgCW ) \\ predSamplesInter[ x ][ y ] &= \text{Clip1}( LmcsPivot[ idxY ] + \\ &\quad ( ( \text{ScaleCoeff}[ idxY ] * ( predSamplesInter[ x ][ y ] - \text{InputPivot}[ idxY ] ) + \\ &\quad ( 1 \ll 10 ) ) \gg 11 ) ) \end{aligned} \quad (991)$$

The prediction samples predSamplesComb[ x ][ y ] with  $x = 0..cbWidth - 1$  and  $y = 0..cbHeight - 1$  are derived as follows:

$$\begin{aligned} predSamplesComb[ x ][ y ] &= ( w * predSamplesIntra[ x ][ y ] + \\ &\quad ( 4 - w ) * predSamplesInter[ x ][ y ] + 2 ) \gg 2 \end{aligned} \quad (992)$$

## 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 predSamplesL of luma prediction samples,
- an (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array predSamplesCb of chroma prediction samples for the component Cb, when sps\_chroma\_format\_idc is not equal to 0,



- an  $(cbWidth / SubWidthC) \times (cbHeight / SubHeightC)$  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  $(cbWidth) \times (cbHeight)$  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  $(cbWidth / SubWidthC) \times (cbHeight / SubHeightC)$  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 subclause 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 subclause 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  $hpellIdx$  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 subclause 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  $hpellIdx$  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 subclause 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  $hpellIdx$  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 subclause 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, `predSamplesCb[xc][yc]` 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 subclause 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 `predSamplesLAcB` and `predSamplesLBcb`, and the variables `angleIdx`, `distanceIdx`, and `cldx` 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, `predSamplesCr[xc][yc]` 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 subclause 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 `predSamplesLAcR` and `predSamplesLBcr`, and the variables `angleIdx`, `distanceIdx`, and `cldx` equal to 2 as inputs.
6. The motion vector storing process for merge geometric partitioning mode specified in subclause 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`.

| <code>merge_gpm_partition_idx</code> | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
|--------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| <code>angleIdx</code>                | 0  | 0  | 2  | 2  | 2  | 2  | 3  | 3  | 3  | 3  | 4  | 4  | 4  | 4  | 5  | 5  |
| <code>distanceIdx</code>             | 1  | 3  | 0  | 1  | 2  | 3  | 0  | 1  | 2  | 3  | 0  | 1  | 2  | 3  | 0  | 1  |
| <code>merge_gpm_partition_idx</code> | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| <code>angleIdx</code>                | 5  | 5  | 8  | 8  | 11 | 11 | 11 | 11 | 12 | 12 | 12 | 12 | 13 | 13 | 13 | 13 |
| <code>distanceIdx</code>             | 2  | 3  | 1  | 3  | 0  | 1  | 2  | 3  | 0  | 1  | 2  | 3  | 0  | 1  | 2  | 3  |
| <code>merge_gpm_partition_idx</code> | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| <code>angleIdx</code>                | 14 | 14 | 14 | 14 | 16 | 16 | 18 | 18 | 18 | 19 | 19 | 19 | 20 | 20 | 20 | 21 |
| <code>distanceIdx</code>             | 0  | 1  | 2  | 3  | 1  | 3  | 1  | 2  | 3  | 1  | 2  | 3  | 1  | 2  | 3  | 1  |
| <code>merge_gpm_partition_idx</code> | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| <code>angleIdx</code>                | 21 | 21 | 24 | 24 | 27 | 27 | 27 | 28 | 28 | 28 | 29 | 29 | 29 | 30 | 30 | 30 |
| <code>distanceIdx</code>             | 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  $(nCbW) \times (nCbH)$  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 `cldx` specifying colour component index.

Output of this process is the  $(nCbW) \times (nCbH)$  array `pbSamples` of prediction sample values.

The variables  $nW$ ,  $nH$ ,  $shift1$ ,  $offset1$ ,  $displacementX$ ,  $displacementY$ ,  $partFlip$  and  $shiftHor$  are derived as follows:

$$nW = (cIdx == 0) ? nCbW : nCbW * SubWidthC \quad (993)$$

$$nH = (cIdx == 0) ? nCbH : nCbH * SubHeightC \quad (994)$$

$$shift1 = \text{Max}(5, 17 - \text{BitDepth}) \quad (995)$$

$$offset1 = 1 \ll (shift1 - 1) \quad (996)$$

$$displacementX = angleIdx \quad (997)$$

$$displacementY = (angleIdx + 8) \% 32 \quad (998)$$

$$partFlip = (angleIdx \geq 13 \ \&\& \ angleIdx \leq 27) ? 0 : 1 \quad (999)$$

$$shiftHor = (angleIdx \% 16 == 8 \ || \ (angleIdx \% 16 != 0 \ \&\& \ nH \geq nW)) ? 0 : 1 \quad (1000)$$

The variables  $offsetX$  and  $offsetY$  are derived as follows:

— If  $shiftHor$  is equal to 0, the following applies:

$$offsetX = (-nW) \gg 1 \quad (1001)$$

$$offsetY = ((-nH) \gg 1) + (angleIdx < 16 ? (distanceIdx * nH) \gg 3 : -((distanceIdx * nH) \gg 3)) \quad (1002)$$

— Otherwise ( $shiftHor$  is equal to 1), the following applies:

$$offsetX = ((-nW) \gg 1) + (angleIdx < 16 ? (distanceIdx * nW) \gg 3 : -((distanceIdx * nW) \gg 3)) \quad (1003)$$

$$offsetY = (-nH) \gg 1 \quad (1004)$$

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:

$$weightIdx = (((xL + offsetX) \ll 1) + 1) * disLut[displacementX] + (((yL + offsetY) \ll 1) + 1) * disLut[displacementY] \quad (1007)$$

$$weightIdxL = partFlip ? 32 + weightIdx : 32 - weightIdx \quad (1008)$$

$$wValue = \text{Clip3}(0, 8, (weightIdxL + 4) \gg 3) \quad (1009)$$

— The prediction sample values are derived as follows:

$$pbSamples[x][y] = \text{Clip3}(0, (1 \ll \text{BitDepth}) - 1, (predSamplesLA[x][y] * wValue + (1010) predSamplesLB[x][y] * (8 - wValue) + offset1) \gg shift1) \quad (1010)$$

**Table 37 — Specification of the geometric partitioning distance array disLut.**

|                      |           |           |           |           |           |           |           |           |           |           |           |           |
|----------------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| <b>idx</b>           | <b>0</b>  | <b>2</b>  | <b>3</b>  | <b>4</b>  | <b>5</b>  | <b>6</b>  | <b>8</b>  | <b>10</b> | <b>11</b> | <b>12</b> | <b>13</b> | <b>14</b> |
| <b>disLut[ idx ]</b> | 8         | 8         | 8         | 4         | 4         | 2         | 0         | -2        | -4        | -4        | -8        | -8        |
| <b>idx</b>           | <b>16</b> | <b>18</b> | <b>19</b> | <b>20</b> | <b>21</b> | <b>22</b> | <b>24</b> | <b>26</b> | <b>27</b> | <b>28</b> | <b>29</b> | <b>30</b> |
| <b>disLut[ idx ]</b> | -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, respectively, are set equal to cbWidth >> 2 and cbHeight >> 2, respectively.

The variables, displacementX, displacementY, isFlip and shiftHor are derived as follows:

$$\text{displacementX} = \text{angleIdx} \quad (1011)$$

$$\text{displacementY} = (\text{angleIdx} + 8) \% 32 \quad (1012)$$

$$\text{isFlip} = (\text{angleIdx} \geq 13 \ \&\& \ \text{angleIdx} \leq 27) ? 1 : 0 \quad (1013)$$

$$\text{shiftHor} = (\text{angleIdx} \% 16 == 8 \ || \ (\text{angleIdx} \% 16 != 0 \ \&\& \ \text{cbHeight} \geq \text{cbWidth})) ? 0 : 1 \quad (1014)$$

The variables offsetX and offsetY are derived as follows:

- If shiftHor is equal to 0, the following applies:

$$\text{offsetX} = (-\text{cbWidth}) \gg 1 \quad (1015)$$

$$\text{offsetY} = ((-\text{cbHeight}) \gg 1) + (\text{angleIdx} < 16 ? (\text{distanceIdx} * \text{cbHeight}) \gg 3 : -((\text{distanceIdx} * \text{cbHeight}) \gg 3)) \quad (1016)$$

— Otherwise (shiftHor is equal to 1), the following applies:

$$\text{offsetX} = ( (-\text{cbWidth}) \gg 1 ) + (\text{angleIdx} < 16 ? (\text{distanceIdx} * \text{cbWidth}) \gg 3 : -((\text{distanceIdx} * \text{cbWidth}) \gg 3)) \quad (1017)$$

$$\text{offsetY} = (-\text{cbHeight}) \gg 1 \quad (1018)$$

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:

$$\text{motionIdx} = ((4 * \text{xSbIdx} + \text{offsetX}) \ll 1) + 5 * \text{disLut}[\text{displacementX}] + ((4 * \text{ySbIdx} + \text{offsetY}) \ll 1) + 5 * \text{disLut}[\text{displacementY}] \quad (1019)$$

— The variable sType is derived as follows:

$$\text{sType} = \text{Abs}(\text{motionIdx}) < 32 ? 2 : (\text{motionIdx} \leq 0 ? (1 - \text{isFlip}) : \text{isFlip}) \quad (1020)$$

— Depending on the value of sType, the following assignments are made:

— If sType is equal to 0, the following applies:

$$\text{predFlagL0} = (\text{predListFlagA} == 0) ? 1 : 0 \quad (1021)$$

$$\text{predFlagL1} = (\text{predListFlagA} == 0) ? 0 : 1 \quad (1022)$$

$$\text{refIdxL0} = (\text{predListFlagA} == 0) ? \text{refIdxA} : -1 \quad (1023)$$

$$\text{refIdxL1} = (\text{predListFlagA} == 0) ? -1 : \text{refIdxA} \quad (1024)$$

$$\text{mvL0}[0] = (\text{predListFlagA} == 0) ? \text{mvA}[0] : 0 \quad (1025)$$

$$\text{mvL0}[1] = (\text{predListFlagA} == 0) ? \text{mvA}[1] : 0 \quad (1026)$$

$$\text{mvL1}[0] = (\text{predListFlagA} == 0) ? 0 : \text{mvA}[0] \quad (1027)$$

$$\text{mvL1}[1] = (\text{predListFlagA} == 0) ? 0 : \text{mvA}[1] \quad (1028)$$

— Otherwise, if sType is equal to 1 or (sType is equal to 2 and predListFlagA + predListFlagB is not equal to 1), the following applies:

$$\text{predFlagL0} = (\text{predListFlagB} == 0) ? 1 : 0 \quad (1029)$$

$$\text{predFlagL1} = (\text{predListFlagB} == 0) ? 0 : 1 \quad (1030)$$

$$\text{refIdxL0} = (\text{predListFlagB} == 0) ? \text{refIdxB} : -1 \quad (1031)$$

$$\text{refIdxL1} = (\text{predListFlagB} == 0) ? -1 : \text{refIdxB} \quad (1032)$$

$$\text{mvL0}[0] = (\text{predListFlagB} == 0) ? \text{mvB}[0] : 0 \quad (1033)$$

$$\text{mvL0}[1] = (\text{predListFlagB} == 0) ? \text{mvB}[1] : 0 \quad (1034)$$

$$\text{mvL1}[0] = (\text{predListFlagB} == 0) ? 0 : \text{mvB}[0] \quad (1035)$$

$$\text{mvL1}[1] = (\text{predListFlagB} == 0) ? 0 : \text{mvB}[1] \quad (1036)$$

- Otherwise (sType is equal to 2 and predListFlagA + predListFlagB is equal to 1), the following applies:

$$\text{predFlagL0} = 1 \quad (1037)$$

$$\text{predFlagL1} = 1 \quad (1038)$$

$$\text{refIdxL0} = (\text{predListFlagA} == 0) ? \text{refIdxA} : \text{refIdxB} \quad (1039)$$

$$\text{refIdxL1} = (\text{predListFlagA} == 0) ? \text{refIdxB} : \text{refIdxA} \quad (1040)$$

$$\text{mvL0}[0] = (\text{predListFlagA} == 0) ? \text{mvA}[0] : \text{mvB}[0] \quad (1041)$$

$$\text{mvL0}[1] = (\text{predListFlagA} == 0) ? \text{mvA}[1] : \text{mvB}[1] \quad (1042)$$

$$\text{mvL1}[0] = (\text{predListFlagA} == 0) ? \text{mvB}[0] : \text{mvA}[0] \quad (1043)$$

$$\text{mvL1}[1] = (\text{predListFlagA} == 0) ? \text{mvB}[1] : \text{mvA}[1] \quad (1044)$$

- The following assignments are made for  $x = \text{xCb}.\text{xCb} + 3$  and  $y = \text{yCb}.\text{yCb} + 3$ :

$$\text{MvL0}[(\text{xSbIdx} \ll 2) + x][(\text{ySbIdx} \ll 2) + y] = \text{mvL0} \quad (1045)$$

$$\text{MvL1}[(\text{xSbIdx} \ll 2) + x][(\text{ySbIdx} \ll 2) + y] = \text{mvL1} \quad (1046)$$

$$\text{MvDmvrL0}[(\text{xSbIdx} \ll 2) + x][(\text{ySbIdx} \ll 2) + y] = \text{mvL0} \quad (1047)$$

$$\text{MvDmvrL1}[(\text{xSbIdx} \ll 2) + x][(\text{ySbIdx} \ll 2) + y] = \text{mvL1} \quad (1048)$$

$$\text{RefIdxL0}[(\text{xSbIdx} \ll 2) + x][(\text{ySbIdx} \ll 2) + y] = \text{refIdxL0} \quad (1049)$$

$$\text{RefIdxL1}[(\text{xSbIdx} \ll 2) + x][(\text{ySbIdx} \ll 2) + y] = \text{refIdxL1} \quad (1050)$$

$$\text{PredFlagL0}[(\text{xSbIdx} \ll 2) + x][(\text{ySbIdx} \ll 2) + y] = \text{predFlagL0} \quad (1051)$$

$$\text{PredFlagL1}[(\text{xSbIdx} \ll 2) + x][(\text{ySbIdx} \ll 2) + y] = \text{predFlagL1} \quad (1052)$$

$$\text{BcwIdx}[(\text{xSbIdx} \ll 2) + x][(\text{ySbIdx} \ll 2) + y] = 0 \quad (1053)$$

### 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  $(\text{xTb0}, \text{yTb0})$  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 cldx specifying the colour component of the current block.

Output of this process is an  $(\text{nTbW}) \times (\text{nTbH})$  array resSamples.



The maximum transform block width  $\text{maxTbWidth}$  and height  $\text{maxTbHeight}$  are derived as follows:

$$\text{maxTbWidth} = (\text{cIdx} == 0) ? \text{MaxTbSizeY} : \text{MaxTbSizeY} / \text{SubWidthC} \quad (1054)$$

$$\text{maxTbHeight} = (\text{cIdx} == 0) ? \text{MaxTbSizeY} : \text{MaxTbSizeY} / \text{SubHeightC} \quad (1055)$$

The luma sample location is derived as follows:

$$(\text{xTbY}, \text{yTbY}) = (\text{cIdx} == 0) ? (\text{xTb0}, \text{yTb0}) : (\text{xTb0} * \text{SubWidthC}, \text{yTb0} * \text{SubHeightC}) \quad (1056)$$

Depending on  $\text{maxTbSize}$ , the following applies:

— If  $\text{nTbW}$  is greater than  $\text{maxTbWidth}$  or  $\text{nTbH}$  is greater than  $\text{maxTbHeight}$ , the following ordered steps apply:

1. The variables  $\text{verSplitFirst}$ ,  $\text{newTbW}$  and  $\text{newTbH}$  are derived as follows:

$$\text{verSplitFirst} = (\text{nTbW} * (\text{cIdx} == 0 ? 1 : \text{SubWidthC}) > \text{nTbH} * (\text{cIdx} == 0 ? 1 : \text{SubHeightC})) \quad (1057)$$

$$\&\& (\text{nTbW} > \text{maxTbWidth})$$

$$\text{newTbW} = \text{verSplitFirst} ? (\text{nTbW} / 2) : \text{nTbW} \quad (1058)$$

$$\text{newTbH} = !\text{verSplitFirst} ? (\text{nTbH} / 2) : \text{nTbH} \quad (1059)$$

2. The decoding process for the residual signal of coding blocks coded in inter prediction mode as specified in this subclause is invoked with  $(\text{xTb0}, \text{yTb0})$ ,  $\text{nCbW}$ ,  $\text{nCbH}$ , the transform block width  $\text{nTbW}$  set equal to  $\text{newTbW}$  and the height  $\text{nTbH}$  set equal to  $\text{newTbH}$ , and  $\text{cIdx}$  as inputs, and the output is assigned to the array  $\text{resSamples}[x][y]$  with  $x = 0.. \text{newTbW} - 1$ ,  $y = 0.. \text{newTbH} - 1$ .
3. The following applies:

— If  $\text{verSplitFirst}$  is equal to TRUE, the decoding process for the residual signal of coding blocks coded in inter prediction mode as specified in this subclause is invoked with the location  $(\text{xTb0}, \text{yTb0})$  set equal to  $(\text{xTb0} + \text{newTbW}, \text{yTb0})$ ,  $\text{nCbW}$ ,  $\text{nCbH}$ , the transform block width  $\text{nTbW}$  set equal to  $\text{newTbW}$  and the height  $\text{nTbH}$  set equal to  $\text{newTbH}$ , and  $\text{cIdx}$  as inputs, and the output is assigned to the array  $\text{resSamples}[x][y]$  with  $x = \text{newTbW}.. \text{nTbW} - 1$ ,  $y = 0.. \text{newTbH} - 1$ .

— Otherwise ( $\text{verSplitFirst}$  is equal to FALSE), the decoding process for the residual signal of coding blocks coded in inter prediction mode as specified in this subclause is invoked with the location  $(\text{xTb0}, \text{yTb0})$  set equal to  $(\text{xTb0}, \text{yTb0} + \text{newTbH})$ ,  $\text{nCbW}$ ,  $\text{nCbH}$ , the transform block width  $\text{nTbW}$  set equal to  $\text{newTbW}$  and the height  $\text{nTbH}$  set equal to  $\text{newTbH}$ , and  $\text{cIdx}$  as inputs, and the output is assigned to the array  $\text{resSamples}[x][y]$  with  $x = 0.. \text{newTbW} - 1$ ,  $y = \text{newTbH}.. \text{nTbH} - 1$ .

— Otherwise, if  $\text{cu\_sbt\_flag}$  is equal to 1, the following applies:

— The variables  $\text{sbtMinNumFourths}$ ,  $\text{wPartIdx}$  and  $\text{hPartIdx}$  are derived as follows:

$$\text{sbtMinNumFourths} = \text{cu\_sbt\_quad\_flag} ? 1 : 2 \quad (1060)$$

$$\text{wPartIdx} = \text{cu\_sbt\_horizontal\_flag} ? 4 : \text{sbtMinNumFourths} \quad (1061)$$

$$\text{hPartIdx} = !\text{cu\_sbt\_horizontal\_flag} ? 4 : \text{sbtMinNumFourths} \quad (1062)$$

— The variables  $\text{xPartIdx}$  and  $\text{yPartIdx}$  are derived as follows:

— If  $\text{cu\_sbt\_pos\_flag}$  is equal to 0,  $\text{xPartIdx}$  and  $\text{yPartIdx}$  are set equal to 0.

- Otherwise ( $cu\_sbt\_pos\_flag$  is equal to 1), the variables  $xPartIdx$  and  $yPartIdx$  are derived as follows:

$$xPartIdx = cu\_sbt\_horizontal\_flag ? 0 : (4 - sbtMinNumFourths) \quad (1063)$$

$$yPartIdx = !cu\_sbt\_horizontal\_flag ? 0 : (4 - sbtMinNumFourths) \quad (1064)$$

- The variables  $xTbYSub$ ,  $yTbYSub$ ,  $xTb0Sub$ ,  $yTb0Sub$ ,  $nTbWSub$  and  $nTbHSub$  are derived as follows:

$$xTbYSub = xTbY + (nTbW * ((cIdx == 0) ? 1 : SubWidthC) * xPartIdx / 4) \quad (1065)$$

$$yTbYSub = yTbY + (nTbH * ((cIdx == 0) ? 1 : SubHeightC) * yPartIdx / 4) \quad (1066)$$

$$xTb0Sub = nTbW * xPartIdx / 4 \quad (1067)$$

$$yTb0Sub = nTbH * yPartIdx / 4 \quad (1068)$$

$$nTbWSub = nTbW * wPartIdx / 4 \quad (1069)$$

$$nTbHSub = nTbH * hPartIdx / 4 \quad (1070)$$

- The scaling and transformation process as specified in subclause 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:

$$resSamples[x][y] = resTbSamples[x - xTb0Sub][y - yTb0Sub] \quad (1071)$$

- Otherwise, the scaling and transformation process as specified in subclause 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  $\text{maxTbWidth}$  and height  $\text{maxTbHeight}$  are derived as follows:

$$\text{maxTbWidth} = \text{MaxTbSizeY} / \text{SubWidthC} \quad (1072)$$

$$\text{maxTbHeight} = \text{MaxTbSizeY} / \text{SubHeightC} \quad (1073)$$

Depending on  $\text{maxTbSize}$ , the following applies:

— If  $\text{nTbW}$  is greater than  $\text{maxTbWidth}$  or  $\text{nTbH}$  is greater than  $\text{maxTbHeight}$ , the following ordered steps apply:

1. The variables  $\text{verSplitFirst}$ ,  $\text{newTbW}$  and  $\text{newTbH}$  are derived as follows:

$$\text{verSplitFirst} = ( \text{nTbW} * \text{SubWidthC} > \text{nTbH} * \text{SubHeightC} ) \ \&\& \ ( \text{nTbW} > \text{maxTbWidth} ) \quad (1074)$$

$$\text{newTbW} = \text{verSplitFirst} ? ( \text{nTbW} / 2 ) : \text{nTbW} \quad (1075)$$

$$\text{newTbH} = !\text{verSplitFirst} ? ( \text{nTbH} / 2 ) : \text{nTbH} \quad (1076)$$

2. The decoding process for the reconstructed signal of chroma coding blocks coded in inter prediction mode as specified in this subclause is invoked with  $(\text{xTb0}, \text{yTb0})$ , the transform block width  $\text{nTbW}$  set equal to  $\text{newTbW}$  and the height  $\text{nTbH}$  set equal to  $\text{newTbH}$ ,  $\text{cldx}$ , the  $(\text{newTbW}) \times (\text{newTbH})$  array  $\text{predSamples}$  set equal to  $\text{predSamples}[\text{x}][\text{y}]$ , and the  $(\text{newTbW}) \times (\text{newTbH})$  array  $\text{resSamples}$  set equal to  $\text{resSamples}[\text{x}][\text{y}]$  with  $\text{x} = 0.. \text{newTbW} - 1$ ,  $\text{y} = 0.. \text{newTbH} - 1$  as inputs, and the output is a modified reconstructed picture before in-loop filtering.

3. The following applies:

— If  $\text{verSplitFirst}$  is equal to TRUE, the decoding process for the reconstructed signal of chroma coding blocks coded in inter prediction mode as specified in this subclause is invoked with the location  $(\text{xTb0}, \text{yTb0})$  set equal to  $(\text{xTb0} + \text{newTbW}, \text{yTb0})$ , the transform block width  $\text{nTbW}$  set equal to  $\text{newTbW}$  and the height  $\text{nTbH}$  set equal to  $\text{newTbH}$ ,  $\text{cldx}$ , the  $(\text{newTbW}) \times (\text{newTbH})$  array  $\text{predSamples}$  set equal to  $\text{predSamples}[\text{x}][\text{y}]$ , and the  $(\text{newTbW}) \times (\text{newTbH})$  array  $\text{resSamples}$  set equal to  $\text{resSamples}[\text{x}][\text{y}]$  with  $\text{x} = \text{newTbW}..2 * \text{newTbW} - 1$ ,  $\text{y} = 0.. \text{newTbH} - 1$  as inputs, and the output is a modified reconstructed picture before in-loop filtering.

— Otherwise ( $\text{verSplitFirst}$  is equal to FALSE), the process for the reconstructed signal of chroma coding blocks coded in inter prediction mode as specified in this subclause is invoked with the location  $(\text{xTb0}, \text{yTb0})$  set equal to  $(\text{xTb0}, \text{yTb0} + \text{newTbH})$ , the transform block width  $\text{nTbW}$  set equal to  $\text{newTbW}$  and the height  $\text{nTbH}$  set equal to  $\text{newTbH}$ ,  $\text{cldx}$ , the  $(\text{newTbW}) \times (\text{newTbH})$  array  $\text{predSamples}$  set equal to  $\text{predSamples}[\text{x}][\text{y}]$ , and the  $(\text{newTbW}) \times (\text{newTbH})$  array  $\text{resSamples}$  set equal to  $\text{resSamples}[\text{x}][\text{y}]$  with  $\text{x} = 0.. \text{newTbW} - 1$ ,  $\text{y} = \text{newTbH}..2 * \text{newTbH} - 1$  as inputs, and the output is a modified reconstructed picture before in-loop filtering.

— Otherwise, if  $\text{cu\_sbt\_flag}$  is equal to 1, the following applies:

— The variables  $\text{nTb0W}$ ,  $\text{nTb0H}$ ,  $\text{nTb1W}$ ,  $\text{nTb1H}$ ,  $\text{xTb1}$ , and  $\text{yTb1}$  are derived as follows:

$$\text{nTb0W} = !\text{cu\_sbt\_horizontal\_flag} ? ( \text{nTbW} * \text{SbtNumFourthsTb0} / 4 ) : \text{nTbW} \quad (1077)$$

$$\text{nTb0H} = !\text{cu\_sbt\_horizontal\_flag} ? \text{nTbH} : ( \text{nTbH} * \text{SbtNumFourthsTb0} / 4 ) \quad (1078)$$

$$\text{nTb1W} = \text{nTbW} - ( !\text{cu\_sbt\_horizontal\_flag} ? \text{nTb0W} : 0 ) \quad (1079)$$

$$nTb1H = nTbH - ( !cu\_sbt\_horizontal\_flag ? 0 : nTb0H ) \quad (1080)$$

$$xTb1 = xTb0 + ( !cu\_sbt\_horizontal\_flag ? nTb0W : 0 ) \quad (1081)$$

$$yTb1 = yTb0 + ( !cu\_sbt\_horizontal\_flag ? 0 : nTb0H ) \quad (1082)$$

- The picture reconstruction process for a colour component as specified in subclause 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, cldx, 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 subclause 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, cldx, 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 subclause 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, cldx, 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 subclause 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:

$$IsGt4by4 = ( cbWidth * cbHeight ) > 16 \quad (1083)$$

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 subclause 8.6.2.1 is invoked with the luma coding block location (  $x_{Cb}, y_{Cb}$  ), 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 subclause 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 subclause 8.6.3 is invoked with the luma coding block location (  $x_{Cb}, y_{Cb}$  ), the luma coding block width  $cbWidth$  and the luma coding block height  $cbHeight$ , the luma block vector  $bvL$ , the variable  $cldx$  set equal to 0 as inputs, and the IBC prediction samples (  $predSamples$  ) that are an (  $cbWidth$  )  $\times$  (  $cbHeight$  ) 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 subclause 8.6.3 is invoked with the luma coding block location (  $x_{Cb}, y_{Cb}$  ), the luma coding block width  $cbWidth$  and the luma coding block height  $cbHeight$ , the chroma block vector  $bvC$  and the variable  $cldx$  set equal to 1 as inputs, and the IBC prediction samples (  $predSamples$  ) that are an (  $cbWidth / SubWidthC$  )  $\times$  (  $cbHeight / SubHeightC$  ) array  $predSamples_{Cb}$  of prediction chroma samples for the chroma components  $Cb$  as outputs.
    - The decoding process for IBC blocks as specified in subclause 8.6.3 is invoked with the luma coding block location (  $x_{Cb}, y_{Cb}$  ), the luma coding block width  $cbWidth$  and the luma coding block height  $cbHeight$ , the chroma block vector  $bvC$  and the variable  $cldx$  set equal to 2 as inputs, and the IBC prediction samples (  $predSamples$  ) that are an (  $cbWidth / SubWidthC$  )  $\times$  (  $cbHeight / SubHeightC$  ) 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 subclause 8.5.8 is invoked with the location (  $x_{Tb0}, y_{Tb0}$  ) set equal to the luma location (  $x_{Cb}, y_{Cb}$  ), 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  $cldx$  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 subclause 8.5.8 is invoked with the location (  $x_{Tb0}, y_{Tb0}$  ) set equal to the chroma location (  $x_{Cb} / SubWidthC, y_{Cb} / SubHeightC$  ), 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  $cldx$  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 subclause 8.5.8 is invoked with the location  $(x_{Tb0}, y_{Tb0})$  set equal to the chroma location  $(x_{Cb} / \text{SubWidthC}, y_{Cb} / \text{SubHeightC})$ , 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 `cldx` set equal to 2 as inputs, and the array `resSamplesCr` 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 subclause 8.7.4.6 is invoked with the variable `nTbW` set equal to `cbWidth`, the variable `nTbH` set equal to `cbHeight`, the array `rY` set equal to `resSamplesL`, the array `rcb` set equal to `resSamplescb`, and the array `rCr` set equal to `resSamplesCr` as inputs, and the output are modified versions of the arrays `resSamplesL`, `resSamplescb` and `resSamplesCr`.
4. The reconstructed samples of the current coding unit are derived as follows:
- The picture reconstruction process for a colour component as specified in subclause 8.7.5.1 is invoked with the block location  $(x_{Curr}, y_{Curr})$  set equal to  $(x_{Cb}, y_{Cb})$ , the block width `nCurrSw` set equal to `cbWidth`, the block height `nCurrSh` set equal to `cbHeight`, the variable `cldx` set equal to 0, the  $(cbWidth) \times (cbHeight)$  array `predSamples` set equal to `predSamplesL`, and the  $(cbWidth) \times (cbHeight)$  array `resSamples` set equal to `resSamplesL` 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 subclause 8.5.9 is invoked with the transform block location  $(x_{Tb0}, y_{Tb0})$  set equal to  $(x_{Cb} / \text{SubWidthC}, y_{Cb} / \text{SubHeightC})$ , the transform block width `nTbW` set equal to `cbWidth / SubWidthC`, the transform block height `nTbH` set equal to `cbHeight / SubHeightC`, the variable `cldx` set equal to 1, the  $(cbWidth / \text{SubWidthC}) \times (cbHeight / \text{SubHeightC})$  array `predSamples` set equal to `predSamplescb` and the  $(cbWidth / \text{SubWidthC}) \times (cbHeight / \text{SubHeightC})$  array `resSamples` set equal to `resSamplescb` 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 subclause 8.5.9 is invoked with the transform block location  $(x_{Tb0}, y_{Tb0})$  set equal to  $(x_{Cb} / \text{SubWidthC}, y_{Cb} / \text{SubHeightC})$ , the transform block width `nTbW` set equal to `cbWidth / SubWidthC`, the transform block height `nTbH` set equal to `cbHeight / SubHeightC`, the variable `cldx` set equal to 2, the  $(cbWidth / \text{SubWidthC}) \times (cbHeight / \text{SubHeightC})$  array `predSamples` set equal to `predSamplesCr` and the  $(cbWidth / \text{SubWidthC}) \times (cbHeight / \text{SubHeightC})$  array `resSamples` set equal to `resSamplesCr` 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  $(x_{Cb}, y_{Cb})$  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 subclause 8.6.2.2 is invoked with the luma location  $(x_{Cb}, y_{Cb})$ , 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 subclause 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:

$$u[0] = (bvL[0] + bvd[0]) \& (2^{18} - 1) \quad (1086)$$

$$bvL[0] = (u[0] \geq 2^{17}) ? (u[0] - 2^{18}) : u[0] \quad (1087)$$

$$u[1] = (bvL[1] + bvd[1]) \& (2^{18} - 1) \quad (1088)$$

$$bvL[1] = (u[1] \geq 2^{17}) ? (u[1] - 2^{18}) : u[1] \quad (1089)$$

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 subclause 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  $((y_{Cb} + (bvL[1] \gg 4)) \& (CtbSizeY - 1)) + cbHeight$ .
- `IbcVirBuf[0][(x + (bvL[0] \gg 4)) \& (IbcBufWidthY - 1)][(y + (bvL[1] \gg 4)) \& (CtbSizeY - 1)]` shall not be equal to -1 for  $x = x_{Cb}..x_{Cb} + cbWidth - 1$  and  $y = y_{Cb}..y_{Cb} + cbHeight - 1$ .

### 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  $(x_{Cb}, y_{Cb})$  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  $(x_{Cb}, y_{Cb})$  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 subclause 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:

```

i = 0
if( availableFlagA1 )
    bvCandList [ i++ ] = bvA1
if( availableFlagB1 )
    bvCandList [ i++ ] = bvB1
    
```

(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 subclause 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  $\text{availableFlagA}_1$  and  $\text{availableFlagB}_1$  of the neighbouring coding units,
- the block vectors in 1/16 fractional-sample accuracy  $\text{bvA}_1$ , and  $\text{bvB}_1$  of the neighbouring coding units.

For the derivation of  $\text{availableFlagA}_1$  and  $\text{mvA}_1$  the following applies:

- The luma location  $(\text{xNbA}_1, \text{yNbA}_1)$  inside the neighbouring luma coding block is set equal to  $(\text{xCb} - 1, \text{yCb} + \text{cbHeight} - 1)$ .
- The derivation process for neighbouring block availability as specified in subclause 6.4.4 is invoked with the current luma location  $(\text{xCurr}, \text{yCurr})$  set equal to  $(\text{xCb}, \text{yCb})$ , the neighbouring luma location  $(\text{xNbA}_1, \text{yNbA}_1)$ ,  $\text{checkPredModeY}$  set equal to TRUE, and  $\text{cIdx}$  set equal to 0 as inputs, and the output is assigned to the block availability flag  $\text{availableA}_1$ .
- The variables  $\text{availableFlagA}_1$  and  $\text{bvA}_1$  are derived as follows:

- If  $\text{availableA}_1$  is equal to FALSE,  $\text{availableFlagA}_1$  is set equal to 0 and both components of  $\text{bvA}_1$  are set equal to 0.
- Otherwise,  $\text{availableFlagA}_1$  is set equal to 1 and the following assignments are made:

$$\text{bvA}_1 = \text{MvL0}[\text{xNbA}_1][\text{yNbA}_1] \quad (1094)$$

For the derivation of  $\text{availableFlagB}_1$  and  $\text{bvB}_1$  the following applies:

- The luma location  $(\text{xNbB}_1, \text{yNbB}_1)$  inside the neighbouring luma coding block is set equal to  $(\text{xCb} + \text{cbWidth} - 1, \text{yCb} - 1)$ .
- The derivation process for neighbouring block availability as specified in subclause 6.4.4 is invoked with the current luma location  $(\text{xCurr}, \text{yCurr})$  set equal to  $(\text{xCb}, \text{yCb})$ , the neighbouring luma location  $(\text{xNbB}_1, \text{yNbB}_1)$ ,  $\text{checkPredModeY}$  set equal to TRUE, and  $\text{cIdx}$  set equal to 0 as inputs, and the output is assigned to the block availability flag  $\text{availableB}_1$ .
- The variables  $\text{availableFlagB}_1$  and  $\text{bvB}_1$  are derived as follows:

- If one or more of the following conditions are true,  $\text{availableFlagB}_1$  is set equal to 0 and both components of  $\text{bvB}_1$  are set equal to 0:
  - $\text{availableB}_1$  is equal to FALSE.
  - $\text{availableA}_1$  is equal to TRUE and the luma locations  $(\text{xNbA}_1, \text{yNbA}_1)$  and  $(\text{xNbB}_1, \text{yNbB}_1)$  have the same block vectors.
- Otherwise,  $\text{availableFlagB}_1$  is set equal to 1 and the following assignments are made:

$$\text{bvB}_1 = \text{MvL0}[\text{xNbB}_1][\text{yNbB}_1] \quad (1095)$$

#### 8.6.2.4 Derivation process for IBC history-based block vector candidates

Inputs to this process are:

- a block vector candidate list  $\text{bvCandList}$ ,
- the number of available block vector candidates in the list  $\text{numCurrCand}$ .

Outputs of this process are:

- the modified block vector candidate list  $\text{bvCandList}$ ,
- the modified number of motion vector candidates in the list  $\text{numCurrCand}$ .

For each candidate in  $\text{HmvpIbcCandList}[\text{NumHmvpIbcCand} - \text{hMvpIdx}]$  with index  $\text{hMvpIdx} = 1..\text{NumHmvpIbcCand}$ , the following ordered steps are repeated until  $\text{numCurrCand}$  is equal to  $\text{MaxNumIbcMergeCand}$ :

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`:
    - $\text{IsGt4by4}$  is equal to `TRUE`.
    - $\text{hMvpIdx}$  is equal to 1.
    - The candidate  $\text{HmvpIbcCandList}[\text{NumHmvpIbcCand} - \text{hMvpIdx}]$  is equal to the block vector candidate  $N$ .
  - Otherwise, `sameMotion` is set equal to `FALSE`.
2. When `sameMotion` is equal to `FALSE`, the candidate  $\text{HmvpIbcCandList}[\text{NumHmvpIbcCand} - \text{hMvpIdx}]$  is added to the block vector candidate list as follows:

$$\text{bvCandList}[\text{numCurrCand}++] = \text{HmvpIbcCandList}[\text{NumHmvpIbcCand} - \text{hMvpIdx}] \quad (1096)$$

#### 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  $\text{bvL}$ .

Output of this process is a chroma block vector in 1/32 fractional-sample accuracy  $\text{bvC}$ .

A chroma block vector is derived from the corresponding luma block vector.

The chroma block vector  $\text{bvC}$  is derived as follows:

$$\text{bvC}[0] = (\text{bvL}[0] \gg (3 + \text{SubWidthC})) * 32 \quad (1097)$$

$$\text{bvC}[1] = (\text{bvL}[1] \gg (3 + \text{SubHeightC})) * 32 \quad (1098)$$

#### 8.6.2.6 Updating process for the history-based block vector predictor candidate list

Inputs to this process are:

- luma block vector  $\text{bvL}$  in 1/16 fractional-sample accuracy.

The candidate list  $\text{HmvpIbcCandList}$  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  $\text{NumHmvpIbcCand}$  is greater than 0, for each index  $\text{hMvpIdx}$  with  $\text{hMvpIdx} = 0..\text{NumHmvpIbcCand} - 1$ , the following steps apply until `identicalCandExist` is equal to `TRUE`:
  - When  $\text{bvL}$  is equal to  $\text{HmvpIbcCandList}[\text{hMvpIdx}]$ , `identicalCandExist` is set equal to `TRUE` and `removeIdx` is set equal to  $\text{hMvpIdx}$ .
3. The candidate list  $\text{HmvpIbcCandList}$  is updated as follows:
  - If `identicalCandExist` is equal to `TRUE` or  $\text{NumHmvpIbcCand}$  is equal to 5, the following applies:
    - For each index  $i$  with  $i = (\text{removeIdx} + 1)..\text{NumHmvpIbcCand} - 1$ ,  $\text{HmvpIbcCandList}[i - 1]$  is set equal to  $\text{HmvpIbcCandList}[i]$ .

- HmvplbcCandList[ NumHmvplbcCand - 1 ] is set equal to bvL.
- Otherwise (identicalCandExist is equal to FALSE and NumHmvplbcCand is less than 5), the following applies:
  - HmvplbcCandList[ NumHmvplbcCand ++ ] 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 cldx specifying the colour component index of the current block.

Outputs of this process are:

- an array predSamples of prediction samples.

When cldx is equal to 0, for  $x = xCb..xCb + cbWidth - 1$  and  $y = yCb..yCb + cbHeight - 1$ , the following applies:

$$xVb = ( x + ( bv[ 0 ] >> 4 ) ) \& ( lbcBufWidthY - 1 ) \quad (1099)$$

$$yVb = ( y + ( bv[ 1 ] >> 4 ) ) \& ( CtbSizeY - 1 ) \quad (1100)$$

$$predSamples[ x - xCb ][ y - yCb ] = lbcVirBuf[ 0 ][ xVb ][ yVb ] \quad (1101)$$

When cldx 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:

$$xVb = ( x + ( bv[ 0 ] >> ( 3 + SubWidthC ) ) ) \& ( lbcBufWidthC - 1 ) \quad (1102)$$

$$yVb = ( y + ( bv[ 1 ] >> ( 3 + SubHeightC ) ) ) \& ( ( CtbSizeY / SubHeightC ) - 1 ) \quad (1103)$$

$$predSamples[ x - ( xCb / SubWidthC ) ][ y - ( yCb / SubHeightC ) ] = lbcVirBuf[ cldx ][ xVb ][ yVb ] \quad (1104)$$

When cldx is equal to 0, the following assignments are made for  $x = 0..cbWidth - 1$  and  $y = 0..cbHeight - 1$ :

$$MvL0[ xCb + x ][ yCb + y ] = bv \quad (1105)$$

$$MvL1[ xCb + x ][ yCb + y ][ 0 ] = 0 \quad (1106)$$

$$MvL1[ xCb + x ][ yCb + y ][ 1 ] = 0 \quad (1107)$$

$$RefldxL0[ xCb + x ][ yCb + y ] = -1 \quad (1108)$$

$$RefldxL1[ xCb + x ][ yCb + y ] = -1 \quad (1109)$$

$$PredFlagL0[ xCb + x ][ yCb + y ] = 0 \quad (1110)$$

$$PredFlagL1[ xCb + x ][ yCb + y ] = 0 \quad (1111)$$

$$\text{BcwIdx}[ \text{xCb} + \text{x} ][ \text{yCb} + \text{y} ] = 0 \quad (1112)$$

## 8.7 Scaling, transformation and array construction process

### 8.7.1 Derivation process for quantization parameters

Inputs to this process are:

- a luma location (  $\text{xCb}, \text{yCb}$  ) specifying the top-left luma sample of the current coding block relative to the top-left luma sample of the current picture,
- a variable  $\text{cbWidth}$  specifying the width of the current coding block in luma samples,
- a variable  $\text{cbHeight}$  specifying the height of the current coding block in luma samples,
- a variable  $\text{treeType}$  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  $\text{Qp}'_{\text{Y}}$  and the chroma quantization parameters  $\text{Qp}'_{\text{Cb}}$ ,  $\text{Qp}'_{\text{Cr}}$  and  $\text{Qp}'_{\text{CbCr}}$  are derived.

The luma location (  $\text{xQg}, \text{yQg}$  ), 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  $\text{xQg}$  and  $\text{yQg}$  are set equal to  $\text{CuQgTopLeftX}$  and  $\text{CuQgTopLeftY}$ , respectively.

NOTE The current quantization group is a rectangular region inside a coding tree block that shares the same  $\text{qP}_{\text{Y,PRED}}$ . 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  $\text{CuQgTopLeftX}$  and  $\text{CuQgTopLeftY}$ .

When  $\text{treeType}$  is equal to SINGLE\_TREE or DUAL\_TREE\_LUMA, the predicted luma quantization parameter  $\text{qP}_{\text{Y,PRED}}$  is derived by the following ordered steps:

1. The variable  $\text{qP}_{\text{Y,PREV}}$  is derived as follows:
  - If one or more of the following conditions are true,  $\text{qP}_{\text{Y,PREV}}$  is set equal to  $\text{SliceQpY}$ :
    - 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,  $\text{qP}_{\text{Y,PREV}}$  is set equal to the luma quantization parameter  $\text{Qp}_{\text{Y}}$  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 subclause 6.4.4 is invoked with the location (  $\text{xCurr}, \text{yCurr}$  ) set equal to (  $\text{xCb}, \text{yCb}$  ), the neighbouring location (  $\text{xNbY}, \text{yNbY}$  ) set equal to (  $\text{xQg} - 1, \text{yQg}$  ),  $\text{checkPredModeY}$  set equal to FALSE, and  $\text{cIdx}$  set equal to 0 as inputs, and the output is assigned to  $\text{availableA}$ . The variable  $\text{qP}_{\text{Y,A}}$  is derived as follows:
  - If one or more of the following conditions are true,  $\text{qP}_{\text{Y,A}}$  is set equal to  $\text{qP}_{\text{Y,PREV}}$ :
    - $\text{availableA}$  is equal to FALSE.
    - The CTB containing the luma coding block covering the luma location (  $\text{xQg} - 1, \text{yQg}$  ) is not equal to the CTB containing the current luma coding block at (  $\text{xCb}, \text{yCb}$  ), i.e. one or more of the following conditions are true:
      - (  $\text{xQg} - 1$  )  $\gg$   $\text{CtbLog2SizeY}$  is not equal to (  $\text{xCb}$  )  $\gg$   $\text{CtbLog2SizeY}$
      - (  $\text{yQg}$  )  $\gg$   $\text{CtbLog2SizeY}$  is not equal to (  $\text{yCb}$  )  $\gg$   $\text{CtbLog2SizeY}$
  - Otherwise,  $\text{qP}_{\text{Y,A}}$  is set equal to the luma quantization parameter  $\text{Qp}_{\text{Y}}$  of the coding unit containing the luma coding block covering (  $\text{xQg} - 1, \text{yQg}$  ).



3. The derivation process for neighbouring block availability as specified in subclause 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 cldx 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:

$$qP_{Y\_PRED} = ( qP_{Y\_A} + qP_{Y\_B} + 1 ) \gg 1 \quad (1113)$$

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:

$$Qp'_{cb} = \text{Clip3}( -QpBdOffset, 63, qP_{cb} + pps\_cb\_qp\_offset + sh\_cb\_qp\_offset + CuQpOffset_{cb} ) + QpBdOffset \quad (1120)$$

$$Qp'_{cr} = \text{Clip3}( -QpBdOffset, 63, qP_{cr} + pps\_cr\_qp\_offset + sh\_cr\_qp\_offset + CuQpOffset_{cr} ) + QpBdOffset \quad (1121)$$

$$Qp'_{cbCr} = \text{Clip3}( -QpBdOffset, 63, qP_{cbCr} + pps\_joint\_cbcr\_qp\_offset\_value + sh\_joint\_cbcr\_qp\_offset + CuQpOffset_{cbCr} ) + QpBdOffset \quad (1122)$$

## 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 subclause 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:

$$res[ x ][ y ] = d[ x ][ y ] \quad (1123)$$

- Otherwise (transform\_skip\_flag[ xTbY ][ yTbY ][ codedCIdx ] is equal to 0), the transformation process for scaled transform coefficients as specified in subclause 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  $\text{resSamples}[x][y]$  with  $x = 0..nTbW - 1$ ,  $y = 0..nTbH - 1$  are derived as follows:

— If  $cIdx$  is equal to  $\text{codedCIdx}$ , the following applies:

$$\text{resSamples}[x][y] = \text{res}[x][y] \quad (1124)$$

— Otherwise, if  $\text{TuCResMode}[xTbY][yTbY]$  is equal to 2, the following applies:

$$\text{resSamples}[x][y] = \text{cSign} * \text{res}[x][y] \quad (1125)$$

— Otherwise, the following applies:

$$\text{resSamples}[x][y] = (\text{cSign} * \text{res}[x][y]) >> 1 \quad (1126)$$

### 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  $\text{predMode}$  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  $(nTbW) \times (nTbH)$  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:

$$qP = Qp'_Y \quad (1127)$$

$$QpActOffset = \text{cu\_act\_enabled\_flag}[xTbY][yTbY] ? -5 : 0 \quad (1128)$$

— Otherwise, if  $\text{TuCResMode}[xTbY][yTbY]$  is equal to 2, the following applies:

$$qP = Qp'_{CbCr} \quad (1129)$$

$$QpActOffset = \text{cu\_act\_enabled\_flag}[xTbY][yTbY] ? 1 : 0 \quad (1130)$$

— Otherwise, if  $cIdx$  is equal to 1, the following applies:

$$qP = Qp'_{Cb} \quad (1131)$$

$$QpActOffset = \text{cu\_act\_enabled\_flag}[xTbY][yTbY] ? 1 : 0 \quad (1132)$$

— Otherwise ( $cIdx$  is equal to 2), the following applies:

$$qP = Qp'_{Cr} \quad (1133)$$

$$QpActOffset = \text{cu\_act\_enabled\_flag}[xTbY][yTbY] ? 3 : 0 \quad (1134)$$

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:

$$qP = Clip3(0, 63 + QpBdOffset, qP + QpActOffset) \quad (1135)$$

$$rectNonTsFlag = ((\log_2(nTbW) + \log_2(nTbH)) \& 1) == 1) ? 1 : 0 \quad (1136)$$

$$bdShift = BitDepth + rectNonTsFlag + ((\log_2(nTbW) + \log_2(nTbH)) / 2) + 10 - \log_2 TransformRange + sh\_dep\_quant\_used\_flag \quad (1137)$$

— Otherwise ( $transform\_skip\_flag[xTbY][yTbY][cIdx]$  is equal to 1), the following applies:

$$qP = Clip3(QpPrimeTsMin, 63 + QpBdOffset, qP + QpActOffset) \quad (1138)$$

$$rectNonTsFlag = 0 \quad (1139)$$

$$bdShift = 10 \quad (1140)$$

The variable  $bdOffset$  is derived as follows:

$$bdOffset = (1 \ll bdShift) \gg 1 \quad (1141)$$

The list  $levelScale[j][k]$  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  $(nTbW) \times (nTbH)$  array  $dz$  is set equal to the  $(nTbW) \times (nTbH)$  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 1 and  $cu\_act\_enabled\_flag[xTbY][yTbY]$ .

— Otherwise, the following applies:

— The variable  $id$  is derived based on  $predMode, cIdx, nTbW$ , and  $nTbH$  as specified in Table 38 and the variable  $\log_2 MatrixSize$  is derived as follows:

$$\log_2 MatrixSize = (id < 2) ? 1 : (id < 8) ? 2 : 3 \quad (1142)$$

— The scaling factor  $m[x][y]$  is derived as follows:

$$m[x][y] = ScalingMatrixRec[id][i][j] \\ \text{with } i = (x \ll \log_2 MatrixSize) \gg \log_2(nTbW), \\ j = (y \ll \log_2 MatrixSize) \gg \log_2(nTbH) \quad (1143)$$

- If  $id$  is greater than 13 and both  $x$  and  $y$  are equal to 0,  $m[0][0]$  is further modified as follows:

$$m[0][0] = \text{ScalingMatrixDcRec}[id - 14] \quad (1144)$$

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:

$$ls[x][y] = (m[x][y] * levelScale[rectNonTsFlag][(qP + 1) \% 6]) << ((qP + 1) / 6) \quad (1145)$$

Otherwise ( $sh\_dep\_quant\_used\_flag$  is equal to 0 or  $transform\_skip\_flag[xTbY][yTbY][cIdx]$  is equal to 1), the following applies:

$$ls[x][y] = (m[x][y] * levelScale[rectNonTsFlag][qP \% 6]) << (qP / 6) \quad (1146)$$

- 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:

$$dz[x][y] = \text{Clip3}(\text{CoeffMin}, \text{CoeffMax}, dz[x - 1][y] + dz[x][y]) \quad (1147)$$

- Otherwise, if  $BdpcmDir[xTbY][yTbY][cIdx]$  is equal to 1 and  $y$  is greater than 0, the following applies:

$$dz[x][y] = \text{Clip3}(\text{CoeffMin}, \text{CoeffMax}, dz[x][y - 1] + dz[x][y]) \quad (1148)$$

- The value  $dnc[x][y]$  is derived as follows:

$$dnc[x][y] = (dz[x][y] * ls[x][y] + bdOffset) >> bdShift \quad (1149)$$

- The scaled transform coefficient  $d[x][y]$  is derived as follows:

$$d[x][y] = \text{Clip3}(\text{CoeffMin}, \text{CoeffMax}, dnc[x][y]) \quad (1150)$$

**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 |
|--------------------------------------------------|----------------------|---|---|----|----|----|----|
| <b>predMode =<br/>MODE_INTRA</b>                 | <b>cIdx = 0 (Y)</b>  |   | 2 | 8  | 14 | 20 | 26 |
|                                                  | <b>cIdx = 1 (Cb)</b> |   | 3 | 9  | 15 | 21 | 21 |
|                                                  | <b>cIdx = 2 (Cr)</b> |   | 4 | 10 | 16 | 22 | 22 |
| <b>predMode =<br/>MODE_INTER or<br/>MODE_IBC</b> | <b>cIdx = 0 (Y)</b>  |   | 5 | 11 | 17 | 23 | 27 |
|                                                  | <b>cIdx = 1 (Cb)</b> | 0 | 6 | 12 | 18 | 24 | 24 |
|                                                  | <b>cIdx = 2 (Cr)</b> | 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 (  $x_{TbY}$ ,  $y_{TbY}$  ) 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  $cldx$  specifying the colour component of the current block,
- an  $(nTbW) \times (nTbH)$  array  $d[x][y]$  of scaled transform coefficients with  $x = 0..nTbW - 1$ ,  $y = 0..nTbH - 1$ .

Output of this process is the  $(nTbW) \times (nTbH)$  array  $res[x][y]$  of residual samples with  $x = 0..nTbW - 1$ ,  $y = 0..nTbH - 1$ .

When  $ApplyLfnstFlag[cldx]$  is equal to 1 and  $transform\_skip\_flag[x_{TbY}][y_{TbY}][cldx]$  is equal to 0, the following applies:

- The variables  $predModeIntra$ ,  $nLfnstOutSize$ ,  $log2LfnstSize$ ,  $nLfnstSize$ , and  $nonZeroSize$  are derived as follows:

$$predModeIntra = (cldx == 0) ? IntraPredModeY[x_{TbY}][y_{TbY}] : IntraPredModeC[x_{TbY}][y_{TbY}] \quad (1151)$$

$$nLfnstOutSize = (nTbW \geq 8 \ \&\& \ nTbH \geq 8) ? 48 : 16 \quad (1152)$$

$$log2LfnstSize = (nTbW \geq 8 \ \&\& \ nTbH \geq 8) ? 3 : 2 \quad (1153)$$

$$nLfnstSize = 1 \ll log2LfnstSize \quad (1154)$$

$$nonZeroSize = ((nTbW == 4 \ \&\& \ nTbH == 4) || (nTbW == 8 \ \&\& \ nTbH == 8)) ? 8 : 16 \quad (1155)$$

- When  $IntraMipFlag[x_{TbY}][y_{TbY}]$  is equal to 1 and  $cldx$  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  $IntraMipFlag[x_{TbY} + nTbW * SubWidthC / 2][y_{TbY} + nTbH * SubHeightC / 2]$  is equal to 1,  $predModeIntra$  is set equal to  $INTRA\_PLANAR$ .
  - Otherwise, if  $CuPredMode[0][x_{TbY} + nTbW * SubWidthC / 2][y_{TbY} + nTbH * SubHeightC / 2]$  is equal to  $MODE\_IBC$  or  $MODE\_PLT$ ,  $predModeIntra$  is set equal to  $INTRA\_DC$ .
  - Otherwise,  $predModeIntra$  is set equal to  $IntraPredModeY[x_{TbY} + nTbW * SubWidthC / 2][y_{TbY} + nTbH * SubHeightC / 2]$ .
- The wide angle intra prediction mode mapping process as specified in subclause 8.4.5.2.7 is invoked with  $predModeIntra$ ,  $nCbW$ ,  $nCbH$ ,  $nTbW$ ,  $nTbH$  and  $cldx$  as inputs, and the modified  $predModeIntra$  as output.



- The values of the list  $u[x]$  with  $x = 0..nonZeroSize - 1$  are derived as follows:

$$xC = \text{DiagScanOrder}[2][2][x][0] \quad (1156)$$

$$yC = \text{DiagScanOrder}[2][2][x][1] \quad (1157)$$

$$u[x] = d[xC][yC] \quad (1158)$$

- The one-dimensional low frequency non-separable transformation process as specified in subclause 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:

$$d[x][y] = (y < 4) ? v[x + (y << \log2LfnstSize)] : ((x < 4) ? v[32 + x + ((y - 4) << 2)] : d[x][y]) \quad (1159)$$

- Otherwise, the following applies:

$$d[x][y] = (x < 4) ? v[y + (x << \log2LfnstSize)] : ((y < 4) ? v[32 + y + ((x - 4) << 2)] : d[x][y]) \quad (1160)$$

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:

$$trTypeHor = (nTbW \geq 4 \ \&\& \ nTbW \leq 16) ? 1 : 0 \quad (1161)$$

$$trTypeVer = (nTbH \geq 4 \ \&\& \ nTbH \leq 16) ? 1 : 0 \quad (1162)$$

- Otherwise,  $trTypeHor$  and  $trTypeVer$  are specified in Table 39 depending on  $mts\_idx$ .

The variables nonZeroW and nonZeroH are derived as follows:

— If ApplyLfnstFlag[ cldx ] is equal to 1, the following applies:

$$\text{nonZeroW} = ( \text{nTbW} == 4 \mid \mid \text{nTbH} == 4 ) ? 4 : 8 \quad (1163)$$

$$\text{nonZeroH} = ( \text{nTbW} == 4 \mid \mid \text{nTbH} == 4 ) ? 4 : 8 \quad (1164)$$

— Otherwise, the following applies:

$$\text{nonZeroW} = \text{Min}( \text{nTbW}, ( \text{trTypeHor} > 0 ) ? 16 : 32 ) \quad (1165)$$

$$\text{nonZeroH} = \text{Min}( \text{nTbH}, ( \text{trTypeVer} > 0 ) ? 16 : 32 ) \quad (1166)$$

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 subclause 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:

$$g[ x ][ y ] = \text{Clip3}( \text{CoeffMin}, \text{CoeffMax}, ( e[ x ][ y ] + 64 ) \gg 7 ) \quad (1167)$$

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 subclause 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:

$$\text{bdShift} = ( \text{nTbH} > 1 \ \&\& \ \text{nTbW} > 1 ) ? ( 5 + \text{Log2TransformRange} - \text{BitDepth} ) : ( 6 + \text{Log2TransformRange} - \text{BitDepth} ) \quad (1168)$$

$$\text{res}[ x ][ y ] = ( r[ x ][ y ] + ( 1 \ll ( \text{bdShift} - 1 ) ) ) \gg \text{bdShift} \quad (1169)$$

**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 subclause 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  $(nTrS) \times (nonZeroSize)$  LFNST matrix lowFreqTransMatrix as output.

The list of transformed samples  $y[i]$  with  $i = 0..nTrS - 1$  is derived as follows:

$$y[i] = \text{Clip3}(\text{CoeffMin}, \text{CoeffMax}, ( (\sum_{j=0}^{nonZeroSize-1} \text{lowFreqTransMatrix}[i][j] * x[j]) + 64 ) >> 7 ) \quad (1170)$$

**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 19 -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 -74 4 39 -5 0 -1 -20 -1 }
{ -17 -28 12 -8 -32 14 -53 36 -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 3 -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 -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  }
},
```



$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol16to31}[m - 16][n]$  with  $m = 16..31$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol16to31} =$

```
{
{ 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 }
},
```

$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol32to47}[m - 32][n]$  with  $m = 32..47$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol32to47} =$

```
{
{ 3 0 -1 0 1 0 0 0 1 0 0 0 1 0 0 0 }
{ 7 2 -2 0 0 -1 1 0 0 2 1 -1 0 0 0 0 }
{ 6 -3 0 0 2 0 0 -1 0 2 -1 0 0 1 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:

$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol0to15}[m][n]$  with  $m = 0..15$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol0to15} =$

```
{
{ -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 }
},
```

$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol16to31}[m - 16][n]$  with  $m = 16..31$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol16to31} =$

```
{
{ 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 }
},
```

$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol32to47}[m - 32][n]$  with  $m = 32..47$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol32to47} =$

```
{
{ 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 2 2 -1 -2 0 1 -1 0 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 -1 }
{ -10 -34 -25 13 -1 0 11 5 1 -1 1 -2 0 0 2 0 }
},
```

- Otherwise, if  $nTrS$  is equal to 48,  $lnstTrSetIdx$  is equal to 1, and  $lnst\_idx$  is equal to 1 the following applies:

$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol0to15}[m][n]$  with  $m = 0..15$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol0to15} =$

```
{
{ 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 }
},
```

$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol16to31}[m - 16][n]$  with  $m = 16..31$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol16to31} =$

```
{
{ -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 }
},
```

$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol32to47}[m - 32][n]$  with  $m = 32..47$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol32to47} =$

```
{
{ -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:

$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol0to15}[m][n]$  with  $m = 0..15$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol0to15} =$

```
{
{ 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 }
},
```

$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol16to31}[m - 16][n]$  with  $m = 16..31$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol16to31} =$

```
{
{ 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 }
},
```

$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol32to47}[m - 32][n]$  with  $m = 32..47$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol32to47} =$

```
{
{ 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:

$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol0to15}[m][n]$  with  $m = 0..15$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol0to15} =$

```
{
{ -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 }
},
```

$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol16to31}[m - 16][n]$  with  $m = 16..31$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol16to31} =$

```
{
{ 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 }
},
```

$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol32to47}[m - 32][n]$  with  $m = 32..47$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol32to47} =$

```
{
{ 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:

$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol0to15}[m][n]$  with  $m = 0..15$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol0to15} =$

```
{
{ 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 }
},
```

$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol16to31}[m - 16][n]$  with  $m = 16..31$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol16to31} =$

```
{
{ 30 -5 -6 1 -1 0 0 0 0 -8 -3 3 0 0 0 0 0 }
{ 72 -29 -2 0 -1 0 -1 0 0 -37 6 7 -2 1 0 0 0 }
{ 7 -38 10 0 1 0 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 0 }
{ 66 17 -24 4 -3 1 -1 0 13 -49 15 1 0 0 0 0 0 }
{ -9 69 -33 5 -2 0 -1 0 -44 -31 10 7 -2 2 0 1 0 }
{ -47 -34 -27 5 4 -1 1 0 -39 -2 27 4 -2 1 0 0 0 }
{ -33 3 22 -2 -4 1 -1 0 -58 -17 6 -6 7 -1 1 0 0 }
{ 7 -8 16 -6 4 -2 1 -1 -15 54 -23 2 -1 0 0 0 0 }
{ -13 17 0 -2 0 -1 0 0 -46 -10 -10 4 -1 1 0 0 0 }
{ 4 51 -3 -6 -1 -1 0 0 -20 6 -34 9 -2 2 -1 0 0 }
{ -1 -4 -68 35 -5 5 -2 1 0 35 43 -4 -6 1 -1 0 0 }
{ -6 -37 -18 -5 2 -2 1 -1 6 -6 -7 25 -6 4 -1 1 0 }
{ -4 -7 -26 -6 -10 6 -4 1 3 8 14 -18 15 -5 2 -1 0 }
{ 1 24 3 5 -1 1 0 0 -3 12 6 -10 1 -1 0 0 0 }
{ 1 4 0 33 -7 5 -2 1 0 -9 53 -22 3 -1 0 0 0 }
},
```

$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol32to47}[m - 32][n]$  with  $m = 32..47$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol32to47} =$

```
{
{ 3 2 -1 0 -2 -1 0 0 1 1 0 0 0 -1 0 0 0 }
{ 12 3 -4 0 -3 -2 1 0 4 0 0 0 -1 0 0 0 0 }
{ 31 -5 -8 3 -14 0 5 -1 6 1 -3 0 -4 -1 1 0 0 }
{ -19 2 0 0 5 1 1 0 -2 0 -1 0 1 0 0 0 0 }
{ -53 34 6 -5 30 -7 -11 3 -11 -2 5 1 4 2 -1 -1 0 }
{ 49 7 2 -6 -23 -3 -2 2 9 4 0 0 -2 -1 -1 0 0 }
{ -11 32 -8 -7 27 -12 -6 6 -13 0 4 -3 3 -1 -2 1 0 }
{ -23 40 -2 5 43 -11 -8 -1 -18 -4 5 2 4 3 0 -1 0 }
{ -42 -25 4 6 34 8 2 -2 -15 -1 0 -1 3 2 0 1 0 }
{ -80 -27 20 -4 -66 23 -2 -2 20 -3 -2 3 -14 2 3 -1 0 }
{ 16 -52 28 1 59 15 -8 -5 -28 -7 2 2 10 3 0 -1 0 }
{ -14 -38 -12 -10 9 5 7 6 -9 7 -4 -3 4 -4 0 3 0 }
{ 16 10 55 -24 15 46 -52 1 35 -43 10 12 -23 13 5 -8 0 }
{ -2 -4 -1 13 0 2 -4 -3 3 -1 2 1 -2 0 -2 -1 0 }
{ -9 -1 -25 10 45 -11 18 2 86 1 -13 -4 -65 -6 7 2 0 }
{ 4 -27 -2 -9 5 36 -13 5 -7 -17 1 2 4 6 4 -1 0 }
},
```

— Otherwise, if  $nTrS$  is equal to 48,  $lfnstTrSetIdx$  is equal to 3, and  $lfnst\_idx$  is equal to 1 the following applies:

$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol0to15}[m][n]$  with  $m = 0..15$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol0to15} =$

```
{
{ -115 37 9 2 2 1 1 0 10 -29 8 0 1 0 1 0 0 }
{ 15 51 -18 0 -3 0 -1 0 -95 7 34 -3 5 -1 2 0 0 }
{ 29 -22 16 -6 3 -2 1 -1 -4 -80 12 15 0 3 0 1 0 }
{ -36 -98 25 5 4 1 2 1 -59 11 -17 1 1 1 0 0 0 }
{ -6 18 3 -3 -1 0 0 0 -50 -5 -38 12 0 2 0 1 0 }
{ 4 15 52 -13 5 -3 2 -1 -17 -45 16 24 -2 4 -1 2 0 }
{ -20 -7 -43 4 0 1 -1 1 -7 35 0 12 -4 1 -1 0 0 }
{ 4 29 1 26 -5 4 -2 1 -17 -7 -73 6 6 2 1 1 0 }
{ 12 13 10 2 -1 3 -1 1 17 -2 -46 12 7 0 2 0 0 }
{ 5 20 90 -17 4 -3 2 -1 6 66 8 28 -7 3 -1 1 0 }
{ -3 -4 -34 -12 2 -1 -1 0 5 25 11 43 -10 4 -2 1 0 }
{ -1 -3 2 19 -2 4 -1 2 9 3 -35 22 11 1 2 0 0 }
{ 10 -4 -6 12 5 1 1 0 11 -9 -12 -2 -7 0 -1 0 0 }
{ 4 6 14 53 -4 4 0 2 0 -1 -20 -13 3 2 -1 1 0 }
{ 2 9 13 37 19 6 2 2 -9 -3 -9 -28 -20 -4 -3 -1 0 }
{ 3 -3 12 84 -12 8 -2 3 6 13 50 -1 45 1 7 0 0 }
},
```



$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol16to31}[m - 16][n]$  with  $m = 16..31$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol16to31} =$

```
{
{ 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 }
},
```

$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol32to47}[m - 32][n]$  with  $m = 32..47$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol32to47} =$

```
{
{ 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:

$\text{lowFreqTransMatrix}[m][n] = \text{lowFreqTransMatrixCol0to15}[m][n]$  with  $m = 0..15$ ,  $n = 0..15$

$\text{lowFreqTransMatrixCol0to15} =$

```
{
{ 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  -5  17  -7 -18   3   3  -1  -5   }
{ -21 -11   1  40  -5  -4 -24  -5  -4   5  -6  -5   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 subclause 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:

$$y[i] = \sum_{j=0}^{\text{nonZeroS}-1} \text{transMatrix}[i][j * 2^{6-\text{Log2}(\text{nTbS})}] * x[j] \quad \text{with } i = 0..nTbS - 1 \quad (1171)$$

- Otherwise (trType is equal to 1 or trType is equal to 2), the following transform matrix multiplication applies:

$$y[i] = \sum_{j=0}^{\text{nonZeroS}-1} \text{transMatrix}[i][j] * x[j] \text{ with } i = 0..n\text{TbS} - 1 \quad (1172)$$

#### 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:

$$\text{transMatrix}[m][n] = \text{transMatrixCol0to15}[m][n] \text{ with } m = 0..15, n = 0..63 \quad (1173)$$

$$\text{transMatrixCol0to15} = \quad (1174)$$

|   |    |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |   |
|---|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|
| { | 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 }
},

```

$$\text{transMatrix}[m][n] = (n \& 1 ? -1 : 1) * \text{transMatrixCol16to31}[47 - m][n] \quad (1177)$$

with  $m = 32..47, n = 0..63$

$$\text{transMatrix}[m][n] = (n \& 1 ? -1 : 1) * \text{transMatrixCol0to15}[63 - m][n] \quad (1178)$$

with  $m = 48..63, n = 0..63$

— Otherwise, if trType is equal to 1 and nTbS is equal to 4, the following applies:

$$\text{transMatrix}[m][n] = \quad (1179)$$

```

{
{ 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:

$$\text{transMatrix}[m][n] = \quad (1180)$$

```

{
{ 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:

$$\text{transMatrix}[m][n] = \quad (1181)$$

```

{
{ 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:

$$\text{transMatrix}[m][n] = \text{transMatrixCol0to15}[m][n] \text{ with } m = 0..15, n = 0..15 \quad (1182)$$

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 -14 -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 -69 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  $(nTbW) \times (nTbH)$  array of luma residual samples  $r_Y$  with elements  $r_Y[x][y]$ ,
- an  $(nTbW) \times (nTbH)$  array of chroma residual samples  $r_{Cb}$  with elements  $r_{Cb}[x][y]$ ,
- an  $(nTbW) \times (nTbH)$  array of chroma residual samples  $r_{Cr}$  with elements  $r_{Cr}[x][y]$ .

Outputs of this process are:

- a modified  $(nTbW) \times (nTbH)$  array  $r_Y$  of luma residual samples,
- a modified  $(nTbW) \times (nTbH)$  array  $r_{Cb}$  of chroma residual samples,
- a modified  $(nTbW) \times (nTbH)$  array  $r_{Cr}$  of chroma residual samples.

The  $(nTbW) \times (nTbH)$  arrays of residual samples  $r_Y$ ,  $r_{Cb}$  and  $r_{Cr}$  are modified as follows:

$$r_Y[x][y] = \text{Clip3}(- (1 \ll (\text{BitDepth} + 1)), (1 \ll (\text{BitDepth} + 1)) - 1, r_Y[x][y]) \quad (1193)$$

$$r_{Cb}[x][y] = \text{Clip3}(- (1 \ll (\text{BitDepth} + 1)), (1 \ll (\text{BitDepth} + 1)) - 1, r_{Cb}[x][y]) \quad (1194)$$

$$r_{Cr}[x][y] = \text{Clip3}(- (1 \ll (\text{BitDepth} + 1)), (1 \ll (\text{BitDepth} + 1)) - 1, r_{Cr}[x][y]) \quad (1195)$$

$$\text{tmp} = r_Y[x][y] - (r_{Cb}[x][y] \gg 1) \quad (1196)$$

$$r_Y[x][y] = \text{tmp} + r_{Cb}[x][y] \quad (1197)$$

$$r_{Cb}[x][y] = \text{tmp} - (r_{Cr}[x][y] \gg 1) \quad (1198)$$

$$r_{Cr}[x][y] += r_{Cb}[x][y] \quad (1199)$$

## 8.7.5 Picture reconstruction process

### 8.7.5.1 General

Inputs to this process are:

- a location  $(x_{\text{Curr}}, y_{\text{Curr}})$  specifying the top-left sample of the current block relative to the top-left sample of the current picture component,
- the variables  $n_{\text{CurrSw}}$  and  $n_{\text{CurrSh}}$  specifying the width and height, respectively, of the current block,
- a variable  $cIdx$  specifying the colour component of the current block,
- an  $n_{\text{CurrSw}} \times n_{\text{CurrSh}}$  array  $\text{predSamples}$  specifying the predicted samples of the current block,
- an  $n_{\text{CurrSw}} \times n_{\text{CurrSh}}$  array  $\text{resSamples}$  specifying the residual samples of the current block.

Output of this process is a reconstructed picture sample array  $\text{recSamples}$ .

Depending on the value of the colour component  $cIdx$ , the following assignments are made:

- If  $cIdx$  is equal to 0,  $\text{recSamples}$  corresponds to the reconstructed picture sample array  $S_L$ .
- Otherwise, if  $cIdx$  is equal to 1,  $\text{tuCbfChroma}$  is set equal to  $(\text{tu\_cb\_coded\_flag}[x_{\text{Curr}} * \text{SubWidthC}][y_{\text{Curr}} * \text{SubHeightC}]) \mid \mid \text{tu\_joint\_cbr\_residual\_flag}[x_{\text{Curr}} * \text{SubWidthC}][y_{\text{Curr}} * \text{SubHeightC}])$ ,  $\text{recSamples}$  corresponds to the reconstructed chroma sample array  $S_{Cb}$ .
- Otherwise ( $cIdx$  is equal to 2),  $\text{tuCbfChroma}$  is set equal to  $(\text{tu\_cr\_coded\_flag}[x_{\text{Curr}} * \text{SubWidthC}][y_{\text{Curr}} * \text{SubHeightC}]) \mid \mid \text{tu\_joint\_cbr\_residual\_flag}[x_{\text{Curr}} * \text{SubWidthC}][y_{\text{Curr}} * \text{SubHeightC}])$ ,  $\text{recSamples}$  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`:
 
$$\text{recSamples}[ \text{xCurr} + i ][ \text{yCurr} + j ] = \text{Clip1}( \text{predSamples}[ i ][ j ] + \text{resSamples}[ i ][ j ] ) \quad (1200)$$
- 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 subclause 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 subclause 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`:

$$\text{xVb} = ( \text{xCurr} + i ) \% ( ( \text{cIdx} == 0 ) ? \text{IbcBufWidthY} : \text{IbcBufWidthC} ) \quad (1201)$$

$$\text{yVb} = ( \text{yCurr} + j ) \% ( ( \text{cIdx} == 0 ) ? \text{CtbSizeY} : ( \text{CtbSizeY} / \text{SubHeightC} ) ) \quad (1202)$$

$$\text{IbcVirBuf}[ \text{cIdx} ][ \text{xVb} ][ \text{yVb} ] = \text{recSamples}[ \text{xCurr} + i ][ \text{yCurr} + j ] \quad (1203)$$

The following assignments are made for `i = 0..nCurrSw * SubWidthC - 1, j = 0..nCurrSh * SubHeightC - 1`:

$$\text{IsAvailable}[ \text{cIdx} ][ \text{xCurr} * \text{SubWidthC} + i ][ \text{yCurr} * \text{SubHeightC} + j ] = \text{TRUE} \quad (1204)$$

### 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`.

- $\text{CuPredMode}[0][x_{\text{Curr}}][y_{\text{Curr}}]$  is equal to  $\text{MODE\_PLT}$ .
- $\text{CuPredMode}[0][x_{\text{Curr}}][y_{\text{Curr}}]$  is equal to  $\text{MODE\_INTER}$  and  $\text{ciip\_flag}[x_{\text{Curr}}][y_{\text{Curr}}]$  is equal to 1.
- Otherwise ( $\text{CuPredMode}[0][x_{\text{Curr}}][y_{\text{Curr}}]$  is equal to  $\text{MODE\_INTER}$  and  $\text{ciip\_flag}[x_{\text{Curr}}][y_{\text{Curr}}]$  is equal to 0), the following applies:
 
$$\begin{aligned} \text{idxY} &= \text{predSamples}[i][j] \gg \text{Log2}(\text{OrgCW}) \\ \text{predMapSamples}[i][j] &= \text{LmcsPivot}[\text{idxY}] + \\ &\quad ((\text{ScaleCoeff}[\text{idxY}] * (\text{predSamples}[i][j] - \text{InputPivot}[\text{idxY}]) + (1 \ll 10)) \gg 11) \quad (1205) \end{aligned}$$
 with  $i = 0..n_{\text{CurrSw}} - 1, j = 0..n_{\text{CurrSh}} - 1$

The reconstructed luma picture sample  $\text{recSamples}$  is derived as follows:

$$\text{recSamples}[x_{\text{Curr}} + i][y_{\text{Curr}} + j] = \text{Clip1}(\text{predMapSamples}[i][j] + \text{resSamples}[i][j]) \quad (1206)$$

with  $i = 0..n_{\text{CurrSw}} - 1, j = 0..n_{\text{CurrSh}} - 1$

### 8.7.5.3 Picture reconstruction with luma dependent chroma residual scaling process for chroma samples

Inputs to this process are:

- a chroma location  $(x_{\text{Curr}}, y_{\text{Curr}})$  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  $n_{\text{CurrSw}}$  specifying the chroma transform block width,
- a variable  $n_{\text{CurrSh}}$  specifying the chroma transform block height,
- a variable  $\text{tuCbfChroma}$  specifying the coded block flag of the current chroma transform block,
- an  $n_{\text{CurrSw}} \times n_{\text{CurrSh}}$  array  $\text{predSamples}$  specifying the chroma prediction samples of the current block,
- an  $n_{\text{CurrSw}} \times n_{\text{CurrSh}}$  array  $\text{resSamples}$  specifying the chroma residual samples of the current block.

Output of this process is a reconstructed chroma picture sample array  $\text{recSamples}$ .

The variable  $\text{sizeY}$  is set equal to  $\text{Min}(\text{CtbSizeY}, 64)$ .

The reconstructed chroma picture sample  $\text{recSamples}$  is derived as follows for  $i = 0..n_{\text{CurrSw}} - 1, j = 0..n_{\text{CurrSh}} - 1$ :

- If one or more of the following conditions are true,  $\text{recSamples}[x_{\text{Curr}} + i][y_{\text{Curr}} + j]$  is set equal to  $\text{Clip1}(\text{predSamples}[i][j] + \text{resSamples}[i][j])$ :
  - $\text{ph\_chroma\_residual\_scale\_flag}$  is equal to 0.
  - $\text{sh\_lmcs\_used\_flag}$  is equal to 0.
  - $n_{\text{CurrSw}} * n_{\text{CurrSh}}$  is less than or equal to 4.
  - $\text{tu\_cb\_coded\_flag}[x_{\text{Curr}} * \text{SubWidthC}][y_{\text{Curr}} * \text{SubHeightC}]$  is equal to 0,  $\text{tu\_cr\_coded\_flag}[x_{\text{Curr}} * \text{SubWidthC}][y_{\text{Curr}} * \text{SubHeightC}]$  is equal to 0, and  $\text{cu\_act\_enabled\_flag}[x_{\text{Curr}} * \text{SubWidthC}][y_{\text{Curr}} * \text{SubHeightC}]$  is equal to 0.
- Otherwise, the following applies:
  - The current luma location  $(x_{\text{CurrY}}, y_{\text{CurrY}})$  is derived as follows:

$$(x_{\text{CurrY}}, y_{\text{CurrY}}) = (x_{\text{Curr}} * \text{SubWidthC}, y_{\text{Curr}} * \text{SubHeightC}) \quad (1207)$$

- The luma location (  $x_{CuCb}$ ,  $y_{CuCb}$  ) is specified as the top-left luma sample location of the coding unit that contains the luma sample at (  $x_{CurrY} / sizeY * sizeY$ ,  $y_{CurrY} / sizeY * sizeY$  ).
- The variables  $availL$  and  $availT$  are derived as follows:
  - The derivation process for neighbouring block availability as specified in subclause 6.4.4 is invoked with the location (  $x_{Curr}$ ,  $y_{Curr}$  ) set equal to (  $x_{CuCb}$ ,  $y_{CuCb}$  ), the neighbouring luma location (  $x_{NbY}$ ,  $y_{NbY}$  ) set equal to (  $x_{CuCb} - 1$ ,  $y_{CuCb}$  ),  $checkPredModeY$  set equal to FALSE, and  $cldx$  set equal to 0 as inputs, and the output is assigned to  $availL$ .
  - The derivation process for neighbouring block availability as specified in subclause 6.4.4 is invoked with the location (  $x_{Curr}$ ,  $y_{Curr}$  ) set equal to (  $x_{CuCb}$ ,  $y_{CuCb}$  ), the neighbouring luma location (  $x_{NbY}$ ,  $y_{NbY}$  ) set equal to (  $x_{CuCb}$ ,  $y_{CuCb} - 1$  ),  $checkPredModeY$  set equal to FALSE, and  $cldx$  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  $currPic[x_{CuCb} - 1][Min(y_{CuCb} + i, pps\_pic\_height\_in\_luma\_samples - 1)]$  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  $currPic[Min(x_{CuCb} + i, pps\_pic\_width\_in\_luma\_samples - 1)][y_{CuCb} - 1]$  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:

$$invAvgLuma = ( \sum_{k=0}^{cnt-1} recLuma[k] + (cnt \gg 1) ) \gg \text{Log2}(cnt) \quad (1208)$$

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

$$invAvgLuma = 1 \ll (BitDepth - 1) \quad (1209)$$

2. The variable  $idxYInv$  is derived by invoking the identification of piece-wise function index process for a luma sample as specified in subclause 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:

$$varScale = \text{ChromaScaleCoeff}[idxYInv] \quad (1210)$$

- The reconstructed chroma picture sample array  $recSamples[x_{Curr} + i][y_{Curr} + j]$  with  $i = 0..n_{CurrSw} - 1$ ,  $j = 0..n_{CurrSh} - 1$  is derived as follows:
  - If  $tuCbfcChroma$  is equal to 1 or  $cu\_act\_enabled\_flag[x_{Curr} * SubWidthC][y_{Curr} * SubHeightC]$  is equal to 1, the following applies:

$$\text{resSamples}[i][j] = \text{Clip3}(- (1 \ll \text{BitDepth}), (1 \ll \text{BitDepth}) - 1, \text{resSamples}[i][j]) \quad (1211)$$

$$\begin{aligned} \text{recSamples}[x\text{Curr} + i][y\text{Curr} + j] = & \text{Clip1}(\text{predSamples}[i][j] + \quad (1212) \\ & \text{Sign}(\text{resSamples}[i][j]) * ((\text{Abs}(\text{resSamples}[i][j]) * \text{varScale} + (1 \ll 10)) \gg \\ & 11)) \end{aligned}$$

- Otherwise ( $\text{tuCbfChroma}$  is equal to 0 and  $\text{cu\_act\_enabled\_flag}[x\text{Curr} * \text{SubWidthC}][y\text{Curr} * \text{SubHeightC}]$  is equal to 0), the following applies:

$$\text{recSamples}[x\text{Curr} + i][y\text{Curr} + j] = \text{predSamples}[i][j] \quad (1213)$$

## 8.8 In-loop filter process

### 8.8.1 General

The picture inverse mapping process with luma samples and the three in-loop filters, namely deblocking filter, sample adaptive offset and adaptive loop filter, are applied as specified by the following ordered steps:

1. When  $\text{sps\_lmcs\_enabled\_flag}$  is equal to 1, the following applies:
  - The picture inverse mapping process for luma samples as specified in subclause 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 subclause 8.8.3.1 is invoked with the reconstructed picture sample array  $S_L$  and, when  $\text{sps\_chroma\_format\_idc}$  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  $\text{sps\_chroma\_format\_idc}$  is not equal to 0, the arrays  $S'_{Cb}$  and  $S'_{Cr}$  after deblocking as outputs.
  - The array  $S'_L$  and, when  $\text{sps\_chroma\_format\_idc}$  is not equal to 0, the arrays  $S'_{Cb}$  and  $S'_{Cr}$  are assigned to the array  $S_L$  and, when  $\text{sps\_chroma\_format\_idc}$  is not equal to 0, the arrays  $S_{Cb}$  and  $S_{Cr}$  (which represent the decoded picture), respectively.
3. When  $\text{sps\_sao\_enabled\_flag}$  is equal to 1, the following applies:
  - The sample adaptive offset process as specified in subclause 8.8.4.1 is invoked with the reconstructed picture sample array  $S_L$  and, when  $\text{sps\_chroma\_format\_idc}$  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  $\text{sps\_chroma\_format\_idc}$  is not equal to 0, the arrays  $S'_{Cb}$  and  $S'_{Cr}$  after sample adaptive offset as outputs.
  - The array  $S'_L$  and, when  $\text{sps\_chroma\_format\_idc}$  is not equal to 0, the arrays  $S'_{Cb}$  and  $S'_{Cr}$  are assigned to the array  $S_L$  and, when  $\text{sps\_chroma\_format\_idc}$  is not equal to 0, the arrays  $S_{Cb}$  and  $S_{Cr}$  (which represent the decoded picture), respectively.
4. When  $\text{sps\_alf\_enabled\_flag}$  is equal to 1, the following applies:
  - The adaptive loop filter process as specified in subclause 8.8.5.1 is invoked with the reconstructed picture sample array  $S_L$  and, when  $\text{sps\_chroma\_format\_idc}$  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  $\text{sps\_chroma\_format\_idc}$  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 subclause 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 subclause 8.8.2.3 with `lumaSample` as the input and `idxYInv` as the output.
2. The variable `invSample` is derived as follows:

$$\text{invSample} = \text{InputPivot}[\text{idxYInv}] + ( (\text{InvScaleCoeff}[\text{idxYInv}] * (\text{lumaSample} - \text{LmcsPivot}[\text{idxYInv}]) + (1 \ll 10) ) \gg 11 ) \quad (1214)$$

3. The inverse mapped luma sample `invLumaSample` is derived as follows:

$$\text{invLumaSample} = \text{Clip1}(\text{invSample}) \quad (1215)$$

- 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 ] )
        break
}
idxYInv = Min( idxYInv, 15 )
```

(1216)

### 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  $\text{recPicture}_L$  and, when  $\text{sps\_chroma\_format\_idc}$  is not equal to 0, the arrays  $\text{recPicture}_{Cb}$  and  $\text{recPicture}_{Cr}$ .

Outputs of this process are the modified reconstructed picture after deblocking, i.e., the array  $\text{recPicture}_L$  and, when  $\text{sps\_chroma\_format\_idc}$  is not equal to 0, the arrays  $\text{recPicture}_{Cb}$  and  $\text{recPicture}_{Cr}$ .

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 document, 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  $\text{subpicIdx}$  and  $\text{sps\_loop\_filter\_across\_subpic\_enabled\_flag}[\text{subpicIdx}]$  is equal to 0,
- Edges that coincide with the virtual boundaries of the picture when  $\text{VirtualBoundariesPresentFlag}$  is equal to 1,
- Edges that coincide with tile boundaries when  $\text{pps\_loop\_filter\_across\_tiles\_enabled\_flag}$  is equal to 0,
- Edges that coincide with slice boundaries when  $\text{pps\_loop\_filter\_across\_slices\_enabled\_flag}$  is equal to 0,
- Edges that coincide with upper or left boundaries of slices with  $\text{sh\_deblocking\_filter\_disabled\_flag}$  equal to 1,
- Edges within slices with  $\text{sh\_deblocking\_filter\_disabled\_flag}$  equal to 1,
- Edges that do not correspond to  $4 \times 4$  sample grid boundaries of the luma component,
- Edges that do not correspond to  $8 \times 8$  sample grid boundaries of the chroma component,
- Edges within the luma component for which both sides of the edge have  $\text{intra\_bdpcm\_luma\_flag}$  equal to 1,
- Edges within the chroma components for which both sides of the edge have  $\text{intra\_bdpcm\_chroma\_flag}$  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  $\text{edgeType}$  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 subclause 8.8.3.2 with the variable treeType, the reconstructed picture prior to deblocking, i.e., the array `recPictureL` and the variable edgeType set equal to EDGE\_VER as inputs, and the modified reconstructed picture after deblocking, i.e., the array `recPictureL` as outputs.
- The horizontal edge are filtered by invoking the deblocking filter process for one direction as specified in subclause 8.8.3.2 with the variable treeType, the modified reconstructed picture after deblocking, i.e., the array `recPictureL` and the variable edgeType set equal to EDGE\_HOR as inputs, and the modified reconstructed picture after deblocking, i.e., the array `recPictureL` 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 subclause 8.8.3.2 with the variable treeType, the reconstructed picture prior to deblocking, i.e., the arrays `recPictureCb` and `recPictureCr`, and the variable edgeType set equal to EDGE\_VER as inputs, and the modified reconstructed picture after deblocking, i.e., the arrays `recPictureCb` and `recPictureCr` as outputs.
  - The horizontal edge are filtered by invoking the deblocking filter process for one direction as specified in subclause 8.8.3.2 with the variable treeType, the modified reconstructed picture after deblocking, i.e., the arrays `recPictureCb` and `recPictureCr`, and the variable edgeType set equal to EDGE\_HOR as inputs, and the modified reconstructed picture after deblocking, i.e., the arrays `recPictureCb` and `recPictureCr` 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 `recPictureL`,
- when `sps_chroma_format_idc` is not equal to 0 and treeType is equal to DUAL\_TREE\_CHROMA, the arrays `recPictureCb` and `recPictureCr`,
- 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 `recPictureL`,
- when `sps_chroma_format_idc` is not equal to 0 and treeType is equal to DUAL\_TREE\_CHROMA, the arrays `recPictureCb` and `recPictureCr`.

The variables `firstCompIdx` and `lastCompIdx` are derived as follows:

$$\text{firstCompIdx} = (\text{treeType} == \text{DUAL\_TREE\_CHROMA}) ? 1 : 0 \quad (1217)$$

$$\text{lastCompIdx} = (\text{treeType} == \text{DUAL\_TREE\_LUMA} \mid \mid \text{sps\_chroma\_format\_idc} == 0) ? 0 : 2 \quad (1218)$$

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 subclause 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 subclause 8.8.3.4 is invoked with the location  $(x_{Cb}, y_{Cb})$ , the coding block width  $n_{CbW}$ , the coding block height  $n_{CbH}$ , 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 subclause 8.8.3.5 is invoked with the picture sample array  $recPicture$ , the location  $(x_{Cb}, y_{Cb})$ , the coding block width  $n_{CbW}$ , the coding block height  $n_{CbH}$ , the variable  $edgeType$ , the variable  $cIdx$ , and the array  $edgeIdc$  as inputs, and an  $(n_{CbW}) \times (n_{CbH})$  array  $bS$  as output.
7. The edge filtering process for one direction is invoked for a coding block as specified in subclause 8.8.3.6 with the variable  $edgeType$ , the variable  $cIdx$ , the reconstructed picture prior to deblocking  $recPicture$ , the location  $(x_{Cb}, y_{Cb})$ , the coding block width  $n_{CbW}$ , the coding block height  $n_{CbH}$ , 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  $(x_{Cb}, y_{Cb})$  specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,
- a variable  $n_{CbW}$  specifying the width of the current coding block,
- a variable  $n_{CbH}$  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  $(n_{CbW}) \times (n_{CbH})$  array  $edgeIdc$ ,
- two-dimensional  $(n_{CbW}) \times (n_{CbH})$  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  $(n_{CbW}) \times (n_{CbH})$  array  $edgeIdc$ ,
- the modified two-dimensional  $(n_{CbW}) \times (n_{CbH})$  arrays  $maxFilterLengthQs$ ,  $maxFilterLengthPs$ .

Depending on  $edgeType$ , the arrays  $edgeIdc$ ,  $maxFilterLengthPs$  and  $maxFilterLengthQs$  are derived as follows:

- The variable  $gridSize$  is set as follows:

$$gridSize = cIdx == 0 ? 4 : 8 \quad (1219)$$

- If edgeType is equal to EDGE\_VER, the following applies:
  - The variable numEdges is set equal to  $\text{Max}(1, \text{nCbW} / \text{gridSize})$ .
  - 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 cldx, edgeIdc[ x ][ y ] is set equal to 1.
  - When edgeIdc[ x ][ y ] is equal to 1, the following applies:
    - If cldx 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 (cldx 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  $\text{Max}(1, \text{nCbH} / \text{gridSize})$ .
  - 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  $\text{edgeIdc}[x][y]$  is derived as follows:
  - If  $y$  is equal to 0,  $\text{edgeIdc}[x][y]$  is set equal to  $\text{filterEdgeFlag}$ .
  - Otherwise, if the transform block covering the location  $(x_{Cb} + x, y_{Cb} + y)$  does not cover the location  $(x_{Cb} + x, y_{Cb} + y - 1)$  for the colour component  $\text{cldx}$ ,  $\text{edgeIdc}[x][y]$  is set equal to 1.
- When  $\text{edgeIdc}[x][y]$  is equal to 1, the following applies:
  - If  $\text{cldx}$  is equal to 0, the following applies:
    - The value of  $\text{maxFilterLengthQs}[x][y]$  is derived as follows:
      - If the height in luma samples of the transform block at luma location  $(x_{Cb} + x, y_{Cb} + y)$  is equal to or less than 4 or the height in luma samples of the transform block at luma location  $(x_{Cb} + x, y_{Cb} + y - 1)$  is equal to or less than 4,  $\text{maxFilterLengthQs}[x][y]$  is set equal to 1.
      - Otherwise, if the height in luma samples of the transform block at luma location  $(x_{Cb} + x, y_{Cb} + y)$  is equal to or greater than 32,  $\text{maxFilterLengthQs}[x][y]$  is set equal to 7.
      - Otherwise,  $\text{maxFilterLengthQs}[x][y]$  is set equal to 3.
    - The value of  $\text{maxFilterLengthPs}[x][y]$  is derived as follows:
      - If the height in luma samples of the transform block at luma location  $(x_{Cb} + x, y_{Cb} + y)$  is equal to or less than 4 or the height in luma samples of the transform block at luma location  $(x_{Cb} + x, y_{Cb} + y - 1)$  is equal to or less than 4,  $\text{maxFilterLengthPs}[x][y]$  is set equal to 1.
      - Otherwise, if the height in luma samples of the transform block at luma location  $(x_{Cb} + x, y_{Cb} + y - 1)$  is equal to or greater than 32,  $\text{maxFilterLengthPs}[x][y]$  is set equal to 7.
      - Otherwise,  $\text{maxFilterLengthPs}[x][y]$  is set equal to 3.
  - Otherwise ( $\text{cldx}$  is not equal to 0), the values of  $\text{maxFilterLengthPs}[x][y]$  and  $\text{maxFilterLengthQs}[x][y]$  are derived as follows:
    - If the height in chroma samples of the transform block at chroma location  $(x_{Cb} + x, y_{Cb} + y)$  and the height in chroma samples of the transform block at chroma location  $(x_{Cb} + x, y_{Cb} + y - 1)$  are both equal to or greater than 8, the following applies:
      - If  $(y_{Cb} + y) \% \text{CtbHeightC}$  is greater than 0, i.e. the horizontal edge do not overlap with the upper chroma CTB boundary, both  $\text{maxFilterLengthPs}[x][y]$  and  $\text{maxFilterLengthQs}[x][y]$  are set equal to 3.
      - Otherwise ( $(y_{Cb} + y) \% \text{CtbHeightC}$  is equal to 0, i.e. the horizontal edge overlaps with the upper chroma CTB boundary),  $\text{maxFilterLengthPs}[x][y]$  is set equal to 1 and  $\text{maxFilterLengthQs}[x][y]$  is set equal to 3.
    - Otherwise,  $\text{maxFilterLengthPs}[x][y]$  and  $\text{maxFilterLengthQs}[x][y]$  are set equal to 1.

#### 8.8.3.4 Derivation process of subblock boundary

Inputs to this process are:

- a location  $(x_{Cb}, y_{Cb})$  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  $(nCbW) \times (nCbH)$  array  $edgeIdc$ ,
- two-dimensional  $(nCbW) \times (nCbH)$  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  $(nCbW) \times (nCbH)$  array  $edgeIdc$ ,
- the modified two-dimensional  $(nCbW) \times (nCbH)$  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:

$$edgeIdc[x][y] = 2 \quad (1220)$$

- 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:

$$maxFilterLengthQs[x][y] = Min(5, maxFilterLengthQs[x][y]) \quad (1221)$$

- When  $InterAffineFlag[xCb - 1][yCb + y]$  is equal to 1 or  $MergeSubblockFlag[xCb - 1][yCb + y]$  is equal to 1, the following applies:

$$maxFilterLengthPs[x][y] = Min(5, maxFilterLengthPs[x][y]) \quad (1222)$$

- Otherwise, if  $edgeTbFlags[x][y]$  is equal to 1, the following applies:

$$maxFilterLengthPs[x][y] = Min(5, maxFilterLengthPs[x][y]) \quad (1223)$$

$$maxFilterLengthQs[x][y] = Min(5, maxFilterLengthQs[x][y]) \quad (1224)$$

— Otherwise, if one or more of the following conditions are true:

- $\text{edgeTbFlags}[x - 4][y]$  is equal to 1,
- $\text{edgeTbFlags}[x + 4][y]$  is equal to 1,

the following applies:

$$\text{maxFilterLengthPs}[x][y] = 1 \quad (1225)$$

$$\text{maxFilterLengthQs}[x][y] = 1 \quad (1226)$$

— Otherwise, if one or more of the following conditions are true:

- $x_{\text{Edge}}$  is equal to 1,
- $x_{\text{Edge}}$  is equal to  $(n_{\text{CbW}} / 8) - 1$ ,
- $\text{edgeTbFlags}[x - \text{sbW}][y]$  is equal to 1,
- $\text{edgeTbFlags}[x + \text{sbW}][y]$  is equal to 1,

the following applies:

$$\text{maxFilterLengthPs}[x][y] = 2 \quad (1227)$$

$$\text{maxFilterLengthQs}[x][y] = 2 \quad (1228)$$

— Otherwise, the following applies:

$$\text{maxFilterLengthPs}[x][y] = 3 \quad (1229)$$

$$\text{maxFilterLengthQs}[x][y] = 3 \quad (1230)$$

— Otherwise, if  $\text{edgeType}$  is equal to  $\text{EDGE\_HOR}$ , the following applies:

- The variable  $\text{sbH}$  is set equal to  $\text{Max}(8, n_{\text{CbH}} / \text{numSbY})$ .
- The array  $\text{edgeTbFlags}$  is set equal to  $\text{edgeIdc}$ .
- For  $y_{\text{Edge}} = 0.. \text{Min}(\text{Max}(1, n_{\text{CbH}} / 8) - 1, \text{numSbY} - 1)$ ,  $x = 0..n_{\text{CbW}} - 1$ :
  - The vertical position  $y$  inside the current coding block is set equal to  $y_{\text{Edge}} * \text{sbH}$ .
  - When  $y$  is greater than 0 or  $\text{filterEdgeFlag}$  is equal to 1, the value of  $\text{edgeIdc}[x][y]$  is modified as follows:

$$\text{edgeIdc}[x][y] = 2 \quad (1231)$$

— When  $\text{edgeIdc}[x][y]$  is equal to 1 or 2, the values of  $\text{maxFilterLengthPs}[x][y]$  and  $\text{maxFilterLengthQs}[x][y]$  are modified as follows:

— If  $y$  is equal to 0, the following applies:

— When  $\text{numSbY}$  is greater than 1, the following applies:

$$\text{maxFilterLengthQs}[x][y] = \text{Min}(5, \text{maxFilterLengthQs}[x][y]) \quad (1232)$$

— When  $\text{InterAffineFlag}[x_{\text{Cb}} + x][y_{\text{Cb}} - 1]$  is equal to 1 or  $\text{MergeSubblockFlag}[x_{\text{Cb}} + x][y_{\text{Cb}} - 1]$  is equal to 1, the following applies:

$$\text{maxFilterLengthPs}[x][y] = \text{Min}(5, \text{maxFilterLengthPs}[x][y]) \quad (1233)$$

- Otherwise, if  $\text{edgeTbFlags}[x][y]$  is equal to 1, the following applies:

$$\text{maxFilterLengthPs}[x][y] = \text{Min}(5, \text{maxFilterLengthPs}[x][y]) \quad (1234)$$

$$\text{maxFilterLengthQs}[x][y] = \text{Min}(5, \text{maxFilterLengthQs}[x][y]) \quad (1235)$$

- Otherwise, if one or more of the following conditions are true:

- $\text{edgeTbFlags}[x][y - 4]$  is equal to 1,

- $\text{edgeTbFlags}[x][y + 4]$  is equal to 1,

the following applies:

$$\text{maxFilterLengthPs}[x][y] = 1 \quad (1236)$$

$$\text{maxFilterLengthQs}[x][y] = 1 \quad (1237)$$

- Otherwise, if one or more of the following conditions are true:

- $y_{\text{Edge}}$  is equal to 1,

- $y_{\text{Edge}}$  is equal to  $(n_{\text{CbH}} / 8) - 1$ ,

- $\text{edgeTbFlags}[x][y - \text{sbH}]$  is equal to 1,

- $\text{edgeTbFlags}[x][y + \text{sbH}]$  is equal to 1,

the following applies:

$$\text{maxFilterLengthPs}[x][y] = 2 \quad (1238)$$

$$\text{maxFilterLengthQs}[x][y] = 2 \quad (1239)$$

- Otherwise, the following applies:

$$\text{maxFilterLengthPs}[x][y] = 3 \quad (1240)$$

$$\text{maxFilterLengthQs}[x][y] = 3 \quad (1241)$$

### 8.8.3.5 Derivation process of boundary filtering strength

Inputs to this process are:

- a picture sample array  $\text{recPicture}$ ,
- a location  $(x_{\text{Cb}}, y_{\text{Cb}})$  specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,
- a variable  $n_{\text{CbW}}$  specifying the width of the current coding block,
- a variable  $n_{\text{CbH}}$  specifying the height of the current coding block,
- a variable  $\text{edgeType}$  specifying whether a vertical (EDGE\_VER) or a horizontal (EDGE\_HOR) edge is filtered,
- a variable  $\text{cIdx}$  specifying the colour component of the current coding block,
- a two-dimensional  $(n_{\text{CbW}}) \times (n_{\text{CbH}})$  array  $\text{edgeIdc}$ .

Output of this process is a two-dimensional  $(n_{\text{CbW}}) \times (n_{\text{CbH}})$  array  $\text{bS}$  specifying the boundary filtering strength.

The variables gridSize, scaleWidth and scaleHeight are derived as follows:

$$\text{gridSize} = \text{cIdx} == 0 ? 4 : 8 \quad (1242)$$

$$\text{scaleWidth} = ( \text{cIdx} == 0 ) ? 1 : \text{SubWidthC} \quad (1243)$$

$$\text{scaleHeight} = ( \text{cIdx} == 0 ) ? 1 : \text{SubHeightC} \quad (1244)$$

The variables xN and yN are derived as follows:

— If edgeType is equal to EDGE\_VER, the following applies:

$$\text{xN} = \text{Max}( 0, ( \text{nCbW} / \text{gridSize} ) - 1 ) \quad (1245)$$

$$\text{yN} = \text{cIdx} == 0 ? ( \text{nCbH} / 4 ) - 1 : ( \text{nCbH} / 2 ) - 1 \quad (1246)$$

— Otherwise (edgeType is equal to EDGE\_HOR), the following applies:

$$\text{xN} = \text{cIdx} == 0 ? ( \text{nCbW} / 4 ) - 1 : ( \text{nCbW} / 2 ) - 1 \quad (1247)$$

$$\text{yN} = \text{Max}( 0, ( \text{nCbH} / \text{gridSize} ) - 1 ) \quad (1248)$$

The variables xD<sub>i</sub> with i = 0..xN and yD<sub>j</sub> with j = 0..yN are derived as follows:

— If edgeType is equal to EDGE\_VER, the following applies:

$$\text{xD}_i = ( i * \text{gridSize} ) \quad (1249)$$

$$\text{yD}_j = \text{cIdx} == 0 ? ( j << 2 ) : ( j << 1 ) \quad (1250)$$

— Otherwise (edgeType is equal to EDGE\_HOR), the following applies:

$$\text{xD}_i = \text{cIdx} == 0 ? ( i << 2 ) : ( i << 1 ) \quad (1251)$$

$$\text{yD}_j = j * \text{gridSize} \quad (1252)$$

For xD<sub>i</sub> with i = 0..xN and yD<sub>j</sub> with j = 0..yN, the following applies:

— If edgeIdx[ xD<sub>i</sub> ][ yD<sub>j</sub> ] is equal to 0, the variable bs[ xD<sub>i</sub> ][ yD<sub>j</sub> ] is set equal to 0.

— Otherwise, if edgeType is equal to EDGE\_VER, VirtualBoundariesPresentFlag is equal to 1, and ( xCb + xD<sub>i</sub> ) is equal to VirtualBoundaryPosX[ n ] / scaleWidth for any n = 0..NumVerVirtualBoundaries – 1, the variable bs[ xD<sub>i</sub> ][ yD<sub>j</sub> ] is set equal to 0.

— Otherwise, if edgeType is equal to EDGE\_HOR, VirtualBoundariesPresentFlag is equal to 1, and ( yCb + yD<sub>j</sub> ) is equal to VirtualBoundaryPosY[ n ] / scaleHeight for any n = 0..NumHorVirtualBoundaries – 1, the variable bs[ xD<sub>i</sub> ][ yD<sub>j</sub> ] is set equal to 0.

— Otherwise, the following applies:

— The samples p<sub>0</sub> and q<sub>0</sub> are derived as follows:

— If edgeType is equal to EDGE\_VER, p<sub>0</sub> is recPicture[ xCb + xD<sub>i</sub> – 1 ][ yCb + yD<sub>j</sub> ] and q<sub>0</sub> is recPicture[ xCb + xD<sub>i</sub> ][ yCb + yD<sub>j</sub> ].

— Otherwise (edgeType is equal to EDGE\_HOR), p<sub>0</sub> is recPicture[ xCb + xD<sub>i</sub> ][ yCb + yD<sub>j</sub> – 1 ] and q<sub>0</sub> is recPicture[ xCb + xD<sub>i</sub> ][ yCb + yD<sub>j</sub> ].

— The variable bs[ xD<sub>i</sub> ][ yD<sub>j</sub> ] is derived as follows:

- If  $cldx$  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  $cldx$  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  $CuPredMode[cldx == 0 ? 0 : 1][x0][y0]$  is equal to  $MODE\_INTRA$  or  $CuPredMode[cldx == 0 ? 0 : 1][x1][y1]$  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.
  - $cldx$  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  $q_0$ .
  - $cldx$  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 to the top-left sample of the Cb transform block containing chroma sample  $q_0$ .
  - $cldx$  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  $cldx$  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  $CuPredMode[cldx == 0 ? 0 : 1][xp0][yp0]$  of the subblock containing the sample  $p_0$  is different from the  $CuPredMode[cldx == 0 ? 0 : 1][xq0][yq0]$  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 subblocks 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 (  $x_{Sb}$ ,  $y_{Sb}$  ) is equal to  $PredFlagL0[x_{Sb}][y_{Sb}] + PredFlagL1[x_{Sb}][y_{Sb}]$ .

NOTE 3 Reference pictures and motion vectors used to predict a subblock containing a sample located at (  $x_S$ ,  $y_S$  ) refer to values of  $RefPicList[X][RefIdxLX[x_S][y_S]]$  and of  $MvLX[x_S][y_S]$ . These reference pictures and motion vectors can differ from those used for fractional sample interpolation specified in subclause 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  $bS[xD_i][yD_i]$  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 (  $x_{Cb}$ ,  $y_{Cb}$  ) 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:

$$\text{gridSize} = \text{cIdx} == 0 ? 4 : 8 \quad (1253)$$

— The variables `subW`, `subH`, `xN`, `yN` are derived as follows:

$$\text{subW} = \text{cIdx} == 0 ? 1 : \text{SubWidthC} \quad (1254)$$

$$\text{subH} = \text{cIdx} == 0 ? 1 : \text{SubHeightC} \quad (1255)$$

$$\text{xN} = \text{edgeType} == \text{EDGE\_VER} ? \text{Max}(0, (\text{nCbW} / \text{gridSize}) - 1) : (\text{nCbW} / (4 / \text{subW})) - 1 \quad (1256)$$

$$\text{yN} = \text{edgeType} == \text{EDGE\_VER} ? (\text{nCbH} / (4 / \text{subH})) - 1 : \text{Max}(0, (\text{nCbH} / \text{gridSize}) - 1) \quad (1257)$$

— The variables `xDk` with  $k = 0..xN$  and `yDm` with  $m = 0..yN$  are derived as follows:

$$\text{xD}_k = \text{edgeType} == \text{EDGE\_VER} ? (k * \text{gridSize}) : (k << (2 / \text{subW})) \quad (1258)$$

$$\text{yD}_m = \text{edgeType} == \text{EDGE\_VER} ? (m << (2 / \text{subH})) : (m * \text{gridSize}) \quad (1259)$$

— For `xDk` with  $k = 0..xN$  and `yDm` with  $m = 0..yN$ , the following applies:

— When `bS[ xDk ][ yDm ]` 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 subclause 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 ( `xDk`, `yDm` ), the edge direction `edgeType`, the boundary filtering strength `bS[ xDk ][ yDm ]`, the maximum filter lengths `maxFilterLengthP` set equal to `maxFilterLengthPs[ xDk ][ yDm ]` and `maxFilterLengthQ` set equal to `maxFilterLengthQs[ xDk ][ yDm ]` as inputs, and the decisions `dE`, `dEp` and `dEq`, the modified maximum filter lengths `maxFilterLengthP` and `maxFilterLengthQ`, and the variable `tc` as outputs.
2. The filtering process for block edges as specified in subclause 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 ( `xDk`, `yDm` ), the edge direction `edgeType`, the decisions `dE`, `dEp` and `dEq`, the maximum filter lengths `maxFilterLengthP` and `maxFilterLengthQ`, and the variable `tc` 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 subclause 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 ( `xDk`, `yDm` ), the edge direction `edgeType`, the variable `cIdx`, the boundary filtering strength `bS[ xDk ][ yDm ]`, the maximum filter lengths `maxFilterLengthP` set equal to `maxFilterLengthPs[ xDk ][ yDm ]` and the maximum filter lengths `maxFilterLengthQ` set equal to `maxFilterLengthQs[ xDk ][ yDm ]` as inputs, and the modified maximum filter lengths `maxFilterLengthP` and `maxFilterLengthQ`, and the variable `tc` as outputs.

2. When `maxFilterLengthQ` is greater than 0, the filtering process for chroma block edges as specified in subclause 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 ( `xDk, yDm` ), the edge direction `edgeType`, the variable `tc`, 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 `tc`.

The sample values `pi,k` and `qj,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:

$$q_{j,k} = \text{recPicture}[ \text{xCb} + \text{xBl} + j ][ \text{yCb} + \text{yBl} + k ] \quad (1260)$$

$$p_{i,k} = \text{recPicture}[ \text{xCb} + \text{xBl} - i - 1 ][ \text{yCb} + \text{yBl} + k ] \quad (1261)$$

- Otherwise ( `edgeType` is equal to `EDGE_HOR` ), the following applies:

$$q_{j,k} = \text{recPicture}[ \text{xCb} + \text{xBl} + k ][ \text{yCb} + \text{yBl} + j ] \quad (1262)$$

$$p_{i,k} = \text{recPicture}[ \text{xCb} + \text{xBl} + k ][ \text{yCb} + \text{yBl} - i - 1 ] \quad (1263)$$

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:

$$\text{lumaLevel} = ( ( p_{0,0} + p_{0,3} + q_{0,0} + q_{0,3} ) \gg 2 ) \quad (1264)$$

- 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 ]
    else
        break
}

```

(1265)

— 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:

$$qP = ( ( Qp_Q + Qp_P + 1 ) \gg 1 ) + qpOffset \quad (1266)$$

The value of the variable  $\beta'$  is determined as specified in Table 43 based on the quantization parameter  $Q$  derived as follows:

$$Q = \text{Clip3}( 0, 63, qP + ( \text{sh\_luma\_beta\_offset\_div2} \ll 1 ) ) \quad (1267)$$

where  $\text{sh\_luma\_beta\_offset\_div2}$  is the value of the syntax element  $\text{sh\_luma\_beta\_offset\_div2}$  for the slice that contains sample  $q_{0,0}$ .

The variable  $\beta$  is derived as follows:

$$\beta = \beta' * ( 1 \ll ( \text{BitDepth} - 8 ) ) \quad (1268)$$

The value of the variable  $t_c'$  is determined as specified in Table 43 based on the quantization parameter  $Q$  derived as follows:

$$Q = \text{Clip3}( 0, 65, qP + 2 * ( bS - 1 ) + ( \text{sh\_luma\_tc\_offset\_div2} \ll 1 ) ) \quad (1269)$$

where  $\text{sh\_luma\_tc\_offset\_div2}$  is the value of the syntax element  $\text{sh\_luma\_tc\_offset\_div2}$  for the slice that contains sample  $q_{0,0}$ .

The variable  $t_c$  is derived as follows:

— If  $\text{BitDepth}$  is less than 10, the following applies:

$$t_c = ( t_c' + ( 1 \ll ( 9 - \text{BitDepth} ) ) ) \gg ( 10 - \text{BitDepth} ) \quad (1270)$$

— Otherwise ( $\text{BitDepth}$  is greater than or equal to 10), the following applies:

$$t_c = t_c' * ( 1 \ll ( \text{BitDepth} - 10 ) ) \quad (1271)$$

The following ordered steps apply:

1. The variables  $dp_0$ ,  $dp_3$ ,  $dq_0$  and  $dq_3$  are derived as follows:

$$dp_0 = \text{Abs}( p_{2,0} - 2 * p_{1,0} + p_{0,0} ) \quad (1272)$$

$$dp_3 = \text{Abs}( p_{2,3} - 2 * p_{1,3} + p_{0,3} ) \quad (1273)$$

$$dq_0 = \text{Abs}( q_{2,0} - 2 * q_{1,0} + q_{0,0} ) \quad (1274)$$

$$dq_3 = \text{Abs}( q_{2,3} - 2 * q_{1,3} + q_{0,3} ) \quad (1275)$$

2. When  $\text{maxFilterLengthP}$  and  $\text{maxFilterLengthQ}$  both are equal to or greater than 3 the variables  $sp_0$ ,  $sq_0$ ,  $spq_0$ ,  $sp_3$ ,  $sq_3$  and  $spq_3$  are derived as follows:

$$sp_0 = \text{Abs}( p_{3,0} - p_{0,0} ) \quad (1276)$$

$$sq0 = \text{Abs}(q_{0,0} - q_{3,0}) \quad (1277)$$

$$spq0 = \text{Abs}(p_{0,0} - q_{0,0}) \quad (1278)$$

$$sp3 = \text{Abs}(p_{3,3} - p_{0,3}) \quad (1279)$$

$$sq3 = \text{Abs}(q_{0,3} - q_{3,3}) \quad (1280)$$

$$spq3 = \text{Abs}(p_{0,3} - q_{0,3}) \quad (1281)$$

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:

$$dp0L = (dp0 + \text{Abs}(p_{5,0} - 2 * p_{4,0} + p_{3,0}) + 1) >> 1 \quad (1282)$$

$$dp3L = (dp3 + \text{Abs}(p_{5,3} - 2 * p_{4,3} + p_{3,3}) + 1) >> 1 \quad (1283)$$

— Otherwise, the following applies:

$$dp0L = dp0 \quad (1284)$$

$$dp3L = dp3 \quad (1285)$$

$$\text{maxFilterLengthP} = 3 \quad (1286)$$

- b. The variables dq0L and dq3L are derived as follows:

— If sideQisLargeBlk is equal to 1, the following applies:

$$dq0L = (dq0 + \text{Abs}(q_{5,0} - 2 * q_{4,0} + q_{3,0}) + 1) >> 1 \quad (1287)$$

$$dq3L = (dq3 + \text{Abs}(q_{5,3} - 2 * q_{4,3} + q_{3,3}) + 1) >> 1 \quad (1288)$$

— Otherwise, the following applies:

$$dq0L = dq0 \quad (1289)$$

$$dq3L = dq3 \quad (1290)$$

- c. The variables sp0L and sp3L are derived as follows:

— If maxFilterLengthP is equal to 7, the following applies:

$$sp0L = sp0 + \text{Abs}(p_{7,0} - p_{6,0} - p_{5,0} + p_{4,0}) \quad (1291)$$

$$sp3L = sp3 + \text{Abs}(p_{7,3} - p_{6,3} - p_{5,3} + p_{4,3}) \quad (1292)$$

— Otherwise, the following applies:

$$sp0L = sp0 \quad (1293)$$

$$sp3L = sp3 \quad (1294)$$

d. The variables sq0L and sq3L are derived as follows:

— If maxFilterLengthQ is equal to 7, the following applies:

$$sq0L = sq0 + \text{Abs}(q_{4,0} - q_{5,0} - q_{6,0} + q_{7,0}) \quad (1295)$$

$$sq3L = sq3 + \text{Abs}(q_{4,3} - q_{5,3} - q_{6,3} + q_{7,3}) \quad (1296)$$

— Otherwise, the following applies:

$$sq0L = sq0 \quad (1297)$$

$$sq3L = sq3 \quad (1298)$$

e. The variables dpq0L, dpq3L, and dL are derived as follows:

$$dpq0L = dp0L + dq0L \quad (1299)$$

$$dpq3L = dp3L + dq3L \quad (1300)$$

$$dL = dpq0L + dpq3L \quad (1301)$$

f. When dL is less than  $\beta$ , 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:

$$p_3 = p_{3,0} \quad (1302)$$

$$p_0 = p_{\text{maxFilterLengthP},0} \quad (1303)$$

— When sideQisLargeBlk is equal to 1, the following applies:

$$q_3 = q_{3,0} \quad (1304)$$

$$q_0 = q_{\text{maxFilterLengthQ},0} \quad (1305)$$

- iv. For the sample location (  $x_{Cb} + x_{Bl}$ ,  $y_{Cb} + y_{Bl}$  ), the decision process for a luma sample as specified in subclause 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,  $\beta$  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:

$$p_3 = p_{3,3} \quad (1306)$$

$$p_0 = p_{\maxFilterLengthP,3} \quad (1307)$$

— When `sideQisLargeBlk` is equal to 1, the following applies:

$$q_3 = q_{3,3} \quad (1308)$$

$$q_0 = q_{\maxFilterLengthQ,3} \quad (1309)$$

- viii. When `edgeType` is equal to `EDGE_VER` for the sample location  $(x_{Cb} + x_{Bl}, y_{Cb} + y_{Bl} + 3)$  or when `edgeType` is equal to `EDGE_HOR` for the sample location  $(x_{Cb} + x_{Bl} + 3, y_{Cb} + y_{Bl})$ , the decision process for a luma sample as specified in subclause 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`,  $\beta$  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  $dpq_0$ ,  $dpq_3$ ,  $dp$ ,  $dq$  and  $d$  are derived as follows:

$$dpq_0 = dp_0 + dq_0 \quad (1310)$$

$$dpq_3 = dp_3 + dq_3 \quad (1311)$$

$$dp = dp_0 + dp_3 \quad (1312)$$

$$dq = dq_0 + dq_3 \quad (1313)$$

$$d = dpq_0 + dpq_3 \quad (1314)$$

- b. The variables `dE`, `dEp`, `dEq`, `sidePisLargeBlk` and `sideQisLargeBlk` are set equal to 0.

- c. When  $d$  is less than  $\beta$  and both `maxFilterLengthP` and `maxFilterLengthQ` are greater than 2, the following ordered steps apply:

- i. The variable  $dpq$  is set equal to  $2 * dpq_0$ .
- ii. The variable  $sp$  is set equal to  $sp_0$ , the variable  $sq$  is set equal to  $sq_0$  and the variable  $spq$  is set equal to  $spq_0$ .
- iii. For the sample location  $(x_{Cb} + x_{Bl}, y_{Cb} + y_{Bl})$ , the decision process for a luma sample as specified in subclause 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`,  $\beta$  and  $t_c$  as inputs, and the output is assigned to the decision `dSam0`.
- iv. The variable  $dpq$  is set equal to  $2 * dpq_3$ .
- v. The variable  $sp$  is set equal to  $sp_3$ , the variable  $sq$  is set equal to  $sq_3$  and the variable  $spq$  is set equal to  $spq_3$ .

- vi. When `edgeType` is equal to `EDGE_VER` for the sample location  $(x_{Cb} + x_{Bl}, y_{Cb} + y_{Bl} + 3)$  or when `edgeType` is equal to `EDGE_HOR` for the sample location  $(x_{Cb} + x_{Bl} + 3, y_{Cb} + y_{Bl})$ , the decision process for a sample as specified in subclause 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  $dpq, sp, sq, spq, sidePisLargeBlk, sideQisLargeBlk, \beta$  and  $t_c$  as inputs, and the output is assigned to the decision  $d_{Sam3}$ .
- d. When  $d$  is less than  $\beta$ , the following ordered steps apply:
- The variable  $dE$  is set equal to 1.
  - When  $d_{Sam0}$  is equal to 1 and  $d_{Sam3}$  is equal to 1, the variable  $dE$  is set equal to 2 and both  $maxFilterLengthP$  and  $maxFilterLengthQ$  are set equal to 3.
  - When  $maxFilterLengthP$  is greater than 1, and  $maxFilterLengthQ$  is greater than 1, and  $dp$  is less than  $(\beta + (\beta \gg 1)) \gg 3$ , the variable  $dEp$  is set equal to 1.
  - When  $maxFilterLengthP$  is greater than 1, and  $maxFilterLengthQ$  is greater than 1, and  $dq$  is less than  $(\beta + (\beta \gg 1)) \gg 3$ , the variable  $dEq$  is set equal to 1.
  - 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  $\beta'$  and  $t_c'$  from input  $Q$ 

|                            |    |    |     |     |     |     |     |     |     |     |     |     |     |     |     |    |    |
|----------------------------|----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|
| <b>Q</b>                   | 0  | 1  | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15 | 16 |
| <b><math>\beta'</math></b> | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0  | 6  |
| <b><math>t_c'</math></b>   | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0  | 0  |
| <b>Q</b>                   | 17 | 18 | 19  | 20  | 21  | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29  | 30  | 31  | 32 | 33 |
| <b><math>\beta'</math></b> | 7  | 8  | 9   | 10  | 11  | 12  | 13  | 14  | 15  | 16  | 17  | 18  | 20  | 22  | 24  | 26 | 28 |
| <b><math>t_c'</math></b>   | 0  | 3  | 4   | 4   | 4   | 4   | 5   | 5   | 5   | 5   | 7   | 7   | 8   | 9   | 10  | 10 | 11 |
| <b>Q</b>                   | 34 | 35 | 36  | 37  | 38  | 39  | 40  | 41  | 42  | 43  | 44  | 45  | 46  | 47  | 48  | 49 | 50 |
| <b><math>\beta'</math></b> | 30 | 32 | 34  | 36  | 38  | 40  | 42  | 44  | 46  | 48  | 50  | 52  | 54  | 56  | 58  | 60 | 62 |
| <b><math>t_c'</math></b>   | 13 | 14 | 15  | 17  | 19  | 21  | 24  | 25  | 29  | 33  | 36  | 41  | 45  | 51  | 57  | 64 | 71 |
| <b>Q</b>                   | 51 | 52 | 53  | 54  | 55  | 56  | 57  | 58  | 59  | 60  | 61  | 62  | 63  | 64  | 65  |    |    |
| <b><math>\beta'</math></b> | 64 | 66 | 68  | 70  | 72  | 74  | 76  | 78  | 80  | 82  | 84  | 86  | 88  | -   | -   |    |    |
| <b><math>t_c'</math></b>   | 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  $(x_{Cb}, y_{Cb})$  specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,
- a location  $(x_{Bl}, y_{Bl})$  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:

$$q_{j,k} = recPicture[xCb + xBl + j][yCb + yBl + k] \quad (1315)$$

$$p_{i,k} = recPicture[xCb + xBl - i - 1][yCb + yBl + k] \quad (1316)$$

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 subclause 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:

$$recPicture[xCb + xBl - i - 1][yCb + yBl + k] = p_i' \quad (1317)$$

- 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:

$$recPicture[xCb + xBl + j][yCb + yBl + k] = q_j' \quad (1318)$$

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 subclause 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:

$$recPicture[xCb + xBl - i - 1][yCb + yBl + k] = p_i' \quad (1319)$$

- c. The filtered sample values  $q_j'$  with  $j = 0..maxFilterLengthQ - 1$  replace the corresponding samples inside the sample array `recPicture` as follows:

$$recPicture[xCb + xBl + j][yCb + yBl + k] = q_j' \quad (1320)$$

— 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:

$$q_{j,k} = recPicture[xCb + xBl + k][yCb + yBl + j] \quad (1321)$$

$$p_{i,k} = recPicture[xCb + xBl + k][yCb + yBl - i - 1] \quad (1322)$$

2. When  $dE$  is not equal to 0 and  $dE$  is not equal to 3, for each sample location  $(x_{Cb} + x_{Bl} + k, y_{Cb} + y_{Bl})$ ,  $k = 0..3$ , the following ordered steps apply:

- a. The filtering process for a luma sample using short filters as specified in subclause 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:

$$recPicture[x_{Cb} + x_{Bl} + k][y_{Cb} + y_{Bl} - i - 1] = p_i' \quad (1323)$$

- 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:

$$recPicture[x_{Cb} + x_{Bl} + k][y_{Cb} + y_{Bl} + j] = q_j' \quad (1324)$$

3. When  $dE$  is equal to 3, for each sample location  $(x_{Cb} + x_{Bl} + k, y_{Cb} + y_{Bl})$ ,  $k = 0..3$ , the following ordered steps apply:

- a. The filtering process for a luma sample using long filters as specified in subclause 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:

$$recPicture[x_{Cb} + x_{Bl} + k][y_{Cb} + y_{Bl} - i - 1] = p_i' \quad (1325)$$

- c. The filtered sample values  $q_j'$  with  $j = 0..maxFilterLengthQ - 1$  replace the corresponding samples inside the sample array  $recPicture$  as follows:

$$recPicture[x_{Cb} + x_{Bl} + k][y_{Cb} + y_{Bl} + j] = q_j' \quad (1326)$$

#### 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  $(x_{Cb}, y_{Cb})$  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  $(x_{Bl}, y_{Bl})$  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  $\text{maxFilterLengthP}$  and  $\text{maxFilterLengthQ}$ ,
- the variable  $t_c$ .

The variable  $\text{maxK}$  is derived as follows:

- If  $\text{edgeType}$  is equal to  $\text{EDGE\_VER}$ , the following applies:

$$\text{maxK} = ( \text{SubHeightC} == 1 ) ? 3 : 1 \quad (1327)$$

- Otherwise ( $\text{edgeType}$  is equal to  $\text{EDGE\_HOR}$ ), the following applies:

$$\text{maxK} = ( \text{SubWidthC} == 1 ) ? 3 : 1 \quad (1328)$$

The values  $p_{i,k}$  and  $q_{j,k}$  with  $i = 0..\text{maxFilterLengthP}$ ,  $j = 0..\text{maxFilterLengthQ}$  and  $k = 0..\text{maxK}$  are derived as follows:

- If  $\text{edgeType}$  is equal to  $\text{EDGE\_VER}$ , the following applies:

$$q_{j,k} = \text{recPicture}[ \text{xCb} + \text{xBl} + j ][ \text{yCb} + \text{yBl} + k ] \quad (1329)$$

$$p_{i,k} = \text{recPicture}[ \text{xCb} + \text{xBl} - i - 1 ][ \text{yCb} + \text{yBl} + k ] \quad (1330)$$

$$\text{subSampleC} = \text{SubHeightC} \quad (1331)$$

- Otherwise ( $\text{edgeType}$  is equal to  $\text{EDGE\_HOR}$ ), the following applies:

$$q_{j,k} = \text{recPicture}[ \text{xCb} + \text{xBl} + k ][ \text{yCb} + \text{yBl} + j ] \quad (1332)$$

$$p_{i,k} = \text{recPicture}[ \text{xCb} + \text{xBl} + k ][ \text{yCb} + \text{yBl} - i - 1 ] \quad (1333)$$

$$\text{subSampleC} = \text{SubWidthC} \quad (1334)$$

The variable  $Q_{pP}$  is derived as follows:

- The luma location  $( \text{xTb}_P, \text{yTb}_P )$  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  $\text{TuCResMode}[ \text{xTb}_P ][ \text{yTb}_P ]$  is equal to 2,  $Q_{pP}$  is set equal to  $Q_{p'_{CbCr}}$  of the transform block containing the sample  $p_{0,0}$ .
- Otherwise, if  $\text{cIdx}$  is equal to 1,  $Q_{pP}$  is set equal to  $Q_{p'_{Cb}}$  of the transform block containing the sample  $p_{0,0}$ .
- Otherwise,  $Q_{pP}$  is set equal to  $Q_{p'_{Cr}}$  of the transform block containing the sample  $p_{0,0}$ .

The variable  $Q_{pQ}$  is derived as follows:

- The luma location  $( \text{xTb}_Q, \text{yTb}_Q )$  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  $\text{TuCResMode}[ \text{xTb}_Q ][ \text{yTb}_Q ]$  is equal to 2,  $Q_{pQ}$  is set equal to  $Q_{p'_{CbCr}}$  of the transform block containing the sample  $q_{0,0}$ .
- Otherwise, if  $\text{cIdx}$  is equal to 1,  $Q_{pQ}$  is set equal to  $Q_{p'_{Cb}}$  of the transform block containing the sample  $q_{0,0}$ .
- Otherwise,  $Q_{pQ}$  is set equal to  $Q_{p'_{Cr}}$  of the transform block containing the sample  $q_{0,0}$ .

The variable  $Q_{pC}$  is derived as follows:

$$Q_{pC} = ( Q_{pQ} - Q_{pBdOffset} + Q_{pP} - Q_{pBdOffset} + 1 ) \gg 1 \quad (1335)$$

The value of the variable  $\beta'$  is determined as specified in Table 43 based on the quantization parameter  $Q$  derived as follows:

$$\begin{aligned} \text{sliceBetaOffsetDiv2} &= ( \text{cIdx} == 1 ? \text{sh\_cb\_beta\_offset\_div2} : \text{sh\_cr\_beta\_offset\_div2} ) \\ Q &= \text{Clip3}( 0, 63, Q_{pC} + ( \text{sliceBetaOffsetDiv2} \ll 1 ) ) \end{aligned} \quad (1336)$$

where  $\text{sh\_cb\_beta\_offset\_div2}$  and  $\text{sh\_cr\_beta\_offset\_div2}$  are the values of the syntax elements  $\text{sh\_cb\_beta\_offset\_div2}$  and  $\text{sh\_cr\_beta\_offset\_div2}$ , respectively, for the slice that contains sample  $q_{0,0}$ .

The variable  $\beta$  is derived as follows:

$$\beta = \beta' * ( 1 \ll ( \text{BitDepth} - 8 ) ) \quad (1337)$$

The value of the variable  $t_c'$  is determined as specified in Table 43 based on the chroma quantization parameter  $Q$  derived as follows:

$$\begin{aligned} \text{sliceTcOffsetDiv2} &= ( \text{cIdx} == 1 ? \text{sh\_cb\_tc\_offset\_div2} : \text{sh\_cr\_tc\_offset\_div2} ) \\ Q &= \text{Clip3}( 0, 65, Q_{pC} + 2 * ( \text{bS} - 1 ) + ( \text{sliceTcOffsetDiv2} \ll 1 ) ) \end{aligned} \quad (1338)$$

where  $\text{sh\_cb\_tc\_offset\_div2}$  and  $\text{sh\_cr\_tc\_offset\_div2}$  are the values of the syntax elements  $\text{sh\_cb\_tc\_offset\_div2}$  and  $\text{sh\_cr\_tc\_offset\_div2}$ , respectively, for the slice that contains sample  $q_{0,0}$ .

The variable  $t_c$  is derived as follows:

— If  $\text{BitDepth}$  is less than 10, the following applies:

$$t_c = ( t_c' + ( 1 \ll ( 9 - \text{BitDepth} ) ) ) \gg ( 10 - \text{BitDepth} ) \quad (1339)$$

— Otherwise ( $\text{BitDepth}$  is greater than or equal to 10), the following applies:

$$t_c = t_c' * ( 1 \ll ( \text{BitDepth} - 10 ) ) \quad (1340)$$

When both  $\text{maxFilterLengthP}$  and  $\text{maxFilterLengthQ}$  are equal to 1 and  $\text{bS}$  is not equal to 2,  $\text{maxFilterLengthP}$  and  $\text{maxFilterLengthQ}$  are both set equal to 0.

When  $\text{maxFilterLengthQ}$  is equal to 3, the following ordered steps apply:

1. The variable  $n1$  is derived as follows:

$$n1 = \text{subSampleC} == 2 ? 1 : 3 \quad (1341)$$

2. When  $\text{maxFilterLengthP}$  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  $\text{dpq0}$ ,  $\text{dpq1}$ ,  $\text{dp}$ ,  $\text{dq}$  and  $d$  are derived as follows:

$$\text{dp0} = \text{Abs}( p_{2,0} - 2 * p_{1,0} + p_{0,0} ) \quad (1342)$$

$$\text{dp1} = \text{Abs}( p_{2,n1} - 2 * p_{1,n1} + p_{0,n1} ) \quad (1343)$$

$$\text{dq0} = \text{Abs}( q_{2,0} - 2 * q_{1,0} + q_{0,0} ) \quad (1344)$$

$$\text{dq1} = \text{Abs}( q_{2,n1} - 2 * q_{1,n1} + q_{0,n1} ) \quad (1345)$$

$$\text{dpq0} = \text{dp0} + \text{dq0} \quad (1346)$$

$$\text{dpq1} = \text{dp1} + \text{dq1} \quad (1347)$$



$$dp = dp0 + dp1 \quad (1348)$$

$$dq = dq0 + dq1 \quad (1349)$$

$$d = dpq0 + dpq1 \quad (1350)$$

4. The variables dSam0 and dSam1 are both set equal to 0.
5. When d is less than  $\beta$ , 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 subclause 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,  $\beta$  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 subclause 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,  $\beta$  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 subclause 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,  $\beta$  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  $t_c$ .

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:

$$\text{maxK} = ( \text{SubHeightC} == 1 ) ? 3 : 1 \quad (1351)$$

— Otherwise (edgeType is equal to EDGE\_HOR), the following applies:

$$\text{maxK} = (\text{SubWidthC} == 1) ? 3 : 1 \quad (1352)$$

The values  $p_{i,k}$  with  $i = 0..\text{maxFilterLengthP}$ ,  $q_{j,k}$  with  $j = 0..\text{maxFilterLengthQ}$ , and  $k = 0..\text{maxK}$  are derived as follows:

— If edgeType is equal to EDGE\_VER, the following applies:

$$q_{j,k} = \text{recPicture}[\text{xCb} + \text{xBl} + j][\text{yCb} + \text{yBl} + k] \quad (1353)$$

$$p_{i,k} = \text{recPicture}[\text{xCb} + \text{xBl} - i - 1][\text{yCb} + \text{yBl} + k] \quad (1354)$$

— Otherwise (edgeType is equal to EDGE\_HOR), the following applies:

$$q_{j,k} = \text{recPicture}[\text{xCb} + \text{xBl} + k][\text{yCb} + \text{yBl} + j] \quad (1355)$$

$$p_{i,k} = \text{recPicture}[\text{xCb} + \text{xBl} + k][\text{yCb} + \text{yBl} - i - 1] \quad (1356)$$

Depending on the value of edgeType, the following applies:

— If edgeType is equal to EDGE\_VER, for each sample location  $(\text{xCb} + \text{xBl}, \text{yCb} + \text{yBl} + k)$ ,  $k = 0..\text{maxK}$ , the following ordered steps apply:

1. The filtering process for a chroma sample as specified in subclause 8.8.3.6.10 is invoked with the variables  $\text{maxFilterLengthP}$  and  $\text{maxFilterLengthQ}$ , the sample values  $p_{i,k}$ ,  $q_{j,k}$  with  $i = 0..\text{maxFilterLengthP}$  and  $j = 0..\text{maxFilterLengthQ}$ , and the variable  $t_c$  as inputs, and the filtered sample values  $p'_i$  and  $q'_j$  with  $i = 0..\text{maxFilterLengthP} - 1$  and  $j = 0..\text{maxFilterLengthQ} - 1$  as outputs.
2. The filtered sample values  $p'_i$  and  $q'_j$  with  $i = 0..\text{maxFilterLengthP} - 1$  and  $j = 0..\text{maxFilterLengthQ} - 1$  replace the corresponding samples inside the sample array  $\text{recPicture}$  as follows:

$$\text{recPicture}[\text{xCb} + \text{xBl} + j][\text{yCb} + \text{yBl} + k] = q'_j \quad (1357)$$

$$\text{recPicture}[\text{xCb} + \text{xBl} - i - 1][\text{yCb} + \text{yBl} + k] = p'_i \quad (1358)$$

— Otherwise (edgeType is equal to EDGE\_HOR), for each sample location  $(\text{xCb} + \text{xBl} + k, \text{yCb} + \text{yBl})$ ,  $k = 0..\text{maxK}$ , the following ordered steps apply:

1. The filtering process for a chroma sample as specified in subclause 8.8.3.6.10 is invoked with the variables  $\text{maxFilterLengthP}$  and  $\text{maxFilterLengthQ}$ , the sample values  $p_{i,k}$ ,  $q_{j,k}$  with  $i = 0..\text{maxFilterLengthP}$  and  $j = 0..\text{maxFilterLengthQ}$ , and the variable  $t_c$  as inputs, and the filtered sample values  $p'_i$  and  $q'_j$  with  $i = 0..\text{maxFilterLengthP} - 1$  and  $j = 0..\text{maxFilterLengthQ} - 1$  as outputs.
2. The filtered sample values  $p'_i$  and  $q'_j$  with  $i = 0..\text{maxFilterLengthP} - 1$  and  $j = 0..\text{maxFilterLengthQ} - 1$  replace the corresponding samples inside the sample array  $\text{recPicture}$  as follows:

$$\text{recPicture}[\text{xCb} + \text{xBl} + k][\text{yCb} + \text{yBl} + j] = q'_j \quad (1359)$$

$$\text{recPicture}[\text{xCb} + \text{xBl} + k][\text{yCb} + \text{yBl} - i - 1] = p'_i \quad (1360)$$

#### 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  $dpq$ ,  $sp$ ,  $sq$ ,  $spq$ ,  $sidePisLargeBlk$ ,  $sideQisLargeBlk$ ,  $\beta$  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:

$$sp = (sp + \text{Abs}(p_3 - p_0) + 1) \gg 1 \quad (1361)$$

— When  $sideQisLargeBlk$  is equal to 1, the following applies:

$$sq = (sq + \text{Abs}(q_3 - q_0) + 1) \gg 1 \quad (1362)$$

The variables  $sThr1$  and  $sThr2$  are derived as follows:

— If  $sidePisLargeBlk$  is equal to 1 or  $sideQisLargeBlk$  is equal to 1, the following applies:

$$sThr1 = 3 * \beta \gg 5 \quad (1363)$$

$$sThr2 = \beta \gg 4 \quad (1364)$$

— Otherwise, the following applies:

$$sThr1 = \beta \gg 3 \quad (1365)$$

$$sThr2 = \beta \gg 2 \quad (1366)$$

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  $(5 * t_c + 1) \gg 1$ .

— 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:

$$p_0' = \text{Clip3}(p_0 - 3 * t_c, p_0 + 3 * t_c, (p_2 + 2 * p_1 + 2 * p_0 + 2 * q_0 + q_1 + 4) >> 3) \quad (1367)$$

$$p_1' = \text{Clip3}(p_1 - 2 * t_c, p_1 + 2 * t_c, (p_2 + p_1 + p_0 + q_0 + 2) >> 2) \quad (1368)$$

$$p_2' = \text{Clip3}(p_2 - 1 * t_c, p_2 + 1 * t_c, (2 * p_3 + 3 * p_2 + p_1 + p_0 + q_0 + 4) >> 3) \quad (1369)$$

$$q_0' = \text{Clip3}(q_0 - 3 * t_c, q_0 + 3 * t_c, (p_1 + 2 * p_0 + 2 * q_0 + 2 * q_1 + q_2 + 4) >> 3) \quad (1370)$$

$$q_1' = \text{Clip3}(q_1 - 2 * t_c, q_1 + 2 * t_c, (p_0 + q_0 + q_1 + q_2 + 2) >> 2) \quad (1371)$$

$$q_2' = \text{Clip3}(q_2 - 1 * t_c, q_2 + 1 * t_c, (p_0 + q_0 + q_1 + 3 * q_2 + 2 * q_3 + 4) >> 3) \quad (1372)$$

— Otherwise, nDp and nDq are set both equal to 0 and the following weak filtering applies:

— The following applies:

$$\Delta = (9 * (q_0 - p_0) - 3 * (q_1 - p_1) + 8) >> 4 \quad (1373)$$

— When Abs( $\Delta$ ) is less than  $t_c * 10$ , the following ordered steps apply:

— The filtered sample values  $p_0'$  and  $q_0'$  are specified as follows:

$$\Delta = \text{Clip3}(-t_c, t_c, \Delta) \quad (1374)$$

$$p_0' = \text{Clip1}(p_0 + \Delta) \quad (1375)$$

$$q_0' = \text{Clip1}(q_0 - \Delta) \quad (1376)$$

— When dEp is equal to 1, the filtered sample value  $p_1'$  is specified as follows:

$$\Delta p = \text{Clip3}(-(t_c >> 1), t_c >> 1, ((p_2 + p_0 + 1) >> 1) - p_1 + \Delta) >> 1) \quad (1377)$$

$$p_1' = \text{Clip1}(p_1 + \Delta p) \quad (1378)$$

— When dEq is equal to 1, the filtered sample value  $q_1'$  is specified as follows:

$$\Delta q = \text{Clip3}(-(t_c >> 1), t_c >> 1, ((q_2 + q_0 + 1) >> 1) - q_1 - \Delta) >> 1) \quad (1379)$$

$$q_1' = \text{Clip1}(q_1 + \Delta q) \quad (1380)$$

— 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  $\text{refMiddle}$  is derived as follows:

- If  $\text{maxFilterLengthP}$  is equal to  $\text{maxFilterLengthQ}$  and  $\text{maxFilterLengthP}$  is equal to 5, the following applies:

$$\text{refMiddle} = (p_4 + p_3 + 2 * (p_2 + p_1 + p_0 + q_0 + q_1 + q_2) + q_3 + q_4 + 8) \gg 4 \quad (1381)$$

- Otherwise, if  $\text{maxFilterLengthP}$  is equal to  $\text{maxFilterLengthQ}$  and  $\text{maxFilterLengthP}$  is not equal to 5, the following applies:

$$\text{refMiddle} = (p_6 + p_5 + p_4 + p_3 + p_2 + p_1 + 2 * (p_0 + q_0) + q_1 + q_2 + q_3 + q_4 + q_5 + q_6 + 8) \gg 4 \quad (1382)$$

- Otherwise, if one of the following conditions is true:

- $\text{maxFilterLengthQ}$  is equal to 7 and  $\text{maxFilterLengthP}$  is equal to 5,
  - $\text{maxFilterLengthQ}$  is equal to 5 and  $\text{maxFilterLengthP}$  is equal to 7,
- the following applies:

$$\text{refMiddle} = (p_5 + p_4 + p_3 + p_2 + 2 * (p_1 + p_0 + q_0 + q_1) + q_2 + q_3 + q_4 + q_5 + 8) \gg 4 \quad (1383)$$

- Otherwise, if one of the following conditions is true:

- $\text{maxFilterLengthQ}$  is equal to 5 and  $\text{maxFilterLengthP}$  is equal to 3,
  - $\text{maxFilterLengthQ}$  is equal to 3 and  $\text{maxFilterLengthP}$  is equal to 5,
- the following applies:

$$\text{refMiddle} = (p_3 + p_2 + p_1 + p_0 + q_0 + q_1 + q_2 + q_3 + 4) \gg 3 \quad (1384)$$

- Otherwise, if  $\text{maxFilterLengthQ}$  is equal to 7 and  $\text{maxFilterLengthP}$  is equal to 3, the following applies:

$$\text{refMiddle} = (2 * (p_2 + p_1 + p_0 + q_0) + p_0 + p_1 + q_1 + q_2 + q_3 + q_4 + q_5 + q_6 + 8) \gg 4 \quad (1385)$$

- Otherwise, the following applies:

$$\text{refMiddle} = (p_6 + p_5 + p_4 + p_3 + p_2 + p_1 + 2 * (q_2 + q_1 + q_0 + p_0) + q_0 + q_1 + 8) \gg 4 \quad (1386)$$

The variables  $\text{refP}$  and  $\text{refQ}$  are derived as follows:

$$\text{refP} = (p_{\text{maxFilterLengthP}} + p_{\text{maxFilterLengthP}-1} + 1) \gg 1 \quad (1387)$$

$$\text{refQ} = (q_{\text{maxFilterLengthQ}} + q_{\text{maxFilterLengthQ}-1} + 1) \gg 1 \quad (1388)$$

The variables  $f_i$  and  $t_{cPD_i}$  are defined as follows:

- If  $\text{maxFilterLengthP}$  is equal to 7, the following applies:

$$f_{0..6} = \{ 59, 50, 41, 32, 23, 14, 5 \} \quad (1389)$$

$$t_{cPD_{0..6}} = \{ 6, 5, 4, 3, 2, 1, 1 \} \quad (1390)$$

- Otherwise, if  $\text{maxFilterLengthP}$  is equal to 5, the following applies:

$$f_{0..4} = \{ 58, 45, 32, 19, 6 \} \quad (1391)$$

$$t_{cPD_{0..4}} = \{ 6, 5, 4, 3, 2 \} \quad (1392)$$

— Otherwise, the following applies:

$$f_{0..2} = \{ 53, 32, 11 \} \quad (1393)$$

$$t_{cPD_{0..2}} = \{ 6, 4, 2 \} \quad (1394)$$

The variables  $g_j$  and  $t_{cQD_j}$  are defined as follows:

— If  $\text{maxFilterLengthQ}$  is equal to 7, the following applies:

$$g_{0..6} = \{ 59, 50, 41, 32, 23, 14, 5 \} \quad (1395)$$

$$t_{cQD_{0..6}} = \{ 6, 5, 4, 3, 2, 1, 1 \} \quad (1396)$$

— Otherwise, if  $\text{maxFilterLengthQ}$  is equal to 5, the following applies:

$$g_{0..4} = \{ 58, 45, 32, 19, 6 \} \quad (1397)$$

$$t_{cQD_{0..4}} = \{ 6, 5, 4, 3, 2 \} \quad (1398)$$

— Otherwise, the following applies:

$$g_{0..2} = \{ 53, 32, 11 \} \quad (1399)$$

$$t_{cQD_{0..2}} = \{ 6, 4, 2 \} \quad (1400)$$

The filtered sample values  $p_i'$  and  $q_j'$  with  $i = 0..\text{maxFilterLengthP} - 1$  and  $j = 0..\text{maxFilterLengthQ} - 1$  are derived as follows:

$$p_i' = \text{Clip3}(p_i - (t_c * t_{cPD_i} >> 1), p_i + (t_c * t_{cPD_i} >> 1), (\text{refMiddle} * f_i + \text{refP} * (64 - f_i) + 32) >> 6) \quad (1401)$$

$$q_j' = \text{Clip3}(q_j - (t_c * t_{cQD_j} >> 1), q_j + (t_c * t_{cQD_j} >> 1), (\text{refMiddle} * g_j + \text{refQ} * (64 - g_j) + 32) >> 6) \quad (1402)$$

When  $\text{pred\_mode\_plt\_flag}$  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  $i = 0..\text{maxFilterLengthP} - 1$ .

When  $\text{pred\_mode\_plt\_flag}$  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  $j = 0..\text{maxFilterLengthQ} - 1$ .

#### 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  $\text{dpq}$ ,  $\beta$  and  $t_c$ .

Output of this process is the variable  $\text{dSam}$  containing a decision.

The variable  $\text{dSam}$  is specified as follows:

- If all of the following conditions are true,  $\text{dSam}$  is set equal to 1:
  - $\text{dpq}$  is less than  $(\beta >> 2)$ ,
  - $\text{Abs}(p_3 - p_0) + \text{Abs}(q_0 - q_3)$  is less than  $(\beta >> 3)$ ,
  - $\text{Abs}(p_0 - q_0)$  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:

$$p'_0 = \text{Clip3}(p_0 - t_c, p_0 + t_c, (p_3 + p_2 + p_1 + 2 * p_0 + q_0 + q_1 + q_2 + 4) >> 3) \quad (1403)$$

$$p'_1 = \text{Clip3}(p_1 - t_c, p_1 + t_c, (2 * p_3 + p_2 + 2 * p_1 + p_0 + q_0 + q_1 + 4) >> 3) \quad (1404)$$

$$p'_2 = \text{Clip3}(p_2 - t_c, p_2 + t_c, (3 * p_3 + 2 * p_2 + p_1 + p_0 + q_0 + 4) >> 3) \quad (1405)$$

$$q'_0 = \text{Clip3}(q_0 - t_c, q_0 + t_c, (p_2 + p_1 + p_0 + 2 * q_0 + q_1 + q_2 + q_3 + 4) >> 3) \quad (1406)$$

$$q'_1 = \text{Clip3}(q_1 - t_c, q_1 + t_c, (p_1 + p_0 + q_0 + 2 * q_1 + q_2 + 2 * q_3 + 4) >> 3) \quad (1407)$$

$$q'_2 = \text{Clip3}(q_2 - t_c, q_2 + t_c, (p_0 + q_0 + q_1 + 2 * q_2 + 3 * q_3 + 4) >> 3) \quad (1408)$$

- Otherwise, if the variable maxFilterLengthP is equal to 1 and maxFilterLengthQ is equal to 3, the following filtering applies:

$$p'_0 = \text{Clip3}(p_0 - t_c, p_0 + t_c, (3 * p_1 + 2 * p_0 + q_0 + q_1 + q_2 + 4) >> 3) \quad (1409)$$

$$q'_0 = \text{Clip3}(q_0 - t_c, q_0 + t_c, (2 * p_1 + p_0 + 2 * q_0 + q_1 + q_2 + q_3 + 4) >> 3) \quad (1410)$$

$$q'_1 = \text{Clip3}(q_1 - t_c, q_1 + t_c, (p_1 + p_0 + q_0 + 2 * q_1 + q_2 + 2 * q_3 + 4) >> 3) \quad (1411)$$

$$q'_2 = \text{Clip3}(q_2 - t_c, q_2 + t_c, (p_0 + q_0 + q_1 + 2 * q_2 + 3 * q_3 + 4) >> 3) \quad (1412)$$

- Otherwise, the following weak filtering applies:

$$\Delta = \text{Clip3}(-t_c, t_c, ((q_0 - p_0) << 2) + p_1 - q_1 + 4) >> 3) \quad (1413)$$

$$p'_0 = \text{Clip1}(p_0 + \Delta) \quad (1414)$$

$$q'_0 = \text{Clip1}(q_0 - \Delta) \quad (1415)$$

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 `recPictureL` and, when `sps_chroma_format_idc` is not equal to 0, the arrays `recPictureCb` and `recPictureCr`.

Outputs of this process are the modified reconstructed picture sample array after sample adaptive offset `saoPictureL` and, when `sps_chroma_format_idc` is not equal to 0, the arrays `saoPictureCb` and `saoPictureCr`.

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 `saoPictureL` and, when `sps_chroma_format_idc` is not equal to 0, the arrays `saoPictureCb` and `saoPictureCr` are initially set equal to the sample values in the reconstructed picture sample array `recPictureL` and, when `sps_chroma_format_idc` is not equal to 0, the arrays `recPictureCb` and `recPictureCr`, 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 subclause 8.8.4.2 is invoked with `recPicture` set equal to `recPictureL`, `cldx` set equal to 0,  $(rx, ry)$ , and both `nCtbSw` and `nCtbSh` set equal to `CtbSizeY` as inputs, and the modified luma picture sample array `saoPictureL` 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 subclause 8.8.4.2 is invoked with `recPicture` set equal to `recPictureCb`, `cldx` set equal to 1,  $(rx, ry)$ , `nCtbSw` set equal to  $(1 \ll CtbLog2SizeY) / SubWidthC$  and `nCtbSh` set equal to  $(1 \ll CtbLog2SizeY) / SubHeightC$  as inputs, and the modified chroma picture sample array `saoPictureCb` 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 subclause 8.8.4.2 is invoked with `recPicture` set equal to `recPictureCr`, `cldx` set equal to 2,  $(rx, ry)$ , `nCtbSw` set equal to  $(1 \ll CtbLog2SizeY) / SubWidthC$  and `nCtbSh` set equal to  $(1 \ll CtbLog2SizeY) / SubHeightC$  as inputs, and the modified chroma picture sample array `saoPictureCr` as output.

### 8.8.4.2 CTB modification process

Inputs to this process are:

- the picture sample array `recPicture` for the colour component `cldx`,
- a variable `cldx` 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 `cldx`.

The variables `scaleWidth` and `scaleHeight` are derived as follows:

$$scaleWidth = (cldx == 0) ? 1 : SubWidthC \quad (1416)$$

$$scaleHeight = (cldx == 0) ? 1 : SubHeightC \quad (1417)$$

The location  $(x_{Ctb}, y_{Ctb})$ , 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:

$$(x_{Ctb}, y_{Ctb}) = (rx * n_{CtbSw}, ry * n_{CtbSh}) \quad (1418)$$

The sample locations inside the current CTB are derived as follows:

$$(x_{Si}, y_{Sj}) = (x_{Ctb} + i, y_{Ctb} + j) \quad (1419)$$

$$(x_{Yi}, y_{Yj}) = (cIdx == 0) ? (x_{Si}, y_{Sj}) : (x_{Si} * SubWidthC, y_{Sj} * SubHeightC) \quad (1420)$$

For all sample locations  $(x_{Si}, y_{Sj})$  and  $(x_{Yi}, y_{Yj})$  with  $i = 0..n_{CtbSw} - 1$  and  $j = 0..n_{CtbSh} - 1$ , the following applies:

- If  $SaoTypeIdx[cIdx][rx][ry]$  is equal to 0,  $saoPicture[x_{Si}][y_{Sj}]$  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  $(x_{Sik'}, y_{Sjk'})$  and  $(x_{Yik'}, y_{Yjk'})$  are derived as follows:
 
$$(x_{Sik'}, y_{Sjk'}) = (x_{Si} + hPos[k], y_{Sj} + vPos[k]) \quad (1421)$$

$$(x_{Yik'}, y_{Yjk'}) = (cIdx == 0) ? (x_{Sik'}, y_{Sjk'}) : (x_{Sik'} * SubWidthC, y_{Sjk'} * SubHeightC) \quad (1422)$$
    - If one or more of the following conditions are true,  $edgeIdx$  is set equal to 0:
      - The sample at location  $(x_{Sik'}, y_{Sjk'})$  for any  $k = 0..1$  is outside the picture boundaries.
      - The sample at location  $(x_{Sik'}, y_{Sjk'})$  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  $recPicture[x_{Si}][y_{Sj}]$  belongs to is equal to 0.
      - $pps\_loop\_filter\_across\_slices\_enabled\_flag$  is equal to 0 and the sample at location  $(x_{Sik'}, y_{Sjk'})$  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  $(x_{Sik'}, y_{Sjk'})$  for any  $k = 0..1$  belongs to a different tile.
      - $VirtualBoundariesPresentFlag$  is equal to 1 and  $x_{Si}$  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  $x_{Si}$  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  $y_{Sj}$  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  $y_{Sj}$  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:

$$\begin{aligned} \text{edgeIdx} = & \\ & 2 + \text{Sign}( \text{recPicture}[xS_i][yS_j] - \text{recPicture}[xS_i + \text{hPos}[0]][yS_j + \text{vPos}[0]] ) + \\ & \text{Sign}( \text{recPicture}[xS_i][yS_j] - \text{recPicture}[xS_i + \text{hPos}[1]][yS_j + \text{vPos}[1]] ) \end{aligned} \quad (1423)$$

— When edgeIdx is equal to 0, 1, or 2, edgeIdx is modified as follows:

$$\text{edgeIdx} = ( \text{edgeIdx} == 2 ) ? 0 : ( \text{edgeIdx} + 1 ) \quad (1424)$$

3. The modified picture sample array  $\text{saoPicture}[xS_i][yS_j]$  is derived as follows:

$$\text{saoPicture}[xS_i][yS_j] = \text{Clip3}( 0, ( 1 \ll \text{BitDepth} ) - 1, \text{recPicture}[xS_i][yS_j] + \text{SaoOffsetVal}[ \text{cIdx} ][ \text{rx} ][ \text{ry} ][ \text{edgeIdx} ] ) \quad (1425)$$

— Otherwise ( $\text{SaoTypeIdx}[ \text{cIdx} ][ \text{rx} ][ \text{ry} ]$  is equal to 1), the following ordered steps apply:

1. The variable bandShift is set equal to  $\text{BitDepth} - 5$ .
2. The variable saoLeftClass is set equal to  $\text{sao\_band\_position}[ \text{cIdx} ][ \text{rx} ][ \text{ry} ]$ .
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:

$$\begin{aligned} & \text{for}( k = 0; k < 4; k++ ) \\ & \quad \text{bandTable}[ ( k + \text{saoLeftClass} ) \& 31 ] = k + 1 \end{aligned} \quad (1426)$$

4. The variable bandIdx is set equal to  $\text{bandTable}[ \text{recPicture}[xS_i][yS_j] \gg \text{bandShift} ]$ .
5. The modified picture sample array  $\text{saoPicture}[xS_i][yS_j]$  is derived as follows:

$$\text{saoPicture}[xS_i][yS_j] = \text{Clip3}( 0, ( 1 \ll \text{BitDepth} ) - 1, \text{recPicture}[xS_i][yS_j] + \text{SaoOffsetVal}[ \text{cIdx} ][ \text{rx} ][ \text{ry} ][ \text{bandIdx} ] ) \quad (1427)$$

**Table 44 — Specification of hPos and vPos according to the sample adaptive offset class**

| <b>SaoEoClass[ cIdx ][ rx ][ ry ]</b> | <b>0</b> | <b>1</b> | <b>2</b> | <b>3</b> |
|---------------------------------------|----------|----------|----------|----------|
| <b>hPos[ 0 ]</b>                      | -1       | 0        | -1       | 1        |
| <b>hPos[ 1 ]</b>                      | 1        | 0        | 1        | -1       |
| <b>vPos[ 0 ]</b>                      | 0        | -1       | -1       | -1       |
| <b>vPos[ 1 ]</b>                      | 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  $\text{recPicture}_L$  and, when  $\text{sps\_chroma\_format\_idc}$  is not equal to 0, the arrays  $\text{recPicture}_{Cb}$  and  $\text{recPicture}_{Cr}$ .

Outputs of this process are the modified reconstructed picture sample array after adaptive loop filter  $\text{alfPicture}_L$  and, when  $\text{sps\_chroma\_format\_idc}$  is not equal to 0, the arrays  $\text{ccAlfPicture}_{Cb}$  and  $\text{ccAlfPicture}_{Cr}$ .

The sample values in the modified reconstructed picture sample array after adaptive loop filter  $\text{alfPicture}_L$  and, when  $\text{sps\_chroma\_format\_idc}$  is not equal to 0, the arrays  $\text{alfPicture}_{Cb}$  and  $\text{alfPicture}_{Cr}$  are initially set equal to the sample values in the reconstructed picture sample array prior to adaptive loop filter  $\text{recPicture}_L$  and, when  $\text{sps\_chroma\_format\_idc}$  is not equal to 0, the arrays  $\text{recPicture}_{Cb}$  and  $\text{recPicture}_{Cr}$ , 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 subclause 8.8.5.2 is invoked with  $recPicture_L$ ,  $alfPicture_L$ , and the luma coding tree block location  $(xCtb, yCtb)$  set equal to  $(rx \ll CtbLog2SizeY, ry \ll CtbLog2SizeY)$  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 subclause 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  $((rx \ll CtbLog2SizeY) / SubWidthC, (ry \ll CtbLog2SizeY) / SubHeightC)$ , 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 subclause 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  $((rx \ll CtbLog2SizeY) / SubWidthC, (ry \ll CtbLog2SizeY) / SubHeightC)$ , 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 subclause 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  $((rx \ll CtbLog2SizeY) / SubWidthC, (ry \ll CtbLog2SizeY) / SubHeightC)$ ,  $ccAlfWidth$  set equal to  $(1 \ll CtbLog2SizeY) / SubWidthC$ ,  $ccAlfHeight$  set equal to  $(1 \ll CtbLog2SizeY) / SubHeightC$ , 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 subclause 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  $((rx \ll CtbLog2SizeY) / SubWidthC, (ry \ll CtbLog2SizeY) / SubHeightC)$ ,  $ccAlfWidth$  set equal to  $(1 \ll CtbLog2SizeY) / SubWidthC$ ,  $ccAlfHeight$  set equal to  $(1 \ll CtbLog2SizeY) / SubHeightC$ , 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 (  $x_{Ctb}, y_{Ctb}$  ) 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 subclause 8.8.5.3 is invoked with the location (  $x_{Ctb}, y_{Ctb}$  ) 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  $alfPicture_L[x_{Ctb} + x][y_{Ctb} + y]$ , each reconstructed luma sample inside the current luma coding tree block  $recPicture[x_{Ctb} + x][y_{Ctb} + y]$  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  $AlfCtbFiltSetIdxY[x_{Ctb} \gg CtbLog2SizeY][y_{Ctb} \gg CtbLog2SizeY]$  is less than 16, the following applies:

$$i = AlfCtbFiltSetIdxY[x_{Ctb} \gg CtbLog2SizeY][y_{Ctb} \gg CtbLog2SizeY] \quad (1428)$$

$$f[j] = AlfFixFiltCoeff[AlfClassToFiltMap[i][filtIdx[x][y]][j]] \quad (1429)$$

$$c[j] = 2^{BitDepth} \quad (1430)$$

— Otherwise ( $AlfCtbFiltSetIdxY[x_{Ctb} \gg CtbLog2SizeY][y_{Ctb} \gg CtbLog2SizeY]$  is greater than or equal to 16), the following applies:

$$i = sh\_alf\_aps\_id\_luma[AlfCtbFiltSetIdxY[x_{Ctb} \gg CtbLog2SizeY][y_{Ctb} \gg CtbLog2SizeY] - 16] \quad (1431)$$

$$f[j] = AlfCoeff_L[i][filtIdx[x][y]][j] \quad (1432)$$

$$c[j] = AlfClip_L[i][filtIdx[x][y]][j] \quad (1433)$$

— 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:

$$idx[] = \{ 9, 4, 10, 8, 1, 5, 11, 7, 3, 0, 2, 6 \} \quad (1434)$$

— Otherwise, if  $transposeIdx[x][y]$  is equal to 2, the following applies:

$$idx[] = \{ 0, 3, 2, 1, 8, 7, 6, 5, 4, 9, 10, 11 \} \quad (1435)$$

— Otherwise, if  $transposeIdx[x][y]$  is equal to 3, the following applies:

$$idx[] = \{ 9, 8, 10, 4, 3, 7, 11, 5, 1, 0, 2, 6 \} \quad (1436)$$

— Otherwise, the following applies:

$$idx[] = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 \} \quad (1437)$$

— The variables  $clipLeftPos$ ,  $clipRightPos$ ,  $clipTopPos$ ,  $clipBottomPos$ ,  $clipTopLeftFlag$  and  $clipBotRightFlag$  are derived by invoking the ALF boundary position derivation process as specified in subclause 8.8.5.5 with (  $x_{Ctb}, y_{Ctb}$  ), (  $x, y$  ) 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:

$$h_{x+i,y+j} = \text{Clip3}(0, \text{pps\_pic\_width\_in\_luma\_samples} - 1, x\text{Ctb} + x + i) \quad (1438)$$

$$v_{y+j} = \text{Clip3}(0, \text{pps\_pic\_height\_in\_luma\_samples} - 1, y\text{Ctb} + y + j) \quad (1439)$$

— The location  $(h_{x+i,y+j}, v_{y+j})$  is modified by invoking the ALF sample padding process as specified in subclause 8.8.5.6 with  $(x\text{Ctb}, y\text{Ctb})$ ,  $(h_{x+i,y+j}, v_{y+j})$ , 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  $\text{pps\_pic\_height\_in\_luma\_samples} - y\text{Ctb} \leq \text{CtbSizeY} - 4$ , `applyAlfLineBufBoundary` is set equal to 0.

— Otherwise, `applyAlfLineBufBoundary` is set equal to 1.

— The vertical sample position offsets  $y_1, y_2, y_3$  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:

$$\text{curr} = \text{recPicture}[h_{x,y}][v_y] \quad (1440)$$

— The variable `sum` is derived as follows:

$$\begin{aligned} \text{sum} = & f[\text{idx}[0]] * (\text{Clip3}(-c[\text{idx}[0]], c[\text{idx}[0]], \text{recPicture}[h_{x,y+y_3}][v_{y+y_3}] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[0]], c[\text{idx}[0]], \text{recPicture}[h_{x,y-y_3}][v_{y-y_3}] - \text{curr})) + \\ & f[\text{idx}[1]] * (\text{Clip3}(-c[\text{idx}[1]], c[\text{idx}[1]], \text{recPicture}[h_{x+1,y+y_2}][v_{y+y_2}] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[1]], c[\text{idx}[1]], \text{recPicture}[h_{x-1,y-y_2}][v_{y-y_2}] - \text{curr})) + \\ & f[\text{idx}[2]] * (\text{Clip3}(-c[\text{idx}[2]], c[\text{idx}[2]], \text{recPicture}[h_{x,y+y_2}][v_{y+y_2}] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[2]], c[\text{idx}[2]], \text{recPicture}[h_{x,y-y_2}][v_{y-y_2}] - \text{curr})) + \\ & f[\text{idx}[3]] * (\text{Clip3}(-c[\text{idx}[3]], c[\text{idx}[3]], \text{recPicture}[h_{x-1,y+y_2}][v_{y+y_2}] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[3]], c[\text{idx}[3]], \text{recPicture}[h_{x+1,y-y_2}][v_{y-y_2}] - \text{curr})) + \\ & f[\text{idx}[4]] * (\text{Clip3}(-c[\text{idx}[4]], c[\text{idx}[4]], \text{recPicture}[h_{x+2,y+y_1}][v_{y+y_1}] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[4]], c[\text{idx}[4]], \text{recPicture}[h_{x-2,y-y_1}][v_{y-y_1}] - \text{curr})) + \\ & f[\text{idx}[5]] * (\text{Clip3}(-c[\text{idx}[5]], c[\text{idx}[5]], \text{recPicture}[h_{x+1,y+y_1}][v_{y+y_1}] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[5]], c[\text{idx}[5]], \text{recPicture}[h_{x-1,y-y_1}][v_{y-y_1}] - \text{curr})) + \\ & f[\text{idx}[6]] * (\text{Clip3}(-c[\text{idx}[6]], c[\text{idx}[6]], \text{recPicture}[h_{x,y+y_1}][v_{y+y_1}] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[6]], c[\text{idx}[6]], \text{recPicture}[h_{x,y-y_1}][v_{y-y_1}] - \text{curr})) + \quad (1441) \\ & f[\text{idx}[7]] * (\text{Clip3}(-c[\text{idx}[7]], c[\text{idx}[7]], \text{recPicture}[h_{x-1,y+y_1}][v_{y+y_1}] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[7]], c[\text{idx}[7]], \text{recPicture}[h_{x+1,y-y_1}][v_{y-y_1}] - \text{curr})) + \\ & f[\text{idx}[8]] * (\text{Clip3}(-c[\text{idx}[8]], c[\text{idx}[8]], \text{recPicture}[h_{x-2,y+y_1}][v_{y+y_1}] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[8]], c[\text{idx}[8]], \text{recPicture}[h_{x+2,y-y_1}][v_{y-y_1}] - \text{curr})) + \\ & f[\text{idx}[9]] * (\text{Clip3}(-c[\text{idx}[9]], c[\text{idx}[9]], \text{recPicture}[h_{x+3,y}][v_y] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[9]], c[\text{idx}[9]], \text{recPicture}[h_{x-3,y}][v_y] - \text{curr})) + \\ & f[\text{idx}[10]] * (\text{Clip3}(-c[\text{idx}[10]], c[\text{idx}[10]], \text{recPicture}[h_{x+2,y}][v_y] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[10]], c[\text{idx}[10]], \text{recPicture}[h_{x-2,y}][v_y] - \text{curr})) + \\ & f[\text{idx}[11]] * (\text{Clip3}(-c[\text{idx}[11]], c[\text{idx}[11]], \text{recPicture}[h_{x+1,y}][v_y] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[11]], c[\text{idx}[11]], \text{recPicture}[h_{x-1,y}][v_y] - \text{curr})) \end{aligned}$$

$$\text{sum} = \text{curr} + ((\text{sum} + (1 \ll (\text{alfShiftY} - 1))) \gg \text{alfShiftY}) \quad (1442)$$

- The modified filtered reconstructed luma picture sample  $\text{alfPicture}_L[\text{xCtb} + \text{x}][\text{yCtb} + \text{y}]$  is derived as follows:

$$\text{alfPicture}_L[\text{xCtb} + \text{x}][\text{yCtb} + \text{y}] = \text{Clip3}(0, (1 \ll \text{BitDepth}) - 1, \text{sum}) \quad (1443)$$

**Table 45 — Specification of y1, y2, y3 and alfShiftY according to the vertical luma sample position y and applyAlfLineBufBoundary**

| Condition                                                                                                      | alfShiftY | y1 | y2 | y3 |
|----------------------------------------------------------------------------------------------------------------|-----------|----|----|----|
| $(y == \text{CtbSizeY} - 5 \mid \mid y == \text{CtbSizeY} - 4) \ \&\& \ (\text{applyAlfLineBufBoundary} == 1)$ | 10        | 0  | 0  | 0  |
| $(y == \text{CtbSizeY} - 6 \mid \mid y == \text{CtbSizeY} - 3) \ \&\& \ (\text{applyAlfLineBufBoundary} == 1)$ | 7         | 1  | 1  | 1  |
| $(y == \text{CtbSizeY} - 7 \mid \mid y == \text{CtbSizeY} - 2) \ \&\& \ (\text{applyAlfLineBufBoundary} == 1)$ | 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  $(\text{xCtb}, \text{yCtb})$  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  $\text{recPicture}$  prior to the adaptive loop filtering process.

Outputs of this process are:

- the classification filter index array  $\text{filtIdx}[\text{x}][\text{y}]$  with  $\text{x}, \text{y} = 0.. \text{CtbSizeY} - 1$ ,
- the transpose index array  $\text{transposeIdx}[\text{x}][\text{y}]$  with  $\text{x}, \text{y} = 0.. \text{CtbSizeY} - 1$ .

The variables  $\text{ac}[\text{x}][\text{y}]$ ,  $\text{sumH}[\text{x}][\text{y}]$ ,  $\text{sumV}[\text{x}][\text{y}]$ ,  $\text{sumD0}[\text{x}][\text{y}]$ ,  $\text{sumD1}[\text{x}][\text{y}]$  and  $\text{sumOfHV}[\text{x}][\text{y}]$  with  $\text{x}, \text{y} = 0..(\text{CtbSizeY} - 1) \gg 2$  are derived as follows:

- The variables  $\text{x4}$  and  $\text{y4}$  are set as  $(\text{x} \ll 2)$  and  $(\text{y} \ll 2)$ , respectively.
- The variables  $\text{minY}$ ,  $\text{maxY}$ , and  $\text{ac}[\text{x}][\text{y}]$  are derived as follows:
  - If  $\text{y4}$  is equal to  $(\text{CtbSizeY} - 8)$  and one of the following conditions is true,  $\text{minY}$  is set equal to  $-2$ ,  $\text{maxY}$  is set equal to  $3$ , and  $\text{ac}[\text{x}][\text{y}]$  is set equal to  $3$ :
    - The bottom boundary of the current coding tree block is the bottom boundary of the picture and  $\text{pps\_pic\_height\_in\_luma\_samples} - \text{yCtb} > \text{CtbSizeY} - 4$ .
    - The bottom boundary of the current coding tree block is not the bottom boundary of the picture.
  - Otherwise, if  $\text{y4}$  is equal to  $(\text{CtbSizeY} - 4)$  and one of the following conditions is true,  $\text{minY}$  is set equal to  $0$ ,  $\text{maxY}$  is set equal to  $5$ , and  $\text{ac}[\text{x}][\text{y}]$  is set equal to  $3$ :
    - The bottom boundary of the current coding tree block is the bottom boundary of the picture and  $\text{pps\_pic\_height\_in\_luma\_samples} - \text{yCtb} > \text{CtbSizeY} - 4$ .
    - The bottom boundary of the current coding tree block is not the bottom boundary of the picture.
  - Otherwise,  $\text{minY}$  is set equal to  $-2$  and  $\text{maxY}$  is set equal to  $5$ , and  $\text{ac}[\text{x}][\text{y}]$  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 subclause 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:

$$h_{x4+i, y4+j} = \text{Clip3}( 0, \text{pps\_pic\_width\_in\_luma\_samples} - 1, x\text{Ctb} + x4 + i ) \quad (1444)$$

$$v_{y4+j} = \text{Clip3}( 0, \text{pps\_pic\_height\_in\_luma\_samples} - 1, y\text{Ctb} + y4 + j ) \quad (1445)$$

- The location (  $h_{x4+i, y4+j}$ ,  $v_{y4+j}$  ) is modified by invoking the ALF sample padding process as specified in subclause 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:

$$\begin{aligned} \text{filtH}[ i ][ j ] = & \text{Abs}( ( \text{recPicture}[ h_{x4+i, y4+j} ][ v_{y4+j} ] << 1 ) - \\ & \text{recPicture}[ h_{x4+i-1, y4+j} ][ v_{y4+j} ] - \\ & \text{recPicture}[ h_{x4+i+1, y4+j} ][ v_{y4+j} ] ) \end{aligned} \quad (1446)$$

$$\begin{aligned} \text{filtV}[ i ][ j ] = & \text{Abs}( ( \text{recPicture}[ h_{x4+i, y4+j} ][ v_{y4+j} ] << 1 ) - \\ & \text{recPicture}[ h_{x4+i, y4+j-1} ][ v_{y4+j-1} ] - \\ & \text{recPicture}[ h_{x4+i, y4+j+1} ][ v_{y4+j+1} ] ) \end{aligned} \quad (1447)$$

$$\begin{aligned} \text{filtD0}[ i ][ j ] = & \text{Abs}( ( \text{recPicture}[ h_{x4+i, y4+j} ][ v_{y4+j} ] << 1 ) - \\ & \text{recPicture}[ h_{x4+i-1, y4+j-1} ][ v_{y4+j-1} ] - \\ & \text{recPicture}[ h_{x4+i+1, y4+j+1} ][ v_{y4+j+1} ] ) \end{aligned} \quad (1448)$$

$$\begin{aligned} \text{filtD1}[ i ][ j ] = & \text{Abs}( ( \text{recPicture}[ h_{x4+i, y4+j} ][ v_{y4+j} ] << 1 ) - \\ & \text{recPicture}[ h_{x4+i+1, y4+j-1} ][ v_{y4+j-1} ] - \\ & \text{recPicture}[ h_{x4+i-1, y4+j+1} ][ v_{y4+j+1} ] ) \end{aligned} \quad (1449)$$

- 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:

$$\text{sumH}[ x ][ y ] = \sum_i \sum_j \text{filtH}[ i ][ j ], \text{ with } i = -2..5, j = \text{minY}.. \text{maxY} \quad (1450)$$

$$\text{sumV}[ x ][ y ] = \sum_i \sum_j \text{filtV}[ i ][ j ], \text{ with } i = -2..5, j = \text{minY}.. \text{maxY} \quad (1451)$$

$$\text{sumD0}[ x ][ y ] = \sum_i \sum_j \text{filtD0}[ i ][ j ], \text{ with } i = -2..5, j = \text{minY}.. \text{maxY} \quad (1452)$$

$$\text{sumD1}[ x ][ y ] = \sum_i \sum_j \text{filtD1}[ i ][ j ], \text{ with } i = -2..5, j = \text{minY}.. \text{maxY} \quad (1453)$$

$$\text{sumOfHV}[ x ][ y ] = \text{sumH}[ x ][ y ] + \text{sumV}[ x ][ y ] \quad (1454)$$

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  $x, y = 0.. \text{CtbSizeY} - 1$  are derived as follows:
  - The variables hv1, hv0 and dirHV are derived as follows:

- If  $\text{sumV}[x \gg 2][y \gg 2]$  is greater than  $\text{sumH}[x \gg 2][y \gg 2]$ , the following applies:

$$\text{hv1} = \text{sumV}[x \gg 2][y \gg 2] \quad (1455)$$

$$\text{hv0} = \text{sumH}[x \gg 2][y \gg 2] \quad (1456)$$

$$\text{dirHV} = 1 \quad (1457)$$

- Otherwise, the following applies:

$$\text{hv1} = \text{sumH}[x \gg 2][y \gg 2] \quad (1458)$$

$$\text{hv0} = \text{sumV}[x \gg 2][y \gg 2] \quad (1459)$$

$$\text{dirHV} = 3 \quad (1460)$$

- The variables  $d1$ ,  $d0$  and  $\text{dirD}$  are derived as follows:

- If  $\text{sumD0}[x \gg 2][y \gg 2]$  is greater than  $\text{sumD1}[x \gg 2][y \gg 2]$ , the following applies:

$$d1 = \text{sumD0}[x \gg 2][y \gg 2] \quad (1461)$$

$$d0 = \text{sumD1}[x \gg 2][y \gg 2] \quad (1462)$$

$$\text{dirD} = 0 \quad (1463)$$

- Otherwise, the following applies:

$$d1 = \text{sumD1}[x \gg 2][y \gg 2] \quad (1464)$$

$$d0 = \text{sumD0}[x \gg 2][y \gg 2] \quad (1465)$$

$$\text{dirD} = 2 \quad (1466)$$

- The variables  $\text{hvd1}$ ,  $\text{hvd0}$ , are derived as follows:

$$\text{hvd1} = (d1 * \text{hv0} > \text{hv1} * d0) ? d1 : \text{hv1} \quad (1467)$$

$$\text{hvd0} = (d1 * \text{hv0} > \text{hv1} * d0) ? d0 : \text{hv0} \quad (1468)$$

- The variables  $\text{dirS}[x][y]$ ,  $\text{dir1}[x][y]$  and  $\text{dir2}[x][y]$  derived as follows:

$$\text{dir1}[x][y] = (d1 * \text{hv0} > \text{hv1} * d0) ? \text{dirD} : \text{dirHV} \quad (1469)$$

$$\text{dir2}[x][y] = (d1 * \text{hv0} > \text{hv1} * d0) ? \text{dirHV} : \text{dirD} \quad (1470)$$

$$\text{dirS}[x][y] = (\text{hvd1} * 2 > 9 * \text{hvd0}) ? 2 : ((\text{hvd1} > 2 * \text{hvd0}) ? 1 : 0) \quad (1471)$$

2. The variable  $\text{avgVar}[x][y]$  with  $x, y = 0..CtbSizeY - 1$  is derived as follows:

$$\text{varTab}[] = \{ 0, 1, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 4 \} \quad (1472)$$

$$\text{avgVar}[x][y] = \text{varTab}[\text{Clip3}(0, 15, (\text{sumOfHV}[x \gg 2][y \gg 2] * \text{ac}[x \gg 2][y \gg 2]) \gg (\text{BitDepth} - 1))] \quad (1473)$$

3. The classification filter index array  $\text{filtIdx}[x][y]$  and the transpose index array  $\text{transposeIdx}[x][y]$  with  $x, y = 0..CtbSizeY - 1$  are derived as follows:

$$\text{transposeTable}[] = \{ 0, 1, 0, 2, 2, 3, 1, 3 \}$$

$$\text{transposeIdx}[x][y] = \text{transposeTable}[\text{dir1}[x][y] * 2 + (\text{dir2}[x][y] \gg 1)]$$

$$\text{filtIdx}[x][y] = \text{avgVar}[x][y]$$

— When  $\text{dirS}[x][y]$  is not equal to 0,  $\text{filtIdx}[x][y]$  is modified as follows:

$$\text{filtIdx}[x][y] += ((\text{dir1}[x][y] \& 0x1) << 1) + \text{dirS}[x][y] * 5 \quad (1474)$$

#### 8.8.5.4 Coding tree block filtering process for chroma samples

Inputs of this process are:

- a reconstructed chroma picture sample array  $\text{recPicture}$  prior to the adaptive loop filtering process,
- a filtered reconstructed chroma picture sample array  $\text{alfPicture}$ ,
- a chroma location  $(x_{\text{CtbC}}, y_{\text{CtbC}})$  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  $\text{altIdx}$ .

Output of this process is the modified filtered reconstructed chroma picture sample array  $\text{alfPicture}$ .

The width and height of the current chroma coding tree block  $\text{ctbWidthC}$  and  $\text{ctbHeightC}$  is derived as follows:

$$\text{ctbWidthC} = \text{CtbSizeY} / \text{SubWidthC} \quad (1475)$$

$$\text{ctbHeightC} = \text{CtbSizeY} / \text{SubHeightC} \quad (1476)$$

For the derivation of the filtered reconstructed chroma samples  $\text{alfPicture}[x_{\text{CtbC}} + x][y_{\text{CtbC}} + y]$ , each reconstructed chroma sample inside the current chroma coding tree block  $\text{recPicture}[x_{\text{CtbC}} + x][y_{\text{CtbC}} + y]$  is filtered as follows with  $x = 0..\text{ctbWidthC} - 1$ ,  $y = 0..\text{ctbHeightC} - 1$ :

- The variables  $\text{clipLeftPos}$ ,  $\text{clipRightPos}$ ,  $\text{clipTopPos}$ ,  $\text{clipBottomPos}$ ,  $\text{clipTopLeftFlag}$  and  $\text{clipBotRightFlag}$  are derived by invoking the ALF boundary position derivation process as specified in subclause 8.8.5.5 with  $(x_{\text{CtbC}} * \text{SubWidthC}, y_{\text{CtbC}} * \text{SubHeightC})$ ,  $(x * \text{SubWidthC}, y * \text{SubHeightC})$  and the variable  $\text{vbOffset}$  set equal to  $2 * \text{SubHeightC}$  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  $\text{recPicture}$  of chroma samples with  $i, j = -2..2$  are derived as follows:

$$h_{x+i, y+j} = \text{Clip3}(0, \text{pps\_pic\_width\_in\_luma\_samples} / \text{SubWidthC} - 1, x_{\text{CtbC}} + x + i) \quad (1477)$$

$$v_{y+j} = \text{Clip3}(0, \text{pps\_pic\_height\_in\_luma\_samples} / \text{SubHeightC} - 1, y_{\text{CtbC}} + y + j) \quad (1478)$$

- The location  $(h_{x+i, y+j}, v_{y+j})$  is modified by invoking the ALF sample padding process as specified in subclause 8.8.5.6 with  $(x_{\text{CtbC}} * \text{SubWidthC}, y_{\text{CtbC}} * \text{SubHeightC})$ ,  $(h_{x+i, y+j}, v_{y+j})$ , the variable  $\text{isChroma}$  set equal to 1,  $\text{clipLeftPos}$ ,  $\text{clipRightPos}$ ,  $\text{clipTopPos}$ ,  $\text{clipBottomPos}$ ,  $\text{clipTopLeftFlag}$  and  $\text{clipBotRightFlag}$  as inputs.
- The variable  $\text{applyAlfLineBufBoundary}$  is derived as follows:
  - If the bottom boundary of the current coding tree block is the bottom boundary of the picture and  $\text{pps\_pic\_height\_in\_luma\_samples} - (y_{\text{CtbC}} * \text{SubHeightC}) < \text{CtbSizeY} - 4$ ,  $\text{applyAlfLineBufBoundary}$  is set equal to 0.
  - Otherwise,  $\text{applyAlfLineBufBoundary}$  is set equal to 1.
- The vertical sample position offsets  $y1$ ,  $y2$  and the variable  $\text{alfShiftC}$  are specified in Table 46 according to the vertical chroma sample position  $y$  and  $\text{applyAlfLineBufBoundary}$ .
- The variable  $\text{curr}$  is derived as follows:

$$\text{curr} = \text{recPicture}[h_{x,y}][v_y] \quad (1479)$$

- The array of chroma filter coefficients  $f[j]$  and the array of chroma clipping values  $c[j]$  is derived as follows with  $j = 0..5$ :

$$f[j] = \text{AlfCoeffc}[\text{sh\_alf\_aps\_id\_chroma}][\text{altIdx}][j] \quad (1480)$$

$$c[j] = \text{AlfClipc}[\text{sh\_alf\_aps\_id\_chroma}][\text{altIdx}][j] \quad (1481)$$

- The variable sum is derived as follows:

$$\begin{aligned} \text{sum} = & f[0] * (\text{Clip3}(-c[0], c[0], \text{recPicture}[h_{x,y+y2}][v_{y+y2}] - \text{curr}) + \\ & \text{Clip3}(-c[0], c[0], \text{recPicture}[h_{x,y-y2}][v_{y-y2}] - \text{curr})) + \\ & f[1] * (\text{Clip3}(-c[1], c[1], \text{recPicture}[h_{x+1,y+y1}][v_{y+y1}] - \text{curr}) + \\ & \text{Clip3}(-c[1], c[1], \text{recPicture}[h_{x-1,y-y1}][v_{y-y1}] - \text{curr})) + \\ & f[2] * (\text{Clip3}(-c[2], c[2], \text{recPicture}[h_{x,y+y1}][v_{y+y1}] - \text{curr}) + \\ & \text{Clip3}(-c[2], c[2], \text{recPicture}[h_{x,y-y1}][v_{y-y1}] - \text{curr})) + \\ & f[3] * (\text{Clip3}(-c[3], c[3], \text{recPicture}[h_{x-1,y+y1}][v_{y+y1}] - \text{curr}) + \\ & \text{Clip3}(-c[3], c[3], \text{recPicture}[h_{x+1,y-y1}][v_{y-y1}] - \text{curr})) + \\ & f[4] * (\text{Clip3}(-c[4], c[4], \text{recPicture}[h_{x+2,y}][v_y] - \text{curr}) + \\ & \text{Clip3}(-c[4], c[4], \text{recPicture}[h_{x-2,y}][v_y] - \text{curr})) + \\ & f[5] * (\text{Clip3}(-c[5], c[5], \text{recPicture}[h_{x+1,y}][v_y] - \text{curr}) + \\ & \text{Clip3}(-c[5], c[5], \text{recPicture}[h_{x-1,y}][v_y] - \text{curr})) \end{aligned} \quad (1482)$$

$$\text{sum} = \text{curr} + ((\text{sum} + (1 \ll (\text{alfShiftC} - 1))) \gg \text{alfShiftC}) \quad (1483)$$

- The modified filtered reconstructed chroma picture sample  $\text{alfPicture}[x\text{CtbC} + x][y\text{CtbC} + y]$  is derived as follows:

$$\text{alfPicture}[x\text{CtbC} + x][y\text{CtbC} + y] = \text{Clip3}(0, (1 \ll \text{BitDepth}) - 1, \text{sum}) \quad (1484)$$

**Table 46 — Specification of  $y1$ ,  $y2$  and  $\text{alfShiftC}$  according to the vertical chroma sample position  $y$  and  $\text{applyAlfLineBufBoundary}$**

| Condition                                                                                                             | $\text{alfShiftC}$ | $y1$ | $y2$ |
|-----------------------------------------------------------------------------------------------------------------------|--------------------|------|------|
| $(y == \text{ctbHeightC} - 2 \mid \mid y == \text{ctbHeightC} - 3) \ \&\&$<br>$(\text{applyAlfLineBufBoundary} == 1)$ | 10                 | 0    | 0    |
| $(y == \text{ctbHeightC} - 1 \mid \mid y == \text{ctbHeightC} - 4) \ \&\&$<br>$(\text{applyAlfLineBufBoundary} == 1)$ | 7                  | 1    | 1    |
| Otherwise                                                                                                             | 7                  | 1    | 2    |

#### 8.8.5.5 ALF boundary position derivation process

Inputs of this process are:

- a luma location  $(x\text{Ctb}, y\text{Ctb})$  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  $\text{vbOffset}$  specifying the offset for ALF virtual boundary.

Output of this process are:

- the left vertical boundary position  $\text{clipLeftPos}$ ,
- the right vertical boundary position  $\text{clipRightPos}$ ,
- the above horizontal boundary position  $\text{clipTopPos}$ ,



- 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  $y - (\text{CtbSizeY} - \text{vbOffset})$  is greater than or equal to 0, the variable clipTopPos is set equal to  $y_{\text{Ctb}} + \text{CtbSizeY} - \text{vbOffset}$ .
- Otherwise, if VirtualBoundariesPresentFlag is equal to 1, and  $y_{\text{Ctb}} + y - \text{VirtualBoundaryPosY}[n]$  is greater than or equal to 0 and less than 3 for any  $n = 0.. \text{NumHorVirtualBoundaries} - 1$ , the following applies:

$$\text{clipTopPos} = \text{VirtualBoundaryPosY}[n] \quad (1485)$$

- Otherwise, if y is less than 3 and one or more of the following conditions are true, the variable clipTopPos is set equal to  $y_{\text{Ctb}}$ :
  - 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 ] -  $y_{\text{Ctb}} - y$  is greater than 0 and less than 5 for any  $n = 0.. \text{NumHorVirtualBoundaries} - 1$ , the following applies:

$$\text{clipBottomPos} = \text{VirtualBoundaryPosY}[n] \quad (1486)$$

- Otherwise, if  $\text{CtbSizeY} - \text{vbOffset} - y$  is greater than 0 and is less than 5, the variable clipBottomPos is set equal to  $y_{\text{Ctb}} + \text{CtbSizeY} - \text{vbOffset}$ .
- Otherwise, if  $\text{CtbSizeY} - y$  is less than 5, and one or more of the following conditions are true, the variable clipBottomPos is set equal to  $y_{\text{Ctb}} + \text{CtbSizeY}$ :
  - 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  $x_{\text{Ctb}} + x - \text{VirtualBoundaryPosX}[n]$  is greater than or equal to 0 and less than 3 for any  $n = 0.. \text{NumVerVirtualBoundaries} - 1$ , the following applies:

$$\text{clipLeftPos} = \text{VirtualBoundaryPosX}[n] \quad (1487)$$

- 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:

$$\text{clipRightPos} = \text{VirtualBoundaryPosX}[ n ] \quad (1488)$$

- Otherwise, if  $\text{CtbSizeY} - x$  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_tiles_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` specifying whether the colour component 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:

$$\text{picWidth} = \text{isChroma} ? \text{pps\_pic\_width\_in\_luma\_samples} / \text{SubWidthC} : \text{pps\_pic\_width\_in\_luma\_samples} \quad (1489)$$

$$\text{picHeight} = \text{isChroma} ? \text{pps\_pic\_height\_in\_luma\_samples} / \text{SubHeightC} : \text{pps\_pic\_height\_in\_luma\_samples} \quad (1490)$$

$$\text{xCtbCur} = \text{isChroma} ? \text{xCtb} / \text{SubWidthC} : \text{xCtb} \quad (1491)$$

$$\text{yCtbCur} = \text{isChroma} ? \text{yCtb} / \text{SubHeightC} : \text{yCtb} \quad (1492)$$

$$\text{CtbSizeHor} = \text{isChroma} ? \text{CtbSizeY} / \text{SubWidthC} : \text{CtbSizeY} \quad (1493)$$

$$\text{CtbSizeVer} = \text{isChroma} ? \text{CtbSizeY} / \text{SubHeightC} : \text{CtbSizeY} \quad (1494)$$

$$\text{topBryPos} = \text{isChroma} ? \text{clipTopPos} / \text{SubHeightC} : \text{clipTopPos} \quad (1495)$$

$$\text{botBryPos} = \text{isChroma} ? \text{clipBottomPos} / \text{SubHeightC} : \text{clipBottomPos} \quad (1496)$$

$$\text{leftBryPos} = \text{isChroma} ? \text{clipLeftPos} / \text{SubWidthC} : \text{clipLeftPos} \quad (1497)$$

$$\text{rightBryPos} = \text{isChroma} ? \text{clipRightPos} / \text{SubWidthC} : \text{clipRightPos} \quad (1498)$$

The variables x and y are modified as follows:

- When topBryPos is not less than 0, the following applies:

$$y = \text{Clip3}(\text{topBryPos}, \text{picHeight} - 1, y) \quad (1499)$$

- When botBryPos is not less than 0, the following applies:

$$y = \text{Clip3}(0, \text{botBryPos} - 1, y) \quad (1500)$$

- When leftBryPos is not less than 0, the following applies:

$$x = \text{Clip3}(\text{leftBryPos}, \text{picWidth} - 1, x) \quad (1501)$$

- When rightBryPos is not less than 0, the following applies:

$$x = \text{Clip3}(0, \text{rightBryPos} - 1, x) \quad (1502)$$

- 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  $x_{CtbCur} + CtbSizeHor - 1$  and  $y$  is greater than  $y_{CtbCur} + CtbSizeVer - 1$ .

### 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  $(x_{CtbC}, y_{CtbC})$  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  $ccAlfPicture[x_{CtbC} + x][y_{CtbC} + y]$ , each reconstructed chroma sample inside the current chroma block of samples  $alfPicture_C[x_{CtbC} + x][y_{CtbC} + y]$  with  $x = 0..ccAlfWidth - 1$ ,  $y = 0..ccAlfHeight - 1$ , is filtered as follows:

- The luma location  $(x_L, y_L)$  corresponding to the current chroma sample at chroma location  $(x_{CtbC} + x, y_{CtbC} + y)$  is set equal to  $((x_{CtbC} + x) * SubWidthC, (y_{CtbC} + y) * SubHeightC)$ .
- The variables  $clipLeftPos$ ,  $clipRightPos$ ,  $clipTopPos$ ,  $clipBottomPos$ ,  $clipTopLeftFlag$  and  $clipBotRightFlag$  are derived by invoking the ALF boundary position derivation process as specified in subclause 8.8.5.5 with  $(x_{CtbC} * SubWidthC, y_{CtbC} * SubHeightC)$ ,  $(x * SubWidthC, y * SubHeightC)$  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:

$$h_{x+i, y+j} = Clip3(0, pps\_pic\_width\_in\_luma\_samples - 1, x_L + i) \quad (1503)$$

$$v_{y+j} = Clip3(0, pps\_pic\_height\_in\_luma\_samples - 1, y_L + j) \quad (1504)$$

- The location  $(h_{x+i, y+j}, v_{y+j})$  is modified by invoking the ALF sample padding process as specified in subclause 8.8.5.6 with  $(x_{CtbC} * SubWidthC, y_{CtbC} * SubHeightC)$ ,  $(h_{x+i, y+j}, v_{y+j})$ , 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  $pps\_pic\_height\_in\_luma\_samples - y_{CtbC} * SubHeightC$  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:

$$curr = alfPicture_C[x_{CtbC} + x][y_{CtbC} + y] \quad (1505)$$

- The array of cross-component filter coefficients  $f[j]$  is derived as follows with  $j = 0..6$ :

$$f[j] = CcAlfCoeff[j] \quad (1506)$$

— The variable sum is derived as follows:

$$\begin{aligned} \text{sum} = & f[0] * (\text{recPicture}_L[h_{x,y-yP1}][v_{y-yP1}] - \text{recPicture}_L[h_{x,y}][v_y]) + \\ & f[1] * (\text{recPicture}_L[h_{x-1,y}][v_y] - \text{recPicture}_L[h_{x,y}][v_y]) + \\ & f[2] * (\text{recPicture}_L[h_{x+1,y}][v_y] - \text{recPicture}_L[h_{x,y}][v_y]) + \\ & f[3] * (\text{recPicture}_L[h_{x-1,y+yP1}][v_{y+yP1}] - \text{recPicture}_L[h_{x,y}][v_y]) + \\ & f[4] * (\text{recPicture}_L[h_{x,y+yP1}][v_{y+yP1}] - \text{recPicture}_L[h_{x,y}][v_y]) + \\ & f[5] * (\text{recPicture}_L[h_{x+1,y+yP1}][v_{y+yP1}] - \text{recPicture}_L[h_{x,y}][v_y]) + \\ & f[6] * (\text{recPicture}_L[h_{x,y+yP2}][v_{y+yP2}] - \text{recPicture}_L[h_{x,y}][v_y]) \end{aligned} \quad (1507)$$

$$\text{scaledSum} = \text{Clip3}(-(1 \ll (\text{BitDepth} - 1)), (1 \ll (\text{BitDepth} - 1)) - 1, (\text{sum} + 64) \gg 7) \quad (1508)$$

$$\begin{aligned} \text{sum} = & (\text{SubHeightC} == 1 \ \&\& \ (y == \text{CtbSizeY} - 3 \ || \ y == \text{CtbSizeY} - 4)) ? \\ & \text{curr} : \text{curr} + \text{scaledSum} \end{aligned} \quad (1509)$$

— The modified filtered reconstructed chroma picture sample  $\text{ccAlfPicture}[x\text{CtbC} + x][y\text{CtbC} + y]$  is derived as follows:

$$\text{ccAlfPicture}[x\text{CtbC} + x][y\text{CtbC} + y] = \text{Clip3}(0, (1 \ll \text{BitDepth}) - 1, \text{sum}) \quad (1510)$$

**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  $\text{ue}(v)$ ,  $\text{se}(v)$ , or  $\text{ae}(v)$  (see subclause 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  $\text{ue}(v)$  or  $\text{se}(v)$ .

Inputs to this process are bits from the RBSP.

Outputs of this process are syntax element values.

Syntax elements coded as  $\text{ue}(v)$  or  $\text{se}(v)$  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++ )
    b = read_bits( 1 )

```

(1511)

The variable codeNum is then assigned as follows:

$$\text{codeNum} = ( 2^{\text{leadingZeroBits} - 1} ) * 2^k + \text{read\_bits}( \text{leadingZeroBits} + k )$$

(1512)

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**

| 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)**

| 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 subclause 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 subclause 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                                   |
| $k$       | $(-1)^{k+1} * \text{Ceil}(k \div 2)$ |

## 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 subclause 9.3.2 is invoked when starting the parsing of the CTU syntax specified in subclause 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 subclause 9.3.3.

The binarization for the syntax element and the sequence of parsed bins determines the decoding process flow as described in subclause 9.3.4.

The storage process for context variables is applied as follows:

- When ending the parsing of the CTU syntax in subclause 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 subclause 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 subclause 7.3.11.2 and the decoding process of the last CU in the CTU in subclause 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 specified in subclause 9.3.2.6 is invoked.

The whole CABAC parsing process for a syntax element `synEl` is illustrated in Figure 11.

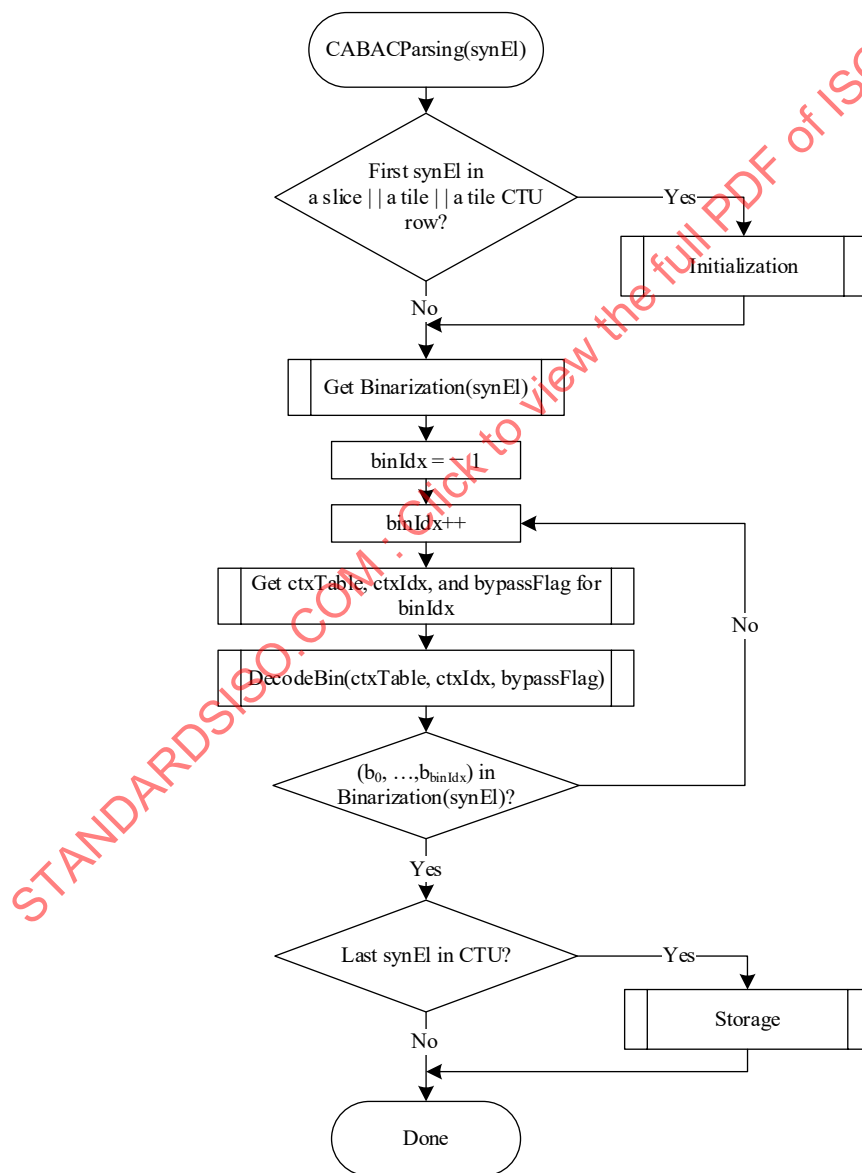


Figure 11 — Flowchart of CABAC parsing process for a syntax element `synEl`

### 9.3.2 Initialization process

#### 9.3.2.1 General

Outputs of this process are initialized CABAC internal variables.

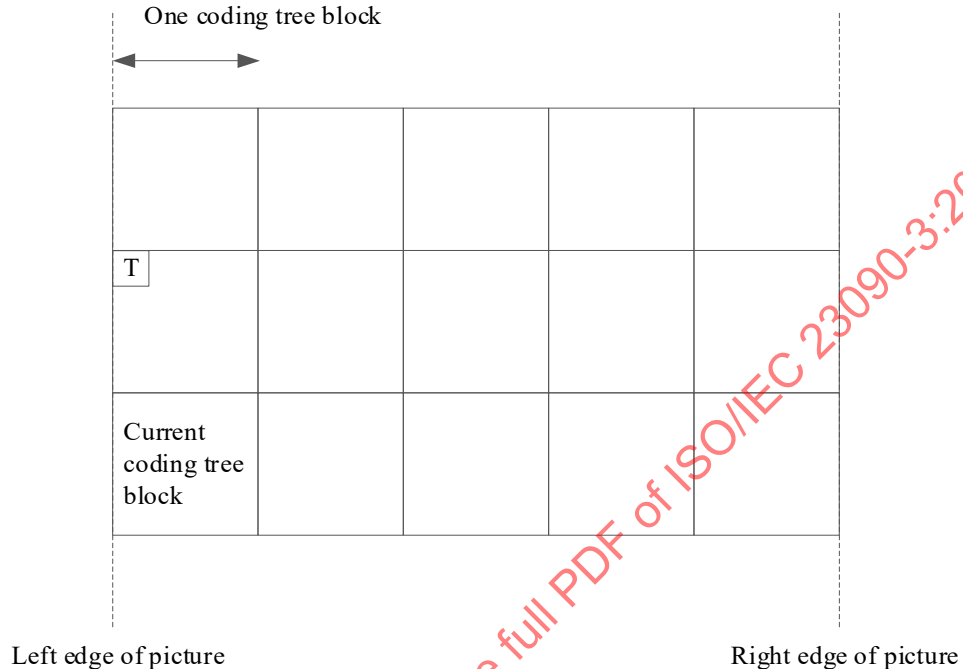


**Figure 12 — Spatial neighbour T that is used to invoke the CTB availability derivation process relative to the current CTB**

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:

$$\text{StatCoeff}[i] = \text{sps\_persistent\_rice\_adaptation\_enabled\_flag} ? 2 * \text{Floor}(\text{Log2}(\text{BitDepth} - 10)) : 0 \quad (1513)$$

- If the CTU is the first CTU in a slice or tile, the initialization process for context variables is invoked as specified in subclause 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:

$$(xNbT, yNbT) = (x0, y0 - \text{CtbSizeY}) \quad (1514)$$

- The derivation process for neighbouring block availability as specified in subclause 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 *cldx* 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 subclause 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 subclause 9.3.2.7 is invoked.
- Otherwise, the initialization process for context variables is invoked as specified in subclause 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 subclause 9.3.2.5.

The whole initialization process for a syntax element synEl is illustrated in Figure 13.

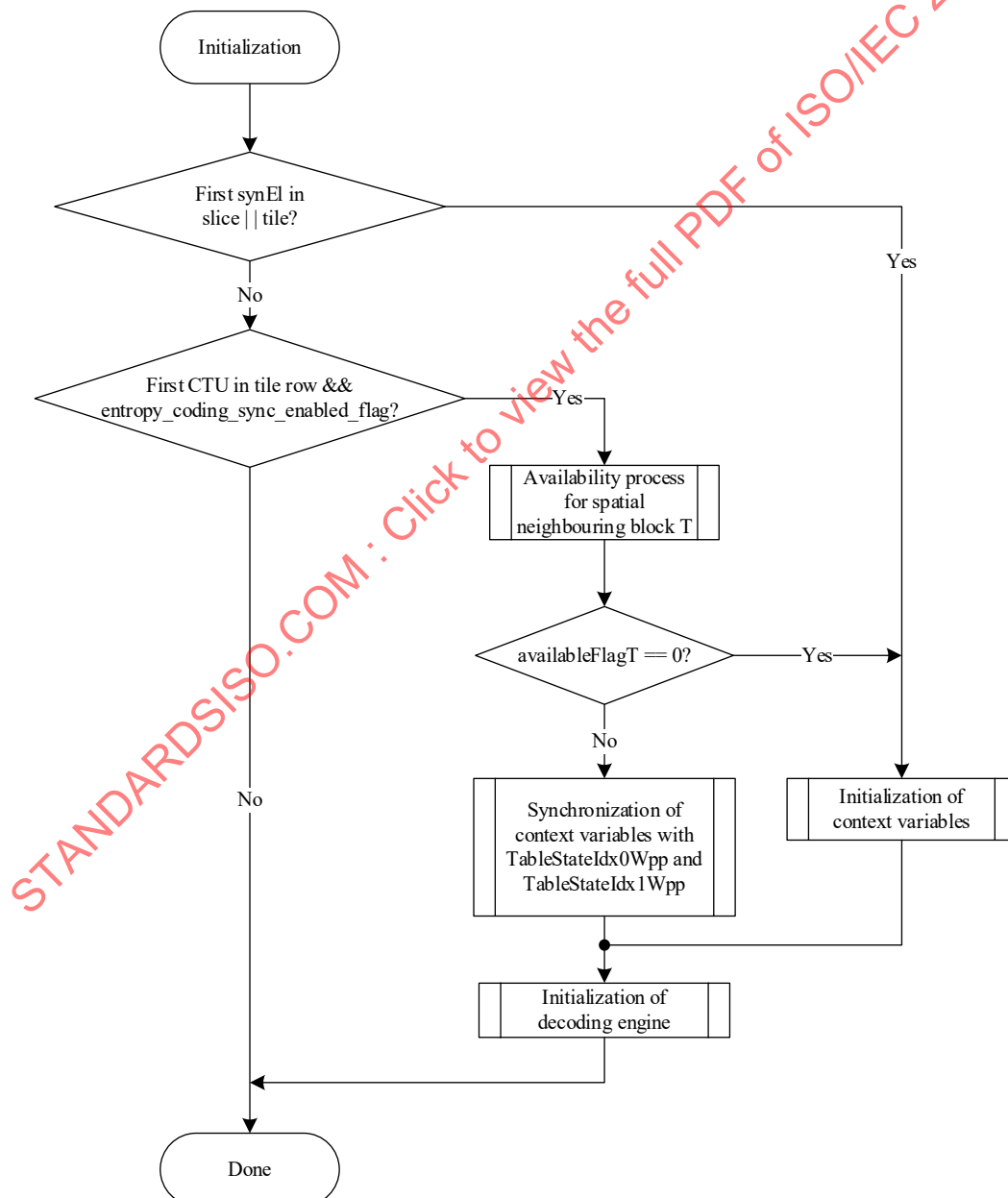


Figure 13 — Flowchart of CABAC initialization process

### 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 subclauses 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:

$$\begin{aligned} \text{slopeIdx} &= \text{initValue} \gg 3 \\ \text{offsetIdx} &= \text{initValue} \& 7 \end{aligned} \quad (1515)$$

- The variables m and n, used in the initialization of context variables, are derived from slopeIdx and offsetIdx as follows:

$$\begin{aligned} m &= \text{slopeIdx} - 4 \\ n &= (\text{offsetIdx} * 18) + 1 \end{aligned} \quad (1516)$$

- The two values assigned to pStateIdx0 and pStateIdx1 for the initialization are derived from SliceQp<sub>V</sub>, which is derived in Formula (140). Given the variables m and n, the initialization is specified as follows:

$$\text{preCtxState} = \text{Clip3}(1, 127, ((m * (\text{Clip3}(0, 63, \text{SliceQp}_V) - 16)) \gg 1) + n) \quad (1517)$$

- The two values assigned to pStateIdx0 and pStateIdx1 for the initialization are derived as follows:

$$\begin{aligned} \text{pStateIdx0} &= \text{preCtxState} \ll 3 \\ \text{pStateIdx1} &= \text{preCtxState} \ll 7 \end{aligned} \quad (1518)$$

NOTE The variables pStateIdx0 and pStateIdx1 correspond to the probability state indices as further described in subclause 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:

$$\begin{aligned} &\text{if}(\text{sh\_slice\_type} == I) \\ &\quad \text{initType} = 0 \\ &\text{else if}(\text{sh\_slice\_type} == P) \\ &\quad \text{initType} = \text{sh\_cabac\_init\_flag} ? 2 : 1 \\ &\text{else} \\ &\quad \text{initType} = \text{sh\_cabac\_init\_flag} ? 1 : 2 \end{aligned} \quad (1519)$$

**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<br>sao_merge_up_flag | Table 57 | 0        | 1     | 2      |
|                    | sao_type_idx_luma<br>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   |
| 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[ ][ ],<br>mvp_l1_flag[ ][ ]  | Table 88 | 0        | 1     | 2      |
|                    | amvr_flag[ ][ ]                          | Table 89 |          | 0..1  | 2..3   |



| Syntax structure   | Syntax element                                                  | ctxTable  | initType |         |          |
|--------------------|-----------------------------------------------------------------|-----------|----------|---------|----------|
|                    |                                                                 |           | 0        | 1       | 2        |
|                    | 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     |
|                    | cu_sbt_pos_flag                                                 | Table 96  |          | 0       | 1        |
|                    | lnst_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[ ][ ]<br>merge_gpm_idx0[ ][ ]<br>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        |
| 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_cbr_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 <code>alf_ctb_flag</code> |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|-------------------------|-------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|                         | 0                                   | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
| <code>initValue</code>  | 62                                  | 39 | 39 | 54 | 39 | 39 | 31 | 39 | 39 | 13 | 23 | 46 | 4  | 61 | 54 | 19 |
| <code>shiftIdx</code>   | 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 |    |    |    |    |    |
| <code>initValue</code>  | 46                                  | 54 | 33 | 52 | 46 | 25 | 61 | 54 | 25 | 61 | 54 |    |    |    |    |    |
| <code>shiftIdx</code>   | 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 <code>alf_use_aps_flag</code> |    |    |
|-------------------------|-----------------------------------------|----|----|
|                         | 0                                       | 1  | 2  |
| <code>initValue</code>  | 46                                      | 46 | 46 |
| <code>shiftIdx</code>   | 0                                       | 0  | 0  |

Table 54 — Specification of `initValue` and `shiftIdx` for `ctxIdx` of `alf_ctb_cc_cb_idc`

| Initialization variable | ctxIdx of <code>alf_ctb_cc_cb_idc</code> |    |    |    |    |    |    |    |    |
|-------------------------|------------------------------------------|----|----|----|----|----|----|----|----|
|                         | 0                                        | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  |
| <code>initValue</code>  | 18                                       | 30 | 31 | 18 | 21 | 38 | 25 | 35 | 38 |
| <code>shiftIdx</code>   | 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 <code>alf_ctb_cc_cr_idc</code> |    |    |    |    |    |    |    |    |
|-------------------------|------------------------------------------|----|----|----|----|----|----|----|----|
|                         | 0                                        | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  |
| <code>initValue</code>  | 18                                       | 30 | 31 | 18 | 21 | 38 | 25 | 28 | 38 |
| <code>shiftIdx</code>   | 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 <code>alf_ctb_filter_alt_idx</code> |    |    |    |    |    |
|-------------------------|-----------------------------------------------|----|----|----|----|----|
|                         | 0                                             | 1  | 2  | 3  | 4  | 5  |
| <code>initValue</code>  | 11                                            | 11 | 20 | 12 | 11 | 26 |
| <code>shiftIdx</code>   | 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  | 1  | 1 | 1  | 1  | 1  | 1  |
| initValue               | 27                      | 6 | 15 | 25 | 19 | 37 | 20 | 14 | 23 | 18 | 19 | 6 | 26 | 36 | 38 | 18 |
| shiftIdx                | 0                       | 8 | 8  | 12 | 12 | 8  | 0  | 8  | 8  | 12 | 12 | 8 | 0  | 8  | 8  | 12 |

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 lfst\_idx

| Initialization variable | ctxIdx of lfst_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  |    |    |    |    |    |    |    |    |