|  |  |
| --- | --- |
| **Joint Video Experts Team (JVET)**  **of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11**  18th Meeting: by teleconference, 15–24 April 2020 | Document: JVET-R2001-v8 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **Versatile Video Coding (Draft 9)** | | |
| *Status:* | Output document approved by JVET | | |
| *Purpose:* | Draft text of video coding specification | | |
| *Author(s) or Contact(s):* | Benjamin Bross Jianle Chen Shan Liu Ye-Kui Wang | Email: | firstname.lastname@hhi.fraunhofer.de cjianle@qti.qualcomm.com shanl@tencent.com yekui.wang@bytedance.com |
| *Source:* | Editors | | |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Abstract

This document is Draft 9 of a new ITU-T Recommendation and ISO/IEC International Standard entitled *Versatile Video Coding*. It has been developed by a joint collaborative team of ITU-T and ISO/IEC experts known as the Joint Video Experts Team (JVET), which is a partnership of ITU-T Study Group 16 Question 6 (known as VCEG) and ISO/IEC JTC 1/SC 29/WG 11 (known as MPEG). This draft new standard has been designed with two primary goals. The first of these is to specify a video coding technology with a compression capability that is substantially beyond that of the prior generations of such standards, and the second is for this technology to be highly versatile for effective use in a broadened range of applications. Some key application areas for the use of this standard particularly include ultra-high-definition video (e.g., with 3840×2160 or 7620×4320 picture resolution and bit depth of 10 or 12 bits as specified in Rec. ITU-R BT.2100), video with a high dynamic range and wide colour gamut (e.g., with the perceptual quantization or hybrid log-gamma transfer characteristics specified in Rec. ITU-R BT.2100), and video for immersive media applications such as 360° omnidirectional video projected using a common projection format such as the equirectangular or cubemap projection format, in addition to the applications that have commonly been addressed by prior video coding standards.

Draft 9 of Versatile Video Coding.

Incorporated the following items:

* JVET-R0041-v2: Changes on the definitions of "associated IRAP picture", "associated GDR picture", and "trailing picture" and the constraints regarding different types of pictures and their relationships in terms of decoding order, output order, and prediction relationship.
* JVET-R0267: Add a constraint on RPLs for an IDR picture when sps\_idr\_rpl\_present\_flag is equal to 1.
* JVET-R0065: 1) Specify that GDR AUs shall be complete – i.e., all of the layers in the CVS shall have a picture in the AU (as with IRAP AUs). 2) Add a flag named aud\_irap\_or\_gdr\_au\_flag to the AUD to specify whether the AU is an IRAP or GDR AU, and mandate the presence of an AUD NAL unit in each IRAP or GDR AU when vps\_max\_layers\_minus1 is greater than 0.
* JVET-R0070: The same types of APSs share the same value space for the APS ID, regardless of whether the APSs are prefix or suffix APS NAL units.
* JVET-R0082: Editorial bug fixes, including to use the syntax element alignment\_bit\_equal\_to\_one to indicate the last bit in the arithmetic decode terminate process and other.
* JVET-R0122: When ph\_non\_reference\_picture\_flag is equal to 1, ph\_pic\_output\_flag is not signalled and inferred to be equal to 1.
* JVET-R0108: Remove the dci\_max\_sublayers\_minus1 SE, but to use 4 reserved bits (at the begin of the DCI syntax) instead of having 8 bits for the number of PTL structures (as proposed), and reserve the value 15 of dci\_num\_ptls\_minus1.
* JVET-R0343 item 1.b (JVET-R0156 proposal 3 / JVET-R0170 / JVET-R0222 proposal 2): Condition the presence of sps\_sublayer\_dpb\_params\_flag on the value of sps\_ptl\_dpb\_hrd\_params\_present\_flag, in addition to sps\_max\_sublayer\_minus1.
* JVET-R0343 item 1.g (JVET-R0275): Change the constraint on when the value of sps\_ptl\_dpb\_hrd\_params\_present\_flag shall be equal to 1
* JVET-R0343 item 2 (JVET-R0105): Infer the value of sps\_ccalf\_enabled\_flag to be equal to 0 when not present.
* JVET-R0343 item 5: Changed poc\_msb\_val to ph\_poc\_msb\_cycle\_val and the semantics accordingly, such that it is clear that all “missing” POC MSBs are inferred to be 0.
* JVET-R0343 item 6 (JVET-R0266 proposal 5, JVET-R0178): Specify that no\_gdr\_constraint\_flag equal to 1 specifies that sps\_gdr\_enabled\_flag shall be equal to 0. no\_gdr\_constraint\_flag equal to 0 does not impose such a constraint.
* JVET-R0343 item 7 (JVET-R0332): Relocate/group syntax elements in SPS.
* JVET-R0343 item 8 (JVET-R0068 proposal 6, JVET-R0262 proposal 1 and 2): Require the value of pps\_conformance\_window\_flag to be equal to 0 when the picture width and height are the maximum picture width and height, and infer the values of the PPS conformance window syntax elements to be the same as those signalled in the SPS if the picture width and height are the maximum picture width and height and to be equal to 0 otherwise.
* JVET-R0343 item 11 (JVET-R0162 proposal 1): Change the signalling for wraparound offset. Signal “picture width minus wraparound offset” instead of “wraparound offset”.
* JVET-R0343 item 12 (JVET-R0266 proposal 4): Change the signalling of the PPS ID from ue(v) to u(6).
* JVET-R0343 item 13 (JVET-R0433): Repurpose the chroma scaling list presence flag in the APS (i.e., aps\_chroma\_present\_flag) and use this flag to condition the presence of chroma presence flags in the APS, and move some APS constraints to the PH and SH semantics.
* JVET-R0406 item 1 (JVET-R0118): Add a general constraint flag pic\_header\_in\_slice\_header\_constraint\_flag.
* JVET-R0406 item 2 (JVET-R0113): When pps\_no\_pic\_partition\_flag is equal to 1, skip the 6 PPS flags 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, pps\_qp\_delta\_info\_in\_ph\_flag and infer them to be equal to 0.
* JVET-R0406 item 4 (JVET-R0202): When sh\_picture\_header\_in\_slice\_header\_flag is equal to 1, require pps\_rpl\_info\_in\_ph\_flag, pps\_dbf\_info\_in\_ph\_flag, pps\_sao\_info\_in\_ph\_flag, pps\_wp\_info\_in\_ph\_flag, pps\_qp\_delta\_info\_in\_ph\_flag to be equal to 0.
* JVET-R0406 item 6: Only allow sh\_picture\_header\_in\_slice\_header\_flag to be equal to 1 when pps\_rect\_slice\_flag is 1.
* JVET-R0406 item 8 (JVET-R0210, R0248): Skip the SH SE sh\_num\_tiles\_in\_slice\_minus1 when NumTilesInPic − sh\_slice\_address is not greater than 1.
* JVET-R0406 item 10.b (JVET-R0189, R0202): Add a constraint such that when sps\_subpic\_info\_present\_flag is equal to 1, the value of sh\_picture\_header\_in\_slice\_header\_flag shall be equal to 0.
* JVET-R0406 item 12 (JVET-R0202): Add a constraint such that when sps\_separate\_colour\_plane\_flag is equal to 1, the value of picture\_header\_in\_slice\_header\_flag shall be equal to 0.
* JVET-R0406 item 13 (JVET- R0124, R0163): Fix the text for determination of the first VCL NAL unit of an AU.
* JVET-R0406 item 15 (JVET-R0251): Fix the semantics of the 6 PPS flags 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, pps\_qp\_delta\_info\_in\_ph\_flag.
* JVET-R0410 item 2.a.iii (JVET-R0324): When certain condition is true, skip ph\_collocated\_from\_l0\_flag, ph\_mvd\_l1\_zero\_flag, ph\_bdof\_disabled\_flag, and ph\_dmvr\_disabled\_flag and infer their values.
* JVET-R0410 item 2.c.iii (JVET-R0324): When certain condition is true, skip num\_l1\_weights.
* JVET-R0410 item 4 (JVET-R0278): Require ph\_inter\_slice\_allowed\_flag to be equal to 0 when the PH flag ph\_gdr\_or\_irap\_pic\_flag is equal to 1 and the PH flag 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.
* JVET-R0410 item 5 (JVET-R0112): Change the semantics of ph\_gdr\_or\_irap\_pic\_flag such that the GDR part is "two-way" while keeping the IRAP part as "one-way".
* JVET-R0410 item 9 (JVET-R0251): Rename the syntax elements pic\_sign\_data\_hiding\_enabled\_flag, sps\_bdof\_pic\_present\_flag, sps\_dmvr\_pic\_present\_flag and sps\_prof\_pic\_present\_flag to ph\_sign\_data\_hiding\_enabled\_flag, sps\_bdof\_control\_present\_in\_ph\_flag, sps\_dmvr\_control\_present\_in\_ph\_flag and sps\_prof\_control\_present\_in\_ph\_flag, respectively.
* JVET-R0410 item 10 (JVET-R0165): Revert to making the entry point signalling optional.
* JVET-R0410 item 11 (JVET-R0298): Move the entry point syntax to the end of the slice header, i.e., behind the slice header extension.
* JVET-R0042 (JVET-R0414 items 1 to 4): Changes for clarifying the meaning of a subpicture in a picture with mixed subpicture types, in terms of relative decoding order, output order, and prediction relationship among the subpicture and the same-layer subpictures with the same subpicture index in preceding and succeeding AUs.
* JVET-R0414 item 5 (JVET-R0276): Relax the existing constraint requiring that a subpicture with a different subpicture ID compared to the collocated subpicture in the previous picture in the CLVS needing to have IRAP NUTs.
* JVET-R0414 item 7 (JVET-R0203): Disallow the mix of an IRAP NUT and a leading picture NUT.
* JVET-R0414 item 8 (JVET-R0203): Allow mixing of more than two NUTs within a coded picture.
* JVET-R0414 item 10 (JVET-R0270): On treating a picture with mixed RASL\_NUT and RADL\_NUT in the output of the decoding process and bitstream conformance tests.
* JVET-R0058-v5: Change the constraint on the combination of subpictures and scalability (in the semantics of sps\_subpic\_treated\_as\_pic\_flag[ i ]), the RPR aspects related to the combination of RPR, subpictures, and scalability, and the decoding processes involving the clipping operations for treating subpicture boundaries in motion compensation and motion prediction as picture boundaries.
* JVET-R0184-v2: Allow wraparound MC for subpictures with sps\_subpic\_treated\_as\_pic\_flag[ i ] equal to 1 and subpicture width same as the picture width.
* Fixed the bug [#1028](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/1028).
* JVET-R0193: Signal vps\_max\_tid\_il\_ref\_pics\_plus1 value separately for each direct reference layer of a layer, i.e. vps\_max\_tid\_il\_ref\_pics\_plus1[ i ][ j ] for each direct reference layer j less than i, instead of single vps\_max\_tid\_il\_ref\_pics\_plus1[ i ] as currently.
* JVET-R0046: Change the description of the bitstream extraction process per the value of vps\_max\_tid\_il\_ref\_pics\_plus1[ ][ ] (aspect 1.2 per JVET-R0046-v4). Add a constraint on an ILRP being either an IRAP picture or having TemporalId less than or equal to Max( 0, vps\_max\_tid\_il\_ref\_pics\_plus1[ refPicVpsLayerId ] − 1 ), with refPicVpsLayerId equal to the value of the nuh\_layer\_id of the referenced picture.
* JVET-R0066: Changes related to DPB memory allocation and the derivation of the variable NoOutputOfPriorPicsFlag.
* JVET-R0067: Bug fixes for the derivation of PictureOutputFlag and not to specify a specific picture output behavior for an AU when the picture of the only output layer is not present (due to e.g. loss or layer down-switching) in a normative manner, but rather describe that in a NOTE.
* JVET-R0274: Remove the redundant constraint "There is at least one VCL NAL unit with nuh\_layer\_id equal to each of the nuh\_layer\_id values in LayerIdInOls[ opOlsIdx ] in BitstreamToDecode."
* JVET-R0068 item 1: Specify that sh\_slice\_type shall be equal to 2 (intra slice) when the NAL unit type is an IRAP NAL unit type and the current picture is the first picture in the current AU.
* JVET-R0194: Change the constraints about parameter set sharing to be as follows: A slice in layerA can refer to a parameter set in layerB only when layerB is less than or equal to layerA and all OLSs in the bitstream that contain layerA also contain layerB.
* JVET-R0411 item 1.b.ii (JVET-R0277 item 1): Remove the redundant SH syntax condition for signalling of numbers of active entries in RPLs as the signalling is not needed for I slices.
* JVET-R0411 item 2 (JVET-R0059 item 2): Skip ltrp\_in\_header\_flag[ listIdx ][ rplsIdx ] when ref\_pic\_list\_struct( ) is directly included in the PH or SH instead of in the SPS.
* JVET-R0411 item 5.b (JVET-R0205): Conditionally signal sps\_inter\_layer\_ref\_pics\_present\_flag, based on sps\_video\_parameter\_set\_id greater than 0, and infer it to be equal to 0 when not signalled.
* JVET-R0411 item 6 (JVET-R0255 item 1): Change the reference picture list structure semantics by replacing the parameters ph\_rpl\_idx[ listIdx ] and slice\_rpl\_idx [ listIdx ] with rpl\_idx[ listIdx ].
* JVET-R0411 item 7 (JVET-R0255 item 3): Modify the inference of rpl\_idx [ i ] when not present: if rpl\_sps\_flag[ i ] is equal to 1 and pps\_rpl1\_idx\_present\_flag is equal to 0, the value of rpl\_idx[ 1 ] is inferred to be equal to rpl\_idx[ 0 ], otherwise the value of rpl\_idx[ i ] is inferred to be equal to 0.
* JVET-R0411 item 14.b (JVET-R0277 item 2): When not present, infer sh\_collocated\_from\_l0\_flag to be equal to 1 for P-slices.
* JVET-R0411 item 15 (JVET-R0059 item 5): Modify the existing constraint in the slice header semantics on the collocated picture by only keeping the 0-valued-RprConstraintsActive[ ][ ] aspect.
* JVET-R0411 items 16 and 17 (JVET-R0059 item 4 and R0323 items 1 and 2): Replace the existing constraint on the value of ph\_temporal\_mvp\_enabled\_flag with a NOTE, with the addition of the scaling window offsets to be also the same, and taking in account that when there is no common reference picture existing among all the slices associated with the PH, the value of ph\_temporal\_mvp\_enabled\_flag has be equal to 0.
* JVET-R0266: Code virtual boundary positions using ue(v).
* JVET-R0415 item 1 (JVET-R0071 item 1, R0156 item 4, R0284 item 1): Condition sps\_independent\_subpics\_flag on "sps\_num\_subpics\_minus1 > 0".
* JVET-R0415 item 3 (JVET-R0071): It is suggested for the editor to specify inference of the value 1 for sps\_independent\_subpics\_flag and the value 1 for sps\_subpic\_treated\_as\_pic\_flag[ i ] and the value 0 for sps\_loop\_filter\_across\_subpic\_enabled\_pic\_flag[ i ] when not present.
* JVET-R0415 item 5 (JVET-R0071): It is suggested for the editor to specify inference of the value 1 for pps\_single\_slice\_per\_subpic\_flag when not present.
* JVET-R0415 item 7 (JVET-R0091 item 1): Introduce constraints to ensure that the slice signalling order in the PPS and the slice decoding order are the same.
* JVET-R0415 item 9 (JVET-R0186 item 1, R0088): Move the signalling of no\_pic\_partition\_flag to be earlier than the signalling of pps\_num\_subpics\_minus1. When the value of no\_pic\_partition\_flag is equal to 1, pps\_num\_subpics\_minus1 is not present and inferred to be equal to 0.
* JVET-R0113 item 1: Condition the presence of pps\_loop\_filter\_across\_tiles\_enabled\_flag.
* JVET-R0062: Changes related to uniform tile and rectangular slice partitioning.
* JVET-R0080/R0211: Change the syntax condition for pps\_tile\_idx\_delta\_present\_flag. When the value of pps\_num\_slices\_in\_pic\_minus1 is greater than 1 instead of 0, the syntax element of pps\_tile\_idx\_delta\_present\_flag is signalled.
* JVET-R0188/R0211/R0209: When the first tile of a rectangular slice is one of the tile(s) at the last tile column of the picture, the syntax element pps\_slice\_width\_in\_tiles\_minus1[ i ] is not present and inferred to be equal to 0. When the first tile of a rectangular slice is one of the tile(s) at the last tile row of the picture, the syntax element pps\_slice\_height\_in\_tiles\_minus1[ i ] is not present and inferred to be equal to 0.
* JVET-R0247: Condition the presence of (pps\_)loop\_filter\_across\_slices\_enabled\_flag on the number of slices in a coded picture or subpicture when known, as follows: "if( !rect\_slice\_flag | | pps\_single\_slice\_per\_subpic\_flag | | pps\_num\_slices\_in\_pic\_minus1 > 0 )".
* JVET-R0239: The semantics of pps\_exp\_slice\_height\_in\_ctus\_minus1[ i ] is clarified.
* JVET-R0078/R0095/R0106/R0152/R0172/R0206/R0218/R0232: Skip the signalling of the chroma tc and β deblocking offset syntax elements (SEs) in the PPS when the chroma format is ( 4:0:0 or (4:4:4 and the separate color plane coding mode is in use) ) and/or when the parameter values for chroma are the same as for luma, and condition the SEs on the existing pps\_chroma\_tool\_offsets\_present\_flag currently for controlling the presence of the QP offsets in the PPS. (The editor may also consider renaming the flag.) If the flag is zero, the chroma offsets (if needed) are inferred from the luma offsets.
* JVET-R0078/R0152/R0232: Skip the signalling of the chroma tc and β deblocking offset syntax elements (SEs) in the PH and the SH when the chroma format is (4:0:0 or (4:4:4 and the separate color plane coding mode is in use) ) and/or when the parameter values for chroma are the same for luma, and condition the SEs on the existing pps\_chroma\_tool\_offsets\_present\_flag currently for controlling the presence of the QP offsets in the PPS. (The editor may also consider renaming the luma beta and tc offset control syntax elements.) If the flag is zero, the chroma offsets (if needed) are inferred from the luma offsets.
* JVET-R0159: Fixing basically editorial bugs of the semantics of DBF control related syntax elements. (with editor discretion on exact form of expression).
* JVET-R0106: Move the signalling location of syntax element pps\_dbf\_info\_in\_ph\_flag to locate it near the other deblocking control parameters signalling.
* JVET-R0388: Change the semantics of the deblocking signalling control syntax elements as editorial improvement. Infer sh\_deblocking\_filter\_override\_flag to be equal to 0 when not present. Skip the signalling of ph\_/sh\_deblocking\_filter\_disabled\_flag under certain condition and infer the value of ph/sh\_deblocking\_filter\_disabled\_flag to be equal to 0 under that condition.
* JVET-R0178: Require that when no\_aps\_constraint\_flag is equal to 1, sps\_lmcs\_enabled\_flag and sps\_scaling\_list\_enabled\_flag shall be equal to 0.
* JVET-R0179: Add a no\_tsrc\_constraint\_flag constraining sh\_ts\_residual\_coding\_disabled\_flag to be equal to 1.
* JVET-R0227: Mandate the value of no\_ccalf\_constraint\_flag to be equal to 1 when no\_alf\_constraint\_flag is equal to 1. Mandate the value of no\_sbtmvp\_constraint\_flag to be equal to 1 when no\_temporal\_mvp\_constraint\_flag is equal to 1. Rename no\_qp\_delta\_constraint\_flag to no\_cu\_qp\_delta\_constraint\_flag. Editor action item: The editor is suggested to consider the names of these syntax elements.
* JVET-R0286: Add sensibility constraints on values of general constraint flags without introducing such syntax conditions. Add various constraint flags on tools: no\_mrl\_constraint\_flag, no\_isp\_constraint\_flag, no\_mip\_constraint\_flag, no\_lfnst\_constraint\_flag, no\_mmvd\_constraint\_flag, no\_smvd\_constraint\_flag, no\_prof\_constraint\_flag, no\_palette\_constraint\_flag, no\_act\_constraint\_flag, no\_lmcs\_constraint\_flag. Add single\_layer\_constraint\_flag and no\_inter\_layer\_pred\_constraint\_flag (perhaps renamed to all\_layers\_independent\_constraint\_flag).
* JVET-R0341: Add a no\_chroma\_qp\_offset\_constraint\_flag.
* JVET-R0073: Regarding the CU-level luma QP delta control, resolve an asserted error in the semantics of pps\_cu\_qp\_delta\_enabled\_flag by using it to specify the presence of cu\_qp\_delta\_abs and cu\_qp\_delta\_sign\_flag in both the transform unit syntax and the palette coding syntax. The editor should also consider the suggestion in “b” (Rename the PPS-level on/off control flag to be pps\_cu\_qp\_delta\_enabled\_flag for clearer wordings).
* JVET-R0404 LMCS item 1.c (JVET-R0089/R0098/R0210/R0200/R0202): Skip the signalling of the SH LMCS enabled flag for the case when the PH is in the SH.
* JVET-R0404 LMCS item 1.d (JVET-R0200): Move the SH flag sh\_lmcs\_enabled\_flag to be just after the ALF parameters.
* JVET-R0404 LMCS item 3 (JVET-R0393): Add a NOTE to caution the reader that enabling chroma residual scaling could cause a GDR problem if there is a virtual boundary that is not aligned with a CTU boundary.
* JVET-R0404 LMCS item 4 (JVET-R0051/R0063/R0160/R0210): Revise the semantics of sps\_lmcs\_enabled\_flag, ph\_lmcs\_enabled\_flag, ph\_chroma\_residual\_scale\_flag, and sh\_lmcs\_enabled\_flag. (editor has discretion over exact expression).
* JVET-R0404 LMCS item 4: It is suggested to remove “one, more, or all” phrases in the text.
* JVET-R0404 scaling list item 1.c (JVET-R0089/R0098/R0202): Skip the signalling of the SH explicit scaling list enabled flag for when the PH is in the SH.
* JVET-R0404 scaling list item 2 (JVET-R0200): Move the SH flag sh\_explicit\_scaling\_list\_used\_flag to be just after the ALF parameters.
* JVET-R0404 ALF/SAO item 2 (JVET-R0225): Use two separate flags (one for Cb, one for Cr) to replace ph\_alf\_chroma\_idc in PH and sh\_alf\_chroma\_idc in SH.
* JVET-R0404 ALF/SAO item 4 (JVET-R0232 section 3.2): In PH/SH, add constraints such that if CCALF is disabled in SPS, an ALF\_APS cannot contain any CCALF filters.
* JVET-R0404 ALF/SAO item 5 (JVET-R0068/R0160/R0251): Revise the semantics of sps/ph\_alf\_enabled\_flag and sps\_sao\_enabled\_flag (editor has discretion over exact expression).
* JVET-R0404 APS item 1 (JVET-R0064): Move scaling\_matrix\_for\_lfnst\_disabled\_flag from the scaling\_list\_data( ) syntax to the SPS.
* JVET-R0404 APS item 2 (JVET-R0201): Add the following constraints: 1) To constrain suffix APS NAL units to be located after the last VCL NAL unit of the PU. 2) To allow prefix and suffix APS NAL units with particular APS identifier and type to have different content. 3) To constrain prefix APS NAL unit to be located before the first VCL NAL unit of the PU.
* JVET-R0344 item 1 (JVET-R0161/R0185/R0204/R0275): Omit the signalling of the index to the list of PTL structures for output layer sets when the number of signalled PTL structures is equal to total number of output layer sets and instead infer its value.
* JVET-R0344 item 2.a (JVET-R0099/R0191): Modify the upper limit of vps\_num\_dpb\_params to be to the total number of multi-layer OLSs.
* JVET-R0344 item 3.a (JVET-R0191): Modify the upper limit of vps\_num\_ols\_hrd\_params\_minus1 to be to the total number of multi-layer OLSs minus 1.
* JVET-R0344 item 4.a (JVET-R0099): Don't signal and instead infer the index of the dpb\_parameters( ) syntax structure that applies to the i-th OLS when the total number of multi-layer OLSs is equal to the number of signalled sets of DPB parameters. Also apply to the HRD parameters.
* JVET-R0344 item 5 (JVET-R0099/R0196): Start the for loop which signals vps\_ols\_dpb\_pic\_width[ i ], vps\_ols\_dpb\_pic\_height[ i ], and vps\_ols\_dpb\_params\_idx[ i ] to start at 1 instead of at 0, since the 0-th OLS is single layer.
* JVET-R0344 item 6 (JVET-R0185/R0196/R0275): Replace if( !vps\_all\_independent\_layers\_flag ) condition on vps\_num\_dpb\_params syntax element with if(!vps\_each\_layer\_is\_an\_ols\_flag).
* JVET-R0344 item 6.a (JVET-R0185/R0196/R0275): Change the semantics of vps\_each\_layer\_is\_an\_ols\_flag to be "two-way", and change vps\_num\_dpb\_params to vps\_num\_dpb\_params\_minus1.
* JVET-R0344 item 6.b (JVET-R0185): Signal DPB parameters in the VPS only under the condition "if( !vps\_each\_layer\_is\_an\_ols\_flag )".
* JVET-R0344 item 10 (JVET-R0191): Constrain that each DPB or HRD parameter structure signalled in VPS shall be associated with at least one multi-layer OLS specified by the VPS and each PTL structure that is signalled is associated with at least one OLS specified by the VPS.
* JVET-R0130: Cleanup DF tC value derivation.
* JVET-R0437: Fix DF boundary strength derivation.
* JVET-R0208: Rounding correction for ALF VB.
* JVET-R0233: Reduce CC-ALF line buffer for 4:2:2, 4:4:4.
* JVET-R0330: Remove average luma value clipping.
* JVET-R0380: Disabling scaling matrices for RGB or YCC.
* JVET-R0055: Define chroma lists for 4:0:0 for prediction.
* JVET-R0131: Chroma QT split in 4:2:2.
* JVET-R0347: Upper limits for min QT, max BT/TT size.
* Fixed some bugs in the semantics of pps\_deblocking\_filter\_disabled\_flag and the inferences of vps\_ols\_dpb\_params\_idx[ i ] and vps\_ols\_hrd\_idx[ i ].
* JVET-R0068: Use the subpicture index instead of the subpicture ID in the subpicture sub-bitstream extraction process.
* JVET-R0294/R0092: Bug fix for picture size rewriting in subpicture sub-bitstream extraction process.
* JVET-R0093/R0294: Adding a constraint that disallows subpictures to be located completely outside the conformance window. Adding conformance window rewriting to the subpicture sub-bitstream extraction process.
* JVET-R0294: Allow decoded picture hash SEI in scalable nesting SEI and utilize in subpicture sub-bistream extraction process.
* JVET-R0295 (JVET-R0342 items 16 to 19): Add subpicture bitrate derivation from indicated bitrate in HRD parameters when present in the subpicture level information SEI message. Fix a cbr\_flag related bug and further minor fixes (#903) and editorial improvements in subpicture sub-bitstream extraction process.
* JVET-R0350: Chroma MIP for 4:4:4 single tree.
* JVET-R0333: Fix palette prediction mismatch.
* JVET-R0334: Disable chroma palette for local dual tree.
* JVET-R0045: Minimum QP of transform skip signalling.
* JVET-R0311: Fix cu\_skip\_flag signaling for IBC.
* JVET-R0319: Interaction between LFNST and BDPCM.
* JVET-R0329: Fix clipping input residuals to IACT.
* Fixed the bug [#1032](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/1032).
* Fixed some bugs, including a typo in the NOTE in the semantics of ph\_chroma\_residual\_scale\_flag.
* JVET-R0370: Add Main 10 Still Picture and Main 4:4:4 10 Still Picture profiles.
* Review of editor's notes: Slightly changed the semantics of general\_sub\_profile\_idc[ i ], and changed the coding of pt\_display\_elemental\_periods\_minus1 from ue(v) to u(4).
* JVET-R0244 & R0245: On level definitions
  + MaxCPB = 80 000 for level 6, 120 000 for level 6.1, 180 000 for level 6.2, and change MinCrScaleFactor for the 4:4:4 profile to 0.75, and change MinCrBase to 8 for level 6.2.
  + Adopt the level value scheme of major \* 16 + minor \* 3 (with the top number retaining its special meaning). The basic idea is to leave the same amount of gap between sublevels but reduce the gap for major levels to enable hypothetical future higher level numbers.
  + Editor action item: It is suggested to put the number correspondence table in the text, along with an informative note that describes the formula and states that future-specified levels could have values selected in a different manner that respects the specified hierarchy.
* JVET-R0221: The editor is asked to ensure that the text adequately expresses the necessary constraints, such that tiles, slices, and subpictures are a proper partitioning of the picture (no overlaps, no gaps, no CTUs that are outside the picture).
* JVET-R0157 item 1: The editor was asked to check and make sure that the constraints related to item 1 of JVET-R0157 (which we believe are already expressed in some form) are sufficiently clear to the reader, i.e., when pps\_mixed\_nalu\_types\_in\_pic\_flag is equal to 1, pps\_no\_pic\_partition\_flag shall be equal to 0 and pps\_rect\_slice\_flag shall be equal to 1.
* JVET-R0344 item 14 (JVET-R0107 Proposal 2): Removing checks that are unnecessary for signalling of vps\_ptl\_max\_temporal\_id[ i ], vps\_dpb\_max\_temporal\_id[ i ], and hrd\_max\_tid[ i ], and editorially simplify the inference rules.
* JVET-R0344 items 15 & 16 (JVET-R0119 items 1 and 2): Change the inferred value of vps\_max\_tid\_il\_ref\_pics\_plus1[ ][ ] when not present from 7 to vps\_max\_sublayers\_minus1 + 1, and, when confirmed not needed and editorially undesirable by the editor, skip the derivation of the NumSubLayersInLayerInOLS[ ][ ] and layerIncludedInOlsFlag[ ][ ] values when vps\_all\_independent\_layers\_flag is equal to 1.
* JVET-R0344 items 20 & 21 (JVET-R0107 proposal 3, JVET-R0296 aspect 1): Change the semantics of vps\_max\_tid\_il\_ref\_pics\_plus1[ ][ ] for the special value 0, define the semantics for special value 0 to include GDR pictures with ph\_recovery\_poc\_cnt equal to 0, and modify the sub-bitstream extraction process to account for GDR pictures with ph\_recovery\_poc\_cnt equal to 0.
* JVET-R0344 item 22 (JVET-R0296 aspect 2): Change the derivation of NumSubLayersInLayerInOLS by separating the cases for vps\_each\_layer\_is\_an\_ols\_flag equal to 1 and vps\_ols\_mode\_idc equal to 0. Note that this was covered in R0193 that was integrated in R2001-v2.
* JVET-R0344 item 29 (JVET-R0158): For the AUD, the value of nuh\_layer\_id should not be constrained (as with DCI, VPS and EOB).
* JVET-R0344 item 30.b (JVET-R0222 aspect 1): Infer vps\_max\_sublayers\_minus1 to be equal to 6 when sps\_video\_parameter\_set\_id is equal to 0 (i.e. VPS is not present).
* JVET-R0342 item 2 (JVET-R0100 Proposal 1): Not signal and infer dui\_sublayer\_delays\_present\_flag[ bp\_max\_sublayers\_minus1 ] to be 1, to make sure du\_spt\_cpb\_removal\_delay\_increment[ bp\_max\_sublayers\_minus1 ], which is used for inference of other syntax, is always signalled in the DUI SEI message.
* JVET-R0342 item 4 (JVET-R0101 Proposal 1): Add missing inference rules for cpb\_alt\_initial\_cpb\_removal\_delay\_delta[ i ][ j ] and cpb\_alt\_initial\_cpb\_removal\_offset\_delta[ i ][ j ].
* JVET-R0342 item 5 (JVET-R0101 Proposal 2): Fix bugs related to unspecified or missing length of u(v) coded syntax elements.
* JVET-R0342 item 6 (JVET-R0103 Proposal 1): Not signal du\_common\_cpb\_removal\_delay\_flag, du\_common\_cpb\_removal\_delay\_increment\_minus1[ i ], num\_nalus\_in\_du\_minus1[ i ] and du\_cpb\_removal\_delay\_increment\_minus1[ i ][ j ] in picture timing SEI message when there is only one DU in an AU and condition the duCpbRemovalDelayInc variable derivation.
* JVET-R0342 item 7 (JVET-R0103 Proposal 2): Eeditorial fixes for the CPB operations for correct use of AU and DU related variable names.
* JVET-R0342 items 8 to 13 (JVET-R0297):
  + Apply an asserted editorial clarification of what n0…n5 mean for the number of conformance tests.
  + Fix the inference rule for syntax elements in the syntax structure sublayer\_hrd\_parameters( ) to use syntax from SPS or VPS as appropriate instead of always using sps\_max\_sublayers\_minus1.
  + Fix n3 specifying the number of conformance tests for IRAPs that are not CRAs with associated RASL pictures and alternative timing.
  + Fix the derivation of InitialCpbRemovalDelay in C.2.3 (Timing of DU removal and decoding of DU) aligning it with derivation in C.2.2 (Timing of DU arrival) to account for alternative timing.
  + Add missing text to update values of CpbDelayOffset and DpbDelayOffset in C.2.3 (Timing of DU removal and decoding of DU).
  + Fix equation C.11 for derivation of NominalRemovalTime[ n ], to account for CpbDelayOffset.
* JVET-R0342 item 14 (JVET-R0413): Signal a separate set of alternative buffering delay parameters for VCL HRD and for NAL HRD and use them for HRD operation.
* JVET-R0342 item 15 (JVET-R0264): Add a constraint that when vps\_max\_tid\_il\_ref\_pics\_plus1[ ][ ] is equal to 0 it would be prohibited to have mixtures of IRAP and non-IRAP NAL units in the picture.
* JVET-R0342 item 1 (JVET-R0094): Signal a fixed DPB output time offset for each temporal sublayer, controlled by a presence flag, within the buffering period SEI message and use these offsets to calculate picDpbOutputDelta[ i ].
* JVET-R0090: Move the two flags about GeneralProgressiveSourceFlag and GeneralInterlacedSourceFlag into the VUI specification.
* JVET-R0137: Define colPic and NoBackwardPredFlag.
* JVET-R0223: Cleanup of DMVR (not used with RPR).
* JVET-R0134: Align deblocking of subblock motion edges to software.
* JVET-R0271 variant #3: The picture header flags ph\_dep\_quant\_enabled\_flag and ph\_sign\_data\_hiding\_enabled\_flag are moved to slice header before sh\_ts\_residual\_coding\_disabled\_flag and the signalling of sh\_ts\_residual\_coding\_disabled\_flag are conditioned on these flags.
* JVET-R0483 Combination 4: Condition sh\_ts\_residual\_coding\_disabled\_flag on "if( sps\_transform\_skip\_enabled\_flag && !sh\_dep\_quant\_enabled\_flag && !sh\_sign\_data\_hiding\_enabled\_flag )".
* JVET-R0068: Change the semantics of sps\_affine\_amvr\_enabled\_flag equal to 1 to use the wording of "may be used" instead of "is used". Adopt (clarify that this has "one way" semantics).
* JVET-R0371: The range of sps\_five\_minus\_max\_num\_subblock\_merge\_cand is set equal to 0 to 5 − sps\_sbtmvp\_enabled\_flag, inclusive, and remove the inference rule.
* JVET-R0214: All aspects with change of flag rather than additional flag.
  + Move the location of sps\_mmvd\_fullpel\_only\_flag to directly follow sps\_mmvd\_enabled\_flag.
  + Infer the value of sps\_mmvd\_fullpel\_only\_flag to be equal to 0 when the flag is not present.
  + Replace/rename the current flag no\_fpel\_mmvd\_constraint\_flag to no\_mmvd\_constraint\_flag and change the semantics accordingly.
* JVET-R0097: If the luma CTB size is not larger than 32, sps\_max\_luma\_transform\_size\_64\_flag is not signalled and inferred to be 0.
* JVET-R0252: semantics cleanup aspect.
* Made some style consistency changes, particularly on the semantics of SPS/PPS/PH/SH enable/disable flags, including the checking of the use of the word "may" in these semantics.
* JVET-R0249: Replace 'slice\_' prefix for slice header syntax elements by 'sh\_', except for slice\_address and slice\_type which are proposed to be renamed to sh\_slice\_address and sh\_slice\_type, respectively. Rename syntax elements in VPS, SPS, PPS, PH, SH to ensure that the names of all syntax elements in these places start with 'vps\_', 'sps\_', 'pps\_', 'ph\_' and 'sh\_', respectively. The editor is asked to consider renaming ph\_disable\_xxx\_flag to ph\_xxx\_disabled\_flag.
* Renamed sps\_fpel\_mmvd\_enabled\_flag and ph\_fpel\_mmvd\_enabled\_flag to sps\_mmvd\_fullpel\_only\_flag and sps\_mmvd\_fullpel\_only\_flag, respectively.
* Fixed the bugs [#945](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/945), [#948](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/948), [#949](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/949), [#953](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/953), [#957](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/957), [#960](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/960), [#965](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/965), [#970](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/970), [#971](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/971), [#972](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/972), [#975](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/975) (partially), [#999](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/999), [#1019](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/1019), [#1033](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/1033), [#1036](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/1036).
* Fixed a bug in Eqn. 30, for the case when sps\_subpic\_info\_present\_flag is equal to 0 and pps\_single\_slice\_per\_subpic\_flag is equal to 1, related to the reference software bug ticket [#1005](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/1005).
* Integrated the following suggestions from Virginie Drugeon:
  + In step 1 in clause C.1, change "…the sub-bitstream that is the output …" to "…the sub-bitstream BitstreamToDecode that is the output …".
  + At the beginning of clause D.5.2, it is clarified that the syntax elements bp\_decoding\_unit\_hrd\_params\_present\_flag, du\_cpb\_params\_in\_pic\_timing\_sei\_flag and dpb\_output\_delay\_du\_length\_minus1 are found in the PB SEI message. Add the syntax element du\_dpb\_params\_in\_pic\_timing\_sei\_flag to that list.
* Fixed the bugs reported in [#964](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/964), [#1037](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/1037), [#1038](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/1038), and [#1039](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/1039).
* In clause C.6, for easier understanding, changed "nal\_unit\_type is not equal to IDR\_W\_RADL, IDR\_N\_LP, CRA\_NUT, or GDR\_NUT, or nal\_unit\_type is equal to GDR\_NUT and the associated ph\_recovery\_poc\_cnt is not equal to 0" to "nal\_unit\_type is equal to TRAIL\_NUT, STSA\_NUT, RADL\_NUT, or RASL\_NUT, or nal\_unit\_type is equal to GDR\_NUT and the associated ph\_recovery\_poc\_cnt is not equal to 0".
* To complete the integration of JVET-R0091, added a constraint to require that the PPS signalling order of rectangular slices belonging to different subpictures to be in increasing order of subpictures indices, to make sure that the signalling order and decoding order of all rectangular slices are the same.

Draft 8 of Versatile Video Coding.

Ed. Notes:

* Fixed the bugs [#672](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/672), [#700](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/700), [#711](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/711), [#715](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/715), [#716](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/716), [#720](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/720), [#729](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/729), [#737](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/737), [#739](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/739).
* Fixed subpicture figure together with accompanying text ([#740](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/740)).
* Removed the editing note comment in the semantics of pps\_tile\_idx\_delta\_val[ i ], which was concluded not needed after a study (input: Hendry).
* Fixed quite some format and style consistency issues in syntax tables, equations, etc. as well as some bugs in those places.
* Made some fixes in the semantics of DCI and VPS.
* Fixed the semantics of sps\_loop\_filter\_across\_virtual\_boundaries\_disabled\_present\_flag and ph\_loop\_filter\_across\_virtual\_boundaries\_disabled\_present\_flag, and changed the syntax element names to sps\_virtual\_boundaries\_present\_flag and ph\_virtual\_boundaries\_present\_flag.
* Fixed the following issues:
  + The definition of the sub-bitstream extraction process still referred to a target LayerId: "A specified process by which NAL units in a bitstream that do not belong to a target set, determined by a target OLS index and a target highest TemporalId and a target LayerId, are removed from the bitstream, …". However, the process in clause C.6 only mentions a target OLS index and a target highest TemporalId in the inputs, but no target LayerId.
  + In the VPS syntax: "ols\_hrd\_parameters( firstSubLayer, hrd\_max\_temporal\_id[ i ] )". hrd\_max\_temporal\_id is not defined anywhere. It should be "hrd\_max\_tid" instead.
  + In the VPS semantics, when deriving LayerUsedAsRefLayerFlag, it was initialised as follows: "LayerUsedAsRefLayerFlag[ j ] = 0." But j is not defined at this point, since it gets defined in the following loop.
  + Fixed a bug in Equation 40 by changing "layerIncludedFlag[ i ][ j ] = 1" to "layerIncludedFlag[ i ][ k ] = 1" and then changed the variable name to layerIncludedInOlsFlag[ ][ ] to be more intuitive.
  + In the semantics of dpb\_size\_only\_flag, only the value 1 was specified, twice, and in a contradicting way. The second sentence should be: "dpb\_size\_only\_flag[ i ] equal to 0 specifies that…".
  + In the semantics of vps\_num\_ols\_hrd\_params\_minus1: "When TotalNumOlss is greater than 1, the value of vps\_num\_ols\_hrd\_params\_minus1 is inferred to be equal to 0". However, vps\_num\_ols\_hrd\_params\_minus1 is signalled when TotalNumOlss is greater than 1. It should be "When TotalNumOlss is equal to 1, the value of vps\_num\_ols\_hrd\_params\_minus1 is inferred to be equal to 0".
  + In the semantics of hrd\_max\_tid, there should be an inference for this syntax element when the value of vps\_max\_sublayers\_minus1 is greater than 0, but vps\_all\_layers\_same\_num\_sublayers\_flag is 1, similarly as in the semantics of vps\_ptl\_max\_temporal\_id[ i ] and vps\_dpb\_max\_temporal\_id[ i ].
  + In the semantics of sps\_ptl\_dpb\_hrd\_params\_present\_flag: "The value of sps\_ptl\_dpb\_hrd\_params\_present\_flag shall be equal to vps\_independent\_layer\_flag[ nuh\_layer\_id ]." Everywhere else vps\_independent\_layer\_flag is indexed by the layer index rather than the layer identifier. It should be the following instead: "The value of sps\_ptl\_dpb\_hrd\_params\_present\_flag shall be equal to vps\_independent\_layer\_flag[ GeneralLayerIdx[ nuh\_layer\_id ] ]."
  + Annex A still mentioned bit\_depth\_luma\_minus8 and bit\_depth\_chroma\_minus8, although these two syntax elements have been replaced by sps\_bit\_depth\_minus8.
  + Annex A still mentioned num\_tile\_columns\_minus1 and num\_tile\_rows\_minus1, although these syntax elements do not exist anymore.
  + The list NestingLayerId[ ] and the variable NestingNumLayers specified in the semantics of the scalable nesting SEI message are not used in other places, thus two variable names were changed to be local variables, i.e., nestingLayerId[ ] and nestingNumLayers.
* Fixed the bugs [#600](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/600), [#713](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/713) and some typos.
* Fixed the bugs [#604](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/604), [#613](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/613), [#710](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/710), [#717](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/717), [#723](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/723) and some typos.
* Fixed the bugs [#742](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/742), [#751](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/751) and some other editorial bugs.
* Changed numrous occurences of recPictureL[ hx, vy ] etc. to recPicture[ hx ][ vy ] etc. (removal of 'L' and to be in the form of two-dimention array; input: Yang Wang).
* Fixed the bugs [#765](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/765), [#709](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/709), [#306](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/306), [#567](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/567), [#791](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/791).
* Incorporated JVET-Q0819 on PH and SH changes, including text for the following adoption items:
  + JVET-Q0482: Remove the mechanism for signalling of constant SH parameter values.
  + JVET-Q0259 aspect 5 (indicate information about collocated picture in picture header when RPL information is signalled in the PH)
  + JVET-Q0781: A two-flag approach. One flag that indicates the presence of intra parameters and one that indicates the presence of inter parameters. When the first flag is 0, no inter slices can be present. When the second flag is 0, no intra slices can be present. If the first flag is 0, the second flag is not present and inferred to be equal to 1. When the first flag is 0, the SH would not contain a sh\_slice\_type.
  + JVET-Q0414 Option 2: constrain the value of ph\_gdr\_pic\_flag based on sps\_gdr\_enabled\_flag.
  + JVET-Q0154 option 1: irap\_or\_gdr\_pic\_flag in the beginning of PH and use it to condition the presence of ph\_gdr\_pic\_flag in the PH. The semantics should be a “one way” indication, not a guarantee that the picture is not an IRAP or GDR.
  + JVET-Q0115: Move POC LSB signalling from SH to PH.
  + JVET-Q0155 aspect 1: Move colour\_plane\_id to SH.
  + JVET-Q0775: A mechnism to allow picture header in slice header. Adopt. Additionally, it was agreed to condition the ph\_no\_output\_of\_prior\_pics\_flag on the indication that the picture is a GDR or IRAP picture and to put ph\_pic\_parameter\_set\_id before ph\_pic\_order\_cnt\_lsb, which would be followed by ph\_recovery\_poc\_cnt. If there is something that can be sent either in the PH NAL unit or in the SH, it shall be in the PH structure when the PH is in the SH. It was agreed that we will keep the PH syntax structure the same in the PH NAL unit and the SH.
  + JVET-Q0200: The flags that specify whether a syntax element of a related coding tool is present in PH or in SH (but not both) are moved from the PH to the PPS. This includes reference picture lists, SAO, ALF, and deblocking.
  + JVET-Q0247: Make the prediction weight table a fifth type of data that can be signalled either in the PH or SH (like ALF, deblocking, RPL, and SAO).
  + JVET-Q0270: Add a PPS flag to determine whether qp delta is sent in the PH or SH, like other things (e.g., ALF, deblocking, SAO).
  + JVET-Q0358: Add a TemporalId constraint between the ALF\_APS NAL unit and the picture associated with PH.
* Made various fixes, including removal of unnessary brackets and merging of some syntax conditions.
* Incorporated JVET-Q0500 on CCLM predictors derivation cleanup.
* Incorporated JVET-Q0194 on CTU row boundary location fix in CCLM.
* Incorporated JVET-Q0275 on bug fixes for CCLM filtering.
* Fixed the bug [#606](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/606) in CCLM derivation process for neighbouring block availability.
* Incorporated JVET-Q0468: Limit the range of min CU size in SPS to 4,8,16,32,64. This is an editorial cleanup of semantics. The subsequent clipping operation min (64, ctusize) shall be kept.
* Incorporated JVET-Q0471: When chroma sampling is 4:2:2, the QT split limit of chroma block would be rectangular (larger in height than in width). The proposed solution is to scale the limit check by multiplying the subsampling ratio factor.
* Incorporated JVET-Q0330: Bug fix in implicit split at a picture boundary.
* Incorporated JVET-Q0267: Resetting CuQpOffsets to 0 at the start of each QG.
* Incorporated JVET-Q0505: Quantization matrix signalling for 4:0:0.
* Incorporated JVET-Q0089: TSRC slice-level switch and BDPCM for chroma in 4:2:0.
* Incorporated JVET-Q0368: Bug fix on MVP and refIdx context models.
* Incorporated JVET-Q0209: Align chroma QP offset between palette and TU signalling.
* Incorporated JVET-Q0293: Removal of chroma N×2 blocks in PDPC.
* Incorporated JVET-Q0446: MIP with constant shifts and offsets.
* Incorporated JVET-Q0371: Bug fix in MIP intra reference sample generation.
* Fixed minor issues and the bugs reported in [#687](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/687), [#761](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/761), [#762](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/762), [#797](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/797).
* Incorporated JVET-Q0044: Align the picture-level slice index and the subpicture-level slice index to be in the same order which also corresponds to the decoding order.
* Incorporated JVET-Q0113: Editorial changes regarding the specification of NAL unit decoding order.
* Incorporated JVET-Q0119: Modifications to the coding and semantics of sps\_num\_subpics\_minus1.
* Incorporated JVET-Q0169: The maximum value of subpicture ID derived by the signalled subpicture ID length shall be greater than or equal to the number of subpicture IDs, and for each signalled subpicture ID, it shall be different from all other subpicture IDs in the same picture.
* Incorporated JVET-Q0271: Add sps\_independent\_subpic\_flag.
* Incorporated JVET-Q0816: Omit the signalling of subpicture layout when only one subpicture is present in the picture.
* Incorporated JVET-Q0817: Remove constraint on pps\_single\_slice\_per\_subpic\_flag when sps\_subpic\_info\_present\_flag is 0.
* Incorporated JVET-Q0222: Skip the signalling of the top-left position of the first subpicture and the size of the last subpicture.
* Incorporated JVET-Q0787: Condition the presence of the syntax elements for signalling of subpictures on their length being non-zero.
* Incorporated JVET-Q0157: Replace the current picture size with the reference picture size in the clipping (motion padding).
* Incorporated JVET-Q0119: Condition sps\_subpic\_id\_mapping\_explicitly\_signalled\_flag on "if( sps\_subpic\_info\_present\_flag )" and infer the value of sps\_subpic\_id\_mapping\_explicitly\_signalled\_flag to be equal to 0 when it is not present.
* Incorporated JVET-Q0119: Remove signalling of the subpicture ID mapping from PH.
* Incorporated JVET-Q0119: Require signalling in the SPS or PPS when sps\_subpic\_id\_mapping\_explicitly\_signalled\_flag is equal to 1.
* Incorporated JVET-Q0119: When sps\_subpic\_info\_present\_flag is equal to 1, the subpicture ID length is explicitly signalled in the SPS regardless of whether subpicture IDs are explicitly signalled, and the length of sh\_subpic\_id is always specified by that length.
* Incorporated JVET-Q0157: Add the following constraint: "When the current picture is not the first picture of the CLVS, for each value of i in the range of 0 to sps\_num\_subpics\_minus1, inclusive, if the value of SubpicId[ i ] is not equal to the value of SubpicId[ i ] of previous picture in decoding order in the same layer, the nal\_unit\_type for all coded slice NAL units of the subpicture with subpicture index i shall be in the range of IDR\_W\_RADL to CRA\_NUT, inclusive."
* Incorporated JVET-Q0246: Add a flag in the SPS for Virtual Boundary enable flag. When the flag is equal to 1, VB signalling may be present in either SPS or PH, but not in both.
* Incorporated JVET-Q0210: Constrain that when subpicture signalling is present, the signalling of virtual boundary position, if present, shall be in SPS.
* Incorporated JVET-Q0417: Constrain that when RPR is enabled, the signalling of virtual boundary position, if present, shall be in PH.
* Incorporated JVET-Q0359: Provide value range for syntax elements pps\_tile\_column\_width\_minus1 and pps\_tile\_column\_width\_minus1.
* Incorporated JVET-Q0218: Condition the presence of pps\_tile\_idx\_delta\_present\_flag on the value of pps\_num\_slices\_in\_pic\_minus1.
* Incorporated JVET-Q0218: Add the default value for pps\_single\_slice\_per\_subpic\_flag when it is not present.
* Incorporated JVET-Q0244: Signal rectangular slice width in tile only when number of tile column is greater than 1; likewise, signal rectangular slice height only when the number of tile row is greater than 1.
* Incorporated JVET-Q0480: Only specify pps\_slice\_height\_in\_tiles\_minus1 syntax element at the start of each rectangular slice row.
* Incorporated JVET-Q0289: Editorial update to clarify the semantics of pps\_tile\_idx\_delta\_val[ i ].
* Incorporated JVET-Q0289: If NumTilesInPic is equal to 1, don’t signal the pps\_rect\_slice\_flag and infer its value to be 1.
* Incorporated JVET-Q0203: Only signal explicit number of slice and explicit slice height for slices that are within a tile.
* Incorporated JVET-Q0244: Condition the signalling of syntax elements for rectangular slices that are less than one tile only when tile row height is greater than 1.
* Incorporated JVET-Q0119: Condition the presence of the sh\_slice\_address on NumSlicesInSubpic[ CurrSubpicIdx ] being greater than 1 and update the semantics, including the inference of the value of sh\_slice\_address accordingly.
* Incorporated JVET-Q0055: Zero-out dependent MTS index signalling.
* Incorporated JVET-Q0516: MTS signalling based on last significant coefficient position.
* Incorporated JVET-Q0784: LFNST signalling, latency reduction and scaling process.
* Incorporated JVET-Q0183: High-level syntax related to transform skip mode.
* Incorporated JVET-Q0820: ACT common text for bug fixes and transform change.
* Fixed the bugs [#601](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/601), [#685](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/685), [#702](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/702), [#725](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/725), [#768](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/768), [#769](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/769), [#789](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/789), [#805](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/805), [#810](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/810).
* Incorporated JVET-Q0435: Bug fix in palette coding syntax structure.
* Incorporated JVET-Q0629: Palette mode excluding small (4×4) blocks.
* Incorporated JVET-Q0504: Palette mode for non-4:4:4 colour format in 4:4:4 profile.
* Incorporated JVET-Q0501: Palette predictor initialization in WPP.
* Incorporated JVET-Q0491: Palette escape binarization.
* Incorporated JVET-Q0291: Reduce palette size for dual tree.
* Incorporated JVET-Q0210: Remove uek(v).
* Incorporated JVET-Q0147: Conditional signalling of sps\_joint\_cbcr\_enabled\_flag.
* Incorporated JVET-Q0265: Conditional signalling of sps\_act\_enabled\_flag.
* Incorporated JVET-Q0152: Move sps\_seq\_parameter\_set\_id to beginning of the SPS.
* Incorporated JVET-Q0420: Conditional signalling of chroma QP-related elements.
* Incorporated JVET-Q0043: Disallow for both sps\_subpic\_info\_present\_flag and sps\_res\_change\_in\_clvs\_allowed\_flag to be equal to 1.
* Incorporated JVET-Q0764: Ref wraparound signalling modifications.
* Incorporated JVET-Q0117: Parmeter sets cleanups.
* Incorporated JVET-Q0280: SPS constraint on VPS id.
* Moved pps\_mixed\_nalu\_types\_in\_pic\_flag location in the PPS.
* Incorporated JVET-Q0399: Inference of scaling window parameters.
* Incorporated JVET-Q0416: Wraparound offset range.
* Move filler data SEI message to SEI specification.
* Incorporated JVET-Q0818: Move frame-field SEI message to SEI spec.
* Incorporated JVET-Q0172: Restrict chroma format and bit depth for inter-layer referencing.
* Incorporated JVET-Q0488: Limit of 4 repetitions of most SEI messages.
* Incorporated JVET-Q0128: Disable DMVR and BDOF when either luma WP or chroma WP is applicable to a coding unit.
* Incorporated JVET-Q0297: Merge estimation regions.
* Incorporated JVET-Q0483: Add clipping to restored MV.
* Incorporated JVET-Q0449: Disable smoothing filter in RPR.
* Incorporated JVET-Q0517: RPR down-sampling filters for affine mode.
* Incorporated JVET-Q0806: Geometric partitioning for merge mode.
* Incorporated JVET-Q0242: Cleanup for CIIP and Geo for 4:0:0 chroma format.
* Fixed the bugs [#723](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/723), [#802](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/802), [#808](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/808), [#811](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/811).
* Incorporated JVET-Q0441: SAO Modification for 12-bit.
* Incorporated JVET-Q0495: Clip ranges for NL-ALF.
* Incorporated JVET-Q0150: Fix for ALF virtual boundary processing.
* Incorporated JVET-Q0249: Clipping flag clean-up for chroma.
* Incorporated JVET-Q0795: Cross-component adaptive loop filter.
* Incorporated JVET-Q0054: Bug fix on long-tap deblocking filter.
* Incorporated cleanups and fixes including the ones reported in [#698](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/698), [#746](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/746), [#757](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/757), [#798](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/798), [#809](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/809), [#815](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/815), [#816](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/816), [#817](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/817).
* Incorporated JVET-Q0786:
  + Repeat PTL info of OLSs containing only one layer in the VPS (in addition to signalling them in the SPS).
  + Not to repeat HRD parameters info of OLSs containing only one layer in the VPS (in addition to signalling them in the SPS).
* Incorporated JVET-Q0118:
  + Require the sh\_slice\_type to be 2, not only for the first picture in an AU that is an IRAP picture, but also for IRAP pictures in independent layers.
  + Specify the variable LayerUsedAsOutputLayerFlag[ i ] and use it in the constraint for expressing that there shall be no layer that is both not used for reference and not an output layer in any OLS.
* Incorporated JVET-Q0814: DPB operations based on the conceptual decoded picture size in the DPB is the largest value of max\_width \* max\_height among all the SPSs in the OLS that is signalled in VPS.
* Incorporated JVET-Q0151/Q0205: Mandate the tile offsets signalling, and move entropy\_coding\_sync\_enabled\_flag and entry\_point\_offsets\_present\_flag from the PPS to the SPS.
* Incorporated JVET-Q0256:
  + It was agreed that the phrase “sublayer representation” should be removed from the language as proposed. Also “identified by a specific value of general\_level\_idc” (and similar tier language) should be removed from the description of a decoder.
  + Add the following statements and rephrase the wording of level accordingly:
    - Decoders are not required to extract a subset of the bitstream; any such extraction process that may be a part of the system is considered to be outside of the scope of the decoding process.
    - The profile of the bitstream is indicated by general\_profile\_idc.
    - The level of the bitstream is indicated by Min( general\_level\_idc, sublayer\_level\_idc[ Htid ].
    - The values of Htid and sublayer\_level\_idc[ Htid ] are not necessary for the operation of the decoding process, and if it is desirable for the decoder to check the conformance of the bitstream, these may be provided by external means.
* Incorporated JVET-Q0112:
  + The first item (using the global maximum picture size to determine the DPB size).
  + The first part of the third item (using the global maximum picture size in computing limits instead of current picture size).
  + The fourth item on using the cumulative worst-case picture size for all pictures in an AU to derive constraints on CPB removal time, etc.
* Incorporated JVET-Q0048: Mandate a bitstream order for BP/PT/DUI SEI message that obeys the parsing dependency.
* Incorporate JVET-Q0216: Not signal and infer cpb\_removal\_delay\_delta\_idx[ i ] when there is only one cpb\_removal\_delay\_delta[ i ] in the BP SEI message.
* Incorporate JVET-Q0219: Signal the alternative times in the PT SEI message per sub-layer.
* Incorporate JVET-Q0394: Non-scalale nested SEI messages apply to bitstreamToDecode, extraction propcess handling scalable-nested SEI messages, add flag to HRD parameter syntax structure to indicate that non-scalable-nested PT SEI messages apply to all OLS in the bitstream.
* Incorporate JVET-Q0221: Add a presence flag to BP SEI message about DPB output delay in PT/DUI SEI message.
* Incorporate JVET-Q0397: Subpicture HRD conformance testing with subpicture extraction process and scalable-nesting HRD timing information.
* Incorporate JVET-Q0404: Add a CBR flag to the subpicture level info SEI message, and change the semantics and the extraction process based on the assocation of VCL and non-VCL NAL units.
* Incorporate JVET-Q0406: CABAC zero word constraint for subpictures.
* Incorporate JVET-Q0630: Resolve parsing dependency of subpicture level SEI message on SPS.
* Incorporate JVET-Q0395: Add constraints for Bitrate and number of tiles to subpicture level info SEI message.
* Incorporate JVET-Q0443: Add MinCR constraint for subpictures, and fix identified bugs in the subpicture level SEI message text, such as incorrect array indices.
* Incorporated JVET-Q0257: Impose scaling ratio constraint after scaling window offset is applied.
* Incorporated JVET-Q0179: Add scaling ratio constraint adressing peak memory bandwith concerns.
* Incorporated JVET-Q0487: aspect 1 - clarify TMVP constraint in sh\_collocated\_ref\_idx semantics.
* Incorporated JVET-Q0487: aspect 3 - modify RefPicIsScaled derivation and rename to RprConstraintsActive to account for 1:1 scaling.
* Incorporated JVET-Q0487: aspect 4 - Signal scaling window in units of chroma samples
* Incorporated JVET-Q0262: Rename sps\_ref\_pic\_resampling\_enabled\_flag to sps\_res\_change\_in\_clvs\_allowed\_flag.
* Incorporated JVET-Q0260: Add conformance window signalling to SPS.
* Incorporated JVET-Q0751: Including the following agreed aspects:
  + Allow mixing of leading with trailing NAL unit types and mixing of RASL with RADL NAL unit types.
  + Not including STSA in the list that can be mixed with IRAP.
  + Allow RASL / RADL to be mixed with STSA.
  + Require that all VCL NAL units within one subpicture have the same NAL unit type.
  + Adding a NOTE expressing that the value of sps\_idr\_rpl\_present\_flag must be equal to 1 in the mixed NAL unit types case.
* Incorporated JVET-Q0217: RPL cleanups.
* Fixed some bugs and made some other editorial changes.
* Incorporated JVET-Q0379: Move the aps\_id related syntax elements in the PH and SH. For the PH, these would immediately follow the POC information. For the SH, it would immediately follow sh\_slice\_type.
* Incorporated JVET-Q0121:
  + Extend abs range of beta offset and tc offset from 6 to 12, PPS and SH as proposed, but also in PH.
  + To have separate deblocking parameters for the two chroma components.
* Incorporated JVET-Q0114:
  + Remove frame\_only\_constraint\_flag, which is redundant to general\_frame\_only\_constraint\_flag.
  + Add a general constraint on no tile partitioning (i.e., one tile only for each picture).
  + Add a general constraint on no slice partitioning (i.e., one slice only for each picture).
  + Add a general constraint on no subpicture partitioning (i.e., one subpicture only for each picture).
  + Add a general constraint on no reference picture resampling.
  + Add general\_non\_projected\_constraint\_flag, the value equal to 1 specifies that there is no omnidirectional projection SEI messages, and change the semantics of general\_non\_packed\_constraint\_flag to cover only frame packing, as frame packing and no omnidirectional projection can be independently applied.
* Incorporated JVET-Q0398: Sublayer wise dependency in multi-layer: when there is a dependent layer, there is an indication of the vps\_max\_tid\_il\_ref\_pics\_plus1 that the layer depends on, and if that value is 0, inter-layer prediction uses only IRAP pictures.
* Incorporated JVET-Q0400: Extra PH and SH bits in early positions.
* Incorporated JVET-Q0197: Miscellaneous editorial fixes to texts related to RPL signalling.
* Added pt\_display\_elemental\_periods\_minus1 plus 1 to the PT SEI message, to complete the incorporation of JVET-Q0818 for moving of the frame-field SEI message to the SEI spec.
* Incorporated JVET-Q0156: Enabling inter-layer prediction for STSA pictures.
* Incorporated JVET-Q0237: Allow STSA pictures in a dependent layer to have TemporalId equal to 0.
* Incorporated JVET-Q0277: Only allow references to SPSs/PPSs/APSs that are in the current or lower layer that is in an OLS that includes the VCL NAL unit.
* Incorporated JVET-Q0346: Add sh\_lmcs\_enabled\_flag.
* Incorporated JVET-Q0346: Enable the functionality of slice-level scaling list enabling/disabling.
* Incorporated JVET-Q0444: Make the presence of the sps\_affine\_amvr\_enabled\_flag in the SPS conditioned on the value of sps\_amvr\_ enabled\_flag.
* Incorporated JVET-Q0481: Reorder the syntax elements in the SPS and PH to group intra syntax elements and inter syntax elements
* Incorporated JVET-Q0402: Establish the semantics of sps\_subpic\_treated\_as\_pic\_flag[ ] to allow SNR scalability with independent subpictures when subpictures are aligned.
* Incorporated JVET-Q0317: When the two sides of a boundary have different values of sps\_loop\_filter\_across\_subpic\_enabled\_flag[ ], disable on both sides when either side has it disabled.
* Incorporated the following decision: Add an enable flag for dependent quantization in the SPS, constrain the SPS flag according to the general\_constraint flag, and condition the presence of the PH flag on the enable flag (and there are no other flags in other places). And do basically the same thing for sign data hiding.
* Incorporated JVET-Q0798: Merge candidate syntax accounting for picture-level TMVP control. Decision (cleanup): Determine the number of subblock merge candidates at the picture level based on the SPS number (sent at the SPS level) and the SPS subblock TMVP enable flag and the PH TMVP enable flag. Text is in Q0798 (v4) option 2.
* Incorporated JVET-Q0273: Use "ph\_" consistently for most PH syntax elements; use a shared syntax structure for RPL syntax in the PH and SH.
* Removed/rephrased all instances of "and that" in the text.
* Incorporated JVET-Q0164: Further clarified that each vertical slice boundary (and consequently each vertical subpicture boundary) is always a vertical tile boundary, including when pps\_single\_slice\_per\_subpic\_flag is equal to 1.
* Fixed the bugs [#747](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/747), [#752](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/752), [#801](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/801), [#823](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/823), [#824](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/824), [#826](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/826), [#828](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/828), [#830](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/830).
* Fixed the bugs [#315](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/315), [#777](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/777)<https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/747>, [#318](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/318), [#488](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/488), [#819](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/819), [#825](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/825), [#829](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/829), [**#**832](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/832), [**#**833](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/833).
* Fixed the bugs [**#**834](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/834), [**#**835](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/835), and [**#**836](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/836), and made various editorial changes, including merging of Annexes D and E, and putting the syntax and semantics of each SEI message next to each other.
* Fixed some bugs, incuding on the setting of PictureOutputFlag in clause 8.1.2, the typo reported in bug ticket [#841](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/841), and some other typo corrections.
* Fixed bugs [#843](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/843), [#848](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/848), [#850](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/850), [#852](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/852), [#860](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/860), [#861](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/861), [#874](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/874).
* Made some editorial corrections and fixed bugs [#878](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/878), [#881](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/881), [#883](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/883), [#884](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/884), [#888](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/888), [#890](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/890), [#893](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/893), [#896](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/896), [#898](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/898), [#905](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/905), [#906](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/906), and [#907](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/907).
* Updated the names of the SPS, PH, and SH syntax elements on enabling/disabling/using of explicit scaling list, clarified their semanitics, and fixed their use for controlling of explicit scaling list usage.
* Updated the names of some binary-valued sytnax elements to have "\_flag" at the end of their syntax element names, including cu\_cbf\_flag -> cu\_coded\_flag, tu\_cbf\_luma -> tu\_y\_coded\_flag, tu\_cbf\_cb -> tu\_cb\_coded\_flag, tu\_cbf\_cr -> tu\_cr\_coded\_flag, coded\_sub\_block\_flag-> sb\_coded\_flag, sao\_offset\_sign -> sao\_offset\_sign\_flag, and intra\_mip\_transposed -> intra\_mip\_transposed\_flag.
* Changed "predMode = MODE\_INTER (INTER, IBC)" to "predMode = MODE\_INTER or MODE\_IBC".
* Removed "It is a requirement of bitstream conformance that, when present, the next AU after an AU that contains an EOS NAL unit shall be a CVSS AU." which is a leftover that confilicts with the design intent that EOS is layer specific.
* Fixed places involving the general\_hrd\_parameters( ) syntax structure that contained some obsolete texts with the assumption of the ols\_hrd\_parameters( ) syntax structure being included within the general\_hrd\_parameters( ) syntax structure.
* Fixed texts on mixed nal\_unit\_type values for VCL NAL units within a picture to be more aligned with the agreed design intent noted in JVET meeting minutes.
* Added the condition "When sps\_video\_parameter\_set\_id is greater than 0" to the constraints that pic\_width/height\_max\_in\_luma\_samples shall be less than or equal to vps\_ols\_dpb\_pic\_width/height[ i ], because when the condition is false there is no VPS and vps\_ols\_dpb\_pic\_width/height[ i ] don't exist.
* Fixed bugs [#908](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/908), [#909](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/909), [#910](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/910), [#914](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/914), [#915](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/915), [#921](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/921), and [#940](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/940).
* Fixed bugs [#837](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/837), [#838](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/838), [#839](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/839), [#847](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/847), [#894](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/894), [#916](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/916), [#931](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/931), [#932](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/932), [#934](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/934), and [#935](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/935).

Draft 7 of Versatile Video Coding.

Ed. Notes:

* Fixed the bugs [#411](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/411), [#412](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/412), [#413](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/413), and [#414](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/414), as well as some other minor editorial bugs.
* Fixed the bug [#422](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/422).
* Made some other editorial bug fixes, including resolving the editor's note "[Ed. (GJS) Cleanup “referred” throughout.]".
* Fixed the bug [#437](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/437).
* Fixed the bugs [#444](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/444), [#457](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/457), [#460](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/460), [#494](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/494), [#498](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/498).
* Added filler data NAL unit and filler payload SEI message ([#497](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/497)).
* Added definitions and abbreviations of HRD and HSS.
* Fixed the constraints on TemporalId values for EOS\_NUT, EOB\_NUT, and AUD\_NUT.
* Fixed the semantics of general\_nal\_hrd\_params\_present\_flag and general\_vcl\_hrd\_params\_present\_flag.
* Fixed the bugs [#508](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/508), [#509](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/509).
* Fixed the bugs [#453](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/453), [#461](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/461), [#463](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/463), [#466](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/466), [#472](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/472), [#473](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/473), [#479](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/479), [#487](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/487), [#492](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/492), [#501](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/501), [#504](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/504), [#513](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/513).
* Fixed the bugs [#436](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/436), [#447](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/447), [#449](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/449), [#452](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/452), [#462](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/462), [#482](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/482), [#486](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/486), [#489](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/489), [#500](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/500), [#511](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/511), [#516](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/516), [#517](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/517).
* Incorporated JVET-P1026: Apply LFNST for ISP Blocks and signal MTS index after LFNST index.
* Incorporated JVET-P0983: Removal of sps\_sbt\_max\_size\_64\_flag.
* Incorporated JVET-P0054: Fixed MIP upsampling order.
* Incorporated JVET-P0199: MipSizeId to be equal to 1 for 4x16 or 16×4.
* Incorporated JVET-P0803: Cleanup of MIP.
* Incorporated JVET-P0329: Removing 1D cases comparisons in planar width and height variables.
* Incorporated JVET-P0111: Map angular modes from luma to chroma for 4:2:2.
* Incorporated JVET-P0418: Modifying MRL to use the same 3 reference lines as CCLM.
* Incorporated JVET-P0599: Cleanup of 4-tap interpolation filtering for intra reference samples.
* Incorporated JVET-P0517: Adaptive colour transform for 4:4:4.
* Incorporated NAL unit header changes
  + JVET-P0362: Reserve nuh\_layer\_id values 56 to 63, inclusive, in VVCv1
  + JVET-P0363: Rearrange/cleanup the NAL unit types
  + JVET-P0588, JVET-P0452: Specify prefix and suffix APSs
  + JVET-P0125: Constraints and rules on values of TemporalId and nuh\_layer\_id
  + JVET-P1006: Picture header (only the part of the new NAL unit type)
  + Changed the term "layer access unit" to "picture unit (PU)"
  + Applied the abbreviations of "AU", "PU", and "DU" in the text
* Incorporated HLS SPS changes
  + JVET-P0590: RPR enabled flag and scaling window
  + JVET-P0243, JVET-P0244, JVET-P0429: Cleanup changes, including coding one bit depth for both luma and chroma, fixed-length coding of SPS IDs, etc.
  + JVET-P0580: Reserve value 3 of log2\_ctu\_size\_minus5
  + JVET-P0578: Disallow the min CU size to be greater than Min( 64, CTU size )
* Incorporated HLS APS changes
  + JVET-P0122: Remove some ALF APS constraints
  + JVET-P0438: Add some ALF APS constraints
* Incorporated JVET-P0184: Cleanup changes on unavailable reference pictures and max\_dec\_pic\_buffering\_minus1
* Incorporated JVET-P0058: Enable transform skip for chroma.
* Incorporated JVET-P1001: Remove transform shift in transform skip mode.
* Incorporated JVET-P0460: Minimum QP setting for transform skip mode in palette mode escape coding.
* Incorporated JVET-P0059: Enable BDPCM for chroma.
* Incorporated JVET-P0516: Modified palette mode flag signalling.
* Incorporated JVET-P0077: Line-based coefficient group palette mode.
* Incorporated JVET-P0400: Remove IBC shared merge list.
* Incorporated JVET-P0457: Fix IBC merge list size signalling.
* Incorporated JVET-P1018: Alternative IBC virtual buffer setting.
* Incorporated JVET-P0325: check B1 before A1 in merge list construction process
* Incorporated JVET-P0090: Limitation of abs\_mvd\_min2 binarization within 32-bins
* Incorporated JVET-P0385: Align rounding of offset vector for sbTMVP
* Incorporated JVET-P0057: CE4-2.1 on 1/32-pel precision of PROF motion refinement
* Incorporated JVET-P0091: Align sample offset calculation of BDOF and PROF
* Incorporated JVET-P0154 (method 1): Clip the PROF sample offset to 14-bit
* Incorporated JVET-P0491: Change clipped range of BDOF and PROF motion refinement to be symmetric, [-31, 31]
* Incorporated JVET-P0653: BDOF and PROF parameter derivation
* Incorporated JVET-P0519: On SAD threshold for BDOF early termination
* Incorporated JVET-P1023: Reference picture conditions in DMVR and BDOF
* Incorporated JVET-P0530: Chroma format dependend TPM blending matrix
* Incorporated JVET-P0595: Align spec with SW on the derivation of weighting factor for CIIP mode
* Incorporated JVET-P0280 (method 1): Fix the behavior between BCW and WP
* Incorporated JVET-P0409: disable PROF for RPR.
* Incorporated JVET-P0088 and JVET-P0353: resampling filters for different picture ratios.
* Incorporated JVET-P0181: BP clean-ups.
* Incorporated JVET-P0183: PT clean-ups.
* Incorporated JVET-P0202, JVET-P0189, JVET-P0203: temporal sublayer timings, BP/PT/DUI clean-ups.
* Incorporated JVET-P0098: Keep VPS\_NUT, DCI\_NUT and EOB\_NUT NAL units when extracting a bitstream.
* Incorporated JVET-P00446: concatenation bug fix, additional concatenation info and AU dropping.
* Incorporated JVET-P0217: PTL clean-ups.
* Incorporated JVET-P0478: Added profile loop in DPS.
* Incorporated JVET-P0366: Added constriant flags for NAL unit types.
* Incorporated JVET-P0984: Added subpicture level information SEI message.
* Incorporated JVET-JVET-P0124, P0095, P0222: Mixed NAL unit types within a picture.
* Incorporated JVET-P1006: Picture header. (This also includes HLS action for JVET-P0407)
* Incorporated JVET-P0314: picture-level on/off control of BDOF, DMVR and PROF.
* Incorporated JVET-P0096, JVET-P1004, JVET-P1012, and JVET-P0252: Tile signalling simplification, removal of bricks, and avoiding signalling last pps\_tile\_idx\_delta\_val[ i ].
* Incorporated JVET-P1024: Add pps\_single\_slice\_per\_subpic\_flag in PPS.
* Incorporated JVET-P0171: Subpicture layout signalling using top left CTU, width, and height.
* Incorporated JVET-P0126, JVET-P0609: Subpicture ID signalling in slice header.
* Incorporated JVET-P0186: Change the name of single\_tile\_in\_pic\_flag and clarify the semantics to indicate that the flag disallows any partitioning of the picture into tiles/slices/subpictures. Also allow there to be only one tile in the picture when the flag is 0 (which may or may not contain multiple slices).
* Incorporated JVET-P0152: Remove pps\_five\_minus\_max\_num\_subblock\_merge\_cand\_plus1.
* Incorporated JVET-P0206: Remove pps\_temporal\_mvp\_enabled\_idc.
* Incorporated JVET-P0144: When single\_tile\_in\_pic\_flag is equal to 1, the value of pps\_num\_slices\_in\_pic\_minus1 is inferred to be equal to 0.
* Incorporated JVET-P0978: RPL constraints for picture types (CRA, GDR, leading pics, trailing pics).
* Incorporated JVET-P0378, JVET-P0572: Bugfix for removing dependency subpictures related to affine.
* Incorporated JVET-P0205: Requiring that vps\_video\_parameter\_set\_id shall be greater than 0.
* Incorporated JVET-P0097, JVET-P0205: Requiring the bitstream is only a single layer bitstream when sps\_video\_parameter\_set\_id is equal to 0.
* Incorporated JVET-P0185: Infer vps\_max\_layers\_minus1 to be equal to 0 when not present.
* Incorporated JVET-P0072: Modified bypass switch and remainder binarization.
* Incorporated JVET-P0298: Disable “level mapping" in bypass mode.
* Incorporated JVET-P0170: Simplified deriviation of ZeroPos[ n ].
* Incorporated JVET-P0562: Set Rice parameter for transform skip abs\_remainder binarization to 1.
* Incorporated residual coding fixes.
* Incorporated JVET-P0488: Remove unnecessary text.
* Incorporated JVET-P0350: Add context for second bin of LFNST index.
* Incorporated JVET-P0042: Cleanup for inter\_pred\_idc coding.
* Incorporated JVET-P0615: Separate contexts for cclm\_mode\_idx and intra\_chroma\_pred\_mode.
* Incorporated various fixes and cleanups including the ones reported in [#400](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/400), [#406](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/406), [#421](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/421), [#545](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/545), [#549](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/549).
* Incorporated JVET-P0168: Removing 2×2 chroma quantization matrices.
* Incorporated JVET-P0365: Scaling matrices for LFNST-coded blocks.
* Incorporated JVET-P1034: Improved coding of user defined quantization matrices.
* Incorporated JVET-P0410: chroma QP mapping table syntax variant with less bits.
* Incorporated JVET-P0469: Efficient coding of qp\_out\_val.
* Incorporated JVET-P0667: Not signalling QP offset table for joint CbCr coding in SPS if that mode is not enabled.
* Incorporated JVET-P0436: CU chroma QP offset signalling consistent with VPDU
* Incorporated various fixes and cleanups including the ones reported in [#438](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/438), [#520](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/520).
* Incorporated JVET-P0894: Profiles, tiers, and levels. Decision: Adopt, but with a maxDpbPicBuf of 8 rather than 7 (see notes for P0133).
* Incorporated JVET-P0188: Decision: It was agreed to establish a picture-level constraint the same as in HEVC (using MinCR as a function of timing).
* Incorporated JVET-P0182, JVET-P0221: Constrain that, when vps\_independent\_layer\_flag[ GeneralLayerIdx[ nuh\_layer\_id ] ] is equal to 1, the value of sps\_inter\_layer\_ref\_pics\_present\_flag shall be equal to 0.
* Incorporated JVET-P0221, JVET-P0589: Adopt the following improvement and bug fix:
  + Update the value range of ilrp\_idx[ listIdx ][ rplsIdx ][ i ] to be in the range of 0 to the NumDependentLayers[ GeneralLayerIdx[ nuh\_layer\_id ] ] − 1, inclusive.
  + Fix the equation for derivation of DirectDependentLayerIdx, by changing "i < " to "i <= ", and "j = i" to "j = i − 1".
* Incorporated JVET-P0228: Adopt the following clean-up changes:
  + Add "&& !vps\_all\_independent\_layers\_flag" to the syntax condition for vps\_output\_layers\_mode, and add the inference rule "When not present, the value of vps\_output\_layers\_mode is inferred to be equal to 1."
  + Replace "ph\_pic\_output\_flag" in the following sentence with "PictureOutputFlag": vps\_output\_layer\_mode is equal to 0 and the current access unit contains a picture that has ph\_pic\_output\_flag equal to 1, has nuh\_layer\_id nuhLid greater than that of the current picture, and belongs to an output layer (i.e., OutputLayerFlag[ GeneralLayerIdx[ nuhLid ] ] is equal to 1)."
* Incorporated JVET-P1019: Adopt the OLS signalling syntax (in the VPS) and semantics in JVET-P1019, with the addition of the following constraint: Each OLS shall have at least one output layer.
* Incorporated JVET-P0115: Adopt the clause 8.1.1 general decoding process text proposed in JVET-P0115-v2, and additionally specify the general decoding process as follows as a means for the decoder to know which OLS is being decoded (including which layers are being output): OlsIdx and Htid are given by external means, add a bitstream requirement that the bitstream does not contain any other layers than included in the OLS with OlsIdx and does not include any NAL unit with TemporalId greater than Htid.
* Incorporated JVET-P0117: Adopt the PTL signalling mechanism in JVET-P0117-v3. PTL signalled for an OLS rather than a layer.
* Incorporated JVET-P0118: Adopt the HRD design in JVET-P0118-v5, as summarized and noted below:
  1. Add vps\_max\_sublayers\_minus1 (this piece is also in JVET-P0185) and vps\_all\_layers\_same\_num\_sublayers\_flag to the VPS.
  2. On signalling of sequence-level HRD parameters, the following applies:
     1. For each independent layer used as an OLS containing only the layer, the sequence-level HRD parameters are signalled in the SPS referred to by the layer. (not in JVET-P0118-v5, suggested and agreed by the HLS BoG).
     2. For other cases, sequence-level HRD parameters are signalled in the VPS.
  3. DPB parameters (size, reordering, latency) for independent layers when used as output layers are signalled in the SPS (for enabling single-layer bitstreams without referring to VPSs), and in the VPS for other cases. Different DPB sizes may be signalled for a layer that may be used both as an output layer (in some OLSs) and as a non-output layer (in some other OLSs).
  4. Sequence-level HRD parameters syntax elements general\_nal\_hrd\_params\_present\_flag, general\_vcl\_hrd\_params\_present\_flag, decoding\_unit\_hrd\_params\_present\_flag, tick\_divisor\_minus2, du\_cpb\_params\_in\_pic\_timing\_sei\_flag, bit\_rate\_scale, cpb\_size\_scale, cpb\_size\_du\_scale, sublayer\_cpb\_params\_present\_flag, num\_layer\_hrd\_params\_minus1, and bp\_cpb\_cnt\_minus1 are signalled at most once for each multi-layer bitstream.
  5. A list of OLS HRD parameter syntax structure is signalled, and an index to the list is signalled for each OLS as needed. Different OLS HRD parameter syntax strutures may include different numbers of sublayer HRD parameter syntax structures.
  6. When sublayer\_cpb\_params\_present\_flag is equal to 0, the OLS HRD parameters for the lower sublayer representations are inferred to be the same as the only set of OLS HRD parameters signalled in the OLS HRD parameters syntax structure for the highest sublayer representation.
  7. The sub-bitstream extraction process is no longer a part of the decoding process, thus moved from clause 10 to Annex C, and technically updated to output a sub-bitstream for a target OLS, and the extraction process removes SEI NAL units that contain buffering period, picture timing, or decoding unit information SEI messages that do not apply to the target OLS.
  8. Only one set of conformance tests, for testing the conformance of the OLSs specified by the VPS, is specified. Therefore, it is possible that there is no OLS specified by the VPS that included all layers of the entire bitstream, in which case the conformance of the entire bitstream is not specified.

Delegate the following to editors: Update the HRD text, particularly the part on describing the conformance tests, such that it sdoes not appear that VPS would be needed even when the bitstream only has a single layer.

* 1. Similarly as in MVC, CPB operations are OLS-specific, i.e., there is only one CPB for the entire OLS bitstream. However, the DPB is specified to conceptually consist of a sub-DPB for each layer, and the DPB operates in a layer-specific manner.
  2. In the list of bitstream conformance constraints, added that the DPB output times derived for all pictures in any particular access unit shall be the same.
  3. In the derivations of the value of the variable NoOutputOfPriorPicsFlag in the DPB operations texts, the syntax elements pps\_pic\_width\_in\_luma\_samples, pps\_pic\_height\_in\_luma\_samples are replaced by sps\_pic\_width\_max\_in\_luma\_samples and sps\_pic\_height\_max\_in\_luma\_samples.
* Incorporated JVET-P0190: Adopt the scalable nesting SEI message in JVET-P0190-v2.
* Incorporated JVET-P0125: Add the set of general constraints and rules on SEI messages of the following categories in section 4 of JVET-P0125-v2:
  + Two constraints on containing of SEI messages in SEI NAL units
  + The specifiation of applicable layers of non-scalable-nested SEI messages
  + Three constraints on the value of nuh\_layer\_id of SEI NAL units
* Incorporated the support of mixed IRAP and non-IRAP pictures within an AU, with the following POC design:
  + For independent layers, use POC MSB cycle signalling. In the SPS, there is a flag to control whether PH has ph\_poc\_msb\_cycle\_present\_flag, and a length in the SPS. When ph\_poc\_msb\_cycle\_present\_flag is equal to 1, the POC MSB cycle, u(v) coded, is signalled in the PH. When POC MSB cycle is present, the POC MSB cycle of the picture is set equal to ph\_poc\_msb\_cycle\_val \* MaxPicOrderCntLsb. [JVET-P0116]
  + For dependent layers, if there is a picture picA in the same AU in a reference layer of the current layer, the POC is derived as equal to the POC of picA and POC LSB values are required to be cross-layer aligned. Otherwise, the current POC derivation process applies. [JVET-P0101]
* Incorporated JVET-P00116: Add a constraint to require that each IRAP AU is complete (i.e., there is a picture in each layer present in the CVS) and all pictures in an IRAP AU are IRAP pictures with the same NAL unit type.
* Incorporated JVET-P0125: Specify EOS NAL units to be layer specific.
* Incorporated JVET-P0097: Adopt a clarification to the bullet items in setting of PicOutputFlag in the decoding process of one coded picture involving variables derived from the VPS, by adding the condition "sps\_video\_parameter\_set\_id is greater than 0", which means that a VPS is referred to by the CLVS.
* Incorporated JVET-P0135: Add a constraint to require that when vps\_independent\_layer\_flag[ i ] is equal to 0 there shall be at least one value of j such that the value of vps\_direct\_dependency\_flag[ i ][ j ] is equal to 1.
* Incorporated USNB comment 1: Removed VUI except sps\_field\_seq\_flag (and the VUI spec except sps\_field\_seq\_flag is to be added to the SEI spec in JVET-P2007).
* Incorporated editorial changes noted in the HLS BoG meeting minutes.
* Incorporated editorial cleanup related to JVET-P1034 and fixes related to interaction between JVET-P0410 and JVET-P0667.
* Incorporated various fixes and cleanups including the ones reported in [#416](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/416), [#519](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/519), [#565](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/565), [#602](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/602), [#607](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/607).
* Incorporated various fixes and cleanups including the ones reported in [#579](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/579), [#585](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/585), [#587](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/587), [#588](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/588), [#589](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/589), [#590](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/590), [#591](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/591), #[598](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/598), [#599](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/599).
* Incorporated JVET-P0406: Generalization of local dual tree for different chroma formats.
* Incorporated JVET-P0063: Disallow 4×4 luma inter blocks introduced by local dual tree.
* Incorporated JVET-P0063: Editorial refinements for the semantics related to local dual tree.
* Incorporated JVET-P0641: Removal of 2×N chroma intra blocks.
* Incorporated JVET-P0347: Maximum MTT Depth and MinCbSize conformance.
* Incorporated JVET-P0360: Remove TU ambiguities.
* Incorporated JVET-P0301: Fixed the decoding process in case of implicit TU partitioning.
* Incorporated JVET-P0626: Cleanup of reference sample padding for intra prediction.
* Incorporated JVET-P0214: Change the chroma intra mode from DM to planar in CIIP.
* Incorporated JVET-P0856: Align spec to SW on the two mismatches in SIF and BCW index
* Incorporated JVET-P0043: Fixes of inconsistency of deblocking in case of affine 4×4 and TPM block edges, when they coincide with TU boundary.
* Incorporated JVET-P1001: Chroma deblocking filter with chroma QP offsets taken into consideration.
* Incorporated JVET-P0162: Remove alf\_ctb\_use\_first\_aps\_flag.
* Incorporated JVET-P0164 (method 2): Remove alf\_luma\_coeff\_signalled\_flag and alf\_luma\_coeff\_flag.
* Incorporated JVET-P0254: Fix number of segments to 32 regardless of bit depth.
* Incorporated JVET-P0505: Fixing non-linear ALF clipping values for 8-bit video.
* Incorporated JVET-P0081: Longer tap filter application at Chroma CTB boundaries.
* Incorporated JVET-P0371: Signalling of corrective values for chroma residual scaling.
* Incorporated JVET-P0554: Clean up of coefficient coding of adaptive loop filter.
* Incorporated JVET-P0665: Spec fix for ALF filter and transpose index calculation.
* Incorporated JVET-P1038: ALF padding method for boundaries.
* Incorporated various fixes and cleanups including the ones reported in [#316](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/316), [#419](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/419), [#457](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/457), [#592](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/592), [#593](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/593), [#597](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/597), [#617](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/617), [#618](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/618), [#626](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/626), [#627](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/627), [#628](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/628), [#629](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/629), [#630](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/630), [#637](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/637), [#638](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/638).
* Incorporated SEI payload types for the newly added SEI messages into both this spec and the VUI and SEI spec.
* Among other editorial changes, replaced "sub-layer" and "sub\_layer" with "sublayer", to be consistent with the VUI and SEI spec.
* Incorporated JVET-P0218: The value of nuh\_layer\_id of AUD NAL units shall be equal to vps\_layer\_id[ 0 ].
* Incorporated a missing piece from JVET-P0206: When no reference picture in the DPB has the same spatial resolution as the current picture, the value of ph\_temporal\_mvp\_enabled\_flag shall be equal to 0.
* Incorporated JVET-P0592: Phase shifts for chroma in RPR with the two-flag solution to indicate 0 or 0.5 offset in horizontal and vertical directions, respectively.
* Added Annex E "Use of ITU-T H.SEI | ISO/IEC 23002-7 VUI parameters and SEI messages".
* Incorporated the syntax and semantics of the reserved SEI message, same as in HEVC.
* Among other editorial changes, fixed the setting of the variable VclAssociatedSeiList.
* Fixed the following bugs reported by Virginie Drugeon:
  + hori\_scale\_fp and vert\_scale\_fp are used in 8.5.6.3.1 General without being defined. Based on the inputs to this process, should these two variables be scalingRatio[ 0 ] and scalingRatio[ 1 ] instead?
  + The inference rule for pt\_sublayer\_delays\_present\_flag seems wrong, since pt\_sublayer\_delays\_present\_flag[ bp\_max\_sublayers\_minus1 ] is not signalled, but is used for DU Cpb removal delays and should be equal to 1. I would therefore suggest the following inference rule: "When not present, the value of pt\_sublayer\_delays\_present\_flag[ i ] is infered to be equal to 1 for i equal to bp\_max\_sublayers\_minus1 and infered to be equal to 0 for all other values of i."
  + Typo in picture timing SEI message semantics: "delay\_for\_concatenation\_ensured\_flag equal to 0 specified that ". It should be "specifies".
  + In subpicture level information SEI message semantics, in the semantics of ref\_level\_fraction\_minus1, when calculating SubpicNumTileRows, it is always set equal to 1. I believe that there is a typo in the derivation, whereby after "if( CtbToTileRowBd[ ctbAddrRs ]  !=  CtbToTileRowBd[ ctbAddrRs − PicWidthInCtbsY ] )", it is SubpicNumTileRows instead of SubpicNumTileCols that should be incremented by 1.
  + In subpicture level information SEI message semantics, in the semantics of ref\_level\_fraction\_minus1, there are two instances of ref\_level\_idc[ i ][ j ], although ref\_level\_idc[ i ] only has one index. The second index should be removed for these two instances.
  + In "Otherwise, if ph\_subpic\_id\_signalling\_present\_flag is equal to 1, the length of sh\_subpic\_id is equal to pps\_subpic\_id\_len\_minus1 + 1", ph\_subpic\_id\_signalling\_present\_flag should be pps\_subpic\_id\_mapping\_present\_flag.
  + In "PocLsbLt[ i ][ j ] = PocLsbLt[ i ][ j ]", the right side of the equation should be PicPocLsbLt[ i ][ j ].
* Fixed the first two aspects reported in bug ticket [#623](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/623).
* Fixed the bugs reported in bug tickets [#475](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/475), [#640](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/640), [#641](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/641), [#645](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/645), and [#658](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/658).
* Incorporated various fixes and cleanups including the ones reported in [#408](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/408), [#429](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/429), [#435](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/435), [#467](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/467), [#495](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/495), [#550](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/550), [#555](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/555), [#564](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/564), [#570](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/570), [#580](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/580), [#581](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/581), [#582](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/582), [#583](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/583), [#603](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/603), [#614](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/614), [#624](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/624), [#644](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/644), [#646](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/646), [#647](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/647), [#649](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/649), [#651](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/651), [#652](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/652), [#653](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/653), [#654](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/654), [#655](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/655), [#657](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/657), [#661](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/661), [#663](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/663), [#667](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/667), [#668](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/668), [#673](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/673), [#674](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/674) and the following bugs reported by Virginie Drugeon:
  + Typo in the syntax of residual\_ts\_coding: at the very end of this syntax structure, TransCoeffLevel is calculated as follows: "TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] ) = ( 1 − 2 \* coeff\_sign\_flag[ n ] ) \* AbsLevel[ xC ][ yC ]" where the ) before = seems to be a typo.
  + Typo in PPS semantics of pps\_mixed\_nalu\_types\_in\_pic\_flag : "...that the VAL NAL units do not have the same value of nal\_unit\_type ..."
  + Typo in the semantics of pps\_subpic\_id[ i ]: "pps\_subpic\_id[ i ] specifies that subpicture ID of the i-th subpicture." : "that" should probably be "the".
  + Table 9-82 is called "Specification of cRiceParam based on locSumAbs, trafoSkip and s". However, trafoSkip and s are not derived in this process anymore, so the name of the table should probably be: "Specification of cRiceParam based on locSumAbs".
* Fixed the bugs [[#344](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/344)](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/344), [#402](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/402) and the following bugs reported by Virginie Drugeon:
  + Editorial only: In 8.5.2.3 Derivation process for spatial merging candidates, for each of the spatial candidates A0, A1, B0, B1 and B2, there is a sentence similar to this one: "– The variables availableFlagB1, refIdxLXB1, predFlagLXB1 and mvLXB1 are derived as follows:" However, there are more variables derived below that sentence. In particular, hpelIfIdxB1 and bcwIdxB1 are also derived. It is therefore suggested to extend the sentence: "– The variables availableFlagB1, refIdxLXB1, predFlagLXB1, mvLXB1, hpelIfIdxB1 and bcwIdxB1 are derived as follows:" (same for the other 4 spatial candidates)
  + In addition, when availableB1 is equal to FALSE, all the variables are set to default values, except for hpelIfIdxB1. Is that intended? Is there a default value set somewhere else? (same for all the other 4 spatial candidates)
  + Fixed derivation of SubpicNumTileRows in subpictures level information SEI semantics.
* Incorporated residual coding fixes and cleanups reported in [#404](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/404), [#578](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/578), [#608](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/608), [#609](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/609), [#678](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/678), [#679](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/679), [#680](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/680).
* Incorporated various fixes and cleanups including the ones reported in [#405](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/405), [#440](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/440), [#446](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/446), [#451](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/451), [#533](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/533), [#541](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/541), [#543](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/543), [#554](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/554), [#605](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/605), [#686](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/686), [#687](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/687), [#689](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/689), [#690](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/690), [#692](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/692), [#693](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/693), [#694](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/694), [#697](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/697), [#699](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/699).
* Minor changes of chroma sample position figures.
* Renumbered equations, figures and tables to use sequential numbering throughout the main body, as requested by ISO CS.
* Remove normative reference to Rec. ITU-T H.273 | ISO/IEC 23091-2 Coding-independent code points for video signal type identification, since the text no longer refers to it.
* Added normative reference to Rec. ITU-T T.35 since the text now refers to it.
* Removed bibliographic reference to Rec. ITU-R BT.2020 since the text no longer refers to it.
* Integrated several cleanups and fixes including:
  + adding a condition for modeType derivation that was accidentally removed as discussed in [#451](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/451),
  + removing Log2( 0 ) case in cpMvLXCorner derivation as reported in [#532](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/532),
  + updating Figure 7 illustrating subpictures to match an OMAF use case,
  + fixing minor typo in conformance\_window\_flag semantic as reported in [#704](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/704).

Draft 6 of Versatile Video Coding.

Ed. Notes:

* Incorporated JVET-O0179: NAL unit header.
* Incorporated JVET-O0610: AUD mandation.
* Incorporated JVET-O0113: No parameter set activation process.
* Incorporated JVET-O0061: Half luma sample MV threshold for deblocking decisions
* Incorporated JVET-O0637: Chroma line selection for gradient computation
* Incorporated JVET-O0159: Deblocking tC table defined for 10-bit video
* Incorporated JVET-O0060: Deblocking at 4×4 block boundaries for luma
* Incorporated the following decisions:
  + JVET-O0181: Non ref flag.
  + Use u(13) for coding of the virtual boundary syntax elements.
  + JVET-O0236: Signal num\_tiles\_in\_pic\_minus1 under the syntax condition "if( uniform\_tile\_spacing\_flag  &&  brick\_splitting\_present\_flag )", to avoid a PPS parsing dependency on SPS.
  + JVET-O0044: Allow signalling of zero or more sub profile fields, use 8 bits for the count, and use 32 bits for the sub profile field.
  + JVET-O0148 (basically editorial): Adopt a constraint to require that, for P and B slices, the number of active entries in RPL 0 is greater than 0, and for B slices, the number of active entries in RPL 1 is greater than 0.
  + JVET-O0244: Additionally signal weighted prediction enabled flags in SPS, which are signalled in the PPS currently, and to syntactically prevent zero delta POC values when weighted prediction is not enabled. Editorial: move the condition out of the RPL structure syntax.
* Incorporated JVET-O1164 & JVET-O1159: Reference picture resampling (RPR) and scalability.
* Incorporated the following decisions:
  + Incorporated JVET-O0176: Moving of slice\_pic\_order\_cnt\_lsb to be before ph\_recovery\_poc\_cnt in the slice header.
  + Incorporated JVET-O0178: Conditionally signalling of sps\_sublayer\_ordering\_info\_present\_flag based on the value of sps\_max\_sublayers\_minus1.
  + Incorporated JVET-O0201: Changing the coding of ph\_recovery\_poc\_cnt to ue(v) instead of se(v), thus disallowing negative values that allow output gaps.
  + Incorporated JVET-O0234: Add slice header extension mechanism that is essentially identical to that in HEVC.
  + Incorporated JVET-O0241: Setting of PictureOutputFlag in the general decoding process for a coded picture and in the process for generating unavailable reference pictures, add the setting the POC value for long-term reference pictures in the RPL decoding process that will be used in the process for generating unavailable reference pictures, and correctly set the slice\_pic\_order\_cnt\_lsb for the generated LTRP in the decoding process for generating unavailable reference pictures.
* Incorporated the following decisions:
  + Incorporated JVET-O0144: Bug fix to the slice data syntax such that byte alignment is ensured at the end of each brick/WPP subset.
  + Incorporated JVET-O0143: Option 2 on signalling of addresses for rectangular slices.
  + Incorporated JVET-O0173, JVET-O0176, JVET-O0338: Replacing num\_brick\_rows\_minus1 with num\_brick\_rows\_minus2.
  + Incorporated JVET-O0176: Adding the following constraint to the semantics of pps\_rect\_slice\_flag: When brick\_splitting\_present\_flag is equal to 1, the value of pps\_rect\_slice\_flag shall be equal to 1.
  + Incorporated JVET-O0452: Not signalling brick\_split\_flag[ i ] but inferring its value when RowHeight[ i ] is equal to 1; not signalling uniform\_brick\_spacing\_flag[ i ] but inferring its value when RowHeight[ i ] is not greater than 2.
  + Incorporated the decision to add a note into the draft text to clarify such cases where sub-bitstream extraction without touching VLC NAL units need to be kept in mind, e.g., the PPS in the extracted sub-bitstream should typically have signalled\_slice\_id\_flag equal to 1, even when pictures referring to the PPS has only one slice in them.
  + Incorporated JVET-O0338: Removing the following constraint for rectangular slices: The slices of a picture shall be in increasing order of their sh\_slice\_address values.
  + Incorporated JVET-O0181: Changing the inference "When pps\_rect\_slice\_flag is equal to 0 and single\_brick\_per\_slice\_flag is equal to 1, the value of num\_bricks\_in\_slice\_minus1 is inferred to be equal to 0." to "When single\_brick\_per\_slice\_flag is equal to 1, the value of num\_bricks\_in\_slice\_minus1 is inferred to be equal to 0."
* Incorporated the following decisions on HRD:
  + Incorporated JVET-O0189: DU based HRD
  + Incorporated JVET-O0180/O0228: Subpicture specific initial removal times, parsing dependencies, editorials
  + Incorporated JVET-O0177: Selective sublayer timing info
  + Incorporated JVET-O0227: Resolving parsing dependency in the picture timing SEI message
  + Incorporated JVET-O0153: HRD related clean ups
* Incorporated JVET-O0472: LFNST index signalling depends on last position.
* Incorporated JVET-O0094: Simplification of LFNST.
* Incorporated JVET-O0368/O0292/O0521/O0466: Disable LFNST for non-DCT2 MTS candidate normatively.
* Incorporated JVET-O0529/O0540: Disable LFNST and MIP for implicit MTS.
* Incorporated JVET-O0219: Use the collocated luma intra prediction mode as the transform set index.
* Incorporated JVET-O0213: No LFNST for CUs larger than the max TU size.
* Incorporated JVET-O0294/O0372/O0380: Context modelling for signalling MTS index.
* Incorporated JVET-O0541: Decouple intra transform selection from inter MTS related SPS flag.
* Incorporated JVET-O0474: No MTS for IBC.
* Incorporated JVET-O0357: Separate the max TB width and height according to the horizontal and vertical subsampling ratio.
* Incorporated JVET-O0545: Enable the max luma transform size to be either 64 or 32 only with a flag at the SPS.
* Incorporated JVET-O0105: Improved joint chroma residual coding.
* Incorporated JVET-O0052: Context-coded bin limit on TB level (threshold 1.75).
* Incorporated JVET-O0122: Improved transform skip residual coding (sign context + level mapping)
* Incorporated JVET-O0619/O0623: Three scan passes for transform skip residual coding.
* Incorporated JVET-O0409: Exclude coded\_subblock\_flag from couting context-coded bins in transform skip.
* Incorporated JVET-O0919/O0405: QP′ clipping for transform skip.
* Incorporated JVET-O0375/O0193: Remove transform depth from ctx derivation for coded block flags.
* Incorporated JVET-O0617: Context reduction for sig\_coeff\_flag in regular transform coefficient coding.
* Incorporated JVET-O0543/O0670: Restrict additional joint chroma modes to intra CUs.
* Incorporated JVET-O0376: SPS enable flag for joint chroma residual coding.
* Incorporated JVET-O0065: CABAC context initialization.
* Incorporated JVET-O1109: Chroma residual scaling implicit derivation
* Incorporated JVET-O0428: LMCS related clean-ups
* Incorporated JVET-O0272: Simplified inverse luma mapping.
* Incorporated JVET-O0057: HPEL AMVR extension with alternative interpolation filter.
* Incorporated JVET-O0163: Remove switching between L0 and L1 for temporal MV.
* Incorporated JVET-O0108: Disable DMVR and BDOF for CIIP coded blocks
* Incorporated JVET-O0055: BDOF 4×4 early termination threshold
* Incorporated JVET-O0634: Allowed BDOF block sizes are unified with DMVR blocks sizes.
* Incorporated JVET-O0681: Disable BCW for CIIP coded blocks.
* Incorporated JVET-O0590: Modified SAD for the center coordinate.
* Incorporated JVET-O0297: Simplification for DMVR padding process.
* Incorporated JVET-O0249: Modified merge syntax.
* Incorporated JVET-O0265: TPM motion storage.
* Incorporated JVET-O0220: Maximum number of subblock merge candidates signalling.
* Incorporated JVET-O0366: Copy BCW index from the first spatial affine merge candidate.
* Incorporated JVET-O0164: Remove spatial AMVP candidate scaling.
* Incorporated JVET-O0284: Condition sym\_mvd\_flag on ph\_mvd\_l1\_zero\_flag
* Incorporated JVET-O0438: SPS affine AMVR control flag.
* Incorporated JVET-O0567: SMVD value range
* Incorporated decision from plenary meeting: extend MVD range to [−217, 217−1]
* Incorporated JVET-O0572: Fix of availability of default reference pictures used in SMVD
* Incorporated JVET-O0414: SMVD only for STRPs
* Incorporated JVET-O0304: Reduction of number of multiplications in BDOF.
* Incorporated JVET-O1140: Unified syntax on DMVR and BDOF flag.
* Incorporated the following decisions:
  + Removed Annex F (non-HRD SEI messages); the removed SEI messages are to be included in JVET-O2007
  + Fixed some x-refs
  + JVET-O0154: Added Annex B text
  + JVET-O0041: Frame-field information SEI message
  + JVET-O0043: Editorial action item: Clarify that the design intent of the HRD picture output is aligned with output patterns of pictures described by syntax elements in the frame-field\_information SEI message (formerly expressed as pic\_struct), including repetitions of pictures. This is intended to essentially work the same way as in HEVC. Delegate to editors to express this in the specification.
* Incorporated JVET-O1143: Subpictures
* Incorporated the following decisions:
  + Incorporated JVET-O0245: Updating constraints for TemporalId of APS, VPS, and DPS; updating constraints for multiple ALF APS; adding constraint for layer id of APS NAL unit such that it shall be the same as the layer id of the current picture of one of the direct reference layer.
  + Incorporated JVET-O0299: Moving scaling list matrices from SPS to APS and adding constraint that all slices of a picture shall refer to the same matrix.
  + Incorporated the decision to clarify that different types of APSs have independent APS ID value spaces.
  + Incorporated JVET-O0428: Limiting LMCS APS ID to be in the range of 0 to 3, inclusive; constraining that all slices of the same picture shall refer to the same LMCS APS; limiting the APS ID range for other APS types as well; adding constraint that all slices of the same picture shall refer to the same scaling APS.
* Incorporated JVET-O0151 and JVET-O0152:
  + The value of ph\_no\_output\_of\_prior\_pics\_flag for CRA pictures starting a new CVS is used similarly as for IDR pictures.
  + The ph\_no\_output\_of\_prior\_pics\_flag is also signalled for GRA pictures and applied for GRA pictures starting a new CVS similarly as for IDR pictures.
  + Adopt to allow association of a buffering period SEI message with a GRA picture, and some minor bug fixes and editorial improvements proposed in JVET-O0152, excluding the change of the trailing picture definition, including changing of the term GRA to GDR
* Incorporated JVET-O0147 and JVET-O0226: Constraints on IRAP, RASL, RADL, and trailing pictures
* Incorporated JVET-O0235 by adding the following constraints:
  + All VCL NAL units of a picture shall have the same NAL unit type. A picture or a PU is also referred to as having a NAL unit type equal to the NAL unit type of the coded slice NAL units of the picture or PU.
  + STSA pictures shall have TemporalId greater than 0.
  + Constraints on RPLs such to make sure that step-wise sublayer switching can occur at STSA pictures.
* Incorporated JVET-O0238 and JVET-O0491:
  + Selectively signal slice header parameters in PPS
  + Removal of tb(v) and tu(v) coding of syntax elements and related parsing process
* Incorporated JVET-O0145 and JVET-O0215: Instead of sending num\_entry\_point\_offsets, send a entry\_point\_offsets\_present\_flag at the PPS level. If it is 1, the number of entry points NumEntryPoints is derived; otherwise, none are signalled.
* Incorporated JVET-O1124: CCLM restrictions for dualtree to reduce latency.
* Incorporated JVET-O1153: Intra chroma mode coding cleanup.
* Incorporated JVET-O0050: Small chroma block size restrictions for shared tree.
* Incorporated JVET-O0640: Restrict picture width, height to be a multiple of Max(8, MinCbSizeY).
* Incorporated JVET-O0106: Use 4xN prediction regions for 4xN and 8×N (N > 4) ISP-coded blocks.
* Incorporated JVET-O0341: Always apply 4-tap cubic interpolation filter for ISP-coded blocks.
* Incorporated JVET-O0651: Chroma DM mode is set as DC if collocated luma block is coded by IBC mode.
* Incorporated JVET-O0925: MIP 8-bit coefficient and simplifications.
* Incorporated JVET-O0502: Signalling of all 67 intra modes for ISP (MPM, remaining modes) and apply PDPC.
* Incorporated JVET-O0364/O0277/O0426: Intra prediction simplifications.
* Incorporated JVET-O0655: Include wide angles in chroma intra angle mapping table for 4:2:2.
* Incorporated JVET-N0222: CIIP intra mode propagation to intra block.
* Incorporated JVET-O0525: Remove PCM.
* Incorporated JVET-O0315: Intra prediction mode alignment for BDPCM.
* Incorporated JVET-O1136: Unified TS and BDPCM signalling.
* Incorporated JVET-O0078: Single HMVP table inside shared merge list region for IBC.
* Incorporated JVET-O0258: Disabling IBC for chroma in case of dual tree and cleanups.
* Incorporated JVET-O0162: IBC mvp flag conditioned on MaxNumIbcMergeCand > 1.
* Incorporated JVET-O0455: Number of IBC merge candidates independent for P/B slices.
* Incorporated JVET-O1170: Bitstream conformance with a virtual IBC buffer concept.
* Incorporated JVET-O0119: Base palette mode from HEVC SCC for 4:4:4 colour format.
* Incorporated JVET-O0070: Prediction refinement with optical flow.
* Incorporated JVET-O0594: Unification of Prediction sample padding for BDOF and PROF.
* Incorporated JVET-O0570: Unified gradient calculations.
* Incorporated JVET-O0616: No signal of SH chroma tool flags in case of 4:0:0.
* Incorporated JVET-O0526: Set Min CTU size to 32×32.
* Editorial refinement of APS semantics and related definitions.
* Reference CICP (Rec. ITU-T H.273 | ISO/IEC 23091-2) where relevant for VUI and fix associated presence bug.
* Incorporated JVET-O0064: Modification of clipping value signalling.
* Incorporated JVET-O0090: Alternative chroma filters and CTU chroma filter selection.
* Incorporated JVET-O0216: Fixed EG order for ALF filter coefficient coding.
* Incorporated JVET-O0247: Remove signalling of alf\_luma\_prev\_filter\_idx\_minus1.
* Incorporated JVET-O0625: Apply ALF VB when the bottom CTU boundary is a slice/tile/brick or “360 virtual boundary”.
* Incorporated JVET-O0288: Unify the maximum number of ALF APSs for inter and intra and remove the inference of chroma APS ID from luma.
* Incorporated JVET-O0669: Simplification of ALF coefficient signalling.
* Incorporated JVET-O0662: Padding in ALF for various boundaries.
* Bug and typo fixes in deblocking and LMCS.
* Alignment of JVET-O0050 and JVET-O1109: allow CRS when DUAL\_TREE is used.
* Incorporated JVET-O0046: always send cu\_qp\_delta in first TU for CU > 64x64.
* Incorporated JVET-O0650: chroma QP table signalling.
* Incorporated JVET-O1168: CU level chroma QP control.
* Incorporated JVET-O0267: Use inter scaling matrices for IBC.
* Incorporated MIP weight matrices.
* Incorporated JVET-O0194: Store CuPredMode[ x ][ y ], cqtDepth[ x ][ y ], CbWidth[ x ][ y ], CbHeight[ x ][ y ], CbPosX[ x ][ y ] and CbPosY[ x ][ y ] for separate trees.
* Incorporated JVET-O0188: Single table for luma and chroma clipping thresholds.
* Incorporated various fixes and cleanups including the ones reported in [#323](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket360), [#359](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket359), [#360](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket360), [#371](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/371), [#376](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/376), [#377](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/377), [#378](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/378), [#382](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/382), [#383](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/383), [#387](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/387), [#389](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/389), [#392](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/389).
* Incorporated various fixes and cleanups, including
  + Replacing of max\_grid\_idxs\_minus1 with max\_subpics\_minus1.
  + Rreplacing the coding of slice\_lmcs\_aps\_id and slice\_scaling\_list\_aps\_id from u(5) to u(2) and u(3), respectively.
  + Resolving the parsing and semantics dependencies of the SPS syntax elements sps\_ref\_wraparound\_enabled\_flag and sps\_ref\_wraparound\_offset\_minus1 on PPS syntax elements.
  + Resolving the parsing and semantics dependencies of the SPS syntax elements subpic\_grid\_col\_width\_minus1 and subpic\_grid\_row\_height\_minus1 on PPS syntax elements.
* Add/expand Abstract, Introduction, and Scope.
* Fixed typos: [#379](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/379).
* Fixed bug [#374](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/374): Wrong BDOF disable threshold.
* Incorporated palette mode fixes.
* Fixed the bugs [#384](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/384), [#385](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/385), [#390](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/390), [#391](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/391), and [#345](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/345), as well as the incorrect use of the sps\_loop\_filter\_across\_subpic\_enabled\_flag[ ] in the adaptive loop filter processes.
* Incorporated frame-field information SEI message, replicating the compound functionality of pic\_struct into separate elements in an SEI message, per JVET-O0042-v3 presentation slide 14, JVET-O1177, and the agreement that "This is intended to essentially work the same way as in HEVC. Delegate to editors to express this in the specification."
* Incorporated neighbouring availability checking process.
* Incorporated and fixed CABAC init value tables with equal probability (EP) values until trained VTM values are final.
* Incorporated various fixes and cleanups including the ones reported in [#394](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket394), [#395](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket395), [#396](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket396), [#397](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket397), [#398](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket398), [#399](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket399), [#401](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket401).
* Incorporated the derivation of CurrPicScalWinWidthL and CurrPicScalWinHeightL as a bug fix.

Draft 5 of Versatile Video Coding.

Ed. Notes:

* Incorporated JVET-N0067: NAL unit header.
* Incorporated JVET-N0278: Layer concept without inter-layer referencing.
* Incorporated JVET-N0349: Decoding parameter set (DPS).
* Incorporated JVET-N0869: Semantics of DPS ID equal to 0.
* Incorporated JVET-N0805: APS types (with loop).
* Incorporated JVET-N0100: Signalling information about long-term reference picture POC LSB.
* Incorporated JVET-N0100: Conditional signalling of num\_ref\_idx\_active\_overwrite\_flag.
* Incorporated JVET-N0288: Inference rule for num\_bricks\_in\_slice\_minus1.
* Incorporated JVET-N0124: Inference rule for single\_brick\_per\_slice\_flag and signalling the bottom right brick index as a delta from top left brick index.
* Incorporated JVET-N0047: Ref pic list for IDR.
* Incorporated JVET-N0276: 5 new constraint flags.
* Incorporated JVET-N0276: Sub-profile indication.
* Incorporated JVET-N0865: Gradual random access (GRA).
* Incorporated JVET-N0101: CRA decoding process.
* Incorporated JVET-N0070: Ref pic wraparound cleanups.
* Incorporated JVET-N0352: Conformance window.
* Incorporated JVET-N0438: Loop filtering disabling aross virtual boundaries.
* Incorporated JVET-N0498: Uniform tile partitioning.
* Incorporated JVET-N0150: WPP with 1-CTU lag.
* Incorporated JVET-N0120: Editorial improvements to the general decoding process to clarify the difference between a CVS and a bitstream.
* Incorporated JVET-N0857: Tile and brick partitioning.
* Incorporated JVET-N0706: Decoded picture hash SEI message.
* Incorporated JVET-N0494: Dependent RAP indication SEI message.
* Incorporated JVET-N0353: Buffering period & picture timing SEI messages.
* Incorporated JVET-N0867: Temporal scalability HRD parameters.
* Incorporated JVET-N0063: VUI design.
* Incorporated JVET-N0423 & N0350: HRD starting point.
* Incorporated JVET-N0266: Remove 4×4 unipred, and 4×8/8×4 bipred regular inter modes, and remove the use of for shared merge candidates in the regular merge list.
* Incorporated JVET-N0821: 6-tap interpolation filter for affine MC.
* Incorporated JVET-N0068: Restriction of memory bandwidth consumption of affine MC.
* Incorporated JVET-N0481: BCW index inheritance for constructed affine merge candidate.
* Incorporated JVET-N0334: MV clipping for MMVD, DMVR and CPMV of constructed affine merge candidadates.
* Incorporated JVET-N0407: Disable 8×8/4×N CUs for DMVR.
* Incorporated JVET-N0178: Implicitly split BDOF application region along 16×16 boundaries.
* Incorporated JVET-N0146: Disabling BDOF for BCW and WP, and align the condidtion of DMVR and BIO.
* Incorporated JVET-N0444: Condition check of block height is not equal to 4 for BDOF.
* Incorporated JVET-N0325: 8-bit fixed precision for BDOF calculations.
* Incorporated JVET-N0127: MMVD enabling signalling in SPS
* Incorporated JVET-N0332: Disable MVD scaling for MMVD for LTRP.
* Incorporated JVET-N0448/JVET-N0380: infer the number of MMVD base merge candidate to 1.
* Incorporated JVET-N0302: CIIP with position-independent weights.
* Incorporated JVET-N0324: Signal a regular merge flag right after the merge flag and skip flag.
* Incorporated JVET-N0851/N0447/N0400/N0500: align triangle merge candidate number and regular merge candidate number.
* Incorporated JVET-N0340: Regular merge candidate list is re-used for triangle.
* Incorporated JVET-N0483: Disable sub-block transform when triangle mode is used.
* Incorporated JVET-M0335: Round MVs toward zero.
* Incorporated JVET-N0235: Add an SPS-level flag to turn on and off symmetric MVD
* Incorporated JVET-N0213: Remove TMVP from merge and AMVP list for 4×8 and 8×4 CUs
* Incorporated JVET-N0868: Revised spec text for DMVR reconciling with software ticket [#214](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/214)
* Fixed minor bugs and typos reported in [#94](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/94), [#221](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/221), [#227](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/227), [#248](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/248).
* Fixed bug [#232](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/232): Mismatch between specification and software for triangle merge mode weighted prediction.
* Fixed minor bugs and typos reported in [#249](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/249), [#253](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/253), [#256](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/256).
* Fixed bug [#247](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/247): Fix of size constraint for triangle merge mode when MMVD is disabled in SPS (also in VTM).
* Incorporated JVET-M0471/N0473: Deblocking with long tap filters (including ISP and SBT transform boundaries).
* Incorporated JVET-M0908: Deblocking of CIIP boundaries.
* Incorporated JVET-M0277: Apply pcm\_loop\_filter\_disabled\_flag for ALF.
* Incorporated JVET-N0242: Non-linear ALF with clipping.
* Incorporated fixes for JVET-N0278: Layer concept without inter-layer referencing.
* Fixed some bugs related to reference picture list, tiles and bricks (including the bug reported in [#261](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/261)), presence of the VUI syntax in the SPS, CABAC initialization, HRD, semantics of the picture timing SEI message, etc.
* Incorporated JVET-N0415: CTU-adaptive ALF with fixed filter.
* Incorporated JVET-N0180: ALF line buffer reduction.
* Incorporated JVET-N0220: Simplification of LMCS and other fixes.
* Incorporated JVET-N0477: LMCS with no chroma residue scaling in case of CBF=0.
* Incorporated JVET-N0383/N0251: IBC search range constraint.
* Incorporated JVET-N0175/N0251/N0384: IBC search range increase for CTUs less than 128×128.
* Incorporated JVET-N0843: IBC motion vector prediction unification for merge and MVP mode.
* Incorporated JVET-N0317: Put (0,0) vector as default in IBC merge list.
* Incorporated JVET-N0318/N0427: Disable IBC for 128×128 blocks.
* Fixed bug [#262](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/262): Missing text for IBC with shared merge list.
* Incorporated JVET-N0185: Unified MPM list for intra mode coding.
* Incorporated JVET-N0137: Disable 2×2/4×2/2×4 in dual tree only.
* Incorporated JVET-N0308: Restrict the maximum CU size for ISP to be 64×64.
* Incorporated JVET-N0435: Harmonization between WAIP and intra smoothing filters.
* Incorporated JVET-N0271: CCLM derived from four neighbouring samples.
* Fixed bug [#223](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/223): Fixed ISP interpolation filter conditions.
* Fixed bug [#165](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/165): Removed top-left sample availability derivation process in for CCLM.
* Added SPS flag for ISP.
* Added SPS flag for MRL.
* Incorporated JVET-N0217: Matrix-based intra prediction (MIP).
* Fixed bug [#276](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/276): Added regular\_merge\_flag to context derivation table.
* Fixed bug [#277](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/277): Replaced all occurances of slice-level slice\_loop\_filter\_across\_slices\_enabled\_flag with PPS loop\_filter\_across\_slices\_enabled\_flag.
* Incorporated JVET-N0470: Consider SMVD flag for BDOF condition.
* Added SPS flag for MIP.
* Fixed bugs [#279](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/279), [#280](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/280), [#282](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/282), [#283](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/283), [#284](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/284).
* Incorporated JVET-N0054: Joint coding of chrominance residuals.
* Incorporated JVET-N0188: Unified rice parameter derivation for coefficient level coding.
* Incorporated JVET-N0492: Chroma TU CBF dependent luma TU CBF signalling.
* Incorporated JVET-N0103: Determine coefficient group size based on TB size instead of colour component.
* Incorporated JVET-N0600: Context reduction for amvr\_flag syntax.
* Incorporated JVET-N0286: Context reduction for bcw\_idx syntax.
* Incorporated JVET-N0194: Context selection of last x/y syntax based on non-reduced TU size in case of zero-out.
* Incorporated JVET-N0280: Transform skip residual coding.
* Incorporated JVET-N0413: Block-based quantized residual domain DPCM (BDPCM).
* Fixed bug [#292](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/292): Minor transform skip residual coding issues.
* Fixed bug [#270](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/270): (Bug fix) Changed nuh\_layer\_id to nuh\_layer\_id\_plus1.
* Fixed bug [#298](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/298): typo in IBC BV candidate list derivation.
* Fixed bug [#275](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/275): Wrong bdofUtilizationFlag indexing.
* Fixed bug [#274](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/274): Undefined variables sGxGym and sGxGys.
* Fixed bug [#273](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/273): Wrong HMVP table size.
* Fixed bug [#299](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/299): Minor intra prediction issues.
* Fixed bug [#302](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/302): Fix CBF flags syntax and semantics for implicit TU split.
* Fixed bugs [#307](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/307), [#308](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/308), [#316](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/316): Deblocking issues.
* Fixed the bug that DeltaPocMsbCycleLt was used without being defined.
* Incorporated JVET-N0193: Low frequency non-separable transform (LFNST).
* Incorporated JVET-N0105: Remove mode dependency in LFNST ctx derivation.
* Incorporated JVET-N0866: Unification of implicit transform selection for ISP and SBT.
* Incorporated JVET-N0246: Include square root of 2 factor in levelScale values.
* Incorporated JVET-N0671: Support of 4:4:4 and 4:2:2 chroma formats.
* Incorporated JVET-N0225: Separate colour plane id.
* Incorporated JVET-N0847: Default and user-defined scaling matrices
* Incorporated DMVR fixes and editorial improvements.
* Fixed bug [#245](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/245): Wrong handling of boundary partitioning for a corner case.
* Fixed bug [#341](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/341): Coefficients outside the first coefficient group are considered in 8×8 LFNST.

Draft 4 of Versatile Video Coding.

Ed. Notes:

* Incorporated JVET-M0102: Intra subpartitions (ISP).
* Incorporated JVET-M0142: Chroma format dependent CCLM downsampling filter.
* Incorporated JVET-M0064: Table reduction in CCLM model parameter calculation.
* Incorporated JVET-M0092: Intra reference sample filtering cleanup.
* Incorporated JVET-M0238: PDPC linear interpolation on the secondary boundary for adjacent angular modes is changed to nearest neighbour.
* Incorporated JVET-M0407: CPR search range.
* Fixed bug [#154](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/154) Availability check for CPR/IBC chroma CU reference block is missing.
* Aligned PDPC filtering for INTRA\_ANGULAR18 and INTRA\_ANGULAR50 to VTM.
* Fixed bug [#167](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/167) on PDPC size condition.
* Incorporated JVET-M0497: Fast DST-7/DCT-8.
* Incorporated JVET-M0303: Shape adaptive transform selection.
* Incorporated JVET-M0464/JVET-M0201: Combined transform skip(TS) and MTS syntax plus extended TS sizes.
* Incorporated JVET-M0297: Zero-out of last 16 samples for 32 samples DST-7/DCT-8.
* Incorporated JVET-M0140: Subblock transform for inter CUs.
* Incorporated JVET-M0251/M0257: Zero-out of last 32 samples for 64 samples DCT-2 fix.
* Incorporated JVET-M0273/M0240/M0116/M0338/M0204: only using left neighbour for SbTMVP fetching
* Incorporated JVET-M0246: AMVR for affine
* Incorporated JVET-M0145: affine sub-block MV clipping
* Incorporated JVET-M0166/M0228/M0477: remove MV comparison for constructed merge candidates
* Incorporated JVET-M0170: Parallel processing for merge mode
* Incorporated JVET-M0147: Decoder side motion vector refinement
* Incorporated JVET-M0361: Fix of cu\_coded\_flag for merge mode
* Incorporated JVET-M0487: Using integer samples instead of bilinear interpolation for extended region of BDOF.
* Incorporated JVET-M0483: IBC signalled as a separate CU prediction mode.
* Incorporated JVET-M0063: Generalization of BDOF bit-depth.
* Incorporated triangular modifications including:
  + JVET-M0118/M0185/M0190/M207(test 1)/M0216(the first aspect)/M0234 (change corresponding to the result table 7 and 8)/M0317(section 2.2)/M0328: Do not signal the triangular prediction mode flag in cases where the combination is not allowed (MMVD, CIIP),
  + JVET-M0328: always use second weight group in triangular prediction.
* Incorporated JVET-M0883: Signalling change of triangular merging candidate which does not need LUT.
* Incorporated JVET-M0193: Pairwise average merging candidate reduction.
* Incorporated HMVP modifications including:
  + JVET-M0436 reduce HMVP number from 6 to 5,
  + JVET-M0300 HMVP initialization for parallel processing with tiles,
  + JVET-M0264 BCW weight is also stored in HMVP,
  + JVET-M0126 reduced HMVP candidate pruning.
* Incorporated JVET-M0255: MMVD mode without fractional sample offsets for screen content coding.
* Incorporated JVET-M0171/M0068: remove redundant MV scaling in MMVD.
* Incorporated JVET-M0444: Symmetrical MVD coding for L0 to L1.
* Incorporated JVET-M0479: MV clipping to 18 bits.
* Incorporated JVET-M0512: TMVP storage reduction.
* Incorporated JVET-M0192: Subblock chroma MV derivation for affine from two luma MVs.
* Incorporated JVET-M0111: Weighted prediction (WP) and disable BCW signalling if WP is enabled.
* Incorporated JVET-M0281/M0117: modified AMVP pruning with rounding.
* Fixed bug [#175](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/175) incorrect derivation of CCLM parameter b.
* Incorporated JVET-M0128: Reference picture management
* Incorporated JVET-M0132: Adaptation parameter set (APS)
* Incorporated JVET-M0853: Adding the support of rectangular slices in addition to the existing raster-scan slices, and enabling extraction of MCTSs without changing VLC NAL units
* Incorporated JVET-M0160: Adding loop\_filter\_across\_slice\_enabled\_flag to the PPS
* Incorporated JVET-M0101:
  + Replace the existing IRAP\_NUT with 3 new NAL unit types: IDR\_W\_RADL, IDR\_N\_LP, CRA\_NUT (from JVET-M0101).
  + Add external means flag HandleCraAsCvsStartFlag, with similar text as in HEVC. Text provided in a v3 of JVET-M0101.
  + Add a NUT value for step-wise temporal access STSA (from JVET-M0101).
  + Add a NUT value for AUD (from JVET-M0101).
  + Add sps\_max\_sublayers\_minus1 syntax element to SPS, and decoding process in 8.1.1, 8.1.2 and 8.1.3 of JVET-M0101.
  + Add text of sections 7.4.2.4 to 7.4.2.4.5 on NAL unit order and AU boundary detection from JVET-M0101, which is primarily editorial, but has some technical aspects.
  + Add profile\_tier\_level( ) syntax structure which includes sublayer level idc (similar to HEVC but without sublayer-specific profiles).
  + Add general\_non\_packed\_constraint\_flag with semantics as in JVET-M0101 (rename the flag to display\_suitability\_flag? – that's editorial).
  + Add the temporal scalability sub-bitstream extraction process in JVET-M0101.
  + Add RASL and RADL NUTs
* Incorporated JVET-M0451: Add new constraint flags corresponding to VVC WD 3 tools.
* Incorporated JVET-M0415: Change the sps\_ref\_wraparound\_offset to sps\_ref\_wraparound\_offset\_minus1 and changing the units to be MinCbSizeY as in option 1 (minor cleanup).
* Incorporated JVET-M0381: Reduce merge idx ctx coded bins (test 2.2.2a).
* Incorporated JVET-M0502: Add one context for pred\_mode\_flag (method 2).
* Incorporated JVET-M0453: Modified CABAC probability estimation (5.1.13\* + init from 5.1.2).
* Fixed bug [#147](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/147) on coefficient coding.
* Incorporated JVET-M0470: Limited EGk for abs\_rem/ dec\_abs\_level.
* Incorporated JVET-M0173: Move rem\_abs\_gt3\_flag into first coding pass.
* Incorporated JVET-M0119: Modified dequantization scaling for TS.
* Incorporated JVET-M0685: QP prediction fix for parallel encoding.
* Incorporated JVET-M0113/M0188: Bug fix for quantization group QP signalling.
* Incorporated JVET-M0421: Split-first signalling for partitioning.
* Incorporated JVET-M0446/M0888/M0905: Inferred QT split to avoid 32×128/128×32 partitions at picture boundaries.
* Incorporated JVET-M0427: Picture reconstruction with luma mapping and chroma scaling (LMCS).

Draft 3 of Versatile Video Coding.

Ed. Notes:

* Fixed bug [#92](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/92) missing MV clipping in interplolation process.
* Incorporated JVET-L0664: disabling 5x5 ALF for luma component.
* Incorporated JVET-L0082/JVET-L0083: reduction of bits for ALF coefficients.
* Incorporated JVET-L0147: subsampling of ALF classifiers.
* Incorporated SAO as found in HEVC.
* Incorporated deblocking filter as found in HEVC with the following modifications:
  + adapt processing to dual tree CTU partitioning,
  + replace RQT based transform block processing with implicit TU split for large blocks (JVET-K0307, JVET-K0237, JVET-K0369, JVET-K0232, JVET-K0315),
  + replace prediction blocks with coding subblocks (JVET-L0074),
  + only apply deblocking on transform block / coding subblock edges if CU is aligned on 8×8 grid in the respecitive direction.
* Incorporated JVET-L0410: tc table fix
* Incorporated JVET-L0414: Luma adaptive deblocking filter.
* Fixed bug [#97](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/97) mismatch between agreed ALF text JVET-K0564 and draft by removing alf\_chroma\_ctb\_present\_flag.
* Incorporated JVET-L0285: 8-bit transform matrix.
* Incorporated JVET-L0118: MTS index fix.
* Incorporated JVET-L0059: Remove dependency on number non-zero coeff. for MTS index signalling.
* Incorporated JVET-L0362: QP signalling.
* Incorporated JVET-L0428: Delta QP and chroma QP offset for dual tree.
* Incorporated JVET-K0251: Increase maximum QP value from 51 to 63 (including fix from JVET-L0553).
* Incorporated JVET-L0217: Fix relation between QT/BT/TT syntax elements.
* Incorporated JVET-L0678: QT/BT/TT syntax overriding in slice header.
* Incorporated JVET-L0081: BT/TT constraint.
* Incorporated JVET-L0191: CCLM simplification using min and max values.
* Incorporated JVET-L0136: CCLM with line buffer restriction.
* Incorporated JVET-L0340: Multi-directional CCLM.
* Incorporated JVET-L0053/JVET-L0272: modified chroma direct mode.
* Incorporated JVET-L0628: Intra 4-tap interpolation filter.
* Incorporated JVET-L0165: Intra 6 MPMs.
* Incorporated JVET-L0283: Multi-line intra prediction.
* Incorporated JVET-L0279: Unification of intra angular prediction.
* Incorporated PCM mode from HEVC and JVET-L0209: PCM mode with dual tree partition.
* Incorporated JVET-L0694: Combination of affine mode and subblock temporal merging candidate modifications including:
  + JVET-L0142/JVET-L0366/JVET-L0632: affine merge candidate list (CE4 4.2.6.d as modifed in L0632).
  + JVET-L0369: moving subblock temporal merging candidate into affine merge candidate list (CE4 4.2.8).
  + JVET-L0045: line buffer reduction for affine inherited candidates, location 1 (CE4 4.1.11.a).
  + JVET-L0047: modification of affine control point motion vector storage (method 1).
  + JVET-L0271: Simplification of affine motion vector predictor candidate list construction (CE4 4.1.6.a).
* Incorporated JVET-L0265: Change chroma subblock size to 4×4 instead of 2×2 for affine motion compensation.
* Incorporated JVET-L0198: Fix subblock size to 8×8 for subblock TMVP (JVET-L0468/JVET-L0104).
* Incorporated JVET-L0198: Use first spatial neighbouring MV for collocated subblock TMVP position
* Incorporated JVET-L0055: Restrict subblock TMVP to CUs with width >= 8 and height >=8.
* Fixed bug [#120](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/120) clipping in vertical subblock TMVP location goes beyond the current CTU row.
* Incorporated JVET-L0274: Reduce number of context coded bins (CE7 7.1.3.b).
* Incorporated missing context derivation for all CABAC-coded syntax elements.
* Incorporated JVET-L0361: Context modeling of CU split modes.
* Incorporated JVET-L0194: Use one context for the first bin of merge\_idx and bypass coding for the others.
* Incorporated JVET-L0266: History-based motion vector prediction.
* Incorporated JVET-L0158: Reset the HMVP FIFO list for each CTU row.
* Incorporated JVET-L0104: Prohibit 4×4 bi-prediction for inter CU.
* Incorporated JVET-L0054: Merge with MVD (MMVD) (CE4 4.5.4.b).
* Incorporated JVET-L0100: Combined inter merge / intra prediction (CIIP).
* Incorporated JVET-L0090: Pairwise average merging candidates.
* Incorporated JVET-L0646: Biprediction with CU weights.
* Incorporated JVET-L0256: Bidirectional optical flow (BDOF).
* Incorporated JVET-L0231: Horizontal MV wrap-around.
* Fixes and cleanups:
  + fixed minor typos,
  + resolved open issues from editors notes,
  + reordered SPS flags to be more aligned with VTM,
  + put all skip/merge related CU syntax in a merge\_data( ) syntax structure (editorial),
  + fixed order of merge syntax by moving merge\_idx and after combined merge/intra syntax,
  + conditioned combined merge/intra syntax on sps\_ciip\_enabled\_flag.
* Incorporated JVET-L0124: Triangular inter-picture prediction mode.
* Incorporated JVET-L0293: Current picture referencing (CPR).
* Fixed bug [#142](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/142) corrected order of derivation process for subblock-based temporal merging base motion data.
* Fixed bug [#137](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/137) wrong CCLM parameter derivation condition.
* Fixed issues related to triangular partitions.
* Incorporated JVET-L0064: Agreement in principle on the need of CRA and its signalling in NAL unit header
* Incorporated JVET-L0248: A restriction on the TemporalId values of the current picture and the active PPS
* Incorporated JVET-L0249: POC signalling and derivation
* Incorporated JVET-L0449: POC LSB in the slice headers for all picture types
* Incorporated JVET-L0686: Tiling and tiling grouping; removed traditional slices
* Incorporated JVET-L0696: Interoperability point and constraint flags

Draft 2 of Versatile Video Coding.

Ed. Notes:

* Incorporated JVET-K0230: Separate trees for intra slices (without multi-DMs) with an implicit split to 64×64;
* Incorporated JVET-K0556: Prohibit ternary split of something bigger than 64 in width or height (and not send the bit to indicate ternary type at that level).
* Incorporated JVET-K0351 (test c): Keep only the TT restriction (preventing binary split with same orientation in center partition of the ternary split)
* Incorporated JVET-K0554: Implicit splitting at picture boundaries and ensure MinQTSize at boundary splits
* Fixed bug [#65](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/65) typos and unused variables in section 6.4
* Fixed bug [#67](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/67) implicit vertical BT split at picture boundary issue
* Incorporated JVET-K0072: Dependent quantization with fallback switch at the picture level and modified entropy coding supporting dependent quantization including:
  + adapted scaling to non-square transform blocks,
  + added binarization process for abs\_remainder,
  + specified CoeffMin and CoeffMax with fixed values,
  + added 0-th order Exp-Golomb code parsing process.
* Incorporated JVET-K0310: Sign data hiding (can only be used when dependent quantization is disabled).
* Incorporated JVET-K0529: Intra prediction using 3MPM on 67 prediction modes (Planar, DC and 65 angular modes)
* Incorporated JVET-K0122: DC prediction without division.
* Incorporated JVET-K0500: Wide-angle intra prediction.
* Incorporated JVET-K0063: Position-dependent intra prediction combination
* Incorporated JVET-K0190: Cross-component linear model intra prediction
* Incorporated multiple transform selection (MTS) for both intra and inter, each controlled by an SPS flag.
* Incorporated transform skip.
* Fixed bug [#68](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/68) various typos
* Fixed bug [#71](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/71) various typos
* Fixed bug [#72](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/72) on CCLM
* Incorporated JVET-K0357: adaptive motion vector resolution (AMVR)
* Incorpor,ated JVET-K0565: affine motion compensation (MC) including:
  + JVET-K0052: Affine merge bug fix
  + JVET-K0184: Affine MC (CE4.1.1a 4×4 fixed subblock size).
  + JVET-K0337: Affine MC coding and models (4.1.3a, affine MVP list construction, and 4.1.3b, MV difference coding, and 4.1.3c, 4/6 parameter model, no slice level switch).
  + JVET-K0367/JVET-K0052/JVET-K0103: Restriction of affine merge mode to CU sizes >= 8×8
* Incorporated 1/16 motion compensation (MC) including:
  + 1/16 MV storage
  + 1/16 merge and affine MVs
  + MVDs in AMVR accuracy (1/4,1,4) shifted to 1/16
  + Inter MVP candidates rounded to AMVR accuracy (1/4,1,4) and shifted to 1/16
  + 1/16 luma and 1/32 chroma interpolation filters
* Incorporated subblock-based temporal merging candidates with 8×8 motion vector storage (JVET-K0346).
* Incorporated JVET-K0371: 4×4 block classification based Adaptive Loop Filter (ALF).
* Fixed bug [#75](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/75) regarding a bottom and right boundary partition issue.
* Fixed bug [#90](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/90) typos in copying the control point vectors to temporal notion vectors.
* Fixed bug [#86](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/86) in intra reference sample filtering.
* Incorporated JVET-K0325: High Level Syntax (HLS) starting point.
* Fixed bug [#82](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/82) on zeroing-out high frequency transform coefficients for larger TUs (>32×32).
* Fixed bug [#85](https://jvet.hhi.fraunhofer.de/trac/vvc/ticket/85) on MTS index coding.

Draft 1 of Versatile Video Coding.

Ed. Notes:

* Incorporated basic definitions, abbreviations and conventions
* Incorporated a basic high-level syntax (HLS) with NAL units, SPS, PPS and slice header.
* Incorporated block partitioning by a quadtree with nested multi-type tree using binary and ternary splits with
  + CU leaf nodes
  + Prediction at CU level
  + Transform at CU level
  + Minimum CU size with 4×4 luma coding block and corresponding chroma coding blocks (2×2 for 4:2:0)
  + Maximum TU size with 64×64 luma transform block and corresponding chroma transform blocks (32×32 for 4:2:0)
  + Minimum TU size with 4×4 luma transform block and corresponding chroma transform blocks (2×2 for 4:2:0)
  + Single tree for luma and chroma

**CONTENTS**

*Page*

Abstract i

Introduction 1

Purpose 1

Profiles, tiers, and levels 1

Overview of the design characteristics 1

How to read this document 2

1 Scope 2

2 Normative references 3

2.1 Identical Recommendations | International Standards 3

2.2 Paired Recommendations | International Standards equivalent in technical content 3

2.3 Additional references 3

3 Definitions 3

4 Abbreviations 11

5 Conventions 14

5.1 General 14

5.2 Arithmetic operators 14

5.3 Logical operators 14

5.4 Relational operators 14

5.5 Bit-wise operators 14

5.6 Assignment operators 15

5.7 Range notation 15

5.8 Mathematical functions 15

5.9 Order of operation precedence 16

5.10 Variables, syntax elements and tables 17

5.11 Text description of logical operations 18

5.12 Processes 19

6 Bitstream and picture formats, partitionings, scanning processes and neighbouring relationships 20

6.1 Bitstream formats 20

6.2 Source, decoded and output picture formats 20

6.3 Partitioning of pictures, subpictures, slices, tiles, and CTUs 22

6.3.1 Partitioning of pictures into subpictures, slices, and tiles 22

6.3.2 Block, quadtree and multi-type tree structures 25

6.3.3 Spatial or component-wise partitionings 25

6.4 Availability processes 26

6.4.1 Allowed quad split process 26

6.4.2 Allowed binary split process 26

6.4.3 Allowed ternary split process 28

6.4.4 Derivation process for neighbouring block availability 29

6.5 Scanning processes 29

6.5.1 CTB raster scanning, tile scanning, and subpicture scanning processes 29

6.5.2 Up-right diagonal scan order array initialization process 33

6.5.3 Horizontal and vertical traverse scan order array initialization process 33

7 Syntax and semantics 35

7.1 Method of specifying syntax in tabular form 35

7.2 Specification of syntax functions and descriptors 36

7.3 Syntax in tabular form 37

7.3.1 NAL unit syntax 37

7.3.2 Raw byte sequence payloads, trailing bits and byte alignment syntax 37

7.3.3 Profile, tier, and level syntax 56

7.3.4 DPB parameters syntax 58

7.3.5 HRD parameters syntax 58

7.3.6 Supplemental enhancement information message syntax 60

7.3.7 Slice header syntax 60

7.3.8 Reference picture lists syntax 63

7.3.9 Reference picture list structure syntax 64

7.3.10 Slice data syntax 64

7.4 Semantics 86

7.4.1 General 86

7.4.2 NAL unit semantics 86

7.4.3 Raw byte sequence payloads, trailing bits and byte alignment semantics 93

7.4.4 Profile, tier, and level semantics 138

7.4.5 DPB parameters semantics 142

7.4.6 HRD parameters semantics 142

7.4.7 Supplemental enhancement information message semantics 146

7.4.8 Slice header semantics 146

7.4.9 Reference picture lists semantics 155

7.4.10 Reference picture list structure semantics 156

7.4.11 Slice data semantics 157

8 Decoding process 178

8.1 General decoding process 178

8.2 NAL unit decoding process 180

8.3 Slice decoding process 180

8.3.1 Decoding process for picture order count 180

8.3.2 Decoding process for reference picture lists construction 181

8.3.3 Decoding process for reference picture marking 185

8.3.4 Decoding process for generating unavailable reference pictures 185

8.3.5 Decoding process for symmetric motion vector difference reference indices 186

8.3.6 Decoding process for collocated picture and no backward prediction 187

8.4 Decoding process for coding units coded in intra prediction mode 187

8.4.1 General decoding process for coding units coded in intra prediction mode 187

8.4.2 Derivation process for luma intra prediction mode 189

8.4.3 Derivation process for chroma intra prediction mode 191

8.4.4 Cross-component chroma intra prediction mode checking process 192

8.4.5 Decoding process for intra blocks 193

8.5 Decoding process for coding units coded in inter prediction mode 224

8.5.1 General decoding process for coding units coded in inter prediction mode 224

8.5.2 Derivation process for motion vector components and reference indices 228

8.5.3 Decoder-side motion vector refinement process 247

8.5.4 Derivation process for geometric partitioning mode motion vector components and reference indices 252

8.5.5 Derivation process for subblock motion vector components and reference indices 253

8.5.6 Decoding process for inter blocks 278

8.5.7 Decoding process for geometric partitioning mode inter blocks 299

8.5.8 Decoding process for the residual signal of coding blocks coded in inter prediction mode 305

8.6 Decoding process for coding units coded in IBC prediction mode 306

8.6.1 General decoding process for coding units coded in IBC prediction mode 306

8.6.2 Derivation process for block vector components for IBC blocks 308

8.6.3 Decoding process for IBC blocks 311

8.7 Scaling, transformation and array construction process 312

8.7.1 Derivation process for quantization parameters 312

8.7.2 Scaling and transformation process 314

8.7.3 Scaling process for transform coefficients 315

8.7.4 Transformation process for scaled transform coefficients 317

8.7.5 Picture reconstruction process 337

8.8 In-loop filter process 340

8.8.1 General 340

8.8.2 Picture inverse mapping process for luma samples 340

8.8.3 Deblocking filter process 341

8.8.4 Sample adaptive offset process 366

8.8.5 Adaptive loop filter process 368

9 Parsing process 379

9.1 General 379

9.2 Parsing process for k-th order Exp-Golomb codes 380

9.2.1 General 380

9.2.2 Mapping process for signed Exp-Golomb codes 381

9.3 CABAC parsing process for slice data 382

9.3.1 General 382

9.3.2 Initialization process 384

9.3.3 Binarization process 407

9.3.4 Decoding process flow 417

Annex A Profiles, tiers and levels 434

A.1 Overview of profiles, tiers and levels 434

A.2 Requirements on video decoder capability 434

A.3 Profiles 434

A.3.1 Main 10 and Main 10 Still Picture profiles 434

A.3.2 Main 4:4:4 10 and Main 4:4:4 10 Still Picture profiles 435

A.4 Tiers and levels 436

A.4.1 General tier and level limits 436

A.4.2 Profile-specific level limits 437

A.4.3 Effect of level limits on picture rate (informative) 439

Annex B Byte stream format 443

B.1 General 443

Annex C Hypothetical reference decoder 446

C.1 General 446

C.2 Operation of the CPB 451

C.2.1 General 451

C.2.2 Timing of DU arrival 451

C.2.3 Timing of DU removal and decoding of DU 453

C.3 Operation of the DPB 456

C.3.1 General 456

C.3.2 Removal of pictures from the DPB before decoding of the current picture 456

C.3.3 Picture output 456

C.3.4 Current decoded picture marking and storage 457

C.4 Bitstream conformance 457

C.5 Decoder conformance 459

C.5.1 General 459

C.5.2 Operation of the output order DPB 460

C.6 Sub-bitstream extraction process 461

C.7 Subpicture sub-bitstream extraction process 462

Annex D Supplemental enhancement information 465

D.1 General 465

D.2 General SEI payload 465

D.2.1 General SEI message syntax 465

D.2.2 General SEI payload semantics 466

D.3 Buffering period SEI message 468

D.3.1 Buffering period SEI message syntax 468

D.3.2 Buffering period SEI message semantics 469

D.4 Picture timing SEI message 472

D.4.1 Picture timing SEI message syntax 472

D.4.2 Picture timing SEI message semantics 474

D.5 DU information SEI message 478

D.5.1 DU information SEI message syntax 478

D.5.2 DU information SEI message semantics 478

D.6 Scalable nesting SEI message 479

D.6.1 Scalable nesting SEI message syntax 479

D.6.2 Scalable nesting SEI message semantics 480

D.7 Subpicture level information SEI message 482

D.7.1 Subpicture level information SEI message syntax 482

D.7.2 Subpicture level information SEI message semantics 482

D.8 Use of ITU-T H.SEI | ISO/IEC 23002-7 VUI parameters 484

D.9 Use of ITU-T H.SEI | ISO/IEC 23002-7 SEI messages 485

D.9.1 General 485

D.9.2 Use of the film grain characteristics SEI message 485

D.9.3 Use of the decoded picture hash SEI message 485

D.9.4 Use of the dependent random access point (DRAP) indication SEI message 485

D.9.5 Use of the equirectangular projection, generalized cubemap projection, and region-wise packing SEI messages 485

D.9.6 Use of the frame-field information SEI message 485

INTERNATIONAL STANDARD

ISO/IEC VVC

ITU-T Rec. H.VVC

ITU-T RECOMMENDATION

Versatile video coding

# Introduction

## Purpose

This Recommendation | International Standard specifies a video coding technology known as *Versatile Video Coding*. It has been drafted by a joint collaborative team of ITU-T and ISO/IEC experts known as the Joint Video Experts Team (JVET), which is a partnership of the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG). It has been designed with two primary goals. The first of these is to specify a video coding technology with a compression capability that is substantially beyond that of the prior generations of such standards, and the second is for this technology to be highly versatile for effective use in a broader range of applications than that addressed by prior standards. Some key application areas for the use of this standard particularly include ultra-high-definition video (e.g., with 3840×2160 or 7620×4320 picture resolution and bit depth of 10 or 12 bits as specified in Rec. ITU-R BT.2100), video with a high dynamic range and wide colour gamut (e.g., with the perceptual quantization or hybrid log-gamma transfer characteristics specified in Rec. ITU-R BT.2100), and video for immersive media applications such as 360° omnidirectional video projected using a common projection format such as the equirectangular or cubemap projection format, in addition to the applications that have commonly been addressed by prior video coding standards.

## Profiles, tiers, and levels

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

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

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

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

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

## 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 may be used to achieve highly efficient compression. Encoding algorithms (not specified within the scope of this Recommendation | International Standard) may select between inter, intra, and intra block copy (IBC) 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 may be specified for a variety of block sizes in the picture. The prediction residual may 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 may 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 (Scope) and moves on to Clause 3 (Terms and definitions). Clause 6 should be read for the geometrical relationship of the source, input, and output of the decoder. Clause 7 (Syntax and semantics) specifies the order to parse syntax elements from the bitstream. See subclauses 7.1 to 7.3 for syntactical order and subclause7.4 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 (Parsing process). Finally, Clause 8 (Decoding process) specifies how the syntax elements are mapped into decoded samples. Throughout reading this document, the reader should refer to Clauses 2 (Normative references), 4 (Abbreviations), and 5 (Conventions) as needed. Annexes A through D also form an integral part of this Recommendation | International Standard.

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

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

# Scope

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

Only the syntax format, semantics, and associated decoding process requirements are specified, while other matters such as pre-processing, the encoding process, system signalling and multiplexing, data loss recovery, post-processing, and video display are considered to be outside the scope of this Recommendation | International Standard. Any encoding process that produces bitstream data that conforms to the specified bitstream syntax format requirements of this Recommendation | International Standard is considered to be in conformance with the requirements of this Recommendation | International Standard. Additionally, the internal processing steps performed within a decoder are also considered to be outside the scope of this Recommendation | International Standard; only the externally observable output behaviour is required to conform to the specifications of this Recommendation | International Standard. The decoding process is specified such that all decoders that conform to a specified combination of capabilities known as the profile, tier, and level will produce numerically identical cropped decoded output pictures when invoking the decoding process associated with that profile for a bitstream conforming to that profile, tier and level. Any decoding process that produces identical cropped decoded output pictures to those produced by the process described herein (with the correct output order or output timing, as specified) is considered to be in conformance with the requirements of this Recommendation | International Standard.

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

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

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

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

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

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

This is the first version of this Recommendation | International Standard.

# Normative references

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

## Identical Recommendations | International Standards

– None

## Paired Recommendations | International Standards equivalent in technical content

– Rec. ITU-T H.SEI | ISO/IEC 23002-7 *Supplemental enhancement information messages for coded video bitstreams*

## Additional references

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

# Definitions

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

* 1. **access unit (AU)**: A set of *PUs* that belong to different *layers* and contain *coded pictures* associated with the same time for output from the *DPB*.
  2. **adaptive loop filter (ALF)**: A filtering process that is applied as part of the *decoding process* and is controlled by parameters conveyed in an *APS*.
  3. **AC transform coefficient**: Any *transform coefficient* for which the *frequency index* in at least one of the two dimensions is non-zero.
  4. **ALF APS**: An *APS* that controls the *ALF* process.
  5. **adaptation parameter set (APS)**: A *syntax structure* containing *syntax elements* that apply to zero or more *slices* as determined by zero or more *syntax elements* found in *slice headers.*
  6. **associated GDR picture (of a particular picture with nuh\_layer\_id equal to a particular value layerId)**: The previous *GDR picture* in *decoding order* with nuh\_layer\_id equal to layerId (when present) between which and the particular *picture* in *decoding order* there is no *IRAP picture* with nuh\_layer\_id equal to layerId.
  7. **associated GDR subpicture (of a particular subpicture with nuh\_layer\_id equal to a particular value layerId and subpicture index equal to a particular value subpicIdx)**: The previous *GDR subpicture* in *decoding order* with nuh\_layer\_id equal to layerId and subpicture index equal to subpicIdx (when present) between which and the particular sub*picture* in *decoding order* there is no *IRAP subpicture* with nuh\_layer\_id equal to layerId and subpicture index equal to subpicIdx.
  8. **associated IRAP picture (of a particular picture** **with nuh\_layer\_id equal to a particular value layerId)**: The previous *IRAP picture* in *decoding order* with nuh\_layer\_id equal to layerId (when present) between which and the particular *picture* in *decoding order* there is no *GDR picture* with nuh\_layer\_id equal to layerId.
  9. **associated IRAP subpicture (of a particular subpicture with nuh\_layer\_id equal to a particular value layerId and subpicture index equal to a particular value subpicIdx)**: The previous *IRAP subpicture* in *decoding order* with nuh\_layer\_id equal to layerId and subpicture index equal to subpicIdx (when present) between which and the particular sub*picture* in *decoding order* there is no *GDR subpicture* with nuh\_layer\_id equal to layerId and subpicture index equal to subpicIdx.
  10. **associated non-VCL NAL unit**: A *non-VCL NAL unit* (when present) for a *VCL NAL unit* where the *VCL NAL unit* is the *associated VCL NAL unit* of the *non-VCL NAL unit*.
  11. **associated VCL NAL unit**: The preceding *VCL NAL unit* in *decoding order* for a *non-VCL NAL unit* with nal\_unit\_type equal to EOS\_NUT, EOB\_NUT, SUFFIX\_APS\_NUT, SUFFIX\_SEI\_NUT, FD\_NUT, RSV\_NVCL\_27, or in the range of UNSPEC\_30..UNSPEC\_31; or otherwise the next *VCL NAL unit* in *decoding order*.
  12. **bin**: One bit of a *bin string*.
  13. **binarization**: A set of *bin strings* for all possible values of a *syntax element*.
  14. **binarization process**: A unique mapping process of all possible values of a *syntax element* onto a set of *bin strings*.
  15. **binary split**: A split of a rectangular MxN *block* of samples into two *blocks* where a vertical split results in a first (M / 2)xN *block* and a second (M / 2)xN *block*, and a horizontal split results in a first Mx(N / 2) *block* and a second Mx(N / 2) *block*.
  16. **bin string**: An intermediate binary representation of values of *syntax elements* from the *binarization* of the *syntax element*.
  17. **bi-predictive (B) slice**: A *slice* that is decoded using *intra* *prediction* or using *inter prediction* with at most two *motion vectors* and *reference indices* to *predict* the sample values of each *block*.
  18. **bitstream**: A sequence of bits, in the form of a *NAL unit stream* or a *byte stream*, that forms the representation of a sequence of *AUs* forming one or more coded video sequences *(CVSs)*.
  19. **block**: An MxN (M-column by N-row) array of samples, or an MxN array of *transform coefficients*.
  20. **block vector**: A two-dimensional vector used for *IBC prediction* that provides an offset from the coordinates of the current *coding block* to the coordinates of the prediction block in the same *decoded picture*.
  21. **byte**: A sequence of 8 bits, within which, when written or read as a sequence of bit values, the left-most and right-most bits represent the most and least significant bits, respectively.
  22. **byte-aligned**: A position in a *bitstream* is byte-aligned when the position is an integer multiple of 8 bits from the position of the first bit in the *bitstream*, and a bit or *byte* or *syntax element* is said to be byte-aligned when the position at which it appears in a *bitstream* is byte-aligned.
  23. **byte stream**: An encapsulation of a *NAL unit stream* containing *start code prefixes* and *NAL units* as specified in Annex B.
  24. **can**: A term used to refer to behaviour that is allowed, but not necessarily required*.*
  25. **chroma**: An adjective, represented by the symbols Cb and Cr, specifying that a sample array or single sample is representing one of the two colour difference signals related to the primary colours.

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

* 1. **clean random access (CRA) PU**: A *PU* in which the *coded picture* is a *CRA picture*.
  2. **clean random access (CRA) picture**: An *IRAP picture* for which each *VCL NAL unit* has nal\_unit\_type equal to CRA\_NUT.

NOTE – A CRA picture does not use inter prediction in its decoding process, and may be the first picture in the bitstream in decoding order, or may appear later in the bitstream. A CRA picture may 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 may not be decodable, as they may contain references to pictures that are not present in the bitstream.

* 1. **clean random access (CRA) subpicture**: An *IRAP subpicture* for which each *VCL NAL unit* has nal\_unit\_type equal to CRA\_NUT.
  2. **coded layer video sequence (CLVS)**: A sequence of *PUs* with the same value of nuh\_layer\_id that consists, in *decoding order*, of a *CLVSS PU*, followed by zero or more *PUs* that are not *CLVSS PUs*, including all subsequent *PUs* up to but not including any subsequent *PU* that is a *CLVSS PU*.

NOTE – A CLVSS PU may 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 HandleCraAsCvsStartFlag 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 decoding order.

* 1. **coded layer video sequence start (CLVSS) PU**: A *PU* in which the *coded picture* is a *CLVSS picture*.
  2. **coded layer video sequence start (CLVSS) picture**: A *coded picture* that is an *IRAP picture* with NoOutputBeforeRecoveryFlag equal to 1 or a *GDR picture* with NoOutputBeforeRecoveryFlag equal to 1.
  3. **coded picture**: A *coded representation* of a *picture* comprising *VCL NAL units* with a particular value of nuh\_layer\_id within an *AU* and containing all *CTUs* of the *picture*.
  4. **coded picture buffer (CPB)**: A first-in first-out buffer containing *DUs* in *decoding order* specified in the *hypothetical reference decoder* in Annex C.
  5. **coded representation**: A data element as represented in its coded form.
  6. **coded video sequence (CVS)**: A sequence of *AUs* that consists, in *decoding order*, of a *CVSS AU*, followed by zero or more *AUs* that are not *CVSS AUs*, including all subsequent *AUs* up to but not including any subsequent *AU* that is a *CVSS AU*.
  7. **coded video sequence start (CVSS) AU**: An *AU* in which there is a *PU* for each layer in the CVS and the *coded picture* in each *PU* is a *CLVSS picture*.
  8. **coding block**: An MxN *block* of samples for some values of M and N such that the division of a *CTB* into *coding blocks* is a *partitioning*.
  9. **coding tree block (CTB)**: An N×N *block* of samples for some value of N such that the division of a *component* into *CTBs* is a *partitioning*.
  10. **coding tree unit (CTU)**: A *CTB* of *luma* samples, two corresponding *CTBs* of *chroma* samples of a *picture* that has three sample arrays, or a *CTB* of samples of a monochrome *picture* or a *picture* that is coded using three separate colour planes and *syntax structures* used to code the samples.
  11. **coding unit (CU)**: A *coding block* of *luma* samples, two corresponding *coding blocks* of *chroma* samples of a *picture* that has three sample arrays, or a *coding block* of samples of a monochrome *picture* or a *picture* that is coded using three separate colour planes and *syntax structures* used to code the samples. [Ed. (BB): Update for dual tree case.]
  12. **component**: An array or single sample from one of the three arrays (*luma* and two *chroma*) that compose a *picture* in 4:2:0, 4:2:2, or 4:4:4 colour format or the array or a single sample of the array that compose a *picture* in monochrome format.
  13. **context variable**: A variable specified for the *adaptive binary arithmetic decoding* *process* of a *bin* by an equation containing recently decoded *bins*.
  14. **deblocking filter**: A filtering process that is applied as part of the *decoding process* in order to minimize the appearance of visual artefacts at the boundaries between *blocks*.
  15. **decoded picture**: A *picture* produced by applying the *decoding process* to a *coded picture*.
  16. **decoded picture buffer (DPB)**: A buffer holding *decoded pictures* for reference, output reordering, or output delay specified for the *hypothetical reference decoder*.
  17. **decoder**: An embodiment of a *decoding process*.
  18. **decoding order**: The order in which *syntax elements* are processed by the *decoding process*.
  19. **decoding process**: The process specified in this Specification that reads a *bitstream* and derives *decoded* *pictures* from it.
  20. **decoding unit (DU)**: An *AU* if DecodingUnitHrdFlag is equal to 0 or a subset of an *AU* otherwise, consisting of one or more *VCL NAL units* in an *AU* and the *associated non-VCL NAL units*.
  21. **emulation prevention byte**: A *byte* equal to 0x03 that is present within a *NAL unit* when the *syntax elements* of the *bitstream* form certain patterns of *byte* values in a manner that ensures that no sequence of consecutive *byte-aligned* *bytes* in the *NAL unit* can contain a *start code prefix*.
  22. **encoder**: An embodiment of an *encoding process*.
  23. **encoding process**: A process not specified in this Specification that produces a *bitstream* conforming to this Specification.
  24. **filler data NAL units**: *NAL units* with nal\_unit\_type equal to FD\_NUT.
  25. **flag**: A variable or single-bit *syntax element* that can take one of the two possible values: 0 and 1.
  26. **frequency index**: A one-dimensional or two-dimensional index associated with a *transform coefficient* prior to the application of a *transform* in the *decoding process.*
  27. **gradual decoding refresh (GDR) AU**: An *AU* in which there is a PU for each layer in the CVS and the *coded picture* in each present *PU* is a *GDR picture*.
  28. **gradual decoding refresh (GDR) PU**: A *PU* in which the *coded picture* is a *GDR picture*.
  29. **gradual decoding refresh (GDR) picture:** A *picture* for which each VCL NAL unit has nal\_unit\_type equal to GDR\_NUT.
  30. **gradual decoding refresh (GDR) subpicture:** A *subpicture* for which each VCL NAL unit has nal\_unit\_type equal to GDR\_NUT.
  31. **hypothetical reference decoder (HRD)**: A hypothetical *decoder* model that specifies constraints on the variability of conforming *NAL unit streams* or conforming *byte streams* that an encoding process may produce.
  32. **hypothetical stream scheduler (HSS)**: A hypothetical delivery mechanism used for checking the conformance of a *bitstream* or a *decoder* with regards to the timing and data flow of the input of a *bitstream* into the *hypothetical reference decoder*.
  33. **informative**: A term used to refer to content provided in this Specification that does not establish any mandatory requirements for conformance to this Specification and thus is not considered an integral part of this Specification.
  34. **instantaneous decoding refresh (IDR) PU**: A *PU* in which the *coded picture* is an *IDR picture*.
  35. **instantaneous decoding refresh (IDR) picture**: An *IRAP* *picture* for which each *VCL NAL unit* has nal\_unit\_type equal to IDR\_W\_RADL or IDR\_N\_LP.

NOTE – An IDR picture does not use inter prediction in its decoding process, and may be the first picture in the bitstream in decoding order, or may 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 may 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.

* 1. **instantaneous decoding refresh (IDR) subpicture**: An *IRA*P subpi*cture* for which each *VCL NAL unit* has nal\_unit\_type equal to IDR\_W\_RADL or IDR\_N\_LP.
  2. **inter-layer reference picture (ILRP)**: A *picture* in the same *AU* with the current *picture*, with nuh\_layer\_id less than the nuh\_layer\_id of the current *picture*, and is marked as "used for long-term reference".
  3. **inter coding**: Coding of a *coding block*, *slice*, or *picture* that uses *inter prediction*.
  4. **inter prediction**: A *prediction* derived in a manner that is dependent on data elements (e.g., sample values or motion vectors) of one or more *reference* *pictures*.
  5. **intra block copy (IBC) prediction**: A *prediction* derived in a manner that is dependent on data elements (e.g., sample values or block vectors) of the same decoded *slice* without referring to a *reference picture*. [Ed. (GJS): This definition needs improvement. It seems the same as the definition for intra prediction. Also, the fact that this starts with the word "intra" seems to imply that it is an intra prediction mode. This makes it confusing, since it is not treated as an intra prediction mode in the text. I strongly suggest removing the word "intra" from the name unless it is considered an intra prediction mode in the text.]
  6. **intra coding**: Coding of a *coding block, slice*, or *picture* that uses *intra prediction*.
  7. **intra prediction**: A *prediction* derived from only data elements (e.g., sample values) of the same decoded *slice* without referring to a *reference picture*.
  8. **intra random access point (IRAP) AU**: An *AU* in which there is a *PU* for each layer in the CVS and the *coded picture* in each *PU* is an *IRAP picture*.
  9. **intra random access point (IRAP) PU**: A *PU* in which the *coded picture* is an *IRAP picture*.
  10. **intra random access point (IRAP) picture**: A *coded picture* for which all *VCL NAL units* have the same value of nal\_unit\_type in the range of IDR\_W\_RADL to CRA\_NUT, inclusive.

NOTE 1 – An IRAP picture does not use inter prediction in its decoding process, and may be a CRA picture or an IDR picture. The first picture in the bitstream in decoding order must be an IRAP or GDR picture. Provided the necessary parameter sets are available when they need to be referenced, the IRAP picture and all subsequent non-RASL pictures in the CVS in decoding order can be correctly decoded without performing the decoding process of any pictures that precede the IRAP picture in decoding order.

NOTE 2 – The value of pps\_mixed\_nalu\_types\_in\_pic\_flag for an IRAP picture is equal to 0. When pps\_mixed\_nalu\_types\_in\_pic\_flag is equal to 0 for a picture, and any slice of the picture has nal\_unit\_type in the range of IDR\_W\_RADL to CRA\_NUT, inclusive, all other slices of the picture have the same value of nal\_unit\_type, and the picture is known to be an IRAP picture.

* 1. **intra random access point (IRAP) subpicture**: A *subpicture* for which all *VCL NAL units* have the same value of nal\_unit\_type in the range of IDR\_W\_RADL to CRA\_NUT, inclusive.
  2. **intra (I) slice**: A *slice* that is decoded using *intra prediction* only.
  3. **layer**: A set of *VCL NAL units* that all have a particular value of nuh\_layer\_id and the *associated non-VCL NAL units*.
  4. **leading picture**: A *picture* that precedes the *associated* *IRAP picture* in *output order*.
  5. **leading subpicture**: A *picture* that precedes the *associated* *IRAP subpicture* in *output order*.
  6. **leaf**: A terminating node of a tree that is a root node of a tree of depth 0.
  7. **level**: A defined set of constraints on the values that may be taken by the *syntax elements* and variables of this Specification, or the value of a *transform coefficient* prior to *scaling*.

NOTE – The same set of levels is defined for all profiles, with most aspects of the definition of each level being in common across different profiles. Individual implementations may, within the specified constraints, support a different level for each supported profile.

* 1. **list 0 (list 1) motion vector**: A *motion vector* associated with a *reference index* pointing into *reference picture list 0* (*list 1*).
  2. **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. **LMCS APS**: An *APS* that controls the *LMCS* process.
  4. **long-term reference picture (LTRP)**: A *picture* with nuh\_layer\_id equal to the nuh\_layer\_id of the current *picture* and marked as "used for long-term reference".
  5. **luma**: An adjective, represented by the symbol or subscript Y or L, specifying that a sample array or single sample is representing the monochrome signal related to the primary colours.

NOTE – The term luma is used rather than the term luminance in order to avoid the implication of the use of linear light transfer characteristics that is often associated with the term luminance. The symbol L is sometimes used instead of the symbol Y to avoid confusion with the symbol y as used for vertical location.

* 1. **luma mapping with chroma scaling (LMCS)**: A process that is applied as part of the *decoding process* that maps *luma* samples to particular values and may apply a scaling operation to the values of *chroma* samples.
  2. **may**: A term that is used to refer to behaviour that is allowed, but not necessarily required*.*

NOTE – In some places where the optional nature of the described behaviour is intended to be emphasized, the phrase "may or may not" is used to provide emphasis.

* 1. **motion vector**: A two-dimensional vector used for *inter prediction* that provides an offset from the coordinates in the *decoded picture* to the coordinates in a *reference picture*.
  2. **multi-type tree**: A *tree* in which a parent node can be split either into two child nodes using a *binary split* or into three child nodes using a *ternary split*, each of which may become parent node for another split into either two or three child nodes.
  3. **must**: A term that is used in expressing an observation about a requirement or an implication of a requirement that is specified elsewhere in this Specification (used exclusively in an *informative* context).
  4. **network abstraction layer (NAL) unit**: A *syntax structure* containing an indication of the type of data to follow and *bytes* containing that data in the form of an *RBSP* interspersed as necessary with *emulation prevention bytes*.
  5. **network abstraction layer (NAL) unit stream**: A sequence of *NAL units*.
  6. **note**: A term that is used to prefix *informative* remarks (used exclusively in an *informative* context).
  7. **operation point (OP)**: A temporal subset of an OLS, identified by an OLS index and a highest value of TemporalId.
  8. **output layer**: A layer of an output layer set that is output.
  9. **output layer set (OLS)**: A set of layers consisting of a specified set of layers, where one or more layers in the set of layers are specified to be output layers.
  10. **output layer set (OLS) layer index**: An index, of a layer in an OLS, to the list of layers in the OLS.
  11. **output order**: The order of pictures or subpictures within a CLVS indicated by increasing POC values, and for decoded pictures that are output output from DPB, this is the order in which the *decoded* *pictures* are output from the *DPB*.
  12. **output time**: A time when a *decoded* *picture* is to be output from the *DPB* (for the *decoded pictures* that are to be output from the *DPB*) as specified by the *HRD* according to the output timing *DPB* operation.
  13. **parameter**: A *syntax element* of a *sequence parameter set (SPS)* or *picture parameter set (PPS)*, or the second word of the defined term *quantization parameter*.
  14. **partitioning**: The division of a set into subsets such that each element of the set is in exactly one of the subsets.
  15. **picture**: An array of *luma* samples in monochrome format or an array of *luma* samples and two corresponding arrays of *chroma* samples in 4:2:0, 4:2:2, and 4:4:4 colour format.

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

* 1. **picture header (PH)**: A *syntax structure* containing *syntax elements* that apply to all *slices* of a coded picture*.*
  2. **picture-level slice index**: An index, defined when pps\_rect\_slice\_flag is equal to 1, of a slice to the list of slices in a picture in the order as the slices are signalled in the PPS when pps\_single\_slice\_per\_subpic\_flag is equal to 0, or in the order of increasing subpicture indices of the subpicture corresponding to the slices when pps\_single\_slice\_per\_subpic\_flag is equal to 1.
  3. **picture order count (POC)**: A variable that is associated with each *picture*, uniquely identifies the associated *picture* among all *pictures* in the *CLVS*, and, when the associated *picture* is to be output from the *DPB*, indicates the position of the associated *picture* in *output order* relative to the *output order* positions of the other *pictures* in the same *CLVS* that are to be output from the *DPB*.
  4. **picture parameter set (PPS)**: A *syntax structure* containing *syntax elements* that apply to zero or more entire *coded pictures* as determined by a *syntax element* found in each *picture header.*
  5. **picture unit (PU)**: A set of *NAL units* that are associated with each other according to a specified classification rule, are consecutive in *decoding order,* and contain exactly one *coded picture*.
  6. **prediction**: An embodiment of the *prediction process*.
  7. **prediction process**: The use of a *predictor* to provide an estimate of the data element (e.g., sample value or motion vector) currently being decoded.
  8. **predictive (P) slice**: A *slice* that is decoded using *intra* *prediction* or using *inter prediction* with at most one *motion vector* and *reference index* to *predict* the sample values of each *block*.
  9. **predictor**: A combination of specified values or previously decoded data elements (e.g., sample value or motion vector) used in the *decoding process* of subsequent data elements.
  10. **profile**: A specified subset of the syntax of this Specification.
  11. **quadtree**: A *tree* in which a parent node can be split into four child nodes, each of which may become parent node for another split into four child nodes.
  12. **quantization parameter**: A variable used by the *decoding process* for *scaling* of *transform coefficient levels*.
  13. **random access**: The act of starting the decoding process for a *bitstream* at a point other than the beginning of the stream.
  14. **random access decodable leading (RADL) PU**: A *PU* in which the *coded picture* is a *RADL picture*.
  15. **random access decodable leading (RADL) picture**: A *coded picture* for which each *VCL NAL unit* has nal\_unit\_type equal to RADL\_NUT.

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

* 1. **random access decodable leading (RADL) subpicture**: A *subpicture* for which each *VCL NAL unit* has nal\_unit\_type equal to RADL\_NUT.
  2. **random access skipped leading (RASL) PU**: A *PU* in which the *coded picture* is a *RASL picture.*
  3. **random access skipped leading (RASL) picture**: A *coded picture* for which there is at least one *VCL NAL unit* with nal\_unit\_type equal to RASL\_NUT and other VCL NAL units all have nal\_unit\_type equal to RASL\_NUT or RADL\_NUT.

NOTE – All RASL pictures are leading pictures of an associated CRA picture. When the associated CRA picture has NoOutputBeforeRecoveryFlag equal to 1, the RASL picture is not output and may not be correctly decodable, as the RASL picture may 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. 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.

* 1. **random access skipped leading (RASL) subpicture**: A *subpicture* for which each *VCL NAL unit* has nal\_unit\_type equal to RASL\_NUT.
  2. **raster scan**: A mapping of a rectangular two-dimensional pattern to a one-dimensional pattern such that the first entries in the one-dimensional pattern are from the first top row of the two-dimensional pattern scanned from left to right, followed similarly by the second, third, etc., rows of the pattern (going down) each scanned from left to right.
  3. **raw byte sequence payload (RBSP)**: A *syntax structure* containing an integer number of *bytes* that is encapsulated in a *NAL unit* and is either empty or has the form of a *string of data bits* containing *syntax elements* followed by an *RBSP stop bit* and zero or more subsequent bits equal to 0.
  4. **raw byte sequence payload (RBSP) stop bit**: A bit equal to 1 present within a *raw byte sequence payload (RBSP)* after a *string of data bits*, for which the location of the end within an *RBSP* can be identified by searching from the end of the *RBSP* for the *RBSP stop bit*, which is the last non-zero bit in the *RBSP.*
  5. **reference index**: An index into a *reference picture list*.
  6. **reference picture**: A *picture* that is a *short-term reference picture*, a *long-term reference picture,* or an *inter-layer reference picture*.

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

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

NOTE – Two reference picture lists, reference picture list 0 and reference picture list 1, are generated for each slice of a non-IDR picture. The set of unique pictures referred to by all entries in the two reference picture lists associated with a picture consists of all reference pictures that may 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 reference picture list 0 is used for inter prediction. For the decoding process of a B slice, both reference picture list 0 and reference picture list 1 are used for inter prediction. For decoding the slice data of an I slice, no reference picture list is used for for inter prediction.

* 1. **reference picture list 0**: The *reference picture list* used for *inter prediction* of a *P* or the first *reference picture list* used for *inter prediction* of a *B* *slice*.
  2. **reference picture list 1**: The second *reference picture list* used for *inter prediction* of a *B slice*.
  3. **reserved**: A term that may be 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 *bitstreams* conforming to this version of this Specification, but may be used in bitstreams conforming to future extensions of this Specification by ITU‑T | ISO/IEC.
  4. **residual**: The decoded difference between a *prediction* of a sample or data element and its decoded value.
  5. **scaling**: The process of multiplying *transform coefficient levels* by a factor, resulting in *transform coefficients*.
  6. **scaling list**: A list that associates each *frequency index* with a scale factor for the *scaling* process.
  7. **scaling list APS**: An *APS* with syntax elements used to construct the *scaling lists*.
  8. **sequence parameter set (SPS)**: A *syntax structure* containing *syntax elements* that apply to zero or more entire *CLVSs* as determined by the content of a *syntax element* found in the *PPS* referred to by a *syntax element* found in each *picture header.*
  9. **shall**: A term used to express mandatory requirements for conformance to this Specification.

NOTE – When used to express a mandatory constraint on the values of syntax elements or on the results obtained by operation of the specified decoding process, it is the responsibility of the encoder to ensure that the constraint is fulfilled. When used in reference to operations performed by the decoding process, any decoding process that produces identical cropped decoded pictures to those output from the decoding process described in this Specification conforms to the decoding process requirements of this Specification.

* 1. **short-term reference picture (STRP)**: A *picture* with nuh\_layer\_id equal to the nuh\_layer\_id of the current *picture* and marked as "used for short-term reference".
  2. **should**: A term used to refer to behaviour of an implementation that is encouraged to be followed under anticipated ordinary circumstances, but is not a mandatory requirement for conformance to this Specification.
  3. **slice**: An integer number of complete *tiles* or an integer number of consecutive complete *CTU* rows within a *tile* of a *picture* that are exclusivelycontained in a single *NAL unit*.
  4. **slice header**: A part of a coded *slice* containing the data elements pertaining to all *tiles* or *CTU rows* within a *tile* represented in the *slice*.
  5. **source**: A term used to describe the video material or some of its attributes before encoding.
  6. **start code prefix**: A unique sequence of three *bytes* equal to 0x000001 embedded in the *byte stream* as a prefix to each *NAL unit*.

NOTE – The location of a start code prefix can be used by a decoder to identify the beginning of a new NAL unit and the end of a previous NAL unit. Emulation of start code prefixes is prevented within NAL units by the inclusion of emulation prevention bytes.

* 1. **step-wise temporal sublayer access (STSA) PU**: A *PU* in which the *coded picture* is an *STSA picture*.
  2. **step-wise temporal sublayer access (STSA) picture**: A *coded picture* for which each *VCL NAL unit* has nal\_unit\_type equal to STSA\_NUT.

NOTE – An STSA picture does not use pictures with the same TemporalId as the STSA picture for inter prediction reference. Pictures following an STSA picture in decoding order with the same TemporalId as the STSA picture do not use pictures prior to the STSA picture in decoding order 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. STSA pictures must have TemporalId greater than 0.

* 1. **step-wise temporal sublayer access (STSA) subpicture**: A *subpicture* for which each *VCL NAL unit* has nal\_unit\_type equal to STSA\_NUT.
  2. **string of data bits (SODB)**: A sequence of some number of bits representing *syntax elements* present within a *raw byte sequence payload* prior to the *raw byte sequence payload stop bit*, where the left-most bit is considered to be the first and most significant bit, and the right-most bit is considered to be the last and least significant bit.
  3. **sub-bitstream extraction process**: A specified process by which *NAL units* in a *bitstream* that do not belong to a target set, determined by a target OLS index and a target highest TemporalId, are removed from the *bitstream*, with the output sub-bitstream consisting of the NAL units in the *bitstream* that belong to the target set.
  4. **sublayer**: A temporal scalable layer of a temporal scalable *bitstream*, consisting of *VCL NAL units* with a particular value of the TemporalId variable and the associated *non-VCL NAL units*.
  5. **sublayer representation**: A subset of the *bitstream* consisting of *NAL units* of a particular *sublayer* and the lower *sublayers*.
  6. **subpicture:** An rectangular region of one or more *slices* within a *picture*.
  7. **subpicture-level slice index**: An index, defined when pps\_rect\_slice\_flag is equal to 1, of a slice to the list of slices in a subpicture in the order as they are signalled in the PPS.
  8. **supplemental enhancement information (SEI) message**: A *syntax structure* with specified semantics that conveys information that is not needed by the *decoding process* in order to determine the values of the samples in *decoded pictures*.
  9. **syntax element**: An element of data represented in the *bitstream*.
  10. **syntax structure**: Zero or more *syntax elements* present together in the *bitstream* in a specified order*.*
  11. **ternary split**: A split of a rectangular MxN *block* of samples into three *blocks* where a vertical split results in a first (M / 4)xN *block*, a second (M / 2)xN *block*, a third (M / 4)xN *block*, and a horizontal split results in a first Mx(N / 4) *block*, a second Mx(N / 2) *block*, a third Mx(N / 4) *block*.
  12. **tier**: A specified category of *level* constraints imposed on values of the *syntax elements* in the *bitstream*, where the *level* constraints are nested within a *tier* and a *decoder* conforming to a certain *tier* and *level* would be capable of decoding all *bitstreams* that conform to the same *tier* or the lower *tier* of that *level* or any *level* below it.
  13. **tile**: A rectangular region of *CTUs* within a particular *tile column* and a particular *tile row* in a *picture*.
  14. **tile column**: A rectangular region of *CTUs* having a height equal to the height of the *picture* and a width specified by *syntax elements* in the *picture parameter set*.
  15. **tile row**: A rectangular region of *CTUs* having a height specified by *syntax elements* in the *picture parameter set* and a width equal to the width of the *picture*.
  16. **tile scan**: A specific sequential ordering of *CTUs* *partitioning* a *picture* in which the *CTUs* are ordered consecutively in *CTU* *raster scan* in a *tile* whereas *tiles* in a *picture* are ordered consecutively in a *raster scan* of the *tiles* of the *picture*.
  17. **trailing picture**: A *picture* for which each *VCL NAL unit* has nal\_unit\_type equal to TRAIL\_NUT.

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

* 1. **trailing subpicture**: A *subpicture* for which each *VCL NAL unit* has nal\_unit\_type equal to TRAIL\_NUT.

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

* 1. **transform**: A part of the *decoding process* by which a *block* of *transform coefficients* is converted to a *block* of spatial-domain values.
  2. **transform block**: A rectangular MxN *block* of samples resulting from a *transform* in the *decoding process*.
  3. **transform coefficient**: A scalar quantity, considered to be in a frequency domain, that is associated with a particular one-dimensional or two-dimensional *frequency index* in a *transform* in the *decoding process*.
  4. **transform coefficient level**: An integer quantity representing the value associated with a particular two‑dimensional frequency index in the *decoding process* prior to *scaling* for computation of a *transform coefficient* value.
  5. **transform unit (TU)**: A *transform block* of *luma* samples and two corresponding *transform blocks* of *chroma* samples of a *picture* when using a single *coding unit tree* for *luma* and *chroma*; or, a *transform block* of *luma* samples or two *transform blocks* of *chroma* samples when using two separate *coding unit trees* for *luma* and *chroma*, and *syntax structures* used to transform the *transform block* samples.
  6. **tree**: A tree is a finite set of nodes with a unique root node.
  7. **unspecified**: A term that may be used to specify some values of a particular *syntax element* to indicate that the values have no specified meaning in this Specification and will not have a specified meaning in the future as an integral part of future versions of this Specification.
  8. **video coding layer (VCL) NAL unit**: A collective term for *coded slice NAL units* and the subset of *NAL units* that have *reserved* values of nal\_unit\_type that are classified as VCL NAL units in this Specification.

# Abbreviations

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

ALF Adaptive Loop Filter

AMVR Adaptive Motion Vector Resolution

APS Adaptation Parameter Set

AU Access Unit

AUD Access Unit Delimiter

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

CPB Coded Picture Buffer

CRA Clean Random Access

CRC Cyclic Redundancy Check

CTB Coding Tree Block

CTU Coding Tree Unit

CU Coding Unit

CVS Coded Video Sequence

DPB Decoded Picture Buffer

DCI Decoding Capability Information

DRAP Dependent Random Access Point

DU Decoding Unit

EG Exponential-Golomb

EGk k-th order Exponential-Golomb

EOB End Of Bitstream

EOS End Of Sequence

FCC Federal Communications Commission (of the United States)

FD Filler Data

FIFO First-In, First-Out

FL Fixed-Length

GBR Green, Blue and Red

GDR Gradual Decoding Refresh

GPM Geometric Partitioning Mode

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

NTSC National Television System Committee (of the United States)

OLS Output Layer Set

OP Operation Point

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 Same as GBR

RPS Reference Picture Set

SAO Sample Adaptive Offset

SAR Sample Aspect Ratio

SEI Supplemental Enhancement Information

SH Slice Header

SMPTE Society of Motion Picture and Television Engineers

SODB String Of Data Bits

SPS Sequence Parameter Set

STRP Short-Term Reference Picture

STSA Step-wise Temporal Sublayer Access

TR Truncated Rice

UCS Universal Coded Character Set

UTF UCS Transmission Format

VBR Variable Bit Rate

VCL Video Coding Layer

VPS Video Parameter Set

VUI Video Usability Information

# Conventions

## General

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

## Arithmetic operators

The following arithmetic operators are defined as follows:

|  |  |
| --- | --- |
| + | Addition |
| − | Subtraction (as a two-argument operator) or negation (as a unary prefix operator) |
| \* | Multiplication, including matrix multiplication |
| xy | Exponentiation. Specifies x to the power of y. In other contexts, such notation is used for superscripting not intended for interpretation as exponentiation. |
| / | Integer division with truncation of the result toward zero. For example, 7 / 4 and −7 / −4 are truncated to 1 and −7 / 4 and 7 / −4 are truncated to −1. |
| ÷ | Used to denote division in mathematical equations where no truncation or rounding is intended. |
|  | Used to denote division in mathematical equations where no truncation or rounding is intended. |
|  | The summation of f( i ) with i taking all integer values from x up to and including y. |
| x % y | Modulus. Remainder of x divided by y, defined only for integers x and y with x >= 0 and y > 0. |

## Logical operators

The following logical operators are defined as follows:

x && y Boolean logical "and" of x and y

x | | y Boolean logical "or" of x and y

! Boolean logical "not"

x ? y : z If x is TRUE, evaluates to the value of y; otherwise, evaluates to the value of z.

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

## Relational operators

The following relational operators are defined as follows:

> Greater than

>= Greater than or equal to

< Less than

<= Less than or equal to

= = Equal to

!= Not equal to

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

## Bit-wise operators

The following bit-wise operators are defined as follows:

& Bit-wise "and". When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.

| Bit-wise "or". When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.

^ Bit-wise "exclusive or". When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.

x >> y Arithmetic right shift of a two's complement integer representation of x by y binary digits. This function is defined only for non-negative integer values of y. Bits shifted into the most significant bits (MSBs) as a result of the right shift have a value equal to the MSB of x prior to the shift operation.

x << y Arithmetic left shift of a two's complement integer representation of x by y binary digits. This function is defined only for non-negative integer values of y. Bits shifted into the least significant bits (LSBs) as a result of the left shift have a value equal to 0.

## Assignment operators

The following arithmetic operators are defined as follows:

= Assignment operator

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

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

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

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

## Range notation

The following notation is used to specify a range of values:

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

## Mathematical functions

The following mathematical functions are defined:

Abs( x ) = (1)

Asin( x ) the trigonometric inverse sine function, operating on an argument x that is  
in the range of −1.0 to 1.0, inclusive, with an output value in the range of   
−π÷2 to π÷2, inclusive, in units of radians (2)

Atan( x ) the trigonometric inverse tangent function, operating on an argument x, with  
an output value in the range of −π÷2 to π÷2, inclusive, in units of radians (3)

Atan2( y, x ) = (4)

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

Clip1( x ) = Clip3( 0, ( 1 << BitDepth ) − 1, x ) (6)

Clip3( x, y, z ) = (7)

ClipH( o, W, x ) = (8)

Cos( x ) the trigonometric cosine function operating on an argument x in units of radians. (9)

Floor( x ) the largest integer less than or equal to x. (10)

GetCurrMsb( a, b, c, d ) = (11)

[Ed. (YK): This function is not used anywhere, thus can be removed. Check also other functions. In the final version, remove those functions that are not used.]

Ln( x ) the natural logarithm of x (the base-e logarithm, where e is the natural logarithm base constant 2.718 281 828...). (12)

Log2( x ) the base-2 logarithm of x. (13)

Log10( x ) the base-10 logarithm of x. (14)

Min( x, y ) = (15)

Max( x, y ) = (16)

Round( x ) = Sign( x ) \* Floor( Abs( x ) + 0.5 ) (17)

Sign( x ) = (18)

Sin( x ) the trigonometric sine function operating on an argument x in units of radians (19)

Sqrt( x ) = (20)

Swap( x, y ) = ( y, x ) (21)

Tan( x ) the trigonometric tangent function operating on an argument x in units of radians (22)

## Order of operation precedence

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

– Operations of a higher precedence are evaluated before any operation of a lower precedence.

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

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

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

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

|  |
| --- |
| **operations (with operands x, y, and z)** |
| "x++", "x− −" |
| "!x", "−x" (as a unary prefix operator) |
| xy |
| "x \* y", "x / y", "x ÷ y", "", "x % y" |
| "x + y", "x − y" (as a two-argument operator), "" |
| "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" |

## 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 may use the values of other variables derived from syntax elements values. Such variables appear in the syntax tables, or text, named by a mixture of lower case and upper case letter and without any underscore characters. Variables starting with an upper case letter are derived for the decoding of the current syntax structure and all depending syntax structures. Variables starting with an upper case letter may be used in the decoding process for later syntax structures without mentioning the originating syntax structure of the variable. Variables starting with a lower case letter are only used within the 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 may contain more upper case letters.

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

Functions that specify properties of the current position in the bitstream are referred to as syntax functions. These functions are specified in clause 7.2 and assume the existence of a bitstream pointer with an indication of the position of the next bit to be read by the decoding process from the bitstream. Syntax functions are described by their names, which are constructed as syntax element names and end with left and right round parentheses including zero or more variable names (for definition) or values (for usage), separated by commas (if more than one variable).

Functions that are not syntax functions (including mathematical functions specified in clause 5.8) are described by their names, which start with an upper case letter, contain a mixture of lower and upper case letters without any underscore character, and end with left and right parentheses including zero or more variable names (for definition) or values (for usage) separated by commas (if more than one variable).

A one-dimensional array is referred to as a list. A two-dimensional array is referred to as a matrix. Arrays can either be syntax elements or variables. Subscripts or square parentheses are used for the indexing of arrays. In reference to a visual depiction of a matrix, the first subscript is used as a row (vertical) index and the second subscript is used as a column (horizontal) index. The indexing order is reversed when using square parentheses rather than subscripts for indexing. Thus, an element of a matrix s at horizontal position x and vertical position y may be denoted either as s[ x ][ y ] or as syx. A single column of a matrix may 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 may be referred to as the list s[ x ].

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

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

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

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

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

## 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

may be described in the following manner:

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

– If condition 0, statement 0

– Otherwise, if condition 1, statement 1

– ...

– Otherwise (informative remark on remaining condition), statement n

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

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

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

may be described in the following manner:

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

– If all of the following conditions are true, statement 0:

– condition 0a

– condition 0b

– Otherwise, if 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

may be described in the following manner:

When condition 0, statement 0

When condition 1, statement 1

## Processes

Processes are used to describe the decoding of syntax elements. A process has a separate specification and invoking. All syntax elements and upper case variables that pertain to the current syntax structure and depending syntax structures are available in the process specification and invoking. A process specification may also have a lower case variable explicitly specified as 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 may be referred to by the variable name having a value equal to the address of the specific coding block.

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

## Bitstream formats

This clause specifies the relationship between the network abstraction layer (NAL) unit stream and byte stream, either of which are referred to as the bitstream.

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

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

## Source, decoded and output picture formats

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

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

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

– Luma (Y) only (monochrome).

– Luma and two chroma (YCbCr or YCgCo).

– Green, blue, and red (GBR, also known as RGB).

– Arrays representing other unspecified monochrome or tri-stimulus colour samplings (for example, YZX, also known as XYZ).

For convenience of notation and terminology in this Specification, the variables and terms associated with these arrays are referred to as luma (or L or Y) and chroma, where the two chroma arrays are referred to as Cb and Cr; regardless of the actual colour representation method in use. The actual colour representation method in use can be indicated in syntax that is specified in VUI parameters as specified in ITU-T H.SEI | ISO/IEC 23002-7.

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 and sps\_separate\_colour\_plane\_flag. Other values of sps\_chroma\_format\_idc, SubWidthC and SubHeightC may be specified in the future by ITU‑T | ISO/IEC.

Table 2 – SubWidthC and SubHeightC values derived from  
sps\_chroma\_format\_idc and sps\_separate\_colour\_plane\_flag

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **sps\_chroma\_format\_idc** | **sps\_separate\_colour\_plane\_flag** | **Chroma format** | **SubWidthC** | **SubHeightC** |
| 0 | 0 | Monochrome | 1 | 1 |
| 1 | 0 | 4:2:0 | 2 | 2 |
| 2 | 0 | 4:2:2 | 2 | 1 |
| 3 | 0 | 4:4:4 | 1 | 1 |
| 3 | 1 | 4:4:4 | 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, depending on the value of sps\_separate\_colour\_plane\_flag, the following applies:

– If sps\_separate\_colour\_plane\_flag is equal to 0, each of the two chroma arrays has the same height and width as the luma array.

– Otherwise (sps\_separate\_colour\_plane\_flag is equal to 1), the three colour planes are separately processed as monochrome sampled pictures.

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, and the number of bits used in the luma array may differ from the number of bits used in the chroma arrays.

When the value of 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 may be indicated in VUI parameters as specified in ITU-T H.SEI | ISO/IEC 23002-7.



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

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

### 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 bounaries within a tile; this occurs when a tile is split into multiple rectangular slices, each of which consists of an integer number of consecutive complete CTU rows within the tile.

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

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

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

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

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

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

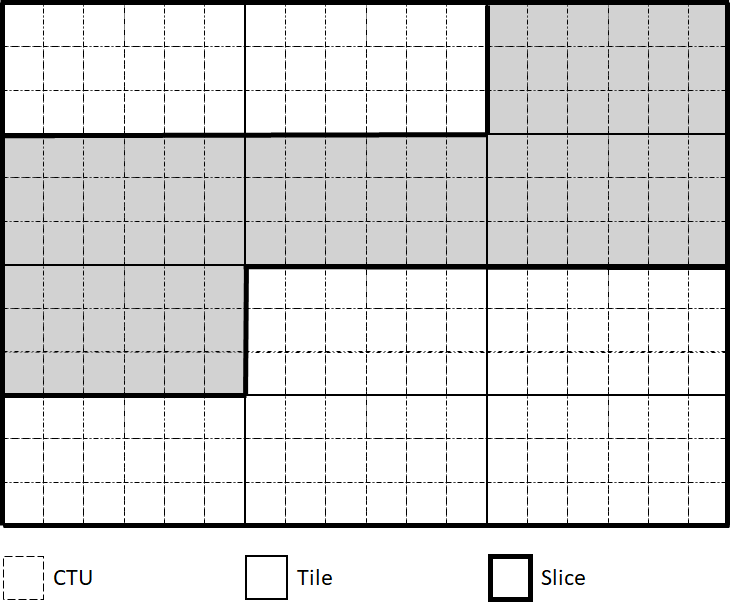


Figure 4 – A picture with 18 by 12 luma CTUs that is partitioned into 12 tiles and 3 raster-scan slices (informative)

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

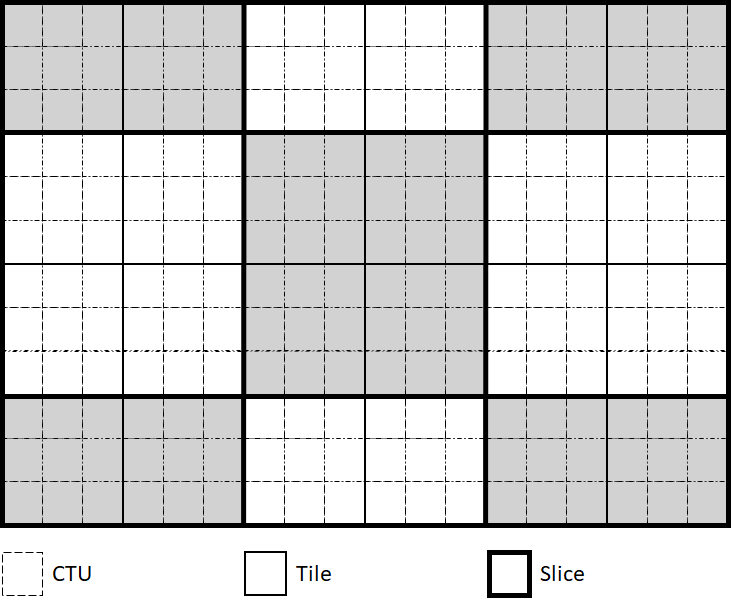


Figure 5 – A picture with 18 by 12 luma CTUs that is partitioned into 24 tiles and 9 rectangular slices (informative)

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

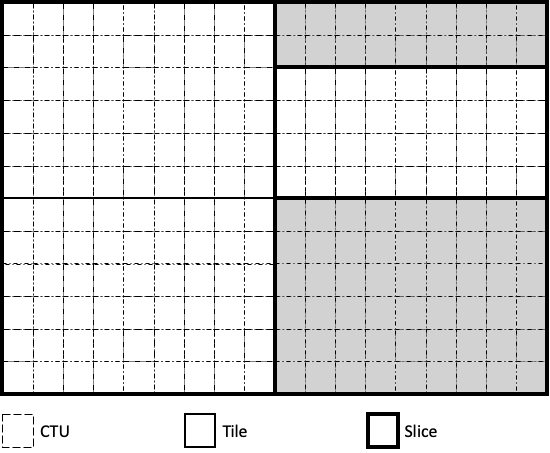


Figure 6 – A picture that is partitioned into 4 tiles and 4 rectangular slices (informative)

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

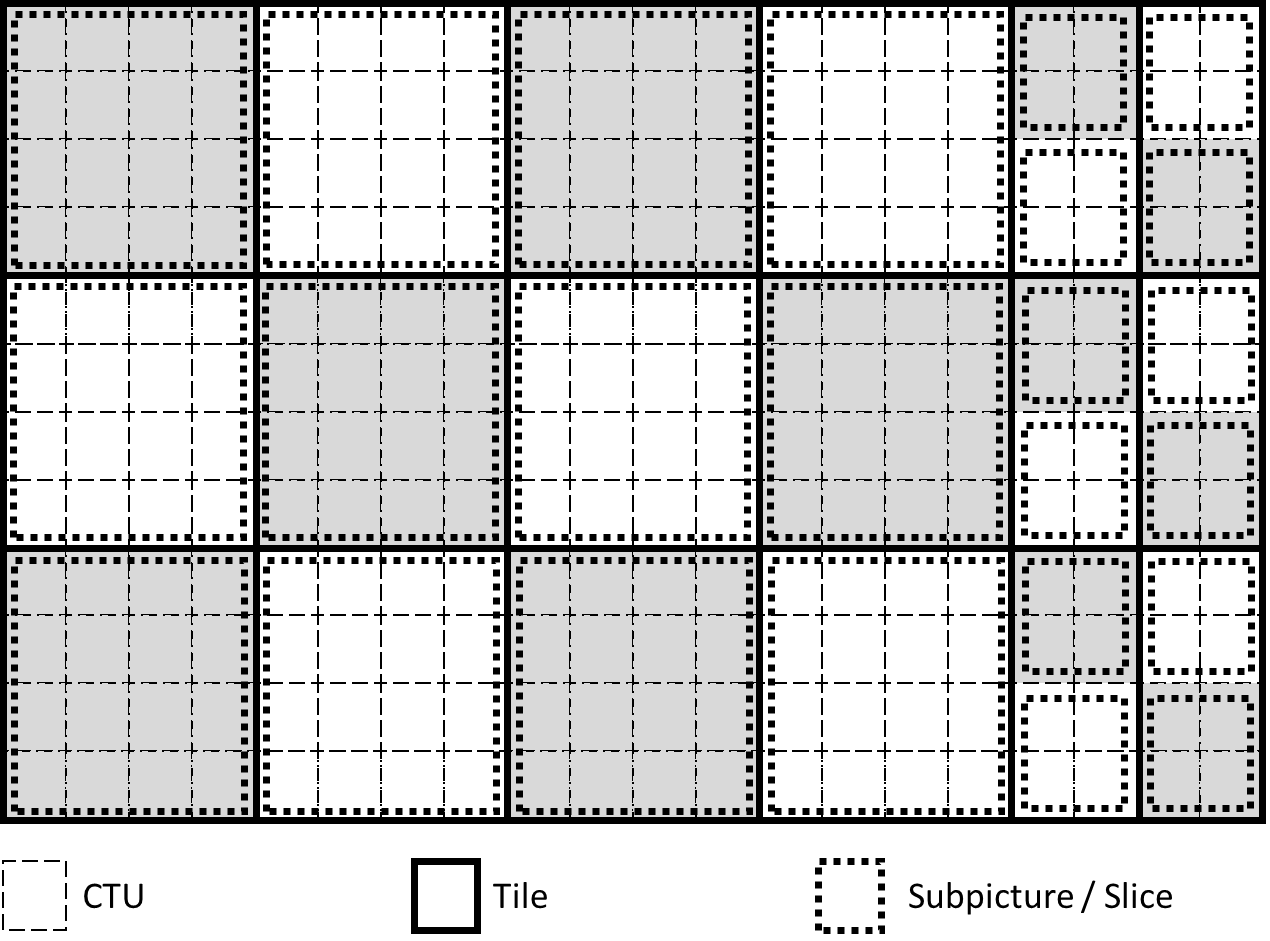


Figure 7 – A picture that is partitioned into 18 tiles, 24 slices and 24 subpictures (informative)

When a picture is coded using three separate colour planes (sps\_separate\_colour\_plane\_flag is equal to 1), a slice contains only CTUs of one colour component being identified by the corresponding value of sh\_colour\_plane\_id, and each colour component array of a picture consists of slices having the same sh\_colour\_plane\_id value. Coded slices with different values of sh\_colour\_plane\_id within a picture may be interleaved with each other under the constraint that for each value of sh\_colour\_plane\_id, the coded slice NAL units with that value of sh\_colour\_plane\_id shall be in the order of increasing CTU address in tile scan order for the first CTU of each coded slice NAL unit.

NOTE – When sps\_separate\_colour\_plane\_flag is equal to 0, each CTU of a picture is contained in exactly one slice. When sps\_separate\_colour\_plane\_flag is equal to 1, each CTU of a colour component is contained in exactly one slice (i.e., information for each CTU of a picture is present in exactly three slices and these three slices have different values of sh\_colour\_plane\_id).

### 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.

[Ed. (BB): Revise the following for QT+MTT.]

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. 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. When the component width is not an integer number 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.

The coding block is the root node of two trees, the prediction tree and the transform tree. The prediction tree specifies the position and size of prediction blocks. The transform tree specifies the position and size of transform blocks. The splitting information for luma and chroma is identical for the prediction tree and may or may 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 sps\_separate\_colour\_plane\_flag is equal to 1) 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 sps\_separate\_colour\_plane\_flag is equal to 1) 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 sps\_separate\_colour\_plane\_flag is equal to 1) or three CTBs (luma and chroma), the associated coding tree syntax structures and the associated coding units are associated with a CTU.

### Spatial or component-wise partitionings

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

– The division of each picture into components

– The division of each component into CTBs

– The division of each picture into subpictures

– The division of each picture into tile columns

– The division of each picture into tile rows

– The division of each tile column into tiles

– The division of each tile row into tiles

– The division of each tile into CTUs

– The division of each picture into slices

– The division of each subpicture into slices

– The division of each slice into CTUs

– The division of each CTU into CTBs

– The 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

– The 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

– The division of each coding unit into transform units

– The division of each coding unit into coding blocks

– The division of each coding block into transform blocks

– The division of each transform unit into transform blocks

For each of the above-listed divisions of an entity A into entities B 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 partitioining, 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.

## Availability processes

### 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), and inter coding modes can be used (MODE\_TYPE\_ALL), or whether only intra and IBC 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.

### Allowed binary split process

Inputs to this process are:

* a binary split mode btSplit,
* a coding block width cbWidth in luma samples,
* a coding block height cbHeight in luma samples,
* a location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture,
* a multi-type tree depth mttDepth,
* a maximum multi-type tree depth with offset maxMttDepth,
* a maximum binary tree size maxBtSize,
* a minimium quadtree size minQtSize,
* a partition index partIdx,
* a variable treeType specifying whether a single tree (SINGLE\_TREE) or a dual tree is used to partition the coding tree node and, when a dual tree is used, whether the luma (DUAL\_TREE\_LUMA) or chroma components (DUAL\_TREE\_CHROMA) are currently processed,
* a variable modeType specifying whether intra (MODE\_INTRA), IBC (MODE\_IBC), and inter coding modes can be used (MODE\_TYPE\_ALL), or whether only intra and IBC coding modes can be used (MODE\_TYPE\_INTRA), or whether only inter coding modes can be used (MODE\_TYPE\_INTER) for coding units inside the coding tree node.

Output of this process is the variable allowBtSplit.

Table 3 – Specification of parallelTtSplit and cbSize based on btSplit

|  |  |  |
| --- | --- | --- |
|  | **btSplit = = SPLIT\_BT\_VER** | **btSplit = = SPLIT\_BT\_HOR** |
| **parallelTtSplit** | SPLIT\_TT\_VER | SPLIT\_TT\_HOR |
| **cbSize** | cbWidth | cbHeight |

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

The variable allowBtSplit is derived as follows:

* If one or more of the following conditions are true, allowBtSplit is set equal to FALSE:
* cbSize is less than or equal to MinBtSizeY
* cbWidth is greater than maxBtSize
* cbHeight is greater than maxBtSize
* mttDepth is greater than or equal to maxMttDepth
* treeType is equal to DUAL\_TREE\_CHROMA and ( cbWidth / SubWidthC ) \* ( cbHeight / SubHeightC ) is less than or equal to 16
* treeType is equal to DUAL\_TREE\_CHROMA and ( cbWidth / SubWidthC ) is equal to 4 and btSplit is equal to SPLIT\_BT\_VER
* treeType is equal to DUAL\_TREE\_CHROMA and modeType is equal to MODE\_TYPE\_INTRA
* cbWidth \* cbHeight is equal to 32 and modeType is equal to MODE\_TYPE\_INTER
* Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE
* btSplit is equal to SPLIT\_BT\_VER
* y0 + cbHeight is greater than pps\_pic\_height\_in\_luma\_samples
* Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE
* btSplit is equal to SPLIT\_BT\_VER
* cbHeight is greater than 64
* x0 + cbWidth is greater than pps\_pic\_width\_in\_luma\_samples
* Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE
* btSplit is equal to SPLIT\_BT\_HOR
* cbWidth is greater than 64
* y0 + cbHeight is greater than pps\_pic\_height\_in\_luma\_samples
* Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE
* x0 + cbWidth is greater than pps\_pic\_width\_in\_luma\_samples
* y0 + cbHeight is greater than pps\_pic\_height\_in\_luma\_samples
* cbWidth is greater than minQtSize
* Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE
* btSplit is equal to SPLIT\_BT\_HOR
* x0 + cbWidth is greater than pps\_pic\_width\_in\_luma\_samples
* y0 + cbHeight is less than or equal to pps\_pic\_height\_in\_luma\_samples
* Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE:
* mttDepth is greater than 0
* partIdx is equal to 1
* MttSplitMode[ x0 ][ y0 ][ mttDepth − 1 ] is equal to parallelTtSplit
* Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE
* btSplit is equal to SPLIT\_BT\_VER
* cbWidth is less than or equal to 64
* cbHeight is greater than 64
* Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE
* btSplit is equal to SPLIT\_BT\_HOR
* cbWidth is greater than 64
* cbHeight is less than or equal to 64

– Otherwise, allowBtSplit is set equal to TRUE.

### 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), and inter coding modes can be used (MODE\_TYPE\_ALL), or whether only intra and IBC coding modes can be used (MODE\_TYPE\_INTRA), or whether only inter coding modes can be used (MODE\_TYPE\_INTER) for coding units inside the coding tree node.

Output of this process is the variable allowTtSplit.

Table 4 – Specification of cbSize based on ttSplit

|  |  |  |
| --- | --- | --- |
|  | **ttSplit = = SPLIT\_TT\_VER** | **ttSplit = = SPLIT\_TT\_HOR** |
| **cbSize** | cbWidth | cbHeight |

The variable cbSize is derived as specified in Table 4.

The variable allowTtSplit is derived as follows:

* If one or more of the following conditions are true, allowTtSplit is set equal to FALSE:
* cbSize is less than or equal to 2 \* MinTtSizeY
* cbWidth is greater than Min( 64, maxTtSize )
* cbHeight is greater than Min( 64, maxTtSize )
* mttDepth is greater than or equal to maxMttDepth
* x0 + cbWidth is greater than pps\_pic\_width\_in\_luma\_samples
* y0 + cbHeight is greater than pps\_pic\_height\_in\_luma\_samples
* treeType is equal to DUAL\_TREE\_CHROMA and ( cbWidth / SubWidthC ) \* ( cbHeight / SubHeightC ) is less than or equal to 32
* treeType is equal to DUAL\_TREE\_CHROMA and ( cbWidth / SubWidthC ) is equal to 8 and ttSplit is equal to SPLIT\_TT\_VER
* treeType is equal to DUAL\_TREE\_CHROMA and modeType is equal to MODE\_TYPE\_INTRA
* cbWidth \* cbHeight is equal to 64 and modeType is equal to MODE\_TYPE\_INTER

– Otherwise, allowTtSplit is set equal to TRUE.

### Derivation process for neighbouring block availability

Inputs to this process are:

* the luma location ( xCurr, yCurr ) of the top-left sample of the current block relative to the top-left luma sample of the current picture,
* the luma location ( xNbY, yNbY ) covered by a neighbouring block relative to the top-left luma sample of the current picture,
* the variable checkPredModeY specifying whether availability depends on the prediction mode,
* the variable cIdx specifying the colour component of the current block.

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

The neighbouring block availability availableN is derived as follows:

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

– xNbY is less than 0.

– yNbY is less than 0.

– xNbY is greater than or equal to pps\_pic\_width\_in\_luma\_samples.

– yNbY is greater than or equal to pps\_pic\_height\_in\_luma\_samples.

– IsAvailable[ cIdx ][ xNbY ][ yNbY ] is equal to FALSE.

– The neighbouring block is contained in a different slice than the current block.

– The neighbouring block is contained in a different tile than the current block.

– sps\_entropy\_coding\_sync\_enabled\_flag is equal to 1 and ( xNbY  >>  CtbLog2SizeY ) is greater than or equal to (xCurr  >>  CtbLog2SizeY ) + 1.

– Otherwise, availableN is set equal to TRUE.

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

– checkPredModeY is equal to TRUE.

– availableN is set equal to TRUE.

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

## Scanning processes

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

The variable NumTileColumns, specifying the number of tile columns, and the list colWidth[ 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++ ) {  
 colWidth[ i ] = pps\_tile\_column\_width\_minus1[ i ] + 1  
 remainingWidthInCtbsY −= colWidth[ i ]  
}  
uniformTileColWidth = pps\_tile\_column\_width\_minus1[ pps\_num\_exp\_tile\_columns\_minus1 ] + 1 (23)  
while( remainingWidthInCtbsY >= uniformTileColWidth ) {  
 colWidth[ i++ ] = uniformTileColWidth  
 remainingWidthInCtbsY −= uniformTileColWidth  
}  
if( remainingWidthInCtbsY > 0 )  
 colWidth[ i++ ] = remainingWidthInCtbsY  
NumTileColumns = i

The variable NumTileRows, specifying the number of tile rows, and the list RowHeight[ 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++ ) {  
 RowHeight[ j ] = pps\_tile\_row\_height\_minus1[ j ] + 1  
 remainingHeightInCtbsY −= RowHeight[ j ]  
}  
uniformTileRowHeight = pps\_tile\_row\_height\_minus1[ pps\_num\_exp\_tile\_rows\_minus1 ] + 1 (24)  
while( remainingHeightInCtbsY >= uniformTileRowHeight ) {  
 RowHeight[ j++ ] = uniformTileRowHeight  
 remainingHeightInCtbsY −= uniformTileRowHeight  
}  
if( remainingHeightInCtbsY > 0 )  
 RowHeight[ j++ ] = remainingHeightInCtbsY  
NumTileRows = j

The variable NumTilesInPic is set equal to NumTileColumns \* NumTileRows.

The list tileColBd[ 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( tileColBd[ 0 ] = 0, i = 0; i < NumTileColumns; i++ )  
 tileColBd[ i + 1 ] = tileColBd[ i ] + colWidth[ i ] (25)

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

The list tileRowBd[ 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( tileRowBd[ 0 ] = 0, j = 0; j < NumTileRows; j++ )  
 tileRowBd[ j + 1 ] = tileRowBd[ j ] + RowHeight[ j ] (26)

NOTE 2 – The size of the array tileRowBd[ ] in the above 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 = = tileColBd[ tileX + 1 ] ) (27)  
 tileX++  
 CtbToTileColBd[ ctbAddrX ] = tileColBd[ tileX ]  
 ctbToTileColIdx[ ctbAddrX ] = tileX  
}

NOTE 3 – The sizes of the arrays CtbToTileColBd[ ] and ctbToTileColIdx[ ] in the above 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 = = tileRowBd[ tileY + 1 ] ) (28)  
 tileY++  
 CtbToTileRowBd[ ctbAddrY ] = tileRowBd[ tileY ]  
 ctbToTileRowIdx[ ctbAddrY ] = tileY  
}

NOTE 4 – The sizes of the arrays CtbToTileRowBd[ ] and ctbToTileRowIdx[ ] in the above 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 ] (29)  
 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 < RowHeight[ ctbToTileRowIdx[ topY ] ] )  
 subpicHeightLessThanOneTileFlag[ i ] = 1  
 else  
 subpicHeightLessThanOneTileFlag[ i ] = 0  
}

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

When pps\_rect\_slice\_flag is equal to 1, the list NumCtusInSlice[ i ] for i ranging from 0 to pps\_num\_slices\_in\_pic\_minus1, inclusive, specifying the number of CTUs in the i-th slice, the list SliceTopLeftTileIdx[ i ] for i ranging from 0 to pps\_num\_slices\_in\_pic\_minus1, inclusive, specifying the tile index of the tile containing the first CTU in the slice, and the matrix CtbAddrInSlice[ i ][ j ] for i ranging from 0 to pps\_num\_slices\_in\_pic\_minus1, inclusive, and j ranging from 0 to NumCtusInSlice[ i ] − 1, inclusive, specifying the picture raster scan address of the j-th CTB within the i-th slice, and the variable NumSlicesInTile[ i ], specifying the number of slices in the tile containing the i-th slice, are derived as follows:

if( pps\_single\_slice\_per\_subpic\_flag ) {  
 if( !sps\_subpic\_info\_present\_flag ) /\* There is no subpicture info and only one slice in a picture. \*/  
 for( j = 0; j < NumTileRows; j++ )  
 for( i = 0; i < NumTileColumns; i++ )  
 AddCtbsToSlice( 0, tileColBd[ i ], tileColBd[ i + 1 ], tileRowBd[ j ], tileRowBd[ 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 = CtbToTileColBd[ sps\_subpic\_ctu\_top\_left\_x[ i ] ]  
 tileY = CtbToTileRowBd[ sps\_subpic\_ctu\_top\_left\_y[ i ] ]  
 for( j = 0; j < SubpicHeightInTiles[ i ]; j++ )  
 for( k = 0; k < SubpicWidthInTiles[ i ]; k++ )  
 AddCtbsToSlice( i, tileColBd[ tileX + k ], tileColBd[ tileX + k + 1 ],  
 tileRowBd[ tileY + j ], tileRowBd[ 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 ) { (30)  
 if( pps\_num\_exp\_slices\_in\_tile[ i ] = = 0 ) {  
 NumSlicesInTile[ i ] = 1  
 sliceHeightInCtus[ i ] = RowHeight[ SliceTopLeftTileIdx[ i ] / NumTileColumns ]  
 } else {  
 remainingHeightInCtbsY = RowHeight[ 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 = tileRowBd[ tileY ]  
 for( j = 0; j < NumSlicesInTile[ i ]; j++ ) {  
 AddCtbsToSlice( i + j, tileColBd[ tileX ], tileColBd[ tileX + 1 ],  
 ctbY, ctbY + sliceHeightInCtus[ i + j ] )  
 ctbY += sliceHeightInCtus[ i + j ]  
 }  
 i += NumSlicesInTile[ i ] − 1  
 } else  
 for( j = 0; j < sliceHeightInTiles[ i ]; j++ )  
 for( k = 0; k < sliceWidthInTiles[ i ]; k++ )  
 AddCtbsToSlice( i, tileColBd[ tileX + k ], tileColBd[ tileX + k + 1 ],  
 tileRowBd[ tileY + j ], tileRowBd[ 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: [Ed. (YK): Consider defining this function in a style consistent with other functions defined in the spec.]

for( ctbY = startY; ctbY < stopY; ctbY++ )  
 for( ctbX = startX; ctbX < stopX; ctbX++ ) {  
 CtbAddrInSlice[ sliceIdx ][ NumCtusInSlice[ sliceIdx ] ] = ctbY \* PicWidthInCtbsY + ctbX (31)  
 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 ] ) && (32)  
 ( 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 ]++  
 }  
 }  
}

### 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 ) { (33)  
 diagScan[ i ][ 0 ] = x  
 diagScan[ i ][ 1 ] = y  
 i++  
 }  
 y− −  
 x++  
 }  
 y = x  
 x = 0  
 if( i >= blkWidth \* blkHeight )  
 stopLoop = TRUE  
}

### Horizontal and vertical traverse scan order array initialization process

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

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

i = 0  
for( y = 0; y < blkHeight; y++ )  
 if( y % 2 = = 0 )  
 for( x = 0; x < blkWidth; x++ ) {  
 hTravScan[ i ][ 0 ] = x  
 hTravScan[ i ][ 1 ] = y  
 i++  
 }  
 else (34)  
 for( x = blkWidth − 1; x >= 0; x− − ) {  
 hTravScan[ i ][ 0 ] = x  
 hTravScan[ i ][ 1 ] = y  
 i++  
 }

i = 0  
for( x = 0; y < blkWidth; y++ )  
 if( x % 2 = = 0 )  
 for( y = 0; y < blkHeight; y++ ) {  
 vTravScan[ i ][ 0 ] = x  
 vTravScan[ i ][ 1 ] = y  
 i++  
 }  
 else (35)  
 for( y = blkHeight − 1; y >= 0; y− − ) {  
 vTravScan[ i ][ 0 ] = x  
 vTravScan[ i ][ 1 ] = y  
 i++  
 }

# Syntax and semantics

## Method of specifying syntax in tabular form

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

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

The following table lists examples of the syntax specification format. When **syntax\_element** appears, it specifies that a syntax element is parsed from the bitstream and the bitstream pointer is advanced to the next position beyond the syntax element in the bitstream parsing process.

|  |  |
| --- | --- |
|  | Descriptor |
| /\* A statement can be a syntax element with an associated descriptor or can be an expression used to specify conditions for the existence, type and quantity of syntax elements, as in the following two examples \*/ |  |
| **syntax\_element** | ue(k) |
| conditioning statement |  |
|  |  |
| /\* A group of statements enclosed in curly brackets is a compound statement and is treated functionally as a single statement. \*/ |  |
| { |  |
| 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 |  |

## Specification of syntax functions and descriptors

The functions presented here are used in the syntactical description. 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 on a byte boundary, i.e., the next bit in the bitstream is the first bit in a byte, the return value of byte\_aligned( ) is equal to TRUE.

– Otherwise, the return value of byte\_aligned( ) is equal to FALSE.

more\_data\_in\_byte\_stream( ), which is used only in the byte stream NAL unit syntax structure specified in Annex B, is specified as follows:

– If more data follow in the byte stream, the return value of more\_data\_in\_byte\_stream( ) is equal to TRUE.

– Otherwise, the return value of more\_data\_in\_byte\_stream( ) is equal to FALSE.

more\_data\_in\_payload( ) is specified as follows:

– If byte\_aligned( ) is equal to TRUE and the current position in the sei\_payload( ) syntax structure is 8 \* payloadSize bits from the beginning of the sei\_payload( ) syntax structure, the return value of more\_data\_in\_payload( ) is equal to FALSE.

– Otherwise, the return value of more\_data\_in\_payload( ) is equal to TRUE.

more\_rbsp\_data( ) is specified as follows:

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

– Otherwise, the RBSP data are searched for the last (least significant, right-most) bit equal to 1 that is present in the RBSP. Given the position of this bit, which is the first bit (rbsp\_stop\_one\_bit) of the rbsp\_trailing\_bits( ) syntax structure, the following applies:

– If there is more data in an RBSP before the rbsp\_trailing\_bits( ) syntax structure, the return value of more\_rbsp\_data( ) is equal to TRUE.

– Otherwise, the return value of more\_rbsp\_data( ) is equal to FALSE.

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

more\_rbsp\_trailing\_data( ) is specified as follows:

– If there is more data in an RBSP, the return value of more\_rbsp\_trailing\_data( ) is equal to TRUE.

– Otherwise, the return value of more\_rbsp\_trailing\_data( ) is equal to FALSE.

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

payload\_extension\_present( ) is specified as follows:

– If the current position in the sei\_payload( ) syntax structure is not the position of the last (least significant, right-most) bit that is equal to 1 that is less than 8 \* payloadSize bits from the beginning of the syntax structure (i.e., the position of the payload\_bit\_equal\_to\_one syntax element), the return value of payload\_extension\_present( ) is equal to TRUE.

– Otherwise, the return value of payload\_extension\_present( ) is equal to FALSE.

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

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

– ae(v): context-adaptive arithmetic entropy-coded syntax element. The parsing process for this descriptor is specified in clause 9.3.

– b(8): byte having any pattern of bit string (8 bits). The parsing process for this descriptor is specified by the return value of the function read\_bits( 8 ).

– f(n): fixed-pattern bit string using n bits written (from left to right) with the left bit first. The parsing process for this descriptor is specified by the return value of the function read\_bits( n ).

– i(n): signed integer using n bits. When n is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function read\_bits( n ) interpreted as a two's complement integer representation with most significant bit written first.

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

– u(n): unsigned integer using n bits. When n is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function read\_bits( n ) interpreted as a binary representation of an unsigned integer with most significant bit written first.

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

## Syntax in tabular form

### NAL unit syntax

#### General NAL unit syntax

|  |  |
| --- | --- |
| nal\_unit( NumBytesInNalUnit ) { | Descriptor |
| nal\_unit\_header( ) |  |
| NumBytesInRbsp = 0 |  |
| for( i = 2; i < NumBytesInNalUnit; i++ ) |  |
| if( i + 2 < NumBytesInNalUnit && next\_bits( 24 ) = = 0x000003 ) { |  |
| **rbsp\_byte**[ NumBytesInRbsp++ ] | b(8) |
| **rbsp\_byte**[ NumBytesInRbsp++ ] | b(8) |
| i += 2 |  |
| **emulation\_prevention\_three\_byte** /\* equal to 0x03 \*/ | f(8) |
| } else |  |
| **rbsp\_byte**[ NumBytesInRbsp++ ] | b(8) |
| } |  |

#### NAL unit header syntax

|  |  |
| --- | --- |
| nal\_unit\_header( ) { | Descriptor |
| **forbidden\_zero\_bit** | f(1) |
| **nuh\_reserved\_zero\_bit** | u(1) |
| **nuh\_layer\_id** | u(6) |
| **nal\_unit\_type** | u(5) |
| **nuh\_temporal\_id\_plus1** | u(3) |
| } |  |

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

#### Decoding capability information RBSP syntax

|  |  |
| --- | --- |
| decoding\_capability\_information\_rbsp( ) { | **Descriptor** |
| **dci\_reserved\_zero\_4bits** | u(4) |
| **dci\_num\_ptls\_minus1** | u(4) |
| for( i = 0; i <= dci\_num\_ptls\_minus1; i++ ) |  |
| profile\_tier\_level( 1, 0 ) |  |
| **dci\_extension\_flag** | u(1) |
| if( dci\_extension\_flag ) |  |
| while( more\_rbsp\_data( ) ) |  |
| **dci\_extension\_data\_flag** | u(1) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

#### Video parameter set RBSP syntax

|  |  |  |
| --- | --- | --- |
| video\_parameter\_set\_rbsp( ) { | Descriptor | |
| **vps\_video\_parameter\_set\_id** | u(4) | |
| **vps\_max\_layers\_minus1** | u(6) | |
| **vps\_max\_sublayers\_minus1** | u(3) | |
| if( vps\_max\_layers\_minus1 > 0 && vps\_max\_sublayers\_minus1 > 0 ) |  | |
| **vps\_all\_layers\_same\_num\_sublayers\_flag** | u(1) | |
| if( vps\_max\_layers\_minus1 > 0 ) |  | |
| **vps\_all\_independent\_layers\_flag** | u(1) | |
| for( i = 0; i <= vps\_max\_layers\_minus1; i++ ) { |  | |
| **vps\_layer\_id**[ i ] | u(6) | |
| if( i > 0 && !vps\_all\_independent\_layers\_flag ) { |  | |
| **vps\_independent\_layer\_flag**[ i ] | u(1) | |
| if( !vps\_independent\_layer\_flag[ i ] ) { |  | |
| **vps\_max\_tid\_ref\_present\_flag**[ i ] | u(1) | |
| for( j = 0; j < i; j++ ) { |  | |
| **vps\_direct\_ref\_layer\_flag**[ i ][ j ] | u(1) | |
| if( vps\_max\_tid\_ref\_present\_flag[ i ] && vps\_direct\_ref\_layer\_flag[ i ][ j ] ) |  | |
| **vps\_max\_tid\_il\_ref\_pics\_plus1**[ i ][ j ] | u(3) | |
| } |  | |
| } |  | |
| } |  | |
| } |  | |
| if( vps\_max\_layers\_minus1 > 0 ) { |  | |
| if( vps\_all\_independent\_layers\_flag ) |  | |
| **vps\_each\_layer\_is\_an\_ols\_flag** | u(1) | |
| if( !vps\_each\_layer\_is\_an\_ols\_flag ) { |  | |
| if( !vps\_all\_independent\_layers\_flag ) |  | |
| **vps\_ols\_mode\_idc** | u(2) | |
| if( vps\_ols\_mode\_idc = = 2 ) { |  | |
| **vps\_num\_output\_layer\_sets\_minus1** | u(8) | |
| for( i = 1; i <= vps\_num\_output\_layer\_sets\_minus1; i ++) |  | |
| for( j = 0; j <= vps\_max\_layers\_minus1; j++ ) |  | |
| **vps\_ols\_output\_layer\_flag**[ i ][ j ] | u(1) | |
| } |  | |
| } |  | |
| } |  | |
| **vps\_num\_ptls\_minus1** | u(8) |
| for( i = 0; i <= vps\_num\_ptls\_minus1; i++ ) { |  | |
| if( i > 0 ) |  | |
| **vps\_pt\_present\_flag**[ i ] | u(1) | |
| if( !vps\_all\_layers\_same\_num\_sublayers\_flag ) |  | |
| **vps\_ptl\_max\_temporal\_id**[ i ] | u(3) | |
| } |  | |
| while( !byte\_aligned( ) ) |  | |
| **vps\_ptl\_alignment\_zero\_bit** /\* equal to 0 \*/ | f(1) | |
| for( i = 0; i <= vps\_num\_ptls\_minus1; i++ ) |  | |
| profile\_tier\_level( vps\_pt\_present\_flag[ i ], vps\_ptl\_max\_temporal\_id[ i ] ) |  |
| for( i = 0; i < TotalNumOlss; i++ ) |  | |
| if( vps\_num\_ptls\_minus1 > 0 && vps\_num\_ptls\_minus1 + 1 != TotalNumOlss ) |  | |
| **vps\_ols\_ptl\_idx**[ i ] | u(8) | |
| if( !vps\_each\_layer\_is\_an\_ols\_flag ) { |  | |
| **vps\_num\_dpb\_params\_minus1** | ue(v) |
| if( vps\_max\_sublayers\_minus1 > 0 ) |  | |
| **vps\_sublayer\_dpb\_params\_present\_flag** | u(1) | |
| for( i = 0; i < VpsNumDpbParams; i++ ) { |  | |
| if( !vps\_all\_layers\_same\_num\_sublayers\_flag ) |  | |
| **vps\_dpb\_max\_temporal\_id**[ i ] | u(3) | |
| dpb\_parameters( vps\_dpb\_max\_temporal\_id[ i ],  vps\_sublayer\_dpb\_params\_present\_flag ) |  |
| } |  |
| for( i = 0; i < NumMultiLayerOlss; i++ ) { |  | |
| **vps\_ols\_dpb\_pic\_width**[ i ] | ue(v) | |
| **vps\_ols\_dpb\_pic\_height**[ i ] | ue(v) | |
| **vps\_ols\_dpb\_chroma\_format**[ i ] | u(2) | |
| **vps\_ols\_dpb\_bitdepth\_minus8**[ i ] | ue(v) | |
| if( VpsNumDpbParams > 1 && vps\_num\_dpb\_params != NumMultiLayerOlss ) |  | |
| **vps\_ols\_dpb\_params\_idx**[ i ] | ue(v) | |
| } |  | |
| **vps\_general\_hrd\_params\_present\_flag** | u(1) | |
| } |  | |
| if( vps\_general\_hrd\_params\_present\_flag ) { |  | |
| general\_hrd\_parameters( ) |  | |
| if( vps\_max\_sublayers\_minus1 > 0 ) |  | |
| **vps\_sublayer\_cpb\_params\_present\_flag** | u(1) | |
| **vps\_num\_ols\_hrd\_params\_minus1** | ue(v) | |
| for( i = 0; i <= vps\_num\_ols\_hrd\_params\_minus1; i++ ) { |  | |
| if( !vps\_all\_layers\_same\_num\_sublayers\_flag ) |  | |
| **hrd\_max\_tid**[ i ] | u(3) | |
| firstSubLayer = vps\_sublayer\_cpb\_params\_present\_flag ? 0 : vps\_hrd\_max\_tid[ i ] |  | |
| ols\_hrd\_parameters( firstSubLayer, vps\_hrd\_max\_tid[ i ] ) |  | |
| } |  | |
| if( vps\_num\_ols\_hrd\_params\_minus1 > 0 &&  vps\_num\_ols\_hrd\_params\_minus1 + 1 != NumMultiLayerOlss ) |  | |
| for( i = 0; i < NumMultiLayerOlss; i++ ) |  | |
| **vps\_ols\_hrd\_idx**[ i ] | ue(v) | |
| } |  | |
| **vps\_extension\_flag** | u(1) | |
| if( vps\_extension\_flag ) |  | |
| while( more\_rbsp\_data( ) ) |  | |
| **vps\_extension\_data\_flag** | u(1) | |
| rbsp\_trailing\_bits( ) |  | |
| } |  | |

#### Sequence parameter set RBSP syntax

|  |  |  |
| --- | --- | --- |
| seq\_parameter\_set\_rbsp( ) { | Descriptor | |
| **sps\_seq\_parameter\_set\_id** | u(4) | |
| **sps\_video\_parameter\_set\_id** | u(4) | |
| **sps\_max\_sublayers\_minus1** | u(3) | |
| **sps\_reserved\_zero\_4bits** | u(4) | |
| **sps\_ptl\_dpb\_hrd\_params\_present\_flag** | u(1) | |
| if( sps\_ptl\_dpb\_hrd\_params\_present\_flag ) |  | |
| profile\_tier\_level( 1, sps\_max\_sublayers\_minus1 ) |  | |
| **sps\_gdr\_enabled\_flag** | u(1) | |
| **sps\_chroma\_format\_idc** | u(2) | |
| if( sps\_chroma\_format\_idc = = 3 ) |  | |
| **sps\_separate\_colour\_plane\_flag** | u(1) | |
| **sps\_ref\_pic\_resampling\_enabled\_flag** | u(1) | |
| if( sps\_ref\_pic\_resampling\_enabled\_flag ) |  | |
| **sps\_res\_change\_in\_clvs\_allowed\_flag** | u(1) | |
| **sps\_pic\_width\_max\_in\_luma\_samples** | ue(v) | |
| **sps\_pic\_height\_max\_in\_luma\_samples** | ue(v) | |
| **sps\_conformance\_window\_flag** | u(1) | |
| if( sps\_conformance\_window\_flag ) { |  | |
| **sps\_conf\_win\_left\_offset** | ue(v) | |
| **sps\_conf\_win\_right\_offset** | ue(v) | |
| **sps\_conf\_win\_top\_offset** | ue(v) | |
| **sps\_conf\_win\_bottom\_offset** | ue(v) | |
| } |  | |
| **sps\_log2\_ctu\_size\_minus5** | u(2) | |
| **sps\_subpic\_info\_present\_flag** | u(1) | |
| if( sps\_subpic\_info\_present\_flag ) { |  | |
| **sps\_num\_subpics\_minus1** | ue(v) | |
| if( sps\_num\_subpics\_minus1 > 0 ) |  | |
| **sps\_independent\_subpics\_flag** | u(1) | |
| for( i = 0; sps\_num\_subpics\_minus1 > 0 && i <= sps\_num\_subpics\_minus1; i++ ) { |  | |
| if( i > 0 && sps\_pic\_width\_max\_in\_luma\_samples > CtbSizeY ) |  | |
| **sps\_subpic\_ctu\_top\_left\_x**[ i ] | u(v) | |
| if( i > 0 && sps\_pic\_height\_max\_in\_luma\_samples > CtbSizeY ) { |  | |
| **sps\_subpic\_ctu\_top\_left\_y**[ i ] | u(v) | |
| if( i < sps\_num\_subpics\_minus1 &&  sps\_pic\_width\_max\_in\_luma\_samples > CtbSizeY ) |  | |
| **sps\_subpic\_width\_minus1**[ i ] | u(v) | |
| if( i < sps\_num\_subpics\_minus1 &&  sps\_pic\_height\_max\_in\_luma\_samples > CtbSizeY ) |  | |
| **sps\_subpic\_height\_minus1**[ i ] | u(v) | |
| if( !sps\_independent\_subpics\_flag) { |  | |
| **sps\_subpic\_treated\_as\_pic\_flag**[ i ] | u(1) | |
| **sps\_loop\_filter\_across\_subpic\_enabled\_flag**[ i ] | u(1) | |
| } |  | |
| } |  | |
| **sps\_subpic\_id\_len\_minus1** | ue(v) | |
| **sps\_subpic\_id\_mapping\_explicitly\_signalled\_flag** | u(1) | |
| if( sps\_subpic\_id\_mapping\_explicitly\_signalled\_flag ) { |  | |
| **sps\_subpic\_id\_mapping\_present\_flag** | u(1) | |
| if( sps\_subpic\_id\_mapping\_present\_flag ) |  | |
| for( i = 0; i <= sps\_num\_subpics\_minus1; i++ ) |  | |
| **sps\_subpic\_id**[ i ] | u(v) | |
| } |  | |
| } |  | |
| **sps\_bit\_depth\_minus8** | ue(v) | |
| **sps\_entropy\_coding\_sync\_enabled\_flag** | u(1) | |
| **sps\_entry\_point\_offsets\_present\_flag** | u(1) | |
| **sps\_log2\_max\_pic\_order\_cnt\_lsb\_minus4** | u(4) | |
| **sps\_poc\_msb\_cycle\_flag** | u(1) | |
| if( sps\_poc\_msb\_cycle\_flag ) |  | |
| **sps\_poc\_msb\_cycle\_len\_minus1** | ue(v) | |
| **sps\_num\_extra\_ph\_bits\_bytes** | u(2) | |
| extra\_ph\_bits\_struct( sps\_num\_extra\_ph\_bits\_bytes ) |  | |
| **sps\_num\_extra\_sh\_bits\_bytes** | u(2) | |
| extra\_sh\_bits\_struct( sps\_num\_extra\_sh\_bits\_bytes ) |  | |
| if( sps\_ptl\_dpb\_hrd\_params\_present\_flag ) { |  | |
| if( sps\_max\_sublayers\_minus1 > 0 ) |  | |
| **sps\_sublayer\_dpb\_params\_flag** | u(1) | |
| dpb\_parameters( sps\_max\_sublayers\_minus1, sps\_sublayer\_dpb\_params\_flag ) |  | |
| } |  | |
| if( ChromaArrayType != 0 ) |  | |
| **sps\_qtbtt\_dual\_tree\_intra\_flag** | u(1) | |
| **sps\_log2\_min\_luma\_coding\_block\_size\_minus2** | ue(v) | |
| **sps\_partition\_constraints\_override\_enabled\_flag** | u(1) | |
| **sps\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_luma** | ue(v) | |
| **sps\_max\_mtt\_hierarchy\_depth\_intra\_slice\_luma** | ue(v) | |
| if( sps\_max\_mtt\_hierarchy\_depth\_intra\_slice\_luma != 0 ) { |  | |
| **sps\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_luma** | ue(v) | |
| **sps\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_luma** | ue(v) | |
| } |  | |
| **sps\_log2\_diff\_min\_qt\_min\_cb\_inter\_slice** | ue(v) | |
| **sps\_max\_mtt\_hierarchy\_depth\_inter\_slice** | ue(v) | |
| if( sps\_max\_mtt\_hierarchy\_depth\_inter\_slice != 0 ) { |  | |
| **sps\_log2\_diff\_max\_bt\_min\_qt\_inter\_slice** | ue(v) | |
| **sps\_log2\_diff\_max\_tt\_min\_qt\_inter\_slice** | ue(v) | |
| } |  | |
| if( sps\_qtbtt\_dual\_tree\_intra\_flag ) { |  | |
| **sps\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_chroma** | ue(v) | |
| **sps\_max\_mtt\_hierarchy\_depth\_intra\_slice\_chroma** | ue(v) | |
| if( sps\_max\_mtt\_hierarchy\_depth\_intra\_slice\_chroma != 0 ) { |  | |
| **sps\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_chroma** | ue(v) | |
| **sps\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_chroma** | ue(v) | |
| } |  | |
| } |  | |
| if( CtbSizeY > 32 ) |  | |
| **sps\_max\_luma\_transform\_size\_64\_flag** | u(1) | |
| if( ChromaArrayType != 0 ) { |  | |
| **sps\_joint\_cbcr\_enabled\_flag** | u(1) | |
| **sps\_same\_qp\_table\_for\_chroma\_flag** | u(1) | |
| numQpTables = sps\_same\_qp\_table\_for\_chroma\_flag ? 1 :  ( sps\_joint\_cbcr\_enabled\_flag ? 3 : 2 ) |  | |
| for( i = 0; i < numQpTables; i++ ) { |  | |
| **sps\_qp\_table\_start\_minus26**[ i ] | se(v) | |
| **sps\_num\_points\_in\_qp\_table\_minus1**[ i ] | ue(v) | |
| for( j = 0; j <= sps\_num\_points\_in\_qp\_table\_minus1[ i ]; j++ ) { |  | |
| **sps\_delta\_qp\_in\_val\_minus1**[ i ][ j ] | ue(v) | |
| **sps\_delta\_qp\_diff\_val**[ i ][ j ] | ue(v) | |
| } |  | |
| } |  | |
| } |  | |
| **sps\_sao\_enabled\_flag** | u(1) | |
| **sps\_alf\_enabled\_flag** | u(1) | |
| if( sps\_alf\_enabled\_flag && ChromaArrayType != 0 ) |  | |
| **sps\_ccalf\_enabled\_flag** | u(1) | |
| **sps\_transform\_skip\_enabled\_flag** | u(1) | |
| if( sps\_transform\_skip\_enabled\_flag ) { |  | |
| **sps\_log2\_transform\_skip\_max\_size\_minus2** | ue(v) | |
| **sps\_bdpcm\_enabled\_flag** | u(1) | |
| } |  | |
| **sps\_weighted\_pred\_flag** | u(1) | |
| **sps\_weighted\_bipred\_flag** | u(1) | |
| **sps\_long\_term\_ref\_pics\_flag** | u(1) | |
| if( sps\_video\_parameter\_set\_id > 0 ) |  | |
| **sps\_inter\_layer\_ref\_pics\_present\_flag** | u(1) | |
| **sps\_idr\_rpl\_present\_flag** | u(1) | |
| **sps\_rpl1\_same\_as\_rpl0\_flag** | u(1) | |
| for( i = 0; i < sps\_rpl1\_same\_as\_rpl0\_flag ? 1 : 2; i++ ) { |  | |
| **sps\_num\_ref\_pic\_lists**[ i ] | ue(v) | |
| for( j = 0; j < sps\_num\_ref\_pic\_lists[ i ]; j++) |  | |
| ref\_pic\_list\_struct( i, j ) |  | |
| } |  | |
| **sps\_ref\_wraparound\_enabled\_flag** | u(1) | |
| **sps\_temporal\_mvp\_enabled\_flag** | u(1) | |
| if( sps\_temporal\_mvp\_enabled\_flag ) |  | |
| **sps\_sbtmvp\_enabled\_flag** | u(1) | |
| **sps\_amvr\_enabled\_flag** | u(1) | |
| **sps\_bdof\_enabled\_flag** | u(1) | |
| if( sps\_bdof\_enabled\_flag ) |  | |
| **sps\_bdof\_control\_present\_in\_ph\_flag** | u(1) | |
| **sps\_smvd\_enabled\_flag** | u(1) | |
| **sps\_dmvr\_enabled\_flag** | u(1) | |
| if( sps\_dmvr\_enabled\_flag) |  | |
| **sps\_dmvr\_control\_present\_in\_ph\_flag** | u(1) | |
| **sps\_mmvd\_enabled\_flag** | u(1) | |
| if( sps\_mmvd\_enabled\_flag ) |  | |
| **sps\_mmvd\_fullpel\_only\_flag** | u(1) | |
| **sps\_six\_minus\_max\_num\_merge\_cand** | ue(v) | |
| **sps\_sbt\_enabled\_flag** | u(1) | |
| **sps\_affine\_enabled\_flag** | u(1) | |
| if( sps\_affine\_enabled\_flag ) { |  | |
| **sps\_five\_minus\_max\_num\_subblock\_merge\_cand** | ue(v) | |
| **sps\_affine\_type\_flag** | u(1) | |
| if( sps\_amvr\_enabled\_flag ) |  | |
| **sps\_affine\_amvr\_enabled\_flag** | u(1) | |
| **sps\_affine\_prof\_enabled\_flag** | u(1) | |
| if( sps\_affine\_prof\_enabled\_flag ) |  | |
| **sps\_prof\_control\_present\_in\_ph\_flag** | u(1) | |
| } |  | |
| **sps\_bcw\_enabled\_flag** | u(1) | |
| **sps\_ciip\_enabled\_flag** | u(1) | |
| if( MaxNumMergeCand >= 2 ) { |  | |
| **sps\_gpm\_enabled\_flag** | u(1) | |
| if( sps\_gpm\_enabled\_flag && MaxNumMergeCand >= 3 ) |  | |
| **sps\_max\_num\_merge\_cand\_minus\_max\_num\_gpm\_cand** | ue(v) | |
| } |  | |
| **sps\_log2\_parallel\_merge\_level\_minus2** | ue(v) | |
| **sps\_isp\_enabled\_flag** | u(1) | |
| **sps\_mrl\_enabled\_flag** | u(1) | |
| **sps\_mip\_enabled\_flag** | u(1) | |
| if( ChromaArrayType != 0 ) |  | |
| **sps\_cclm\_enabled\_flag** | u(1) | |
| if( sps\_chroma\_format\_idc = = 1 ) { |  | |
| **sps\_chroma\_horizontal\_collocated\_flag** | u(1) | |
| **sps\_chroma\_vertical\_collocated\_flag** | u(1) | |
| } |  | |
| **sps\_mts\_enabled\_flag** | u(1) | |
| if( sps\_mts\_enabled\_flag ) { |  | |
| **sps\_explicit\_mts\_intra\_enabled\_flag** | u(1) | |
| **sps\_explicit\_mts\_inter\_enabled\_flag** | u(1) | |
| } |  | |
| **sps\_palette\_enabled\_flag** | u(1) | |
| if( ChromaArrayType = = 3 && !sps\_max\_luma\_transform\_size\_64\_flag ) |  | |
| **sps\_act\_enabled\_flag** | u(1) | |
| if( sps\_transform\_skip\_enabled\_flag | | sps\_palette\_enabled\_flag ) |  | |
| **sps\_internal\_bit\_depth\_minus\_input\_bit\_depth** | ue(v) | |
| **sps\_ibc\_enabled\_flag** | u(1) | |
| if( sps\_ibc\_enabled\_flag ) |  | |
| **sps\_six\_minus\_max\_num\_ibc\_merge\_cand** | ue(v) | |
| **sps\_lmcs\_enabled\_flag** | u(1) | |
| **sps\_lfnst\_enabled\_flag** | u(1) | |
| **sps\_ladf\_enabled\_flag** | u(1) | |
| if( sps\_ladf\_enabled\_flag ) { |  | |
| **sps\_num\_ladf\_intervals\_minus2** | u(2) | |
| **sps\_ladf\_lowest\_interval\_qp\_offset** | se(v) | |
| for( i = 0; i < sps\_num\_ladf\_intervals\_minus2 + 1; i++ ) { |  | |
| **sps\_ladf\_qp\_offset**[ i ] | se(v) | |
| **sps\_ladf\_delta\_threshold\_minus1**[ i ] | ue(v) | |
| } |  | |
| } |  | |
| **sps\_explicit\_scaling\_list\_enabled\_flag** | u(1) | |
| if( sps\_lfnst\_enabled\_flag && sps\_explicit\_scaling\_list\_enabled\_flag ) |  | |
| **sps\_scaling\_matrix\_for\_lfnst\_disabled\_flag** | u(1) |
| if( sps\_act\_enabled\_flag && sps\_explicit\_scaling\_list\_enabled\_flag ) |  |
| **sps\_scaling\_matrix\_for\_alternative\_colour\_space\_disabled\_flag** | u(1) |
| if( sps\_scaling\_matrix\_for\_alternative\_colour\_space\_disabled\_flag ) |  |
| **sps\_scaling\_matrix\_designated\_colour\_space\_flag** | u(1) |
| **sps\_dep\_quant\_enabled\_flag** | u(1) | |
| if( !sps\_dep\_quant\_enabled\_flag ) |  | |
| **sps\_sign\_data\_hiding\_enabled\_flag** | u(1) | |
| **sps\_virtual\_boundaries\_enabled\_flag** | u(1) | |
| if( sps\_virtual\_boundaries\_enabled\_flag ) { |  | |
| **sps\_virtual\_boundaries\_present\_flag** | u(1) | |
| if( sps\_virtual\_boundaries\_present\_flag ) { |  | |
| **sps\_num\_ver\_virtual\_boundaries** | u(2) | |
| for( i = 0; i < sps\_num\_ver\_virtual\_boundaries; i++ ) |  | |
| **sps\_virtual\_boundary\_pos\_x**[ i ] | ue(v) | |
| **sps\_num\_hor\_virtual\_boundaries** | u(2) | |
| for( i = 0; i < sps\_num\_hor\_virtual\_boundaries; i++ ) |  | |
| **sps\_virtual\_boundary\_pos\_y**[ i ] | ue(v) | |
| } |  | |
| } |  | |
| if( sps\_ptl\_dpb\_hrd\_params\_present\_flag ) { |  | |
| **sps\_general\_hrd\_params\_present\_flag** | u(1) | |
| if( sps\_general\_hrd\_params\_present\_flag ) { |  | |
| general\_hrd\_parameters( ) |  | |
| if( sps\_max\_sublayers\_minus1 > 0 ) |  | |
| **sps\_sublayer\_cpb\_params\_present\_flag** | u(1) | |
| firstSubLayer = sps\_sublayer\_cpb\_params\_present\_flag ? 0 :  sps\_max\_sublayers\_minus1 |  | |
| ols\_hrd\_parameters( firstSubLayer, sps\_max\_sublayers\_minus1 ) |  | |
| } |  | |
| } |  | |
| **sps\_field\_seq\_flag** | u(1) | |
| **sps\_vui\_parameters\_present\_flag** | u(1) | |
| if( sps\_vui\_parameters\_present\_flag ) |  | |
| vui\_parameters( ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  | |
| **sps\_extension\_flag** | u(1) | |
| if( sps\_extension\_flag ) |  | |
| while( more\_rbsp\_data( ) ) |  | |
| **sps\_extension\_data\_flag** | u(1) | |
| rbsp\_trailing\_bits( ) |  | |
| } |  | |

#### Picture parameter set RBSP syntax

|  |  |
| --- | --- |
| pic\_parameter\_set\_rbsp( ) { | Descriptor |
| **pps\_pic\_parameter\_set\_id** | u(6) |
| **pps\_seq\_parameter\_set\_id** | u(4) |
| **pps\_mixed\_nalu\_types\_in\_pic\_flag** | u(1) |
| **pps\_pic\_width\_in\_luma\_samples** | ue(v) |
| **pps\_pic\_height\_in\_luma\_samples** | ue(v) |
| **pps\_conformance\_window\_flag** | u(1) |
| if( pps\_conformance\_window\_flag ) { |  |
| **pps\_conf\_win\_left\_offset** | ue(v) |
| **pps\_conf\_win\_right\_offset** | ue(v) |
| **pps\_conf\_win\_top\_offset** | ue(v) |
| **pps\_conf\_win\_bottom\_offset** | ue(v) |
| } |  |
| **pps\_scaling\_window\_explicit\_signalling\_flag** | u(1) |
| if( pps\_scaling\_window\_explicit\_signalling\_flag ) { |  |
| **pps\_scaling\_win\_left\_offset** | se(v) |
| **pps\_scaling\_win\_right\_offset** | se(v) |
| **pps\_scaling\_win\_top\_offset** | se(v) |
| **pps\_scaling\_win\_bottom\_offset** | se(v) |
| } |  |
| **pps\_output\_flag\_present\_flag** | u(1) |
| **pps\_no\_pic\_partition\_flag** | u(1) |
| **pps\_subpic\_id\_mapping\_present\_flag** | u(1) |
| if( pps\_subpic\_id\_mapping\_present\_flag ) { |  |
| if( !pps\_no\_pic\_partition\_flag ) |  |
| **pps\_num\_subpics\_minus1** | ue(v) |
| **pps\_subpic\_id\_len\_minus1** | ue(v) |
| for( i = 0; i <= pps\_num\_subpic\_minus1; i++ ) |  |
| **pps\_subpic\_id**[ i ] | u(v) |
| } |  |
| if( !pps\_no\_pic\_partition\_flag ) { |  |
| **pps\_log2\_ctu\_size\_minus5** | u(2) |
| **pps\_num\_exp\_tile\_columns\_minus1** | ue(v) |
| **pps\_num\_exp\_tile\_rows\_minus1** | ue(v) |
| for( i = 0; i <= pps\_num\_exp\_tile\_columns\_minus1; i++ ) |  |
| **pps\_tile\_column\_width\_minus1**[ i ] | ue(v) |
| for( i = 0; i <= pps\_num\_exp\_tile\_rows\_minus1; i++ ) |  |
| **pps\_tile\_row\_height\_minus1**[ i ] | ue(v) |
| if( NumTilesInPic > 1 ) { |  |
| **pps\_loop\_filter\_across\_tiles\_enabled\_flag** | u(1) |
| **pps\_rect\_slice\_flag** | u(1) |
| } |  |
| if( pps\_rect\_slice\_flag ) |  |
| **pps\_single\_slice\_per\_subpic\_flag** | u(1) |
| if( pps\_rect\_slice\_flag && !pps\_single\_slice\_per\_subpic\_flag ) { |  |
| **pps\_num\_slices\_in\_pic\_minus1** | ue(v) |
| if( pps\_num\_slices\_in\_pic\_minus1 > 1 ) |  |
| **pps\_tile\_idx\_delta\_present\_flag** | u(1) |
| for( i = 0; i < pps\_num\_slices\_in\_pic\_minus1; i++ ) { |  |
| if( SliceTopLeftTileIdx[ i ] % NumTileColumns != NumTileColumns − 1 ) |  |
| **pps\_slice\_width\_in\_tiles\_minus1**[ i ] | ue(v) |
| if( SliceTopLeftTileIdx[ i ] / NumTileColumns != NumTileRows − 1 &&  ( pps\_tile\_idx\_delta\_present\_flag | |  SliceTopLeftTileIdx[ i ] % NumTileColumns = = 0 ) ) |  |
| **pps\_slice\_height\_in\_tiles\_minus1**[ i ] | ue(v) |
| if( pps\_slice\_width\_in\_tiles\_minus1[ i ] = = 0 &&  pps\_slice\_height\_in\_tiles\_minus1[ i ] = = 0 &&  RowHeight[ SliceTopLeftTileIdx[ i ] / NumTileColumns ] > 1 ) { |  |
| **pps\_num\_exp\_slices\_in\_tile**[ i ] | ue(v) |
| for( j = 0; j < pps\_num\_exp\_slices\_in\_tile[ i ]; j++ ) |  |
| **pps\_exp\_slice\_height\_in\_ctus\_minus1**[ i ][ j ] | ue(v) |
| i += NumSlicesInTile[ i ] − 1 |  |
| } |  |
| if( pps\_tile\_idx\_delta\_present\_flag && i < pps\_num\_slices\_in\_pic\_minus1 ) |  |
| **pps\_tile\_idx\_delta\_val**[ i ] | se(v) |
| } |  |
| } |  |
| if( !pps\_rect\_slice\_flag | | pps\_single\_slice\_per\_subpic\_flag | |  pps\_num\_slices\_in\_pic\_minus1 > 0 ) |  |
| **pps\_loop\_filter\_across\_slices\_enabled\_flag** | u(1) |
| } |  |
| **pps\_cabac\_init\_present\_flag** | u(1) |
| for( i = 0; i < 2; i++ ) |  |
| **pps\_num\_ref\_idx\_default\_active\_minus1**[ i ] | ue(v) |
| **pps\_rpl1\_idx\_present\_flag** | u(1) |
| **pps\_init\_qp\_minus26** | se(v) |
| **pps\_cu\_qp\_delta\_enabled\_flag** | u(1) |
| **pps\_chroma\_tool\_offsets\_present\_flag** | u(1) |
| if( pps\_chroma\_tool\_offsets\_present\_flag ) { |  |
| **pps\_cb\_qp\_offset** | se(v) |
| **pps\_cr\_qp\_offset** | se(v) |
| **pps\_joint\_cbcr\_qp\_offset\_present\_flag** | u(1) |
| if( pps\_joint\_cbcr\_qp\_offset\_present\_flag ) |  |
| **pps\_joint\_cbcr\_qp\_offset\_value** | se(v) |
| **pps\_slice\_chroma\_qp\_offsets\_present\_flag** | u(1) |
| **pps\_cu\_chroma\_qp\_offset\_list\_enabled\_flag** | u(1) |
| } |  |
| if( pps\_cu\_chroma\_qp\_offset\_list\_enabled\_flag ) { |  |
| **pps\_chroma\_qp\_offset\_list\_len\_minus1** | ue(v) |
| for( i = 0; i <= pps\_chroma\_qp\_offset\_list\_len\_minus1; i++ ) { |  |
| **pps\_cb\_qp\_offset\_list**[ i ] | se(v) |
| **pps\_cr\_qp\_offset\_list**[ i ] | se(v) |
| if( pps\_joint\_cbcr\_qp\_offset\_present\_flag ) |  |
| **pps\_joint\_cbcr\_qp\_offset\_list**[ i ] | se(v) |
| } |  |
| } |  |
| **pps\_weighted\_pred\_flag** | u(1) |
| **pps\_weighted\_bipred\_flag** | u(1) |
| **pps\_deblocking\_filter\_control\_present\_flag** | u(1) |
| if( pps\_deblocking\_filter\_control\_present\_flag ) { |  |
| **pps\_deblocking\_filter\_override\_enabled\_flag** | u(1) |
| **pps\_deblocking\_filter\_disabled\_flag** | u(1) |
| if( !pps\_no\_pic\_partition\_flag && pps\_deblocking\_filter\_override\_enabled\_flag ) |  |
| **pps\_dbf\_info\_in\_ph\_flag** | u(1) |
| if( !pps\_deblocking\_filter\_disabled\_flag ) { |  |
| **pps\_luma\_beta\_offset\_div2** | se(v) |
| **pps\_luma\_tc\_offset\_div2** | se(v) |
| if( pps\_chroma\_tool\_offsets\_present\_flag ) { |  |
| **pps\_cb\_beta\_offset\_div2** | se(v) |
| **pps\_cb\_tc\_offset\_div2** | se(v) |
| **pps\_cr\_beta\_offset\_div2** | se(v) |
| **pps\_cr\_tc\_offset\_div2** | se(v) |
| } |  |
| } |  |
| } |  |
| if( !pps\_no\_pic\_partition\_flag ) { |  |
| **pps\_rpl\_info\_in\_ph\_flag** | u(1) |
| **pps\_sao\_info\_in\_ph\_flag** | u(1) |
| **pps\_alf\_info\_in\_ph\_flag** | u(1) |
| if( ( pps\_weighted\_pred\_flag | | pps\_weighted\_bipred\_flag ) &&  pps\_rpl\_info\_in\_ph\_flag ) |  |
| **pps\_wp\_info\_in\_ph\_flag** | u(1) |
| **pps\_qp\_delta\_info\_in\_ph\_flag** | u(1) |
| } |  |
| **pps\_ref\_wraparound\_enabled\_flag** | u(1) |
| if( pps\_ref\_wraparound\_enabled\_flag ) |  |
| **pps\_pic\_width\_minus\_wraparound\_offset** | ue(v) |
| **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( ) |  |
| } |  |

#### Adaptation parameter set RBSP syntax

|  |  |
| --- | --- |
| adaptation\_parameter\_set\_rbsp( ) { | Descriptor |
| **aps\_adaptation\_parameter\_set\_id** | u(5) |
| **aps\_params\_type** | u(3) |
| **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( ) |  |
| } |  |

#### Picture header RBSP syntax

|  |  |
| --- | --- |
| picture\_header\_rbsp( ) { | Descriptor |
| picture\_header\_structure( ) |  |
| rbsp\_trailing\_bits( ) |  |
| } |  |

#### Picture header structure syntax

[Ed. (YK): Move this subclause to be somewhere after the subclause of the RBSP trailing bits syntax.]

|  |  |
| --- | --- |
| picture\_header\_structure( ) { | Descriptor |
| **ph\_gdr\_or\_irap\_pic\_flag** | u(1) |
| if( ph\_gdr\_or\_irap\_pic\_flag ) |  |
| **ph\_gdr\_pic\_flag** | u(1) |
| **ph\_inter\_slice\_allowed\_flag** | u(1) |
| if( ph\_inter\_slice\_allowed\_flag ) |  |
| **ph\_intra\_slice\_allowed\_flag** | u(1) |
| **ph\_non\_reference\_picture\_flag** | u(1) |
| **ph\_pic\_parameter\_set\_id** | ue(v) |
| **ph\_pic\_order\_cnt\_lsb** | u(v) |
| if( ph\_gdr\_or\_irap\_pic\_flag ) |  |
| **ph\_no\_output\_of\_prior\_pics\_flag** | u(1) |
| if( ph\_gdr\_pic\_flag ) |  |
| **ph\_recovery\_poc\_cnt** | ue(v) |
| for( i = 0; i < NumExtraPhBits; i++ ) |  |
| **ph\_extra\_bit**[ i ] | u(1) |
| if( sps\_poc\_msb\_cycle\_flag ) { |  |
| **ph\_poc\_msb\_cycle\_present\_flag** | u(1) |
| if( ph\_poc\_msb\_cycle\_present\_flag ) |  |
| **ph\_poc\_msb\_cycle\_val** | u(v) |
| } |  |
| if( sps\_alf\_enabled\_flag && pps\_alf\_info\_in\_ph\_flag ) { |  |
| **ph\_alf\_enabled\_flag** | u(1) |
| if( ph\_alf\_enabled\_flag ) { |  |
| **ph\_num\_alf\_aps\_ids\_luma** | u(3) |
| for( i = 0; i < ph\_num\_alf\_aps\_ids\_luma; i++ ) |  |
| **ph\_alf\_aps\_id\_luma**[ i ] | u(3) |
| if( ChromaArrayType != 0 ) { |  |
| **ph\_alf\_cb\_flag** | u(1) |
| **ph\_alf\_cr\_flag** | u(1) |
| } |  |
| if( ph\_alf\_cb\_flag | | ph\_alf\_cr\_flag ) |  |
| **ph\_alf\_aps\_id\_chroma** | u(3) |
| if( sps\_ccalf\_enabled\_flag ) { |  |
| **ph\_cc\_alf\_cb\_enabled\_flag** | u(1) |
| if( ph\_cc\_alf\_cb\_enabled\_flag ) |  |
| **ph\_cc\_alf\_cb\_aps\_id** | u(3) |
| **ph\_cc\_alf\_cr\_enabled\_flag** | u(1) |
| if( ph\_cc\_alf\_cr\_enabled\_flag ) |  |
| **ph\_cc\_alf\_cr\_aps\_id** | u(3) |
| } |  |
| } |  |
| } |  |
| if( sps\_lmcs\_enabled\_flag ) { |  |
| **ph\_lmcs\_enabled\_flag** | u(1) |
| if( ph\_lmcs\_enabled\_flag ) { |  |
| **ph\_lmcs\_aps\_id** | u(2) |
| if( ChromaArrayType != 0 ) |  |
| **ph\_chroma\_residual\_scale\_flag** | u(1) |
| } |  |
| } |  |
| if( sps\_explicit\_scaling\_list\_enabled\_flag ) { |  |
| **ph\_explicit\_scaling\_list\_enabled\_flag** | u(1) |
| if( ph\_explicit\_scaling\_list\_enabled\_flag ) |  |
| **ph\_scaling\_list\_aps\_id** | u(3) |
| } |  |
| if( sps\_virtual\_boundaries\_enabled\_flag && !sps\_virtual\_boundaries\_present\_flag ) { |  |
| **ph\_virtual\_boundaries\_present\_flag** | u(1) |
| if( ph\_virtual\_boundaries\_present\_flag ) { |  |
| **ph\_num\_ver\_virtual\_boundaries** | u(2) |
| for( i = 0; i < ph\_num\_ver\_virtual\_boundaries; i++ ) |  |
| **ph\_virtual\_boundary\_pos\_x**[ i ] | ue(v) |
| **ph\_num\_hor\_virtual\_boundaries** | u(2) |
| for( i = 0; i < ph\_num\_hor\_virtual\_boundaries; i++ ) |  |
| **ph\_virtual\_boundary\_pos\_y**[ i ] | ue(v) |
| } |  |
| } |  |
| if( pps\_output\_flag\_present\_flag && !ph\_non\_reference\_picture\_flag ) |  |
| **ph\_pic\_output\_flag** | u(1) |
| if( pps\_rpl\_info\_in\_ph\_flag ) |  |
| ref\_pic\_lists( ) |  |
| if( sps\_partition\_constraints\_override\_enabled\_flag ) |  |
| **ph\_partition\_constraints\_override\_flag** | u(1) |
| if( ph\_intra\_slice\_allowed\_flag ) { |  |
| if( ph\_partition\_constraints\_override\_flag ) { |  |
| **ph\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_luma** | ue(v) |
| **ph\_max\_mtt\_hierarchy\_depth\_intra\_slice\_luma** | ue(v) |
| if( ph\_max\_mtt\_hierarchy\_depth\_intra\_slice\_luma != 0 ) { |  |
| **ph\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_luma** | ue(v) |
| **ph\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_luma** | ue(v) |
| } |  |
| if( sps\_qtbtt\_dual\_tree\_intra\_flag ) { |  |
| **ph\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_chroma** | ue(v) |
| **ph\_max\_mtt\_hierarchy\_depth\_intra\_slice\_chroma** | ue(v) |
| if( ph\_max\_mtt\_hierarchy\_depth\_intra\_slice\_chroma != 0 ) { |  |
| **ph\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_chroma** | ue(v) |
| **ph\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_chroma** | ue(v) |
| } |  |
| } |  |
| } |  |
| if( pps\_cu\_qp\_delta\_enabled\_flag ) |  |
| **ph\_cu\_qp\_delta\_subdiv\_intra\_slice** | ue(v) |
| if( pps\_cu\_chroma\_qp\_offset\_list\_enabled\_flag ) |  |
| **ph\_cu\_chroma\_qp\_offset\_subdiv\_intra\_slice** | ue(v) |
| } |  |
| if( ph\_inter\_slice\_allowed\_flag ) { |  |
| if( ph\_partition\_constraints\_override\_flag ) { |  |
| **ph\_log2\_diff\_min\_qt\_min\_cb\_inter\_slice** | ue(v) |
| **ph\_max\_mtt\_hierarchy\_depth\_inter\_slice** | ue(v) |
| if( ph\_max\_mtt\_hierarchy\_depth\_inter\_slice != 0 ) { |  |
| **ph\_log2\_diff\_max\_bt\_min\_qt\_inter\_slice** | ue(v) |
| **ph\_log2\_diff\_max\_tt\_min\_qt\_inter\_slice** | ue(v) |
| } |  |
| } |  |
| if( pps\_cu\_qp\_delta\_enabled\_flag ) |  |
| **ph\_cu\_qp\_delta\_subdiv\_inter\_slice** | ue(v) |
| if( pps\_cu\_chroma\_qp\_offset\_list\_enabled\_flag ) |  |
| **ph\_cu\_chroma\_qp\_offset\_subdiv\_inter\_slice** | ue(v) |
| if( sps\_temporal\_mvp\_enabled\_flag ) { |  |
| **ph\_temporal\_mvp\_enabled\_flag** | u(1) |
| if( ph\_temporal\_mvp\_enabled\_flag && pps\_rpl\_info\_in\_ph\_flag ) { |  |
| if( num\_ref\_entries[ 1 ][ RplsIdx[ 1 ] ] > 0 ) |  |
| **ph\_collocated\_from\_l0\_flag** | u(1) |
| if( ( ph\_collocated\_from\_l0\_flag &&  num\_ref\_entries[ 0 ][ RplsIdx[ 0 ] ] > 1 ) | |  ( !ph\_collocated\_from\_l0\_flag &&  num\_ref\_entries[ 1 ][ RplsIdx[ 1 ] ] > 1 ) ) |  |
| **ph\_collocated\_ref\_idx** | ue(v) |
| } |  |
| } |  |
| if( sps\_mmvd\_fullpel\_only\_flag ) |  |
| **ph\_mmvd\_fullpel\_only\_flag** | u(1) |
| if( !pps\_rpl\_info\_in\_ph\_flag | | num\_ref\_entries[ 1 ][ RplsIdx[ 1 ] ] > 0 ) { |  |
| **ph\_mvd\_l1\_zero\_flag** | u(1) |
| if( sps\_bdof\_control\_present\_in\_ph\_flag ) |  |
| **ph\_bdof\_disabled\_flag** | u(1) |
| if( sps\_dmvr\_control\_present\_in\_ph\_flag ) |  |
| **ph\_dmvr\_disabled\_flag** | u(1) |
| } |  |
| if( sps\_prof\_control\_present\_in\_ph\_flag ) |  |
| **ph\_prof\_disabled\_flag** | u(1) |
| if( ( pps\_weighted\_pred\_flag | | pps\_weighted\_bipred\_flag ) &&  pps\_wp\_info\_in\_ph\_flag ) |  |
| pred\_weight\_table( ) |  |
| } |  |
| if( pps\_qp\_delta\_info\_in\_ph\_flag ) |  |
| **ph\_qp\_delta** | se(v) |
| if( sps\_joint\_cbcr\_enabled\_flag ) |  |
| **ph\_joint\_cbcr\_sign\_flag** | u(1) |
| if( sps\_sao\_enabled\_flag && pps\_sao\_info\_in\_ph\_flag ) { |  |
| **ph\_sao\_luma\_enabled\_flag** | u(1) |
| if( ChromaArrayType != 0 ) |  |
| **ph\_sao\_chroma\_enabled\_flag** | u(1) |
| } |  |
| if( pps\_deblocking\_filter\_override\_enabled\_flag && pps\_dbf\_info\_in\_ph\_flag ) { |  |
| **ph\_deblocking\_filter\_override\_flag** | u(1) |
| if( ph\_deblocking\_filter\_override\_flag ) { |  |
| if( !pps\_deblocking\_filter\_disabled\_flag ) |  |
| **ph\_deblocking\_filter\_disabled\_flag** | u(1) |
| if( !ph\_deblocking\_filter\_disabled\_flag ) { |  |
| **ph\_luma\_beta\_offset\_div2** | se(v) |
| **ph\_luma\_tc\_offset\_div2** | se(v) |
| if( pps\_chroma\_tool\_offsets\_present\_flag ) { |  |
| **ph\_cb\_beta\_offset\_div2** | se(v) |
| **ph\_cb\_tc\_offset\_div2** | se(v) |
| **ph\_cr\_beta\_offset\_div2** | se(v) |
| **ph\_cr\_tc\_offset\_div2** | se(v) |
| } |  |
| } |  |
| } |  |
| } |  |
| if( pps\_picture\_header\_extension\_present\_flag ) { |  |
| **ph\_extension\_length** | ue(v) |
| for( i = 0; i < ph\_extension\_length; i++) |  |
| **ph\_extension\_data\_byte**[ i ] | u(8) |
| } |  |
| } |  |

#### Supplemental enhancement information RBSP syntax

|  |  |
| --- | --- |
| sei\_rbsp( ) { | Descriptor |
| do |  |
| sei\_message( ) |  |
| while( more\_rbsp\_data( ) ) |  |
| rbsp\_trailing\_bits( ) |  |
| } |  |

#### AU delimiter RBSP syntax

|  |  |
| --- | --- |
| access\_unit\_delimiter\_rbsp( ) { | Descriptor |
| **aud\_irap\_or\_gdr\_au\_flag** | u(1) |
| **aud\_pic\_type** | u(3) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

#### End of sequence RBSP syntax

|  |  |
| --- | --- |
| end\_of\_seq\_rbsp( ) { | Descriptor |
| } |  |

#### End of bitstream RBSP syntax

|  |  |
| --- | --- |
| end\_of\_bitstream\_rbsp( ) { | Descriptor |
| } |  |

#### Filler data RBSP syntax

|  |  |
| --- | --- |
| filler\_data\_rbsp( ) { | Descriptor |
| while( next\_bits( 8 ) = = 0xFF ) |  |
| **fd\_ff\_byte** /\* equal to 0xFF \*/ | f(8) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

#### Slice layer RBSP syntax

|  |  |
| --- | --- |
| slice\_layer\_rbsp( ) { | Descriptor |
| slice\_header( ) |  |
| slice\_data( ) |  |
| rbsp\_slice\_trailing\_bits( ) |  |
| } |  |

#### RBSP slice trailing bits syntax

|  |  |
| --- | --- |
| rbsp\_slice\_trailing\_bits( ) { | Descriptor |
| rbsp\_trailing\_bits( ) |  |
| while( more\_rbsp\_trailing\_data( ) ) |  |
| **cabac\_zero\_word** /\* equal to 0x0000 \*/ | f(16) |
| } |  |

#### RBSP trailing bits syntax

|  |  |
| --- | --- |
| rbsp\_trailing\_bits( ) { | Descriptor |
| **rbsp\_stop\_one\_bit** /\* equal to 1 \*/ | f(1) |
| while( !byte\_aligned( ) ) |  |
| **rbsp\_alignment\_zero\_bit** /\* equal to 0 \*/ | f(1) |
| } |  |

#### Byte alignment syntax

|  |  |
| --- | --- |
| byte\_alignment( ) { | Descriptor |
| **alignment\_bit\_equal\_to\_one** /\* equal to 1 \*/ | f(1) |
| while( !byte\_aligned( ) ) |  |
| **alignment\_bit\_equal\_to\_zero** /\* equal to 0 \*/ | f(1) |
| } |  |

#### Extra picture header bits structure syntax

|  |  |
| --- | --- |
| extra\_ph\_bits\_struct( numExtraBtyes ) { | Descriptor |
| for( i = 0; i < ( numExtraBtyes \* 8 ); i++ ) |  |
| **extra\_ph\_bit\_present\_flag**[ i ] | u(1) |
| } |  |

#### Extra slice header bits structure syntax

|  |  |
| --- | --- |
| extra\_sh\_bits\_struct( numExtraBtyes ) { | Descriptor |
| for( i = 0; i < ( numExtraBtyes \* 8 ); i++ ) |  |
| **extra\_sh\_bit\_present\_flag**[ i ] | u(1) |
| } |  |

#### Adaptive loop filter data syntax

|  |  |
| --- | --- |
| alf\_data( ) { | Descriptor |
| **alf\_luma\_filter\_signal\_flag** | u(1) |
| if( aps\_chroma\_present\_flag ) { |  |
| **alf\_chroma\_filter\_signal\_flag** | u(1) |
| **alf\_cc\_cb\_filter\_signal\_flag** | u(1) |
| **alf\_cc\_cr\_filter\_signal\_flag** | u(1) |
| } |  |
| if( alf\_luma\_filter\_signal\_flag ) { |  |
| **alf\_luma\_clip\_flag** | u(1) |
| **alf\_luma\_num\_filters\_signalled\_minus1** | ue(v) |
| if( alf\_luma\_num\_filters\_signalled\_minus1 > 0 ) |  |
| for( filtIdx = 0; filtIdx < NumAlfFilters; filtIdx++ ) |  |
| **alf\_luma\_coeff\_delta\_idx**[ filtIdx ] | u(v) |
| for( sfIdx = 0; sfIdx <= alf\_luma\_num\_filters\_signalled\_minus1; sfIdx++ ) |  |
| for( j = 0; j < 12; j++ ) { |  |
| **alf\_luma\_coeff\_abs**[ sfIdx ][ j ] | ue(v) |
| if( alf\_luma\_coeff\_abs[ sfIdx ][ j ] ) |  |
| **alf\_luma\_coeff\_sign**[ sfIdx ][ j ] | u(1) |
| } |  |
| if( alf\_luma\_clip\_flag ) |  |
| for( sfIdx = 0; sfIdx <= alf\_luma\_num\_filters\_signalled\_minus1; sfIdx++ ) |  |
| for( j = 0; j < 12; j++ ) |  |
| **alf\_luma\_clip\_idx**[ sfIdx ][ j ] | u(2) |
| } |  |
| if( alf\_chroma\_filter\_signal\_flag ) { |  |
| **alf\_chroma\_clip\_flag** | u(1) |
| **alf\_chroma\_num\_alt\_filters\_minus1** | ue(v) |
| for( altIdx = 0; altIdx <= alf\_chroma\_num\_alt\_filters\_minus1; altIdx++ ) { |  |
| for( j = 0; j < 6; j++ ) { |  |
| **alf\_chroma\_coeff\_abs**[ altIdx ][ j ] | ue(v) |
| if( alf\_chroma\_coeff\_abs[ altIdx ][ j ] > 0 ) |  |
| **alf\_chroma\_coeff\_sign**[ altIdx ][ j ] | u(1) |
| } |  |
| if( alf\_chroma\_clip\_flag ) |  |
| for( j = 0; j < 6; j++ ) |  |
| **alf\_chroma\_clip\_idx**[ altIdx ][ j ] | u(2) |
| } |  |
| } |  |
| if( alf\_cc\_cb\_filter\_signal\_flag ) { |  |
| **alf\_cc\_cb\_filters\_signalled\_minus1** | ue(v) |
| for( k = 0; k < alf\_cc\_cb\_filters\_signalled\_minus1 + 1; k++ ) { |  |
| for( j = 0; j < 7; j++ ) { |  |
| **alf\_cc\_cb\_mapped\_coeff\_abs**[ k ][ j ] | u(3) |
| if( alf\_cc\_cb\_mapped\_coeff\_abs[ k ][ j ] ) |  |
| **alf\_cc\_cb\_coeff\_sign**[ k ][ j ] | u(1) |
| } |  |
| } |  |
| } |  |
| if( alf\_cc\_cr\_filter\_signal\_flag ) { |  |
| **alf\_cc\_cr\_filters\_signalled\_minus1** | ue(v) |
| for( k = 0; k < alf\_cc\_cr\_filters\_signalled\_minus1 + 1; k++ ) { |  |
| for( j = 0; j < 7; j++ ) { |  |
| **alf\_cc\_cr\_mapped\_coeff\_abs**[ k ][ j ] | u(3) |
| if( alf\_cc\_cr\_mapped\_coeff\_abs[ k ][ j ] ) |  |
| **alf\_cc\_cr\_coeff\_sign**[ k ][ j ] | u(1) |
| } |  |
| } |  |
| } |  |
| } |  |

#### Luma mapping with chroma scaling data syntax

|  |  |
| --- | --- |
| lmcs\_data( ) { | **Descriptor** |
| **lmcs\_min\_bin\_idx** | ue(v) |
| **lmcs\_delta\_max\_bin\_idx** | ue(v) |
| **lmcs\_delta\_cw\_prec\_minus1** | ue(v) |
| for( i = lmcs\_min\_bin\_idx; i <= LmcsMaxBinIdx; i++ ) { |  |
| **lmcs\_delta\_abs\_cw**[ i ] | u(v) |
| if( lmcs\_delta\_abs\_cw[ i ] > 0 ) |  |
| **lmcs\_delta\_sign\_cw\_flag**[ i ] | u(1) |
| } |  |
| if( aps\_chroma\_present\_flag ) |  |
| **lmcs\_delta\_abs\_crs** | u(3) |
| if( lmcs\_delta\_abs\_crs > 0 ) |  |
| **lmcs\_delta\_sign\_crs\_flag** | u(1) |
| } |  |

#### Scaling list data syntax

|  |  |
| --- | --- |
| scaling\_list\_data( ) { | **Descriptor** |
| for( id = 0; id < 28; id ++ ) { |  |
| matrixSize = id < 2 ? 2 : ( id < 8 ? 4 : 8 ) |  |
| if( aps\_chroma\_present\_flag | | id % 3  = =  2 | | id  = =  27 ) { |  |
| **scaling\_list\_copy\_mode\_flag**[ id ] | u(1) |
| if( !scaling\_list\_copy\_mode\_flag[ id ] ) |  |
| **scaling\_list\_pred\_mode\_flag**[ id ] | u(1) |
| if( ( scaling\_list\_copy\_mode\_flag[ id ] | | scaling\_list\_pred\_mode\_flag[ id ] ) &&  id != 0 && id != 2 && id != 8 ) |  |
| **scaling\_list\_pred\_id\_delta**[ id ] | ue(v) |
| if( !scaling\_list\_copy\_mode\_flag[ id ] ) { |  |
| nextCoef = 0 |  |
| if( id > 13 ) { |  |
| **scaling\_list\_dc\_coef**[ id − 14 ] | se(v) |
| nextCoef += scaling\_list\_dc\_coef[ id − 14 ] |  |
| } |  |
| for( i = 0; i < matrixSize \* matrixSize; i++ ) { |  |
| x = DiagScanOrder[ 3 ][ 3 ][ i ][ 0 ] |  |
| y = DiagScanOrder[ 3 ][ 3 ][ i ][ 1 ] |  |
| if( !( id > 25 && x >= 4 && y >= 4 ) ) { |  |
| **scaling\_list\_delta\_coef**[ id ][ i ] | se(v) |
| nextCoef += scaling\_list\_delta\_coef[ id ][ i ] |  |
| } |  |
| ScalingList[ id ][ i ] = nextCoef |  |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |

### Profile, tier, and level syntax

#### General profile, tier, and level syntax

|  |  |
| --- | --- |
| profile\_tier\_level( profileTierPresentFlag, maxNumSubLayersMinus1 ) { | **Descriptor** |
| if( profileTierPresentFlag ) { |  |
| **general\_profile\_idc** | u(7) |
| **general\_tier\_flag** | u(1) |
| general\_constraint\_info( ) |  |
| } |  |
| **general\_level\_idc** | u(8) |
| if( profileTierPresentFlag ) { |  |
| **num\_sub\_profiles** | u(8) |
| for( i = 0; i < num\_sub\_profiles; i++ ) |  |
| **general\_sub\_profile\_idc**[ i ] | u(32) |
| } |  |
| for( i = 0; i < maxNumSubLayersMinus1; i++ ) |  |
| **sublayer\_level\_present\_flag**[ i ] | u(1) |
| while( !byte\_aligned( ) ) |  |
| **ptl\_alignment\_zero\_bit** | f(1) |
| for( i = 0; i < maxNumSubLayersMinus1; i++ ) |  |
| if( sublayer\_level\_present\_flag[ i ] ) |  |
| **sublayer\_level\_idc**[ i ] | u(8) |
| } |  |

#### General constraint information syntax

|  |  |
| --- | --- |
| general\_constraint\_info( ) { | **Descriptor** |
| **general\_non\_packed\_constraint\_flag** | u(1) |
| **general\_frame\_only\_constraint\_flag** | u(1) |
| **general\_non\_projected\_constraint\_flag** | u(1) |
| **general\_one\_picture\_only\_constraint\_flag** | u(1) |
| **intra\_only\_constraint\_flag** | u(1) |
| **max\_bitdepth\_constraint\_idc** | u(4) |
| **max\_chroma\_format\_constraint\_idc** | u(2) |
| **single\_layer\_constraint\_flag** | u(1) |
| **all\_layers\_independent\_constraint\_flag** | u(1) |
| **no\_ref\_pic\_resampling\_constraint\_flag** | u(1) |
| **no\_res\_change\_in\_clvs\_constraint\_flag** | u(1) |
| **one\_tile\_per\_pic\_constraint\_flag** | u(1) |
| **pic\_header\_in\_slice\_header\_constraint\_flag** | u(1) |
| **one\_slice\_per\_pic\_constraint\_flag** | u(1) |
| **one\_subpic\_per\_pic\_constraint\_flag** | u(1) |
| **no\_qtbtt\_dual\_tree\_intra\_constraint\_flag** | u(1) |
| **no\_partition\_constraints\_override\_constraint\_flag** | u(1) |
| **no\_sao\_constraint\_flag** | u(1) |
| **no\_alf\_constraint\_flag** | u(1) |
| **no\_ccalf\_constraint\_flag** | u(1) |
| **no\_joint\_cbcr\_constraint\_flag** | u(1) |
| **no\_mrl\_constraint\_flag** | u(1) |
| **no\_isp\_constraint\_flag** | u(1) |
| **no\_mip\_constraint\_flag** | u(1) |
| **no\_ref\_wraparound\_constraint\_flag** | u(1) |
| **no\_temporal\_mvp\_constraint\_flag** | u(1) |
| **no\_sbtmvp\_constraint\_flag** | u(1) |
| **no\_amvr\_constraint\_flag** | u(1) |
| **no\_bdof\_constraint\_flag** | u(1) |
| **no\_dmvr\_constraint\_flag** | u(1) |
| **no\_cclm\_constraint\_flag** | u(1) |
| **no\_mts\_constraint\_flag** | u(1) |
| **no\_sbt\_constraint\_flag** | u(1) |
| **no\_lfnst\_constraint\_flag** | u(1) |
| **no\_affine\_motion\_constraint\_flag** | u(1) |
| **no\_mmvd\_constraint\_flag** | u(1) |
| **no\_smvd\_constraint\_flag** | u(1) |
| **no\_prof\_constraint\_flag** | u(1) |
| **no\_bcw\_constraint\_flag** | u(1) |
| **no\_ibc\_constraint\_flag** | u(1) |
| **no\_ciip\_constraint\_flag** | u(1) |
| **no\_gpm\_constraint\_flag** | u(1) |
| **no\_ladf\_constraint\_flag** | u(1) |
| **no\_transform\_skip\_constraint\_flag** | u(1) |
| **no\_bdpcm\_constraint\_flag** | u(1) |
| **no\_palette\_constraint\_flag** | u(1) |
| **no\_act\_constraint\_flag** | u(1) |
| **no\_lmcs\_constraint\_flag** | u(1) |
| **no\_cu\_qp\_delta\_constraint\_flag** | u(1) |
| **no\_chroma\_qp\_offset\_constraint\_flag** | u(1) |
| **no\_dep\_quant\_constraint\_flag** | u(1) |
| **no\_sign\_data\_hiding\_constraint\_flag** | u(1) |
| **no\_tsrc\_constraint\_flag** | u(1) |
| **no\_mixed\_nalu\_types\_in\_pic\_constraint\_flag** | u(1) |
| **no\_trail\_constraint\_flag** | u(1) |
| **no\_stsa\_constraint\_flag** | u(1) |
| **no\_rasl\_constraint\_flag** | u(1) |
| **no\_radl\_constraint\_flag** | u(1) |
| **no\_idr\_constraint\_flag** | u(1) |
| **no\_cra\_constraint\_flag** | u(1) |
| **no\_gdr\_constraint\_flag** | u(1) |
| **no\_aps\_constraint\_flag** | u(1) |
| while( !byte\_aligned( ) ) |  |
| **gci\_alignment\_zero\_bit** | f(1) |
| **gci\_num\_reserved\_bytes** | u(8) |
| for( i = 0; i < gci\_num\_reserved\_bytes; i++ ) |  |
| **gci\_reserved\_byte**[ i ] | u(8) |
| } |  |

### DPB parameters syntax

|  |  |
| --- | --- |
| dpb\_parameters( maxSubLayersMinus1, subLayerInfoFlag ) { | Descriptor |
| for( i = ( subLayerInfoFlag ? 0 : maxSubLayersMinus1 );  i <= maxSubLayersMinus1; i++ ) { |  |
| **max\_dec\_pic\_buffering\_minus1**[ i ] | ue(v) |
| **max\_num\_reorder\_pics**[ i ] | ue(v) |
| **max\_latency\_increase\_plus1**[ i ] | ue(v) |
| } |  |
| } |  |

### HRD parameters syntax

#### General HRD parameters syntax

|  |  |
| --- | --- |
| general\_hrd\_parameters( ) { | Descriptor |
| **num\_units\_in\_tick** | u(32) |
| **time\_scale** | u(32) |
| **general\_nal\_hrd\_params\_present\_flag** | u(1) |
| **general\_vcl\_hrd\_params\_present\_flag** | u(1) |
| **general\_same\_pic\_timing\_in\_all\_ols\_flag** | u(1) |
| **general\_decoding\_unit\_hrd\_params\_present\_flag** | u(1) |
| if( general\_decoding\_unit\_hrd\_params\_present\_flag ) |  |
| **tick\_divisor\_minus2** | u(8) |
| **bit\_rate\_scale** | u(4) |
| **cpb\_size\_scale** | u(4) |
| if( general\_decoding\_unit\_hrd\_params\_present\_flag ) |  |
| **cpb\_size\_du\_scale** | u(4) |
| **hrd\_cpb\_cnt\_minus1** | ue(v) |
| } |  |

#### OLS HRD parameters syntax

|  |  |
| --- | --- |
| ols\_hrd\_parameters( firstSubLayer, maxSubLayers ) { | Descriptor |
| for( i = firstSubLayer; i <= maxSubLayers; i++ ) { |  |
| **fixed\_pic\_rate\_general\_flag**[ i ] | u(1) |
| if( !fixed\_pic\_rate\_general\_flag[ i ] ) |  |
| **fixed\_pic\_rate\_within\_cvs\_flag**[ i ] | u(1) |
| if( fixed\_pic\_rate\_within\_cvs\_flag[ i ] ) |  |
| **elemental\_duration\_in\_tc\_minus1**[ i ] | ue(v) |
| else if( hrd\_cpb\_cnt\_minus1 = = 0 ) |  |
| **low\_delay\_hrd\_flag**[ i ] | u(1) |
| if( general\_nal\_hrd\_params\_present\_flag ) |  |
| sublayer\_hrd\_parameters( i ) |  |
| if( general\_vcl\_hrd\_params\_present\_flag ) |  |
| sublayer\_hrd\_parameters( i ) |  |
| } |  |
| } |  |

#### Sublayer HRD parameters syntax

|  |  |
| --- | --- |
| sublayer\_hrd\_parameters( subLayerId ) { | Descriptor |
| for( j = 0; j <= hrd\_cpb\_cnt\_minus1; j++ ) { |  |
| **bit\_rate\_value\_minus1**[ subLayerId ][ j ] | ue(v) |
| **cpb\_size\_value\_minus1**[ subLayerId ][ j ] | ue(v) |
| if( general\_decoding\_unit\_hrd\_params\_present\_flag ) { |  |
| **cpb\_size\_du\_value\_minus1**[ subLayerId ][ j ] | ue(v) |
| **bit\_rate\_du\_value\_minus1**[ subLayerId ][ j ] | ue(v) |
| } |  |
| **cbr\_flag**[ subLayerId ][ j ] | u(1) |
| } |  |
| } |  |

### Supplemental enhancement information message syntax

|  |  |
| --- | --- |
| sei\_message( ) { | **Descriptor** |
| payloadType = 0 |  |
| do { |  |
| **payload\_type\_byte** | u(8) |
| payloadType += payload\_type\_byte |  |
| } while( payload\_type\_byte = = 0xFF ) |  |
| payloadSize = 0 |  |
| do { |  |
| **payload\_size\_byte** | u(8) |
| payloadSize += payload\_size\_byte |  |
| } while( payload\_size\_byte = = 0xFF ) |  |
| sei\_payload( payloadType, payloadSize ) |  |
| } |  |

### Slice header syntax

#### General slice header syntax

|  |  |
| --- | --- |
| slice\_header( ) { | Descriptor |
| **sh\_picture\_header\_in\_slice\_header\_flag** | u(1) |
| if( sh\_picture\_header\_in\_slice\_header\_flag ) |  |
| picture\_header\_structure( ) |  |
| if( sps\_subpic\_info\_present\_flag ) |  |
| **sh\_subpic\_id** | u(v) |
| if( ( pps\_rect\_slice\_flag && NumSlicesInSubpic[ CurrSubpicIdx ] > 1 ) | |  ( !pps\_rect\_slice\_flag && NumTilesInPic > 1 ) ) |  |
| **sh\_slice\_address** | u(v) |
| for( i = 0; i < NumExtraShBits; i++ ) |  |
| **sh\_extra\_bit**[ i ] | u(1) |
| if( !pps\_rect\_slice\_flag && NumTilesInPic − sh\_slice\_address > 1 ) |  |
| **sh\_num\_tiles\_in\_slice\_minus1** | ue(v) |
| if( ph\_inter\_slice\_allowed\_flag ) |  |
| **sh\_slice\_type** | ue(v) |
| if( sps\_alf\_enabled\_flag && !pps\_alf\_info\_in\_ph\_flag ) { |  |
| **sh\_alf\_enabled\_flag** | u(1) |
| if( sh\_alf\_enabled\_flag ) { |  |
| **sh\_num\_alf\_aps\_ids\_luma** | u(3) |
| for( i = 0; i < sh\_num\_alf\_aps\_ids\_luma; i++ ) |  |
| **sh\_alf\_aps\_id\_luma**[ i ] | u(3) |
| if( ChromaArrayType != 0 ) { |  |
| **sh\_alf\_cb\_flag** | u(1) |
| **sh\_alf\_cr\_flag** | u(1) |
| } |  |
| if( sh\_alf\_cb\_flag | | sh\_alf\_cr\_flag ) |  |
| **sh\_alf\_aps\_id\_chroma** | u(3) |
| if( sps\_ccalf\_enabled\_flag ) { |  |
| **sh\_cc\_alf\_cb\_enabled\_flag** | u(1) |
| if( sh\_cc\_alf\_cb\_enabled\_flag ) |  |
| **sh\_cc\_alf\_cb\_aps\_id** | u(3) |
| **sh\_cc\_alf\_cr\_enabled\_flag** | u(1) |
| if( sh\_cc\_alf\_cr\_enabled\_flag ) |  |
| **sh\_cc\_alf\_cr\_aps\_id** | u(3) |
| } |  |
| } |  |
| } |  |
| if( ph\_lmcs\_enabled\_flag && !sh\_picture\_header\_in\_slice\_header\_flag ) |  |
| **sh\_lmcs\_enabled\_flag** | u(1) |
| if( ph\_explicit\_scaling\_list\_enabled\_flag && !sh\_picture\_header\_in\_slice\_header\_flag ) |  |
| **sh\_explicit\_scaling\_list\_used\_flag** | u(1) |
| if( sps\_separate\_colour\_plane\_flag = = 1 ) |  |
| **sh\_colour\_plane\_id** | u(2) |
| if( !pps\_rpl\_info\_in\_ph\_flag && ( ( nal\_unit\_type != IDR\_W\_RADL &&  nal\_unit\_type != IDR\_N\_LP ) | | sps\_idr\_rpl\_present\_flag ) ) |  |
| ref\_pic\_lists( ) |  |
| if( ( ( ( sh\_slice\_type != I && num\_ref\_entries[ 0 ][ RplsIdx[ 0 ] ] > 1 ) | |  ( sh\_slice\_type = = B && num\_ref\_entries[ 1 ][ RplsIdx[ 1 ] ] > 1 ) ) ) { |  |
| **sh\_num\_ref\_idx\_active\_override\_flag** | u(1) |
| if( sh\_num\_ref\_idx\_active\_override\_flag ) |  |
| for( i = 0; i < ( sh\_slice\_type = = B ? 2: 1 ); i++ ) |  |
| if( num\_ref\_entries[ i ][ RplsIdx[ i ] ] > 1 ) |  |
| **sh\_num\_ref\_idx\_active\_minus1**[ i ] | ue(v) |
| } |  |
| if( sh\_slice\_type != I ) { |  |
| if( pps\_cabac\_init\_present\_flag ) |  |
| **sh\_cabac\_init\_flag** | u(1) |
| if( ph\_temporal\_mvp\_enabled\_flag && !pps\_rpl\_info\_in\_ph\_flag ) { |  |
| if( sh\_slice\_type = = B ) |  |
| **sh\_collocated\_from\_l0\_flag** | u(1) |
| if( ( sh\_collocated\_from\_l0\_flag && NumRefIdxActive[ 0 ] > 1 ) | |  ( ! sh\_collocated\_from\_l0\_flag && NumRefIdxActive[ 1 ] > 1 ) ) |  |
| **sh\_collocated\_ref\_idx** | ue(v) |
| } |  |
| if( !pps\_wp\_info\_in\_ph\_flag &&  ( ( pps\_weighted\_pred\_flag && sh\_slice\_type = = P ) | |  ( pps\_weighted\_bipred\_flag && sh\_slice\_type = = B ) ) ) |  |
| pred\_weight\_table( ) |  |
| } |  |
| if( !pps\_qp\_delta\_info\_in\_ph\_flag ) |  |
| **sh\_qp\_delta** | se(v) |
| if( pps\_slice\_chroma\_qp\_offsets\_present\_flag ) { |  |
| **sh\_cb\_qp\_offset** | se(v) |
| **sh\_cr\_qp\_offset** | se(v) |
| if( sps\_joint\_cbcr\_enabled\_flag ) |  |
| **sh\_joint\_cbcr\_qp\_offset** | se(v) |
| } |  |
| if( pps\_cu\_chroma\_qp\_offset\_list\_enabled\_flag ) |  |
| **sh\_cu\_chroma\_qp\_offset\_enabled\_flag** | u(1) |
| if( sps\_sao\_enabled\_flag && !pps\_sao\_info\_in\_ph\_flag ) { |  |
| **sh\_sao\_luma\_flag** | u(1) |
| if( ChromaArrayType != 0 ) |  |
| **sh\_sao\_chroma\_flag** | u(1) |
| } |  |
| if( pps\_deblocking\_filter\_override\_enabled\_flag && !pps\_dbf\_info\_in\_ph\_flag ) |  |
| **sh\_deblocking\_filter\_override\_flag** | u(1) |
| if( sh\_deblocking\_filter\_override\_flag ) { |  |
| if( !pps\_deblocking\_filter\_disabled\_flag ) |  |
| **sh\_deblocking\_filter\_disabled\_flag** | u(1) |
| if( !sh\_deblocking\_filter\_disabled\_flag ) { |  |
| **sh\_luma\_beta\_offset\_div2** | se(v) |
| **sh\_luma\_tc\_offset\_div2** | se(v) |
| if( pps\_chroma\_tool\_offsets\_present\_flag ) { |  |
| **sh\_cb\_beta\_offset\_div2** | se(v) |
| **sh\_cb\_tc\_offset\_div2** | se(v) |
| **sh\_cr\_beta\_offset\_div2** | se(v) |
| **sh\_cr\_tc\_offset\_div2** | se(v) |
| } |  |
| } |  |
| } |  |
| if( sps\_dep\_quant\_enabled\_flag ) |  |
| **sh\_dep\_quant\_enabled\_flag** | u(1) |
| if( sps\_sign\_data\_hiding\_enabled\_flag && !sh\_dep\_quant\_enabled\_flag ) |  |
| **sh\_sign\_data\_hiding\_enabled\_flag** | u(1) |
| if( sps\_transform\_skip\_enabled\_flag && !sh\_dep\_quant\_enabled\_flag &&  !sh\_sign\_data\_hiding\_enabled\_flag ) |  |
| **sh\_ts\_residual\_coding\_disabled\_flag** | u(1) |
| if( pps\_slice\_header\_extension\_present\_flag ) { |  |
| **sh\_slice\_header\_extension\_length** | ue(v) |
| for( i = 0; i < sh\_slice\_header\_extension\_length; i++) |  |
| **sh\_slice\_header\_extension\_data\_byte**[ i ] | u(8) |
| } |  |
| if( NumEntryPoints > 0 ) { |  |
| **sh\_entry\_offset\_len\_minus1** | ue(v) |
| for( i = 0; i < NumEntryPoints; i++ ) |  |
| **sh\_entry\_point\_offset\_minus1**[ i ] | u(v) |
| } |  |
| byte\_alignment( ) |  |
| } |  |

#### Weighted prediction parameters syntax

[Ed. (YK): Move this syntax strucutre to a better place now that it can be included in either PH or SH.]

|  |  |
| --- | --- |
| pred\_weight\_table( ) { | Descriptor |
| **luma\_log2\_weight\_denom** | ue(v) |
| if( ChromaArrayType != 0 ) |  |
| **delta\_chroma\_log2\_weight\_denom** | se(v) |
| if( pps\_wp\_info\_in\_ph\_flag ) |  |
| **num\_l0\_weights** | ue(v) |
| for( i = 0; i < NumWeightsL0; i++ ) |  |
| **luma\_weight\_l0\_flag**[ i ] | u(1) |
| if( ChromaArrayType != 0 ) |  |
| for( i = 0; i < NumWeightsL0; i++ ) |  |
| **chroma\_weight\_l0\_flag**[ i ] | u(1) |
| for( i = 0; i < NumWeightsL0; i++ ) { |  |
| if( luma\_weight\_l0\_flag[ i ] ) { |  |
| **delta\_luma\_weight\_l0**[ i ] | se(v) |
| **luma\_offset\_l0**[ i ] | se(v) |
| } |  |
| if( chroma\_weight\_l0\_flag[ i ] ) |  |
| for( j = 0; j < 2; j++ ) { |  |
| **delta\_chroma\_weight\_l0**[ i ][ j ] | se(v) |
| **delta\_chroma\_offset\_l0**[ i ][ j ] | se(v) |
| } |  |
| } |  |
| if( pps\_weighted\_bipred\_flag && pps\_wp\_info\_in\_ph\_flag &&  num\_ref\_entries[ 1 ][ RplsIdx[ 1 ] ] > 0 ) |  |
| **num\_l1\_weights** | ue(v) |
| for( i = 0; i < NumWeightsL1; i++ ) |  |
| **luma\_weight\_l1\_flag**[ i ] | u(1) |
| if( ChromaArrayType != 0 ) |  |
| for( i = 0; i < NumWeightsL1; i++ ) |  |
| **chroma\_weight\_l1\_flag**[ i ] | u(1) |
| for( i = 0; i < NumWeightsL1; i++ ) { |  |
| if( luma\_weight\_l1\_flag[ i ] ) { |  |
| **delta\_luma\_weight\_l1**[ i ] | se(v) |
| **luma\_offset\_l1**[ i ] | se(v) |
| } |  |
| if( chroma\_weight\_l1\_flag[ i ] ) |  |
| for( j = 0; j < 2; j++ ) { |  |
| **delta\_chroma\_weight\_l1**[ i ][ j ] | se(v) |
| **delta\_chroma\_offset\_l1**[ i ][ j ] | se(v) |
| } |  |
| } |  |
| } |  |

### Reference picture lists syntax

|  |  |
| --- | --- |
| ref\_pic\_lists( ) { | Descriptor |
| for( i = 0; i < 2; i++ ) { |  |
| if( sps\_num\_ref\_pic\_lists[ i ] > 0 &&  ( i = = 0 | | ( i = = 1 && pps\_rpl1\_idx\_present\_flag ) ) ) |  |
| **rpl\_sps\_flag**[ i ] | u(1) |
| if( rpl\_sps\_flag[ i ] ) { |  |
| if( sps\_num\_ref\_pic\_lists[ i ] > 1 &&  ( i = = 0 | | ( i = = 1 && pps\_rpl1\_idx\_present\_flag ) ) ) |  |
| **rpl\_idx**[ i ] | u(v) |
| } else |  |
| ref\_pic\_list\_struct( i, sps\_num\_ref\_pic\_lists[ i ] ) |  |
| for( j = 0; j < NumLtrpEntries[ i ][ RplsIdx[ i ] ]; j++ ) { |  |
| if( ltrp\_in\_header\_flag[ i ][ RplsIdx[ i ] ] ) |  |
| **poc\_lsb\_lt**[ i ][ j ] | u(v) |
| **delta\_poc\_msb\_cycle\_present\_flag**[ i ][ j ] | u(1) |
| if( delta\_poc\_msb\_cycle\_present\_flag[ i ][ j ] ) |  |
| **delta\_poc\_msb\_cycle\_lt**[ i ][ j ] | ue(v) |
| } |  |
| } |  |
| } |  |

### Reference picture list structure syntax

|  |  |
| --- | --- |
| ref\_pic\_list\_struct( listIdx, rplsIdx ) { | Descriptor |
| **num\_ref\_entries**[ listIdx ][ rplsIdx ] | ue(v) |
| if( sps\_long\_term\_ref\_pics\_flag && rplsIdx < sps\_num\_ref\_pic\_lists[ listIdx ] ) |  |
| **ltrp\_in\_header\_flag**[ listIdx ][ rplsIdx ] | u(1) |
| for( i = 0, j = 0; i < num\_ref\_entries[ listIdx ][ rplsIdx ]; i++) { |  |
| if( sps\_inter\_layer\_ref\_pics\_present\_flag ) |  |
| **inter\_layer\_ref\_pic\_flag**[ listIdx ][ rplsIdx ][ i ] | u(1) |
| if( !inter\_layer\_ref\_pic\_flag[ listIdx ][ rplsIdx ][ i ] ) { |  |
| if( sps\_long\_term\_ref\_pics\_flag ) |  |
| **st\_ref\_pic\_flag**[ listIdx ][ rplsIdx ][ i ] | u(1) |
| if( st\_ref\_pic\_flag[ listIdx ][ rplsIdx ][ i ] ) { |  |
| **abs\_delta\_poc\_st**[ listIdx ][ rplsIdx ][ i ] | ue(v) |
| if( AbsDeltaPocSt[ listIdx ][ rplsIdx ][ i ] > 0 ) |  |
| **strp\_entry\_sign\_flag**[ listIdx ][ rplsIdx ][ i ] | u(1) |
| } else if( !ltrp\_in\_header\_flag[ listIdx ][ rplsIdx ] ) |  |
| **rpls\_poc\_lsb\_lt**[ listIdx ][ rplsIdx ][ j++ ] | u(v) |
| } else |  |
| **ilrp\_idx**[ listIdx ][ rplsIdx ][ i ] | ue(v) |
| } |  |
| } |  |

### Slice data syntax

#### General slice data syntax

|  |  |
| --- | --- |
| slice\_data( ) { | Descriptor |
| FirstCtbRowInSlice = 1 |  |
| for( i = 0; i < NumCtusInCurrSlice; i++ ) { |  |
| CtbAddrInRs = CtbAddrInCurrSlice[ i ] |  |
| CtbAddrX = ( CtbAddrInRs % PicWidthInCtbsY ) |  |
| CtbAddrY = ( CtbAddrInRs / PicWidthInCtbsY ) |  |
| if( CtbAddrX = = CtbToTileColBd[ CtbAddrX ] ) { |  |
| NumHmvpCand = 0 |  |
| NumHmvpIbcCand = 0 |  |
| ResetIbcBuf = 1 |  |
| } |  |
| coding\_tree\_unit( ) |  |
| if( i = = NumCtusInCurrSlice − 1 ) |  |
| **end\_of\_slice\_one\_bit** /\* equal to 1 \*/ | ae(v) |
| else if( CtbAddrX = = CtbToTileColBd[ CtbAddrX + 1 ] − 1 ) { |  |
| if( CtbAddrY = = CtbToTileRowBd[ CtbAddrY + 1 ] − 1 ) { |  |
| **end\_of\_tile\_one\_bit** /\* equal to 1 \*/ | ae(v) |
| byte\_alignment( ) |  |
| } else if( sps\_entropy\_coding\_sync\_enabled\_flag ) { |  |
| **end\_of\_subset\_one\_bit** /\* equal to 1 \*/ | ae(v) |
| byte\_alignment( ) |  |
| } |  |
| FirstCtbRowInSlice = 0 |  |
| } |  |
| } |  |
| } |  |

#### Coding tree unit syntax

|  |  |
| --- | --- |
| coding\_tree\_unit( ) { | Descriptor |
| xCtb = CtbAddrX  <<  CtbLog2SizeY |  |
| yCtb = CtbAddrY  <<  CtbLog2SizeY |  |
| if( sh\_sao\_luma\_flag | | sh\_sao\_chroma\_flag ) |  |
| sao( CtbAddrX, CtbAddrY ) |  |
| if( sh\_alf\_enabled\_flag ){ |  |
| **alf\_ctb\_flag**[ 0 ][ CtbAddrX ][ CtbAddrY ] | ae(v) |
| if( alf\_ctb\_flag[ 0 ][ CtbAddrX ][ CtbAddrY ] ) { |  |
| if( sh\_num\_alf\_aps\_ids\_luma > 0 ) |  |
| **alf\_use\_aps\_flag** | ae(v) |
| if( alf\_use\_aps\_flag ) { |  |
| if( sh\_num\_alf\_aps\_ids\_luma > 1 ) |  |
| **alf\_luma\_prev\_filter\_idx** | ae(v) |
| } else |  |
| **alf\_luma\_fixed\_filter\_idx** | ae(v) |
| } |  |
| if( sh\_alf\_cb\_flag ) { |  |
| **alf\_ctb\_flag**[ 1 ][ CtbAddrX ][ CtbAddrY ] | ae(v) |
| if( alf\_ctb\_flag[ 1 ][ CtbAddrX ][ CtbAddrY ]  && alf\_chroma\_num\_alt\_filters\_minus1 > 0 ) |  |
| **alf\_ctb\_filter\_alt\_idx**[ 0 ][ CtbAddrX ][ CtbAddrY ] | ae(v) |
| } |  |
| if( sh\_alf\_cr\_flag ) { |  |
| **alf\_ctb\_flag**[ 2 ][ CtbAddrX ][ CtbAddrY ] | ae(v) |
| if( alf\_ctb\_flag[ 2 ][ CtbAddrX ][ CtbAddrY ]  && alf\_chroma\_num\_alt\_filters\_minus1 > 0 ) |  |
| **alf\_ctb\_filter\_alt\_idx**[ 1 ][ CtbAddrX ][ CtbAddrY ] | ae(v) |
| } |  |
| } |  |
| if( sh\_cc\_alf\_cb\_enabled\_flag ) |  |
| **alf\_ctb\_cc\_cb\_idc**[ CtbAddrX ][ CtbAddrY ] | ae(v) |
| if( sh\_cc\_alf\_cr\_enabled\_flag ) |  |
| **alf\_ctb\_cc\_cr\_idc**[ CtbAddrX ][ CtbAddrY ] | ae(v) |
| if( sh\_slice\_type = = I  &&  sps\_qtbtt\_dual\_tree\_intra\_flag ) |  |
| dual\_tree\_implicit\_qt\_split( xCtb, yCtb, CtbSizeY, 0 ) |  |
| else |  |
| coding\_tree( xCtb, yCtb, CtbSizeY, CtbSizeY, 1, 1, 0, 0, 0, 0, 0,  SINGLE\_TREE, MODE\_TYPE\_ALL ) |  |
| } |  |

|  |  |
| --- | --- |
| dual\_tree\_implicit\_qt\_split( x0, y0, cbSize, cqtDepth ) { | Descriptor |
| cbSubdiv = 2 \* cqtDepth |  |
| if( cbSize > 64 ) { |  |
| if( pps\_cu\_qp\_delta\_enabled\_flag && cbSubdiv <= CuQpDeltaSubdiv ) { |  |
| IsCuQpDeltaCoded = 0 |  |
| CuQpDeltaVal = 0 |  |
| CuQgTopLeftX = x0 |  |
| CuQgTopLeftY = y0 |  |
| } |  |
| if( sh\_cu\_chroma\_qp\_offset\_enabled\_flag &&  cbSubdiv <= CuChromaQpOffsetSubdiv ) { |  |
| IsCuChromaQpOffsetCoded = 0 |  |
| 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 ) |  |
| } |  |
| } |  |

#### Sample adaptive offset syntax

|  |  |
| --- | --- |
| sao( rx, ry ) { | Descriptor |
| if( rx > 0 ) { |  |
| leftCtbAvailable = rx != CtbToTileColBd[ rx ] |  |
| if( leftCtbAvailable ) |  |
| **sao\_merge\_left\_flag** | ae(v) |
| } |  |
| if( ry > 0 && !sao\_merge\_left\_flag ) { |  |
| upCtbAvailable = ry != CtbToTileRowBd[ ry ] && !FirstCtbRowInSlice |  |
| if( upCtbAvailable ) |  |
| **sao\_merge\_up\_flag** | ae(v) |
| } |  |
| if( !sao\_merge\_up\_flag && !sao\_merge\_left\_flag ) |  |
| for( cIdx = 0; cIdx < ( ChromaArrayType != 0 ? 3 : 1 ); cIdx++ ) |  |
| if( ( sh\_sao\_luma\_flag && cIdx = = 0 ) | |  ( sh\_sao\_chroma\_flag && cIdx > 0 ) ) { |  |
| if( cIdx = = 0 ) |  |
| **sao\_type\_idx\_luma** | ae(v) |
| else if( cIdx = = 1 ) |  |
| **sao\_type\_idx\_chroma** | ae(v) |
| if( SaoTypeIdx[ cIdx ][ rx ][ ry ] != 0 ) { |  |
| for( i = 0; i < 4; i++ ) |  |
| **sao\_offset\_abs**[ cIdx ][ rx ][ ry ][ i ] | ae(v) |
| if( SaoTypeIdx[ cIdx ][ rx ][ ry ] = = 1 ) { |  |
| for( i = 0; i < 4; i++ ) |  |
| if( sao\_offset\_abs[ cIdx ][ rx ][ ry ][ i ] != 0 ) |  |
| **sao\_offset\_sign\_flag**[ cIdx ][ rx ][ ry ][ i ] | ae(v) |
| **sao\_band\_position**[ cIdx ][ rx ][ ry ] | ae(v) |
| } else { |  |
| if( cIdx = = 0 ) |  |
| **sao\_eo\_class\_luma** | ae(v) |
| if( cIdx = = 1 ) |  |
| **sao\_eo\_class\_chroma** | ae(v) |
| } |  |
| } |  |
| } |  |
| } |  |

#### Coding tree syntax

|  |  |
| --- | --- |
| coding\_tree( x0, y0, cbWidth, cbHeight, qgOnY, qgOnC, cbSubdiv, cqtDepth, mttDepth, depthOffset,    partIdx, treeTypeCurr, modeTypeCurr ) { | Descriptor |
| if( ( allowSplitBtVer | | allowSplitBtHor | | allowSplitTtVer | | allowSplitTtHor | |  allowSplitQT ) && ( x0 + cbWidth <= pps\_pic\_width\_in\_luma\_samples ) &&  ( y0 + cbHeight <= pps\_pic\_height\_in\_luma\_samples ) ) |  |
| **split\_cu\_flag** | ae(v) |
| if( pps\_cu\_qp\_delta\_enabled\_flag && qgOnY && cbSubdiv <= CuQpDeltaSubdiv ) { |  |
| IsCuQpDeltaCoded = 0 |  |
| CuQpDeltaVal = 0 |  |
| CuQgTopLeftX = x0 |  |
| CuQgTopLeftY = y0 |  |
| } |  |
| if( sh\_cu\_chroma\_qp\_offset\_enabled\_flag && qgOnC &&  cbSubdiv <= CuChromaQpOffsetSubdiv ) { |  |
| IsCuChromaQpOffsetCoded = 0 |  |
| CuQpOffsetCb = 0 |  |
| CuQpOffsetCr = 0 |  |
| CuQpOffsetCbCr = 0 |  |
| } |  |
| if( split\_cu\_flag ) { |  |
| if( ( allowSplitBtVer | | allowSplitBtHor | | allowSplitTtVer | | allowSplitTtHor ) &&  allowSplitQT ) |  |
| **split\_qt\_flag** | ae(v) |
| if( !split\_qt\_flag ) { |  |
| if( ( allowSplitBtHor | | allowSplitTtHor ) && ( allowSplitBtVer | | allowSplitTtVer ) ) |  |
| **mtt\_split\_cu\_vertical\_flag** | ae(v) |
| if( ( allowSplitBtVer && allowSplitTtVer && mtt\_split\_cu\_vertical\_flag ) | |  ( allowSplitBtHor && allowSplitTtHor && !mtt\_split\_cu\_vertical\_flag ) ) |  |
| **mtt\_split\_cu\_binary\_flag** | ae(v) |
| } |  |
| if( modeTypeCondition = = 1 ) |  |
| modeType = MODE\_TYPE\_INTRA |  |
| else if( modeTypeCondition = = 2 ) { |  |
| **mode\_constraint\_flag** | ae(v) |
| modeType = mode\_constraint\_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, cbHeightY, qgOnY, qgOnC, cbSubdiv + 1,   cqtDepth, mttDepth + 1, depthOffset, 1, treeType, modeType ) |  |
| } else if( MttSplitMode[ x0 ][ y0 ][ mttDepth ] = = SPLIT\_BT\_HOR ) { |  |
| depthOffset  +=  ( y0 + cbHeight > pps\_pic\_height\_in\_luma\_samples ) ? 1 : 0 |  |
| y1 = y0 + ( cbHeight / 2 ) |  |
| coding\_tree( x0, y0, cbWidth, cbHeight / 2, qgOnY, qgOnC, cbSubdiv + 1,  cqtDepth, mttDepth + 1, depthOffset, 0, treeType, modeType ) |  |
| if( y1 < pps\_pic\_height\_in\_luma\_samples ) |  |
| coding\_tree( x0, y1, cbWidth, cbHeight / 2, qgOnY, qgOnC, cbSubdiv + 1,  cqtDepth, mttDepth + 1, depthOffset, 1, treeType, modeType ) |  |
| } else if( MttSplitMode[ x0 ][ y0 ][ mttDepth ] = = SPLIT\_TT\_VER ) { |  |
| x1 = x0 + ( cbWidth / 4 ) |  |
| x2 = x0 + ( 3 \* cbWidth / 4 ) |  |
| qgNextOnY = qgOnY && ( cbSubdiv + 2 <= CuQpDeltaSubdiv ) |  |
| qgNextOnC = qgOnC && ( cbSubdiv + 2 <= CuChromaQpOffsetSubdiv ) |  |
| coding\_tree( x0, y0, cbWidth / 4, cbHeight, qgNextOnY, qgNextOnC, cbSubdiv + 2,  cqtDepth, mttDepth + 1, depthOffset, 0, treeType, modeType ) |  |
| coding\_tree( x1, y0, cbWidth / 2, cbHeight, qgNextOnY, qgNextOnC, cbSubdiv + 1,  cqtDepth, mttDepth + 1, depthOffset, 1, treeType, modeType ) |  |
| coding\_tree( x2, y0, cbWidth / 4, cbHeight, qgNextOnY, qgNextOnC, cbSubdiv + 2,  cqtDepth, mttDepth + 1, depthOffset, 2, treeType, modeType ) |  |
| } else { /\* SPLIT\_TT\_HOR \*/ |  |
| y1 = y0 + ( cbHeight / 4 ) |  |
| y2 = y0 + ( 3 \* cbHeight / 4 ) |  |
| qgNextOnY = qgOnY && ( cbSubdiv + 2 <= CuQpDeltaSubdiv ) |  |
| qgNextOnC = qgOnC && ( cbSubdiv + 2 <= CuChromaQpOffsetSubdiv ) |  |
| coding\_tree( x0, y0, cbWidth, cbHeight / 4, qgNextOnY, qgNextOnC, cbSubdiv + 2,  cqtDepth, mttDepth + 1, depthOffset, 0, treeType, modeType ) |  |
| coding\_tree( x0, y1, cbWidth, cbHeight / 2, qgNextOnY, qgNextOnC, cbSubdiv + 1,  cqtDepth, mttDepth + 1, depthOffset, 1, treeType, modeType ) |  |
| coding\_tree( x0, y2, cbWidth, cbHeight / 4, qgNextOnY, qgNextOnC, cbSubdiv + 2,  cqtDepth, mttDepth + 1, depthOffset, 2, treeType, modeType ) |  |
| } |  |
| } else { |  |
| x1 = x0 + ( cbWidth / 2 ) |  |
| y1 = y0 + ( cbHeight / 2 ) |  |
| coding\_tree( x0, y0, cbWidth / 2, cbHeight / 2, qgOnY, qgOnC, cbSubdiv + 2,  cqtDepth + 1, 0, 0, 0, treeType, modeType ) |  |
| if( x1 < pps\_pic\_width\_in\_luma\_samples ) |  |
| coding\_tree( x1, y0, cbWidth / 2, cbHeight / 2, qgOnY, qgOnC, cbSubdiv + 2,  cqtDepth + 1, 0, 0, 1, treeType, modeType ) |  |
| if( y1 < pps\_pic\_height\_in\_luma\_samples ) |  |
| coding\_tree( x0, y1, cbWidth / 2, cbHeight / 2, qgOnY, qgOnC, cbSubdiv + 2,  cqtDepth + 1,  0, 0, 2, treeType, modeType ) |  |
| if( y1 < pps\_pic\_height\_in\_luma\_samples && x1 < pps\_pic\_width\_in\_luma\_samples ) |  |
| coding\_tree( x1, y1, cbWidth / 2, cbHeight / 2, qgOnY, qgOnC, cbSubdiv + 2,  cqtDepth + 1,  0, 0, 3, treeType, modeType ) |  |
| } |  |
| if( modeTypeCur = = MODE\_TYPE\_ALL && modeType = = MODE\_TYPE\_INTRA ) |  |
| coding\_tree( x0, y0, cbWidth, cbHeight, 0, qgOnC, cbSubdiv, cqtDepth, mttDepth, 0, 0,   DUAL\_TREE\_CHROMA , modeType ) |  |
| } else |  |
| coding\_unit( x0, y0, cbWidth, cbHeight, cqtDepth, treeTypeCurr , modeTypeCurr ) |  |
| } |  |

#### Coding unit syntax

|  |  |
| --- | --- |
| coding\_unit( x0, y0, cbWidth, cbHeight, cqtDepth, treeType, modeType ) { | Descriptor |
| if( sh\_slice\_type = = I && ( cbWidth > 64 | | cbHeight > 64 ) ) |  |
| modeType = MODE\_TYPE\_INTRA |  |
| chType = treeType = = DUAL\_TREE\_CHROMA ? 1 : 0 |  |
| if( sh\_slice\_type != I | | sps\_ibc\_enabled\_flag ) { |  |
| if( treeType != DUAL\_TREE\_CHROMA &&  ( ( !( cbWidth = = 4 && cbHeight = = 4 ) &&  modeType != MODE\_TYPE\_INTRA ) | |  ( sps\_ibc\_enabled\_flag && cbWidth <= 64 && cbHeight <= 64 ) ) ) |  |
| **cu\_skip\_flag**[ x0 ][ y0 ] | ae(v) |
| if( cu\_skip\_flag[ x0 ][ y0 ] = = 0 && sh\_slice\_type != I &&  !( cbWidth = = 4 && cbHeight = = 4 ) && modeType = = MODE\_TYPE\_ALL ) |  |
| **pred\_mode\_flag** | ae(v) |
| if( ( ( sh\_slice\_type = = I && cu\_skip\_flag[ x0 ][ y0 ] = =0 ) | |  ( sh\_slice\_type != I && ( CuPredMode[ chType ][ x0 ][ y0 ] != MODE\_INTRA | |  ( ( ( cbWidth = = 4 && cbHeight = = 4 ) | | modeType = = MODE\_TYPE\_INTRA )  && cu\_skip\_flag[ x0 ][ y0 ] = = 0 ) ) ) ) &&  cbWidth <= 64 && cbHeight <= 64 && modeType != MODE\_TYPE\_INTER &&  sps\_ibc\_enabled\_flag && treeType != DUAL\_TREE\_CHROMA ) |  |
| **pred\_mode\_ibc\_flag** | ae(v) |
| } |  |
| if( CuPredMode[ chType ][ x0 ][ y0 ] = = MODE\_INTRA && sps\_palette\_enabled\_flag &&  cbWidth <= 64 && cbHeight <= 64 && cu\_skip\_flag[ x0 ][ y0 ] = = 0 &&  modeType != MODE\_TYPE\_INTER && ( ( cbWidth \* cbHeight ) >   ( treeType != DUAL\_TREE\_CHROMA ? 16 : 16 \* SubWidthC \* SubHeightC ) ) &&  ( modeType != MODE\_TYPE\_INTRA | | treeType != DUAL\_TREE\_CHROMA ) ) |  |
| **pred\_mode\_plt\_flag** | ae(v) |
| if( CuPredMode[ chType ][ x0 ][ y0 ] = = MODE\_INTRA && sps\_act\_enabled\_flag &&  treeType = = SINGLE\_TREE ) |  |
| **cu\_act\_enabled\_flag** | ae(v) |
| if( CuPredMode[ chType ][ x0 ][ y0 ] = = MODE\_INTRA | |  CuPredMode[ chType ][ x0 ][ y0 ] = = MODE\_PLT ) { |  |
| if( treeType = = SINGLE\_TREE | | treeType = = DUAL\_TREE\_LUMA ) { |  |
| if( pred\_mode\_plt\_flag ) |  |
| palette\_coding( x0, y0, cbWidth, cbHeight, treeType ) |  |
| else { |  |
| if( sps\_bdpcm\_enabled\_flag &&  cbWidth <= MaxTsSize && cbHeight <= MaxTsSize ) |  |
| **intra\_bdpcm\_luma\_flag** | ae(v) |
| if( intra\_bdpcm\_luma\_flag ) |  |
| **intra\_bdpcm\_luma\_dir\_flag** | ae(v) |
| else { |  |
| if( sps\_mip\_enabled\_flag ) |  |
| **intra\_mip\_flag**[ x0 ][ y0 ] | ae(v) |
| if( intra\_mip\_flag[ x0 ][ y0 ] ) { |  |
| **intra\_mip\_transposed\_flag**[ x0 ][ y0 ] | ae(v) |
| **intra\_mip\_mode**[ x0 ][ y0 ] | ae(v) |
| } else { |  |
| if( sps\_mrl\_enabled\_flag && ( ( y0 % CtbSizeY ) > 0 ) ) |  |
| **intra\_luma\_ref\_idx**[ x0 ][ y0 ] | ae(v) |
| if( sps\_isp\_enabled\_flag && intra\_luma\_ref\_idx[ x0 ][ y0 ] = = 0 &&  ( cbWidth <= MaxTbSizeY && cbHeight <= MaxTbSizeY ) &&  ( cbWidth \* cbHeight > MinTbSizeY \* MinTbSizeY ) &&  !cu\_act\_enabled\_flag ) |  |
| **intra\_subpartitions\_mode\_flag**[ x0 ][ y0 ] | ae(v) |
| if( intra\_subpartitions\_mode\_flag[ x0 ][ y0 ] = = 1 ) |  |
| **intra\_subpartitions\_split\_flag**[ x0 ][ y0 ] | ae(v) |
| if( intra\_luma\_ref\_idx[ x0 ][ y0 ] = = 0 ) |  |
| **intra\_luma\_mpm\_flag**[ x0 ][ y0 ] | ae(v) |
| if( intra\_luma\_mpm\_flag[ x0 ][ y0 ] ) { |  |
| if( intra\_luma\_ref\_idx[ x0 ][ y0 ] = = 0 ) |  |
| **intra\_luma\_not\_planar\_flag**[ x0 ][ y0 ] | ae(v) |
| if( intra\_luma\_not\_planar\_flag[ x0 ][ y0 ] ) |  |
| **intra\_luma\_mpm\_idx**[ x0 ][ y0 ] | ae(v) |
| } else |  |
| **intra\_luma\_mpm\_remainder**[ x0 ][ y0 ] | ae(v) |
| } |  |
| } |  |
| } |  |
| } |  |
| if( ( treeType = = SINGLE\_TREE | | treeType = = DUAL\_TREE\_CHROMA ) &&  ChromaArrayType != 0 ) { |  |
| if( pred\_mode\_plt\_flag && treeType = = DUAL\_TREE\_CHROMA ) |  |
| palette\_coding( x0, y0, cbWidth / SubWidthC, cbHeight / SubHeightC, treeType ) |  |
| else if( !pred\_mode\_plt\_flag ) { |  |
| if( !cu\_act\_enabled\_flag ) { |  |
| if( cbWidth / SubWidthC <= MaxTsSize && cbHeight / SubHeightC <= MaxTsSize  && sps\_bdpcm\_enabled\_flag ) |  |
| **intra\_bdpcm\_chroma\_flag** | ae(v) |
| if( intra\_bdpcm\_chroma\_flag ) |  |
| **intra\_bdpcm\_chroma\_dir\_flag** | ae(v) |
| else { |  |
| if( CclmEnabled ) |  |
| **cclm\_mode\_flag** | ae(v) |
| if( cclm\_mode\_flag ) |  |
| **cclm\_mode\_idx** | ae(v) |
| else |  |
| **intra\_chroma\_pred\_mode** | ae(v) |
| } |  |
| } |  |
| } |  |
| } |  |
| } else if( treeType != DUAL\_TREE\_CHROMA ) { /\* MODE\_INTER or MODE\_IBC \*/ |  |
| if( cu\_skip\_flag[ x0 ][ y0 ] = = 0 ) |  |
| **general\_merge\_flag**[ x0 ][ y0 ] | ae(v) |
| if( general\_merge\_flag[ x0 ][ y0 ] ) |  |
| merge\_data( x0, y0, cbWidth, cbHeight, chType ) |  |
| else if( CuPredMode[ chType ][ x0 ][ y0 ] = = MODE\_IBC ) { |  |
| mvd\_coding( x0, y0, 0, 0 ) |  |
| if( MaxNumIbcMergeCand > 1 ) |  |
| **mvp\_l0\_flag**[ x0 ][ y0 ] | ae(v) |
| if( sps\_amvr\_enabled\_flag &&  ( MvdL0[ x0 ][ y0 ][ 0 ] != 0 | | MvdL0[ x0 ][ y0 ][ 1 ] != 0 ) ) |  |
| **amvr\_precision\_idx**[ x0 ][ y0 ] | ae(v) |
| } else { |  |
| if( sh\_slice\_type = = B ) |  |
| **inter\_pred\_idc**[ x0 ][ y0 ] | ae(v) |
| if( sps\_affine\_enabled\_flag && cbWidth >= 16 && cbHeight >= 16 ) { |  |
| **inter\_affine\_flag**[ x0 ][ y0 ] | ae(v) |
| if( sps\_affine\_type\_flag && inter\_affine\_flag[ x0 ][ y0 ] ) |  |
| **cu\_affine\_type\_flag**[ x0 ][ y0 ] | ae(v) |
| } |  |
| if( sps\_smvd\_enabled\_flag && !ph\_mvd\_l1\_zero\_flag &&  inter\_pred\_idc[ x0 ][ y0 ] = = PRED\_BI &&  !inter\_affine\_flag[ x0 ][ y0 ] && RefIdxSymL0 > −1 && RefIdxSymL1 > −1 ) |  |
| **sym\_mvd\_flag**[ x0 ][ y0 ] | ae(v) |
| if( inter\_pred\_idc[ x0 ][ y0 ] != PRED\_L1 ) { |  |
| if( NumRefIdxActive[ 0 ] > 1 && !sym\_mvd\_flag[ x0 ][ y0 ] ) |  |
| **ref\_idx\_l0**[ x0 ][ y0 ] | ae(v) |
| mvd\_coding( x0, y0, 0, 0 ) |  |
| if( MotionModelIdc[ x0 ][ y0 ] > 0 ) |  |
| mvd\_coding( x0, y0, 0, 1 ) |  |
| if(MotionModelIdc[ x0 ][ y0 ] > 1 ) |  |
| mvd\_coding( x0, y0, 0, 2 ) |  |
| **mvp\_l0\_flag**[ x0 ][ y0 ] | 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 ] ) |  |
| **ref\_idx\_l1**[ x0 ][ y0 ] | 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 ) |  |
| } |  |
| **mvp\_l1\_flag**[ 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 &&  ( MvdL0[ x0 ][ y0 ][ 0 ] != 0 | | MvdL0[ x0 ][ y0 ][ 1 ] != 0 | |  MvdL1[ x0 ][ y0 ][ 0 ] != 0 | | MvdL1[ x0 ][ y0 ][ 1 ] != 0 ) ) | |  ( sps\_affine\_amvr\_enabled\_flag && inter\_affine\_flag[ x0 ][ y0 ] = = 1 &&  ( MvdCpL0[ x0 ][ y0 ][ 0 ][ 0 ] != 0 | | MvdCpL0[ x0 ][ y0 ][ 0 ][ 1 ] != 0 | |  MvdCpL1[ x0 ][ y0 ][ 0 ][ 0 ] != 0 | | MvdCpL1[ x0 ][ y0 ][ 0 ][ 1 ] != 0 | |  MvdCpL0[ x0 ][ y0 ][ 1 ][ 0 ] != 0 | | MvdCpL0[ x0 ][ y0 ][ 1 ][ 1 ] != 0 | |  MvdCpL1[ x0 ][ y0 ][ 1 ][ 0 ] != 0 | | MvdCpL1[ x0 ][ y0 ][ 1 ][ 1 ] != 0 | |  MvdCpL0[ x0 ][ y0 ][ 2 ][ 0 ] != 0 | | MvdCpL0[ x0 ][ y0 ][ 2 ][ 1 ] != 0 | |  MvdCpL1[ x0 ][ y0 ][ 2 ][ 0 ] != 0 | | MvdCpL1[ x0 ][ y0 ][ 2 ][ 1 ] != 0 ) ) { |  |
| **amvr\_flag**[ x0 ][ y0 ] | ae(v) |
| if( amvr\_flag[ x0 ][ y0 ] ) |  |
| **amvr\_precision\_idx**[ x0 ][ y0 ] | ae(v) |
| } |  |
| if( sps\_bcw\_enabled\_flag && inter\_pred\_idc[ x0 ][ y0 ] = = PRED\_BI &&  luma\_weight\_l0\_flag[ ref\_idx\_l0 [ x0 ][ y0 ] ] = = 0 &&  luma\_weight\_l1\_flag[ ref\_idx\_l1 [ x0 ][ y0 ] ] = = 0 &&  chroma\_weight\_l0\_flag[ ref\_idx\_l0 [ x0 ][ y0 ] ] = = 0 &&  chroma\_weight\_l1\_flag[ ref\_idx\_l1 [ x0 ][ y0 ] ] = = 0 &&  cbWidth \* cbHeight >= 256 ) |  |
| **bcw\_idx**[ x0 ][ y0 ] | ae(v) |
| } |  |
| } |  |
| if( CuPredMode[ chType ][ x0 ][ y0 ] != MODE\_INTRA && !pred\_mode\_plt\_flag &&  general\_merge\_flag[ x0 ][ y0 ] = = 0 ) |  |
| **cu\_coded\_flag** | ae(v) |
| if( cu\_coded\_flag ) { |  |
| if( CuPredMode[ chType ][ x0 ][ y0 ] = = MODE\_INTER && sps\_sbt\_enabled\_flag &&   !ciip\_flag[ x0 ][ y0 ] && cbWidth  <=  MaxTbSizeY && cbHeight  <=  MaxTbSizeY ) { |  |
| allowSbtVerH = cbWidth  >=  8 |  |
| allowSbtVerQ = cbWidth  >=  16 |  |
| allowSbtHorH = cbHeight  >=  8 |  |
| allowSbtHorQ = cbHeight  >=  16 |  |
| if( allowSbtVerH | | allowSbtHorH ) |  |
| **cu\_sbt\_flag** | ae(v) |
| if( cu\_sbt\_flag ) { |  |
| if( ( allowSbtVerH | | allowSbtHorH ) && ( allowSbtVerQ | | allowSbtHorQ ) ) |  |
| **cu\_sbt\_quad\_flag** | ae(v) |
| if( ( cu\_sbt\_quad\_flag && allowSbtVerQ && allowSbtHorQ ) | |  ( !cu\_sbt\_quad\_flag && allowSbtVerH && allowSbtHorH ) ) |  |
| **cu\_sbt\_horizontal\_flag** | ae(v) |
| **cu\_sbt\_pos\_flag** | ae(v) |
| } |  |
| } |  |
| if( sps\_act\_enabled\_flag && CuPredMode[ chType ][ x0 ][ y0 ] != MODE\_INTRA &&  treeType = = SINGLE\_TREE ) |  |
| **cu\_act\_enabled\_flag** | ae(v) |
| LfnstDcOnly = 1 |  |
| LfnstZeroOutSigCoeffFlag = 1 |  |
| MtsDcOnly = 1 |  |
| MtsZeroOutSigCoeffFlag = 1 |  |
| transform\_tree( x0, y0, cbWidth, cbHeight, treeType, chType ) |  |
| lfnstWidth = ( treeType = = DUAL\_TREE\_CHROMA ) ? cbWidth / SubWidthC :  ( ( IntraSubPartitionsSplitType = = ISP\_VER\_SPLIT ) ?  cbWidth / NumIntraSubPartitions : cbWidth ) |  |
| lfnstHeight = ( treeType = = DUAL\_TREE\_CHROMA ) ? cbHeight / SubHeightC :  ( ( IntraSubPartitionsSplitType = = ISP\_HOR\_SPLIT) ?  cbHeight / NumIntraSubPartitions : cbHeight ) |  |
| lfnstNotTsFlag = ( treeType = = DUAL\_TREE\_CHROMA | |  !tu\_y\_coded\_flag[ x0 ][ y0 ] | |  transform\_skip\_flag[ x0 ][ y0 ][ 0 ] = = 0 ) &&  ( treeType = = DUAL\_TREE\_LUMA | |  ( ( !tu\_cb\_coded\_flag[ x0 ][ y0 ] | |  transform\_skip\_flag[ x0 ][ y0 ][ 1 ] = = 0 ) &&  ( !tu\_cr\_coded\_flag[ x0 ][ y0 ] | |  transform\_skip\_flag[ x0 ][ y0 ][ 2 ] = = 0 ) ) ) |  |
| if( Min( lfnstWidth, lfnstHeight ) >= 4 && sps\_lfnst\_enabled\_flag = = 1 &&  CuPredMode[ chType ][ x0 ][ y0 ] = = MODE\_INTRA && lfnstNotTsFlag = = 1 &&  ( treeType = = DUAL\_TREE\_CHROMA | | !intra\_mip\_flag[ x0 ][ y0 ] | |  Min( lfnstWidth, lfnstHeight ) >= 16 ) &&  Max( cbWidth, cbHeight ) <= MaxTbSizeY) { |  |
| if( ( IntraSubPartitionsSplitType != ISP\_NO\_SPLIT | | LfnstDcOnly = = 0 ) &&  LfnstZeroOutSigCoeffFlag = = 1 ) |  |
| **lfnst\_idx** | ae(v) |
| } |  |
| if( treeType != DUAL\_TREE\_CHROMA && lfnst\_idx = = 0 &&  transform\_skip\_flag[ x0 ][ y0 ][ 0 ] = = 0 && Max( cbWidth, cbHeight ) <= 32 &&  IntraSubPartitionsSplitType = = ISP\_NO\_SPLIT && cu\_sbt\_flag = = 0 &&  MtsZeroOutSigCoeffFlag = = 1 && MtsDcOnly = = 0 ) { |  |
| if( ( ( CuPredMode[ chType ][ x0 ][ y0 ] = = MODE\_INTER &&  sps\_explicit\_mts\_inter\_enabled\_flag ) | |  ( CuPredMode[ chType ][ x0 ][ y0 ] = = MODE\_INTRA &&  sps\_explicit\_mts\_intra\_enabled\_flag ) ) ) |  |
| **mts\_idx** | ae(v) |
| } |  |
| } |  |
| } |  |

#### Palette coding syntax

|  |  |
| --- | --- |
| palette\_coding( x0, y0, cbWidth, cbHeight, treeType ) { | Descriptor |
| startComp = ( treeType = = DUAL\_TREE\_CHROMA ) ? 1 : 0 |  |
| numComps = ( treeType = = SINGLE\_TREE ) ? ( ChromaArrayType = = 0 ? 1 : 3 ) :  ( treeType = = DUAL\_TREE\_CHROMA ) ? 2 : 1 |  |
| maxNumPaletteEntries = ( treeType = = SINGLE\_TREE ) ? 31 : 15 |  |
| palettePredictionFinished = 0 |  |
| NumPredictedPaletteEntries = 0 |  |
| for( predictorEntryIdx = 0; predictorEntryIdx < PredictorPaletteSize[ startComp ] &&  !palettePredictionFinished &&  NumPredictedPaletteEntries < maxNumPaletteEntries; predictorEntryIdx++ ) { |  |
| **palette\_predictor\_run** | ae(v) |
| if( palette\_predictor\_run != 1 ) { |  |
| if( palette\_predictor\_run > 1 ) |  |
| predictorEntryIdx += palette\_predictor\_run − 1 |  |
| PalettePredictorEntryReuseFlags[ predictorEntryIdx ] = 1 |  |
| NumPredictedPaletteEntries++ |  |
| } else |  |
| palettePredictionFinished = 1 |  |
| } |  |
| if( NumPredictedPaletteEntries < maxNumPaletteEntries ) |  |
| **num\_signalled\_palette\_entries** | ae(v) |
| for( cIdx = startComp; cIdx < ( startComp + numComps ); cIdx++ ) |  |
| for( i = 0; i < num\_signalled\_palette\_entries; i++ ) |  |
| **new\_palette\_entries**[ cIdx ][ i ] | ae(v) |
| if( CurrentPaletteSize[ startComp ] > 0 ) |  |
| **palette\_escape\_val\_present\_flag** | ae(v) |
| if( MaxPaletteIndex > 0 ) { |  |
| adjust = 0 |  |
| **palette\_transpose\_flag** | ae(v) |
| } |  |
| if( treeType != DUAL\_TREE\_CHROMA && palette\_escape\_val\_present\_flag ) |  |
| if( pps\_cu\_qp\_delta\_enabled\_flag && !IsCuQpDeltaCoded ) { |  |
| **cu\_qp\_delta\_abs** | ae(v) |
| if( cu\_qp\_delta\_abs ) |  |
| **cu\_qp\_delta\_sign\_flag** | ae(v) |
| } |  |
| if( treeType != DUAL\_TREE\_LUMA && palette\_escape\_val\_present\_flag ) |  |
| if( sh\_cu\_chroma\_qp\_offset\_enabled\_flag && !IsCuChromaQpOffsetCoded ) { |  |
| **cu\_chroma\_qp\_offset\_flag** | ae(v) |
| if( cu\_chroma\_qp\_offset\_flag && pps\_chroma\_qp\_offset\_list\_len\_minus1 > 0 ) |  |
| **cu\_chroma\_qp\_offset\_idx** | ae(v) |
| } |  |
| PreviousRunPosition = 0 |  |
| PreviousRunType = 0 |  |
| for( subSetId = 0; subSetId <= ( cbWidth \* cbHeight − 1 ) / 16; subSetId++ ) { |  |
| minSubPos = subSetId \* 16 |  |
| if( minSubPos + 16 > cbWidth \* cbHeight) |  |
| maxSubPos = cbWidth \* cbHeight |  |
| else |  |
| maxSubPos = minSubPos + 16 |  |
| RunCopyMap[ x0 ][ y0 ] = 0 |  |
| PaletteScanPos = minSubPos |  |
| log2CbWidth = Log2( cbWidth ) |  |
| log2CbHeight = Log2( cbHeight ) |  |
| while( PaletteScanPos < maxSubPos ) { |  |
| xC = x0 + TraverseScanOrder[ log2CbWidth ][ log2CbHeight ][ PaletteScanPos ][ 0 ] |  |
| yC = y0 + TraverseScanOrder[ log2CbWidth ][ log2CbHeight ][ PaletteScanPos ][ 1 ] |  |
| if( PaletteScanPos > 0 ) { |  |
| xcPrev = x0 +   TraverseScanOrder[ log2CbWidth ][ log2CbHeight ][ PaletteScanPos − 1 ][ 0 ] |  |
| ycPrev = y0 +   TraverseScanOrder[ log2CbWidth ][ log2CbHeight ][ PaletteScanPos − 1 ][ 1 ] |  |
| if( MaxPaletteIndex > 0 && PaletteScanPos > 0 ) { |  |
| **run\_copy\_flag** | ae(v) |
| RunCopyMap[ xC ][ yC ] = run\_copy\_flag |  |
| } |  |
| CopyAboveIndicesFlag[ xC ][ yC ] = 0 |  |
| if( MaxPaletteIndex > 0 && !RunCopyMap[ xC ][ yC ] ) { |  |
| if( ( ( !palette\_transpose\_flag && yC > 0 ) | | ( palette\_transpose\_flag && xC > 0 ) )   && CopyAboveIndicesFlag[ xcPrev ][ ycPrev ] = = 0 ) { |  |
| **copy\_above\_palette\_indices\_flag** | ae(v) |
| CopyAboveIndicesFlag[ xC ][ yC ] = copy\_above\_palette\_indices\_flag |  |
| } |  |
| PreviousRunType = CopyAboveIndicesFlag[ xC ][ yC ] |  |
| PreviousRunPosition = PaletteScanPos |  |
| } else |  |
| CopyAboveIndicesFlag[ xC ][ yC ] = CopyAboveIndicesFlag[ xcPrev ][ ycPrev ] |  |
| PaletteScanPos ++ |  |
| } |  |
| PaletteScanPos = minSubPos |  |
| while( PaletteScanPos < maxSubPos ) { |  |
| xC = x0 + TraverseScanOrder[ log2CbWidth ][ log2CbHeight ][ PaletteScanPos ][ 0 ] |  |
| yC = y0 + TraverseScanOrder[ log2CbWidth ][ log2CbHeight ][ PaletteScanPos ][ 1 ] |  |
| if( PaletteScanPos > 0 ) { |  |
| xcPrev =x0 +   TraverseScanOrder[ log2CbWidth ][ log2CbHeight ][ PaletteScanPos − 1 ][ 0 ] |  |
| ycPrev = y0 +   TraverseScanOrder[ log2CbWidth ][ log2CbHeight ][ PaletteScanPos − 1 ][ 1 ] |  |
| } |  |
| if( MaxPaletteIndex > 0 && !RunCopyMap[ xC ][ yC ] &&  CopyAboveIndicesFlag[ xC ][ yC ] = = 0 ) { |  |
| if( MaxPaletteIndex − adjust > 0 ) |  |
| **palette\_idx\_idc** | ae(v) |
| adjust = 1 |  |
| } |  |
| if( !RunCopyMap[ xC ][ yC ] && CopyAboveIndicesFlag[ xC ][ yC ] = = 0 ) |  |
| CurrPaletteIndex = palette\_idx\_idc |  |
| if( CopyAboveIndicesFlag[ xC ][ yC ] = = 0 ) |  |
| PaletteIndexMap[ xC ][ yC ] = CurrPaletteIndex |  |
| else if( !palette\_transpose\_flag ) |  |
| PaletteIndexMap[ xC ][ yC ] = PaletteIndexMap[ xC ][ yC − 1 ] |  |
| else |  |
| PaletteIndexMap[ xC ][ yC ] = PaletteIndexMap[ xC − 1 ][ yC ] |  |
| PaletteScanPos ++ |  |
| } |  |
| if( palette\_escape\_val\_present\_flag ) |  |
| for( cIdx = startComp; cIdx < ( startComp + numComps ); cIdx++ ) |  |
| for( sPos = minSubPos; sPos < maxSubPos; sPos++ ) { |  |
| xC = x0 + TraverseScanOrder[ log2CbWidth][ log2CbHeight ][ sPos ][ 0 ] |  |
| yC = y0 + TraverseScanOrder[ log2CbWidth][ log2CbHeight ][ sPos ][ 1 ] |  |
| if( !( treeType = = SINGLE\_TREE && cIdx != 0 &&   xC % SubWidthC != 0 && yC % SubHeightC != 0 ) ) { |  |
| if( PaletteIndexMap[ cIdx ][ xC ][ yC ] = = MaxPaletteIndex ) { |  |
| **palette\_escape\_val** | ae(v) |
| PaletteEscapeVal[ cIdx ][ xC ][ yC ] = palette\_escape\_val |  |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |

#### Merge data syntax

|  |  |
| --- | --- |
| merge\_data( x0, y0, cbWidth, cbHeight, chType ) { | Descriptor |
| if( CuPredMode[ chType ][ x0 ][ y0 ] = = MODE\_IBC ) { |  |
| if( MaxNumIbcMergeCand > 1 ) |  |
| **merge\_idx**[ x0 ][ y0 ] | ae(v) |
| } else { |  |
| if( MaxNumSubblockMergeCand > 0 && cbWidth >= 8 && cbHeight >= 8 ) |  |
| **merge\_subblock\_flag**[ x0 ][ y0 ] | ae(v) |
| if( merge\_subblock\_flag[ x0 ][ y0 ] = = 1 ) { |  |
| if( MaxNumSubblockMergeCand > 1 ) |  |
| **merge\_subblock\_idx**[ x0 ][ y0 ] | ae(v) |
| } else { |  |
| if( cbWidth < 128 && cbHeight < 128 &&  ( ( sps\_ciip\_enabled\_flag && cu\_skip\_flag[ x0 ][ y0 ] = = 0 &&  ( cbWidth \* cbHeight ) >= 64 ) | |  ( sps\_gpm\_enabled\_flag &&  sh\_slice\_type = = B && cbWidth >= 8 && cbHeight >= 8 &&  cbWidth < ( 8 \* cbHeight ) && cbHeight < ( 8 \* cbWidth ) ) ) ) |  |
| **regular\_merge\_flag**[ x0 ][ y0 ] | ae(v) |
| if( regular\_merge\_flag[ x0 ][ y0 ] = = 1 ) { |  |
| if( sps\_mmvd\_enabled\_flag ) |  |
| **mmvd\_merge\_flag**[ x0 ][ y0 ] | ae(v) |
| if( mmvd\_merge\_flag[ x0 ][ y0 ] = = 1 ) { |  |
| if( MaxNumMergeCand > 1 ) |  |
| **mmvd\_cand\_flag**[ x0 ][ y0 ] | ae(v) |
| **mmvd\_distance\_idx**[ x0 ][ y0 ] | ae(v) |
| **mmvd\_direction\_idx**[ x0 ][ y0 ] | ae(v) |
| } else if( MaxNumMergeCand > 1 ) |  |
| **merge\_idx**[ x0 ][ y0 ] | ae(v) |
| } else { |  |
| if( sps\_ciip\_enabled\_flag && sps\_gpm\_enabled\_flag &&  sh\_slice\_type = = B &&  cu\_skip\_flag[ x0 ][ y0 ] = = 0 && cbWidth >= 8 && cbHeight >= 8 &&  cbWidth < ( 8 \* cbHeight ) && cbHeight < ( 8 \* cbWidth ) &&  cbWidth < 128 && cbHeight < 128 ) |  |
| **ciip\_flag**[ x0 ][ y0 ] | ae(v) |
| if( ciip\_flag[ x0 ][ y0 ] && MaxNumMergeCand > 1 ) |  |
| **merge\_idx**[ x0 ][ y0 ] | ae(v) |
| if( !ciip\_flag[ x0 ][ y0 ] ) { |  |
| **merge\_gpm\_partition\_idx**[ x0 ][ y0 ] | ae(v) |
| **merge\_gpm\_idx0**[ x0 ][ y0 ] | ae(v) |
| if( MaxNumGpmMergeCand > 2 ) |  |
| **merge\_gpm\_idx1**[ x0 ][ y0 ] | ae(v) |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |

#### Motion vector difference syntax

|  |  |
| --- | --- |
| mvd\_coding( x0, y0, refList ,cpIdx ) { | **Descriptor** |
| **abs\_mvd\_greater0\_flag**[ 0 ] | ae(v) |
| **abs\_mvd\_greater0\_flag**[ 1 ] | ae(v) |
| if( abs\_mvd\_greater0\_flag[ 0 ] ) |  |
| **abs\_mvd\_greater1\_flag**[ 0 ] | ae(v) |
| if( abs\_mvd\_greater0\_flag[ 1 ] ) |  |
| **abs\_mvd\_greater1\_flag**[ 1 ] | ae(v) |
| if( abs\_mvd\_greater0\_flag[ 0 ] ) { |  |
| if( abs\_mvd\_greater1\_flag[ 0 ] ) |  |
| **abs\_mvd\_minus2**[ 0 ] | ae(v) |
| **mvd\_sign\_flag**[ 0 ] | ae(v) |
| } |  |
| if( abs\_mvd\_greater0\_flag[ 1 ] ) { |  |
| if( abs\_mvd\_greater1\_flag[ 1 ] ) |  |
| **abs\_mvd\_minus2**[ 1 ] | ae(v) |
| **mvd\_sign\_flag**[ 1 ] | ae(v) |
| } |  |
| } |  |

#### Transform tree syntax

|  |  |
| --- | --- |
| transform\_tree( x0, y0, tbWidth, tbHeight , treeType, chType ) { | Descriptor |
| InferTuCbfLuma = 1 |  |
| if( IntraSubPartitionsSplitType = = ISP\_NO\_SPLIT && !cu\_sbt\_flag ) { |  |
| if( tbWidth > MaxTbSizeY | | tbHeight > MaxTbSizeY ) { |  |
| verSplitFirst = ( tbWidth > MaxTbSizeY && tbWidth > tbHeight ) ? 1 : 0 |  |
| trafoWidth = verSplitFirst ? ( tbWidth / 2 ) : tbWidth |  |
| trafoHeight = !verSplitFirst ? ( tbHeight / 2 ) : tbHeight |  |
| transform\_tree( x0, y0, trafoWidth,  trafoHeight, treeType, chType ) |  |
| if( verSplitFirst ) |  |
| transform\_tree( x0 + trafoWidth, y0, trafoWidth, trafoHeight, treeType, chType ) |  |
| else |  |
| transform\_tree( x0, y0 + trafoHeight, trafoWidth, trafoHeight, treeType, chType ) |  |
| 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 ) |  |
| } |  |
| } |  |

#### Transform unit syntax

|  |  |
| --- | --- |
| transform\_unit( x0, y0, tbWidth, tbHeight, treeType, subTuIndex, chType ) { | Descriptor |
| if( IntraSubPartitionsSplitType != ISP\_NO\_SPLIT &&  treeType = = SINGLE\_TREE && subTuIndex = = NumIntraSubPartitions − 1 ) { |  |
| xC = CbPosX[ chType ][ x0 ][ y0 ] |  |
| yC = CbPosY[ chType ][ x0 ][ y0 ] |  |
| wC = CbWidth[ chType ][ x0 ][ y0 ] / SubWidthC |  |
| hC = CbHeight[ chType ][ x0 ][ y0 ] / SubHeightC |  |
| } else { |  |
| xC = x0 |  |
| yC = y0 |  |
| wC = tbWidth / SubWidthC |  |
| hC = tbHeight / SubHeightC |  |
| } |  |
| chromaAvailable = treeType != DUAL\_TREE\_LUMA && ChromaArrayType != 0 &&  ( IntraSubPartitionsSplitType = = ISP\_NO\_SPLIT  | |  ( IntraSubPartitionsSplitType != ISP\_NO\_SPLIT &&  subTuIndex = = NumIntraSubPartitions − 1 ) ) |  |
| if( ( treeType = = SINGLE\_TREE | | treeType = = DUAL\_TREE\_CHROMA ) &&  ChromaArrayType != 0 && ( IntraSubPartitionsSplitType = = ISP\_NO\_SPLIT &&  ( ( subTuIndex  = = 0  &&  cu\_sbt\_pos\_flag )  | |  ( subTuIndex  = = 1  &&  !cu\_sbt\_pos\_flag ) ) ) )  | |  ( IntraSubPartitionsSplitType != ISP\_NO\_SPLIT &&  ( subTuIndex = = NumIntraSubPartitions − 1 ) ) ) { |  |
| **tu\_cb\_coded\_flag**[ xC ][ yC ] | ae(v) |
| **tu\_cr\_coded\_flag**[ xC ][ yC ] | ae(v) |
| } |  |
| if( treeType = = SINGLE\_TREE | | treeType = = DUAL\_TREE\_LUMA ) { |  |
| if( ( IntraSubPartitionsSplitType = = ISP\_NO\_SPLIT && !( cu\_sbt\_flag &&  ( ( subTuIndex  = = 0  &&  cu\_sbt\_pos\_flag )  | |  ( subTuIndex  = = 1  &&  !cu\_sbt\_pos\_flag ) ) ) &&  ( CuPredMode[ chType ][ x0 ][ y0 ] = = MODE\_INTRA | |  ( chromaAvailable && ( tu\_cb\_coded\_flag[ xC ][ yC ] | |  tu\_cr\_coded\_flag[ xC ][ yC ] ) ) | |  CbWidth[ chType ][ x0 ][ y0 ] > MaxTbSizeY | |  CbHeight[ chType ][ x0 ][ y0 ] > MaxTbSizeY ) )  | |  ( IntraSubPartitionsSplitType != ISP\_NO\_SPLIT &&  ( subTuIndex < NumIntraSubPartitions − 1 | | !InferTuCbfLuma ) ) ) |  |
| **tu\_y\_coded\_flag**[ x0 ][ y0 ] | ae(v) |
| if(IntraSubPartitionsSplitType != ISP\_NO\_SPLIT ) |  |
| InferTuCbfLuma = InferTuCbfLuma && !tu\_y\_coded\_flag[ x0 ][ y0 ] |  |
| } |  |
| if( ( CbWidth[ chType ][ x0 ][ y0 ]  >  64 | | CbHeight[ chType ][ x0 ][ y0 ]  >  64 | |  tu\_y\_coded\_flag[ x0 ][ y0 ] | | ( chromaAvailable && ( tu\_cb\_coded\_flag[ xC ][ yC ] | |  tu\_cr\_coded\_flag[ xC ][ yC ] ) ) && treeType != DUAL\_TREE\_CHROMA &&  pps\_cu\_qp\_delta\_enabled\_flag && !IsCuQpDeltaCoded ) { |  |
| **cu\_qp\_delta\_abs** | ae(v) |
| if( cu\_qp\_delta\_abs ) |  |
| **cu\_qp\_delta\_sign\_flag** | ae(v) |
| } |  |
| if( ( CbWidth[ chType ][ x0 ][ y0 ]  >  64 | | CbHeight[ chType ][ x0 ][ y0 ]  >  64 | |  ( chromaAvailable && ( tu\_cb\_coded\_flag[ xC ][ yC ] | |  tu\_cr\_coded\_flag[ xC ][ yC ] ) ) ) &&  treeType != DUAL\_TREE\_LUMA && sh\_cu\_chroma\_qp\_offset\_enabled\_flag &&  !IsCuChromaQpOffsetCoded ) { |  |
| **cu\_chroma\_qp\_offset\_flag** | ae(v) |
| if( cu\_chroma\_qp\_offset\_flag && pps\_chroma\_qp\_offset\_list\_len\_minus1 > 0 ) |  |
| **cu\_chroma\_qp\_offset\_idx** | ae(v) |
| } |  |
| if( sps\_joint\_cbcr\_enabled\_flag && ( ( CuPredMode[ chType ][ x0 ][ y0 ] = = MODE\_INTRA  && ( tu\_cb\_coded\_flag[ xC ][ yC ] | | tu\_cr\_coded\_flag[ xC ][ yC ] ) ) | |  ( tu\_cb\_coded\_flag[ xC ][ yC ] && tu\_cr\_coded\_flag[ xC ][ yC ] ) ) &&  chromaAvailable ) |  |
| **tu\_joint\_cbcr\_residual\_flag**[ xC ][ yC ] | ae(v) |
| if( tu\_y\_coded\_flag[ x0 ][ y0 ] && treeType  !=  DUAL\_TREE\_CHROMA ) { |  |
| if( sps\_transform\_skip\_enabled\_flag && !BdpcmFlag[ x0 ][ y0 ][ 0 ] && tbWidth <= MaxTsSize && tbHeight <= MaxTsSize && ( IntraSubPartitionsSplitType  = =  ISP\_NO\_SPLIT ) && !cu\_sbt\_flag ) |  |
| **transform\_skip\_flag**[ x0 ][ y0 ][ 0 ] | ae(v) |
| if( !transform\_skip\_flag[ x0 ][ y0 ][ 0 ] | | sh\_ts\_residual\_coding\_disabled\_flag ) |  |
| residual\_coding( x0, y0, Log2( tbWidth ), Log2( tbHeight ), 0 ) |  |
| else |  |
| residual\_ts\_coding( x0, y0, Log2( tbWidth ), Log2( tbHeight ), 0 ) |  |
| } |  |
| if( tu\_cb\_coded\_flag[ xC ][ yC ] && treeType  !=  DUAL\_TREE\_LUMA ) { |  |
| if( sps\_transform\_skip\_enabled\_flag && !BdpcmFlag[ x0 ][ y0 ][ 1 ] && wC <= MaxTsSize && hC <= MaxTsSize && !cu\_sbt\_flag ) |  |
| **transform\_skip\_flag**[ xC ][ yC ][ 1 ] | ae(v) |
| if( !transform\_skip\_flag[ xC ][ yC ][ 1 ] | | sh\_ts\_residual\_coding\_disabled\_flag ) |  |
| residual\_coding( xC, yC, Log2( wC ), Log2( hC ), 1 ) |  |
| else |  |
| residual\_ts\_coding( xC, yC, Log2( wC ), Log2( hC ), 1 ) |  |
| } |  |
| if( tu\_cr\_coded\_flag[ xC ][ yC ] && treeType  !=  DUAL\_TREE\_LUMA &&  !( tu\_cb\_coded\_flag[ xC ][ yC ] && tu\_joint\_cbcr\_residual\_flag[ xC ][ yC ] ) ) { |  |
| if( sps\_transform\_skip\_enabled\_flag && !BdpcmFlag[ x0 ][ y0 ][ 2 ] && wC <= MaxTsSize && hC <= MaxTsSize && !cu\_sbt\_flag ) |  |
| **transform\_skip\_flag**[ xC ][ yC ][ 2 ] | ae(v) |
| if( !transform\_skip\_flag[ xC ][ yC ][ 2 ] | | sh\_ts\_residual\_coding\_disabled\_flag ) |  |
| residual\_coding( xC, yC, Log2( wC ), Log2( hC ), 2 ) |  |
| else |  |
| residual\_ts\_coding( xC, yC, Log2( wC ), Log2( hC ), 2 ) |  |
| } |  |
| } |  |

#### Residual coding syntax

|  |  |  |
| --- | --- | --- |
| residual\_coding( x0, y0, log2TbWidth, log2TbHeight, cIdx ) { | Descriptor | |
| if( sps\_mts\_enabled\_flag && cu\_sbt\_flag && cIdx  = =  0 && log2TbWidth  = =  5 && log2TbHeight < 6 ) |  | |
| log2ZoTbWidth = 4 |  | |
| else |  | |
| log2ZoTbWidth = Min( log2TbWidth, 5 ) |  | |
| if( sps\_mts\_enabled\_flag && cu\_sbt\_flag && cIdx  = =  0 &&   log2TbWidth < 6 && log2TbHeight  = =  5 ) |  | |
| log2ZoTbHeight = 4 |  | |
| else |  | |
| log2ZoTbHeight = Min( log2TbHeight, 5 ) |  | |
| if( log2TbWidth > 0 ) |  | |
| **last\_sig\_coeff\_x\_prefix** | ae(v) |
| if( log2TbHeight > 0 ) |  |
| **last\_sig\_coeff\_y\_prefix** | ae(v) |
| if( last\_sig\_coeff\_x\_prefix > 3 ) |  |
| **last\_sig\_coeff\_x\_suffix** | ae(v) |
| if( last\_sig\_coeff\_y\_prefix > 3 ) |  |
| **last\_sig\_coeff\_y\_suffix** | ae(v) |
| log2TbWidth = log2ZoTbWidth |  |
| log2TbHeight = log2ZoTbHeight |  |
| remBinsPass1 = ( ( 1 << ( log2TbWidth + log2TbHeight ) ) \* 7 ) >> 2 |  |
| 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 ) |  |
| lastScanPos = numSbCoeff |  |
| lastSubBlock = ( 1  <<  ( log2TbWidth + log2TbHeight − ( log2SbW + log2SbH ) ) ) − 1 |  |
| do { |  |
| if( lastScanPos = = 0 ) { |  |
| lastScanPos = numSbCoeff |  |
| lastSubBlock− − |  |
| } |  |
| lastScanPos− − |  |
| xS = DiagScanOrder[ log2TbWidth − log2SbW ][ log2TbHeight − log2SbH ]  [ lastSubBlock ][ 0 ] |  |
| yS = DiagScanOrder[ log2TbWidth − log2SbW ][ log2TbHeight − 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 && log2TbWidth >= 2 && log2TbHeight >= 2 &&  !transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] && lastScanPos > 0 ) |  |
| LfnstDcOnly = 0 |  |
| if( ( lastSubBlock > 0 && log2TbWidth >= 2 && log2TbHeight >= 2 ) | |  ( lastScanPos > 7 && ( log2TbWidth = = 2 | | log2TbWidth = = 3 ) &&  log2TbWidth = = log2TbHeight ) ) |  |
| LfnstZeroOutSigCoeffFlag = 0 |  |
| if( ( lastSubBlock > 0 | | lastScanPos > 0 ) && cIdx = = 0 ) |  |
| MtsDcOnly = 0 |  |
| QState = 0 |  |
| for( i = lastSubBlock; i >= 0; i− − ) { |  |
| startQStateSb = QState |  |
| xS = DiagScanOrder[ log2TbWidth − log2SbW ][ log2TbHeight − log2SbH ]  [ i ][ 0 ] |  |
| yS = DiagScanOrder[ log2TbWidth − log2SbW ][ log2TbHeight − log2SbH ]  [ i ][ 1 ] |  |
| inferSbDcSigCoeffFlag = 0 |  |
| if( i < lastSubBlock && i > 0 ) { |  |
| **sb\_coded\_flag**[ xS ][ yS ] | ae(v) |
| inferSbDcSigCoeffFlag = 1 |  |
| } |  |
| if( sb\_coded\_flag[ xS ][ yS ] && ( xS > 3 | | yS > 3 ) && cIdx = = 0 ) |  |
| MtsZeroOutSigCoeffFlag = 0 |  |
| firstSigScanPosSb = numSbCoeff |  |
| lastSigScanPosSb = −1 |  |
| firstPosMode0 = ( i = = lastSubBlock ? lastScanPos : numSbCoeff − 1 ) |  |
| firstPosMode1 = firstPosMode0 |  |
| for( n = firstPosMode0; n >= 0 && remBinsPass1 >= 4; n− − ) { |  |
| xC = ( xS << log2SbW ) + DiagScanOrder[ log2SbW ][ log2SbH ][ n ][ 0 ] |  |
| yC = ( yS << log2SbH ) + DiagScanOrder[ log2SbW ][ log2SbH ][ n ][ 1 ] |  |
| if( sb\_coded\_flag[ xS ][ yS ] && ( n > 0 | | !inferSbDcSigCoeffFlag ) &&  ( xC != LastSignificantCoeffX | | yC != Last SignificantCoeffY ) ) { |  |
| **sig\_coeff\_flag**[ xC ][ yC ] | ae(v) |
| remBinsPass1− − |  |
| if( sig\_coeff\_flag[ xC ][ yC ] ) |  |
| inferSbDcSigCoeffFlag = 0 |  |
| } |  |
| if( sig\_coeff\_flag[ xC ][ yC ] ) { |  | |
| **abs\_level\_gtx\_flag**[ n ][ 0 ] | ae(v) |
| remBinsPass1− − |  |
| if( abs\_level\_gtx\_flag[ n ][ 0 ] ) { |  |
| **par\_level\_flag**[ n ] | ae(v) |
| remBinsPass1− − |  |
| **abs\_level\_gtx\_flag**[ n ][ 1 ] | ae(v) |
| remBinsPass1− − |  |
| } |  |
| if( lastSigScanPosSb = = −1 ) |  |
| lastSigScanPosSb = n |  |
| firstSigScanPosSb = n |  |
| } |  |
| AbsLevelPass1[ xC ][ yC ] = sig\_coeff\_flag[ xC ][ yC ] + par\_level\_flag[ n ] +  abs\_level\_gtx\_flag[ n ][ 0 ] + 2 \* abs\_level\_gtx\_flag[ n ][ 1 ] |  |
| if( sh\_dep\_quant\_enabled\_flag ) |  |
| QState = QStateTransTable[ QState ][ AbsLevelPass1[ xC ][ yC ] & 1 ] |  |
| firstPosMode1 = n − 1 |  |
| } |  |
| for( n = firstPosMode0; n > firstPosMode1; n− − ) { |  |
| xC = ( xS << log2SbW ) + DiagScanOrder[ log2SbW ][ log2SbH ][ n ][ 0 ] |  |
| yC = ( yS << log2SbH ) + DiagScanOrder[ log2SbW ][ log2SbH ][ n ][ 1 ] |  |
| if( abs\_level\_gtx\_flag[ n ][ 1 ] ) |  |
| **abs\_remainder**[ n ] | ae(v) |
| AbsLevel[ xC ][ yC ] = AbsLevelPass1[ xC ][ yC ] +2 \* abs\_remainder[ n ] |  |
| } |  |
| for( n = firstPosMode1; n >= 0; n− − ) { |  |
| xC = ( xS << log2SbW ) + DiagScanOrder[ log2SbW ][ log2SbH ][ n ][ 0 ] |  |
| yC = ( yS << log2SbH ) + DiagScanOrder[ log2SbW ][ log2SbH ][ n ][ 1 ] |  |
| if( sb\_coded\_flag[ xS ][ yS ] ) |  |
| **dec\_abs\_level**[ n ] | ae(v) |
| if( AbsLevel[ xC ][ yC ] > 0 ) { |  |
| if( lastSigScanPosSb = = −1 ) |  |
| lastSigScanPosSb = n |  |
| firstSigScanPosSb = n |  |
| } |  |
| if( sh\_dep\_quant\_enabled\_flag ) |  |
| QState = QStateTransTable[ QState ][ AbsLevel[ xC ][ yC ] & 1 ] |  |
| } |  |
| if( sh\_dep\_quant\_enabled\_flag | | !sh\_sign\_data\_hiding\_enabled\_flag ) |  |
| signHidden = 0 |  |
| else |  |
| signHidden = ( 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 ) &&  ( !signHidden | | ( n != firstSigScanPosSb ) ) ) |  |
| **coeff\_sign\_flag**[ n ] | ae(v) |
| } |  |
| if( sh\_dep\_quant\_enabled\_flag ) { |  |
| QState = startQStateSb |  |
| for( n = numSbCoeff − 1; n >= 0; n− − ) { |  |
| xC = ( xS << log2SbW ) + DiagScanOrder[ log2SbW ][ log2SbH ][ n ][ 0 ] |  |
| yC = ( yS << log2SbH ) + DiagScanOrder[ log2SbW ][ log2SbH ][ n ][ 1 ] |  |
| if( AbsLevel[ xC ][ yC ] > 0 ) |  |
| TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] =  ( 2 \* AbsLevel[ xC ][ yC ] − ( QState > 1 ? 1 : 0 ) ) \*  ( 1 − 2 \* coeff\_sign\_flag[ n ] ) |  |
| QState = QStateTransTable[ QState ][ AbsLevel[ xC ][ yC ] & 1 ] |  |
| } else { |  |
| sumAbsLevel = 0 |  |
| for( n = numSbCoeff − 1; n >= 0; n− − ) { |  |
| xC = ( xS << log2SbW ) + DiagScanOrder[ log2SbW ][ log2SbH ][ n ][ 0 ] |  |
| yC = ( yS << log2SbH ) + DiagScanOrder[ log2SbW ][ log2SbH ][ n ][ 1 ] |  |
| if( AbsLevel[ xC ][ yC ] > 0 ) { |  |
| TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] =  AbsLevel[ xC ][ yC ] \* ( 1 − 2 \* coeff\_sign\_flag[ n ] ) |  |
| if( signHidden ) { |  |
| sumAbsLevel += AbsLevel[ xC ][ yC ] |  |
| if( ( n = = firstSigScanPosSb ) && ( sumAbsLevel % 2 ) = = 1 ) ) |  |
| TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] =  −TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] |  |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  | |

|  |  |
| --- | --- |
| residual\_ts\_coding( x0, y0, log2TbWidth, log2TbHeight, cIdx ) { | Descriptor |
| log2SbW = ( Min( log2TbWidth, log2TbHeight ) < 2 ? 1 : 2 ) |  |
| log2SbH = log2SbW |  |
| if( log2TbWidth + log2TbHeight > 3 ) |  |
| if( log2TbWidth < 2 ) { |  |
| log2SbW = log2TbWidth |  |
| log2SbH = 4 − log2SbW |  |
| } else if( log2TbHeight < 2 ) { |  |
| log2SbH = log2TbHeight |  |
| log2SbW = 4 − log2SbH |  |
| } |  |
| numSbCoeff = 1 << ( log2SbW + log2SbH ) |  |
| lastSubBlock = ( 1  <<  ( log2TbWidth + log2TbHeight − ( log2SbW + log2SbH ) ) ) − 1 |  |
| inferSbCbf = 1 |  |
| RemCcbs = ( ( 1 << ( log2TbWidth + log2TbHeight ) ) \* 7 ) >> 2 |  |
| for( i =0; i <= lastSubBlock; i++ ) { |  |
| xS = DiagScanOrder[ log2TbWidth − log2SbW ][ log2TbHeight − log2SbH ][ i ][ 0 ] |  |
| yS = DiagScanOrder[ log2TbWidth − log2SbW ][ log2TbHeight − log2SbH ][ i ][ 1 ] |  |
| if( i != lastSubBlock | | !inferSbCbf ) |  |
| **sb\_coded\_flag**[ xS ][ yS ] | ae(v) |
| if( sb\_coded\_flag[ xS ][ yS ] && i < lastSubBlock ) |  |
| inferSbCbf = 0 |  |
| /\* First scan pass \*/ |  |
| inferSbSigCoeffFlag = 1 |  |
| lastScanPosPass1 = −1 |  |
| for( n = 0; n <= numSbCoeff − 1 && RemCcbs >= 4; n++ ) { |  |
| xC = ( xS << log2SbW ) + DiagScanOrder[ log2SbW ][ log2SbH ][ n ][ 0 ] |  |
| yC = ( yS << log2SbH ) + DiagScanOrder[ log2SbW ][ log2SbH ][ n ][ 1 ] |  |
| if( sb\_coded\_flag[ xS ][ yS ] &&  ( n != numSbCoeff − 1 | | !inferSbSigCoeffFlag ) ) { |  |
| **sig\_coeff\_flag**[ xC ][ yC ] | ae(v) |
| RemCcbs− − |  |
| if( sig\_coeff\_flag[ xC ][ yC ] ) |  |
| inferSbSigCoeffFlag = 0 |  |
| } |  |
| CoeffSignLevel[ xC ][ yC ] = 0 |  |
| if( sig\_coeff\_flag[ xC ][ yC ] { |  |
| **coeff\_sign\_flag**[ n ] | ae(v) |
| RemCcbs− − |  |
| CoeffSignLevel[ xC ][ yC ] = ( coeff\_sign\_flag[ n ] > 0 ? −1 : 1 ) |  |
| **abs\_level\_gtx\_flag**[ n ][ 0 ] | ae(v) |
| RemCcbs− − |  |
| if( abs\_level\_gtx\_flag[ n ][ 0 ] ) { |  |
| **par\_level\_flag**[ n ] | ae(v) |
| RemCcbs− − |  |
| } |  |
| } |  |
| AbsLevelPass1[ xC ][ yC ] =  sig\_coeff\_flag[ xC ][ yC ] + par\_level\_flag[ n ] + abs\_level\_gtx\_flag[ n ][ 0 ] |  |
| lastScanPosPass1 = n |  |
| } |  |
| /\* Greater than X scan pass (numGtXFlags=5) \*/ |  |
| lastScanPosPass2 = −1 |  |
| for( n = 0; n <= numSbCoeff − 1 && RemCcbs >= 4; n++ ) { |  |
| xC = ( xS << log2SbW ) + DiagScanOrder[ log2SbW ][ log2SbH ][ n ][ 0 ] |  |
| yC = ( yS << log2SbH ) + DiagScanOrder[ log2SbW ][ log2SbH ][ n ][ 1 ] |  |
| AbsLevelPass2[ xC ][ yC ] = AbsLevelPass1[ xC ][ yC ] |  |
| for( j = 1; j < 5; j++ ) { |  |
| if( abs\_level\_gtx\_flag[ n ][ j − 1 ] ) { |  |
| **abs\_level\_gtx\_flag**[ n ][ j ] | ae(v) |
| RemCcbs− − |  |
| } |  |
| AbsLevelPass2[ xC ][ yC ]  +=  2 \* abs\_level\_gtx\_flag[ n ][ j ] |  |
| } |  |
| lastScanPosPass2 = n |  |
| } |  |
| /\* remainder scan pass \*/ |  |
| for( n = 0; n <= numSbCoeff − 1; n++ ) { |  |
| xC = ( xS << log2SbW ) + DiagScanOrder[ log2SbW ][ log2SbH ][ n ][ 0 ] |  |
| yC = ( yS << log2SbH ) + DiagScanOrder[ log2SbW ][ log2SbH ][ n ][ 1 ] |  |
| if( ( n <= lastScanPosPass2 && AbsLevelPass2[ xC ][ yC ] >= 10 ) | |  ( n > lastScanPosPass2 && n <= lastScanPosPass1 &&  AbsLevelPass1[ xC ][ yC ] >= 2 ) | |  ( n > lastScanPosPass1 && sb\_coded\_flag[ xS ][ yS ] ) ) |  |
| **abs\_remainder**[ n ] | ae(v) |
| if( n <= lastScanPosPass2 ) |  |
| AbsLevel[ xC ][ yC ] = AbsLevelPass2[ xC ][ yC ] + 2 \* abs\_remainder[ n ] |  |
| else if(n <= lastScanPosPass1 ) |  |
| AbsLevel[ xC ][ yC ] = AbsLevelPass1[ xC ][ yC ] + 2 \* abs\_remainder[ n ] |  |
| else { /\* bypass \*/ |  |
| AbsLevel[ xC ][ yC ] = abs\_remainder[ n ] |  |
| if( abs\_remainder[ n ] ) |  |
| **coeff\_sign\_flag**[ n ] | ae(v) |
| } |  |
| if( BdpcmFlag[ x0 ][ y0 ][ cIdx ] = = 0 && n <= lastScanPosPass1 ) { |  |
| absLeftCoeff = xC > 0 ? AbsLevel[ xC − 1 ][ yC ] ) : 0 |  |
| absAboveCoeff = yC > 0 ? AbsLevel[ xC ][ yC − 1 ] ) : 0 |  |
| predCoeff = Max( absLeftCoeff, absAboveCoeff ) |  |
| if( AbsLevel[ xC ][ yC ] = = 1 && predCoeff > 0 ) |  |
| AbsLevel[ xC ][ yC ] = predCoeff |  |
| else if( AbsLevel[ xC ][ yC ] > 0 && AbsLevel[ xC ][ yC ] <= predCoeff ) |  |
| AbsLevel[ xC ][ yC ]− − |  |
| } |  |
| } |  |
| TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] = ( 1 − 2 \* coeff\_sign\_flag[ n ] ) \*   AbsLevel[ xC ][ yC ] |  |
| } |  |
| } |  |

## Semantics

### General

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

### NAL unit semantics

#### 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 may be specified outside of this Specification.

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

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

The RBSP contains a string of data bits (SODB) as follows:

– If the SODB is empty (i.e., zero bits in length), the RBSP is also empty.

– Otherwise, the RBSP contains the SODB as follows:

1) The first byte of the RBSP contains the first (most significant, left-most) eight bits of the SODB; the next byte of the RBSP contains the next eight bits of the SODB, etc., until fewer than eight bits of the SODB remain.

2) The rbsp\_trailing\_bits( ) syntax structure is present after the SODB as follows:

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

ii) The next bit consists of a single bit equal to 1 (i.e., rbsp\_stop\_one\_bit).

iii) When the rbsp\_stop\_one\_bit is not the last bit of a byte-aligned byte, one or more zero-valued bits (i.e., instances of rbsp\_alignment\_zero\_bit) are present to result in byte alignment.

3) One or more cabac\_zero\_word 16-bit syntax elements equal to 0x0000 may be present in some RBSPs after the rbsp\_trailing\_bits( ) at the end of the RBSP.

Syntax structures having these RBSP properties are denoted in the syntax tables using an "\_rbsp" suffix. These structures 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 can extract the SODB from the RBSP by concatenating the bits of the bytes of the RBSP and discarding the rbsp\_stop\_one\_bit, which is the last (least significant, right-most) bit equal to 1, and discarding any following (less significant, farther to the right) bits that follow it, which are equal to 0. The data necessary for the decoding process is contained in the SODB part of the RBSP.

**emulation\_prevention\_three\_byte** is a byte equal to 0x03. When an emulation\_prevention\_three\_byte is present in the NAL unit, it shall be discarded by the decoding process.

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

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

– 0x000000

– 0x000001

– 0x000002

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

– 0x00000300

– 0x00000301

– 0x00000302

– 0x00000303

#### 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 may be specified in the future by ITU‑T | ISO/IEC. Decoders 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.

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, EOS\_NUT, or FD\_NUT, nuh\_layer\_id shall be equal to the nuh\_layer\_id of associated VCL NAL unit.

NOTE 1 – The value of nuh\_layer\_id for DCI, 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.

The value of nal\_unit\_type shall be the same for all pictures of a CVSS AU.

NAL units that have nal\_unit\_type in the range of UNSPEC\_28..UNSPEC\_31, inclusive, for which semantics are not specified, shall not affect the decoding process specified in this Specification.

NOTE 2 – NAL unit types in the range of UNSPEC\_28..UNSPEC\_31 may be used as determined by the application. No decoding process for these values of nal\_unit\_type is specified in this Specification. Since different applications might use these NAL unit types for different purposes, particular care must be exercised in the design of encoders that generate NAL units with these nal\_unit\_type values, and in the design of decoders that interpret the content of NAL units with these nal\_unit\_type values. This Specification does not define any management for these values. These nal\_unit\_type values might only be suitable for use in contexts in which "collisions" of usage (i.e., different definitions of the meaning of the NAL unit content for the same nal\_unit\_type value) are unimportant, or not possible, or are managed – e.g., defined or managed in the controlling application or transport specification, or by controlling the environment in which bitstreams are distributed.

For purposes other than determining the amount of data in the DUs of the bitstream (as specified in Annex C), decoders shall ignore (remove from the bitstream and discard) the contents of all NAL units that use reserved values of nal\_unit\_type.

NOTE 3 – This requirement allows future definition of compatible extensions to this Specification.

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

|  |  |  |  |
| --- | --- | --- | --- |
| **nal\_unit\_type** | **Name of nal\_unit\_type** | **Content of NAL unit and RBSP syntax structure** | **NAL unit type class** |
| 0 | TRAIL\_NUT | Coded slice of a trailing picture or subpicture\* slice\_layer\_rbsp( ) | VCL |
| 1 | STSA\_NUT | Coded slice of an STSA picture or subpicture\* slice\_layer\_rbsp( ) | VCL |
| 2 | RADL\_NUT | Coded slice of a RADL picture or subpicture\* slice\_layer\_rbsp( ) | VCL |
| 3 | RASL\_NUT | Coded slice of a RASL picture or subpicture\* slice\_layer\_rbsp( ) | VCL |
| 4..6 | RSV\_VCL\_4.. RSV\_VCL\_6 | Reserved non-IRAP VCL NAL unit types | VCL |
| 7 8 | IDR\_W\_RADL IDR\_N\_LP | Coded slice of an IDR picture or subpicture\* slice\_layer\_rbsp( ) | VCL |
| 9 | CRA\_NUT | Coded slice of a CRA picture or subpicture\* silce\_layer\_rbsp( ) | VCL |
| 10 | GDR\_NUT | Coded slice of a GDR picture or subpicture\* slice\_layer\_rbsp( ) | VCL |
| 11 12 | RSV\_IRAP\_11 RSV\_IRAP\_12 | Reserved IRAP VCL NAL unit types | VCL |
| 13 | DCI\_NUT | Decoding capability information decoding\_capability\_information\_rbsp( ) | non-VCL |
| 14 | VPS\_NUT | Video parameter set video\_parameter\_set\_rbsp( ) | non-VCL |
| 15 | SPS\_NUT | Sequence parameter set seq\_parameter\_set\_rbsp( ) | non-VCL |
| 16 | PPS\_NUT | Picture parameter set pic\_parameter\_set\_rbsp( ) | non-VCL |
| 17 18 | PREFIX\_APS\_NUT SUFFIX\_APS\_NUT | Adaptation parameter set adaptation\_parameter\_set\_rbsp( ) | non-VCL |
| 19 | PH\_NUT | Picture header picture\_header\_rbsp( ) | non-VCL |
| 20 | AUD\_NUT | AU delimiter access\_unit\_delimiter\_rbsp( ) | non-VCL |
| 21 | EOS\_NUT | End of sequence end\_of\_seq\_rbsp( ) | non-VCL |
| 22 | EOB\_NUT | End of bitstream end\_of\_bitstream\_rbsp( ) | non-VCL |
| 23 24 | PREFIX\_SEI\_NUT SUFFIX\_SEI\_NUT | Supplemental enhancement information sei\_rbsp( ) | non-VCL |
| 25 | FD\_NUT | Filler data filler\_data\_rbsp( ) | non-VCL |
| 26 27 | RSV\_NVCL\_26 RSV\_NVCL\_27 | Reserved non-VCL NAL unit types | non-VCL |
| 28..31 | UNSPEC\_28.. UNSPEC\_31 | Unspecified non-VCL NAL unit types | non-VCL |
| \* indicates a property of a picture when pps\_mixed\_nalu\_types\_in\_pic\_flag is equal to 0 and a property of the subpicture when pps\_mixed\_nalu\_types\_in\_pic\_flag is equal to 1. | | | |

NOTE 4 – A clean random access (CRA) picture may 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 may 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.

When any two subpictures in a picture have different NAL unit types, the value of sps\_subpic\_treated\_as\_pic\_flag[ ] shall be equal to 1 for all subpictures in the picture that contain at least one P or B slice. [Ed. (YK): Consider rephrasing this constraint to be on the values of slice-level syntax elements based on the SPS-level syntax elements instead.]

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), the following applies:

– 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 the VCL NAL units of at least one subpicture of the picture have a particular value of nal\_unit\_type equal to IDR\_W\_RADL, IDR\_N\_LP, or CRA\_NUT, the VCL NAL units of other subpictures in the picture shall all have nal\_unit\_type equal to TRAIL\_NUT.

When 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, the current picture shall not have both VCL NAL units with a particular value of nal\_unit\_type equal to IDR\_W\_RADL, IDR\_N\_LP, or CRA\_NUT and VCL NAL units with nal\_unit\_type equal to a different value than that particular value.

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 PU by discarding all PUs before the IRAP PU (and to correctly decode the IRAP picture and all the subsequent non-RASL pictures in decoding order), provided each parameter set is available (either in the bitstream or by external means not specified in this Specification) when it is referenced.

– No RADL subpictures shall be present in the bitstream that are associated with an IDR subpicture having nal\_unit\_type equal to IDR\_N\_LP.

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

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

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

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

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

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

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

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

– If sps\_field\_seq\_flag is equal to 0 and 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, 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 and 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, 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:

TemporalId = nuh\_temporal\_id\_plus1 − 1 (36)

When nal\_unit\_type is in the range of IDR\_W\_RADL to RSV\_IRAP\_12, 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 not be equal to 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, 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 may be greater than or equal to the TemporalId of the containing AU, as all PPSs and APSs may 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.

#### Encapsulation of an SODB within an RBSP (informative)

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

The form of encapsulation of an SODB within an RBSP and the use of the emulation\_prevention\_three\_byte for encapsulation of an RBSP within a NAL unit is described for the following purposes:

– To prevent the emulation of start codes within NAL units while allowing any arbitrary SODB to be represented within a NAL unit,

– To enable identification of the end of the SODB within the NAL unit by searching the RBSP for the rbsp\_stop\_one\_bit starting at the end of the RBSP,

– To enable a NAL unit to have a size greater than that of the SODB under some circumstances (using one or more 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 cabac\_zero\_word), a final byte equal to 0x03 is appended to the end of the data. The last zero byte of a byte‑aligned three-byte sequence 0x000000 in the RBSP (which is replaced by the four-byte sequence 0x00000300) is taken into account when searching the RBSP data for the next occurrence of byte-aligned bits with the binary patterns specified above.

1. 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.

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

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

#### Order of NAL units in the bitstream

##### General

The subclauses of clause 7.4.2.4 specify constraints on the order of NAL units in the bitstream.

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

Within a NAL unit, the syntax in clauses 7.3 and D.2 specifies the decoding order of syntax elements. When the VUI parameters or any SEI message specified in ITU-T H.SEI | ISO/IEC 23002-7 is included in a NAL unit specified in this Specification, the syntax of the VUI parameters or the SEI message specified in ITU-T H.SEI | 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.

##### Order of AUs and their association to CVSs

A bitstream consists of one or more CVSs.

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

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

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

##### Order of PUs and their association to AUs

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

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

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

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

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

– The value of nuh\_layer\_id of the VCL NAL unit is less than 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),

– 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, can only be one of the above-listed NAL units.

It is a requirement of bitstream conformance that, when present, the next PU of a particular layer after a PU that belongs to the same layer and contains an EOS NAL unit shall be a CLVSS PU, which is either an IRAP PU with NoOutputBeforeRecoveryFlag equal to 1 or a GDR PU with NoOutputBeforeRecoveryFlag equal to 1.

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

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

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

When a VCL NAL unit has sh\_picture\_header\_in\_slice\_header\_flag equal to 1 or is the first VCL NAL unit that follows a PH NAL unit, the VCL NAL unit is the first VCL NAL unit of a picture.

The order of the non-VCL NAL units (other than the AUD 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 unit of the PU.

– When any suffix APS NAL units are present in a PU, they shall follow the last VCL 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 with in the PU other than an EOB NAL unit (when present).

[Ed. A Figure should be added here]

##### 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 sliceddrB 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.

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

#### Decoding capability information RBSP semantics

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

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

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

**dci\_reserved\_zero\_4bits** shall be equal to 0 in bitstreams conforming to this version of this Specification. The values greater than 0 for dci\_reserved\_zero\_4bits are reserved for future use by ITU-T | ISO/IEC.

**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 conforms to at least one of the profile\_tier\_level( ) syntax structures in the DCI NAL unit. [Ed. (JB): Should this constraint be moved to the PTL semantics?]

NOTE 2 – The DCI NAL unit may 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 there are dci\_extension\_data\_flag syntax elements present in the DCI RBSP syntax structure.

**dci\_extension\_data\_flag** may have any value. Its presence and value do not affect decoder conformance to profiles specified in Annex A. Decoders conforming to this version of this Specification shall ignore all dci\_extension\_data\_flag syntax elements.

#### Video parameter set RBSP semantics

A VPS RBSP shall be available to the decoding process prior to it being referenced, included in at least one AU with TemporalId equal to 0 or provided through external means.

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\_max\_layers\_minus1** plus 1 specifies 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 in each CVS referring to the VPS. The value of vps\_max\_sublayers\_minus1 shall be in the range of 0 to 6, inclusive.

**vps\_all\_layers\_same\_num\_sublayers\_flag** equal to 1 specifies that the number of temporal sublayers is the same for all the layers in each CVS referring to the VPS. vps\_all\_layers\_same\_num\_sublayers\_flag equal to 0 specifies that the layers in each CVS referring to the VPS may or may not have the same number of temporal sublayers. When not present, the value of vps\_all\_layers\_same\_num\_sublayers\_flag is inferred to be equal to 1.

**vps\_all\_independent\_layers\_flag** equal to 1 specifies that all layers in the CVS are independently coded without using inter-layer prediction. vps\_all\_independent\_layers\_flag equal to 0 specifies that one or more of the layers in the CVS may 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 may 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 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 ] is 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 ], RefLayerIdx[ 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++ ) { (37)  
 if( vps\_direct\_ref\_layer\_flag[ i ][ j ] ) {  
 DirectRefLayerIdx[ i ][ d++ ] = j  
 LayerUsedAsRefLayerFlag[ j ] = 1  
 }  
 if( dependencyFlag[ i ][ j ] )  
 RefLayerIdx[ i ][ r++ ] = j  
 }  
 NumDirectRefLayers[ i ] = d  
 NumRefLayers[ i ] = r  
}

The variable GeneralLayerIdx[ i ], specifying the layer index of the layer with nuh\_layer\_id equal to vps\_layer\_id[ i ], is derived as follows:

for( i = 0; i <= vps\_max\_layers\_minus1; i++ ) (38)  
 GeneralLayerIdx[ vps\_layer\_id[ i ] ] = i

For any two different values of i and j, both in the range of 0 to vps\_max\_layers\_minus1, inclusive, when dependencyFlag[ i ][ j ] equal to 1, it is a requirement of bitstream conformance that the values of sps\_chroma\_format\_idc and sps\_bit\_depth\_minus8 that apply to the i-th layer shall be equal to the values of sps\_chroma\_format\_idc and sps\_bit\_depth\_minus8, respectively, that apply to the j-th layer.

**vps\_max\_tid\_il\_ref\_pics\_plus1**[ i ][ j ] equal to 0 specifies that the pictures of the j-th layer that are neither IRAP pictures nor GDR pictures with ph\_recovery\_poc\_cnt equal to 0 are not used as ILRPs for decoding of pictures of the i-th layer. vps\_max\_tid\_il\_ref\_pics\_plus1[ i ][ j ] greater than 0 specifies that, for decoding pictures of the i-th layer, no picture from the j-th layer with TemporalId greater than vps\_max\_tid\_il\_ref\_pics\_plus1[ i ][ j ] − 1 is used as ILRP. When not present, the value of vps\_max\_tid\_il\_ref\_pics\_plus1[ i ][ j ] is inferred to be equal to vps\_max\_sublayers\_minus1 + 1.

**vps\_each\_layer\_is\_an\_ols\_flag** equal to 1 specifies that each OLS contains only one layer and each layer itself in a CVS referring to 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 that at least one OLS contains more than one layer. If vps\_max\_layers\_minus1 is equal to 0, the value of vps\_each\_layer\_is\_an\_ols\_flag is inferred to be equal to 1. Otherwise, when vps\_all\_independent\_layers\_flag is equal to 0, the value of vps\_each\_layer\_is\_an\_ols\_flag is inferred to be equal to 0.

**vps\_ols\_mode\_idc** equal to 0 specifies that the total number of OLSs specified by the VPS is equal to vps\_max\_layers\_minus1 + 1, the i-th OLS includes the layers with layer indices from 0 to i, inclusive, and for each OLS only the highest layer in the OLS is an output layer.

vps\_ols\_mode\_idc equal to 1 specifies that the total number of OLSs specified by the VPS is equal to vps\_max\_layers\_minus1 + 1, the i-th OLS includes the layers with layer indices from 0 to i, inclusive, and for each OLS all layers in the OLS are output layers.

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

The value of vps\_ols\_mode\_idc shall be in the range of 0 to 2, inclusive. The value 3 of vps\_ols\_mode\_idc is reserved for future use by ITU-T | ISO/IEC.

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\_minus1** plus 1 specifies the total number of OLSs specified by the VPS when vps\_ols\_mode\_idc is equal to 2.

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

if( vps\_max\_layers\_minus1 = = 0 )  
 TotalNumOlss = 1  
else if( vps\_each\_layer\_is\_an\_ols\_flag | | vps\_ols\_mode\_idc = = 0 | | vps\_ols\_mode\_idc = = 1 )  
 TotalNumOlss = vps\_max\_layers\_minus1 + 1 (39)  
else if( vps\_ols\_mode\_idc = = 2 )  
 TotalNumOlss = vps\_num\_output\_layer\_sets\_minus1 + 1

**vps\_ols\_output\_layer\_flag**[ i ][ j ] equal to 1 specifies that the layer with nuh\_layer\_id equal to vps\_layer\_id[ j ] is an output layer of the i-th OLS when vps\_ols\_mode\_idc is equal to 2. vps\_ols\_output\_layer\_flag[ i ][ j ] equal to 0 specifies that the layer with nuh\_layer\_id equal to vps\_layer\_id[ j ] is not an output layer of the i-th OLS when vps\_ols\_mode\_idc is equal to 2.

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

NumOutputLayersInOls[ 0 ] = 1  
OutputLayerIdInOls[ 0 ][ 0 ] = vps\_layer\_id[ 0 ]  
NumSubLayersInLayerInOLS[ 0 ][ 0 ] = vps\_max\_sub\_layers\_minus1 + 1  
LayerUsedAsOutputLayerFlag[ 0 ] = 1  
for( i = 1, i <= vps\_max\_layers\_minus1; i++ ) {  
 if( vps\_each\_layer\_is\_an\_ols\_flag | | vps\_ols\_mode\_idc < 2 )  
 LayerUsedAsOutputLayerFlag[ i ] = 1  
 else /\*( !vps\_each\_layer\_is\_an\_ols\_flag && vps\_ols\_mode\_idc = = 2 ) \*/  
 LayerUsedAsOutputLayerFlag[ i ] = 0  
}  
for( i = 1; i < TotalNumOlss; i++ )  
 if( vps\_each\_layer\_is\_an\_ols\_flag | | vps\_ols\_mode\_idc = = 0 ) {  
 NumOutputLayersInOls[ i ] = 1  
 OutputLayerIdInOls[ i ][ 0 ] = vps\_layer\_id[ i ]  
 if( vps\_each\_layer\_is\_an\_ols\_flag )  
 NumSubLayersInLayerInOLS[ i ][ 0 ] = vps\_max\_sub\_layers\_minus1 + 1  
 else {  
 NumSubLayersInLayerInOLS[ i ][ i ] = vps\_max\_sub\_layers\_minus1 + 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\_max\_sub\_layers\_minus1 + 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  
 numLayerInOls = 0  
 for( k = 0, j = 0; k <= vps\_max\_layers\_minus1; k++ ) (40)  
 if( vps\_ols\_output\_layer\_flag[ i ][ k ] ) {  
 layerIncludedInOlsFlag[ i ][ k ] = 1  
 highestIncludedLayer = k  
 numLayerInOls++  
 LayerUsedAsOutputLayerFlag[ k ] = 1  
 OutputLayerIdx[ i ][ j ] = k  
 OutputLayerIdInOls[ i ][ j++ ] = vps\_layer\_id[ k ]  
 NumSubLayersInLayerInOLS[ i ][ k ] = vps\_max\_sub\_layers\_minus1 + 1  
 }  
 NumOutputLayersInOls[ i ] = j  
 for( j = 0; j < NumOutputLayersInOls[ i ]; j++ ) {  
 idx = OutputLayerIdx[ i ][ j ]  
 for( k = 0; k < NumRefLayers[ idx ]; k++ ) {  
 if (!layerIncludedInOlsFlag[ i ][ RefLayerIdx[ idx ][ k ] ] )  
 numLayerInOls++  
 layerIncludedInOlsFlag[ i ][ RefLayerIdx[ 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 be both equal to 0. In other words, there shall be no layer that is neither an output layer of at least one OLS nor a direct reference layer of any other layer.

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

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

NumLayersInOls[ 0 ] = 1  
LayerIdInOls[ 0 ][ 0 ] = vps\_layer\_id[ 0 ]  
NumMultiLayerOlss = 0  
for( i = 1; i < TotalNumOlss; i++ ) {  
 if( vps\_each\_layer\_is\_an\_ols\_flag ) {  
 NumLayersInOls[ i ] = 1  
 LayerIdInOls[ i ][ 0 ] = vps\_layer\_id[ i ] (41)  
 } else if( vps\_ols\_mode\_idc = = 0 | | vps\_ols\_mode\_idc = = 1 ) {  
 NumLayersInOls[ i ] = i + 1  
 for( j = 0; j < NumLayersInOls[ i ]; j++ )  
 LayerIdInOls[ i ][ j ] = vps\_layer\_id[ j ]  
 } else if( vps\_ols\_mode\_idc = = 2 ) {  
 for( k = 0, j = 0; k <= vps\_max\_layers\_minus1; k++ )  
 if( layerIncludedInOlsFlag[ i ][ k ] )  
 LayerIdInOls[ i ][ j++ ] = vps\_layer\_id[ k ]  
 NumLayersInOls[ i ] = j  
 }  
 if( NumLayersInOls[ i ] > 1 ) {  
 MultiLayerOlsIdx[ i ] = NumMultiLayerOlss  
 NumMultiLayerOlss++  
 }  
}

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

The variable OlsLayerIdx[ i ][ j ], specifying the OLS layer index of the layer with nuh\_layer\_id equal to LayerIdInOls[ i ][ j ], is derived as follows:

for( i = 0; i < TotalNumOlss; i++ )  
 for j = 0; j < NumLayersInOls[ i ]; j++ ) (42)  
 OlsLayerIdx[ i ][ LayerIdInOls[ i ][ j ] ] = j

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

Each layer shall be included in at least one OLS specified by the VPS. In other words, for each layer with a particular value of nuh\_layer\_id nuhLayerId equal to one of vps\_layer\_id[ k ] for k in the range of 0 to vps\_max\_layers\_minus1, inclusive, there shall be at least one pair of values of i and j, where i is in the range of 0 to TotalNumOlss − 1, inclusive, and j is in the range of NumLayersInOls[ i ] − 1, inclusive, such that the value of LayerIdInOls[ i ][ j ] is equal to nuhLayerId.

**vps\_num\_ptls\_minus1** plus 1 specifies the number of profile\_tier\_level( ) syntax structures in the VPS. The value of vps\_num\_ptls\_minus1 shall be less than TotalNumOlss.

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

**vps\_ptl\_max\_temporal\_id**[ i ] specifies the TemporalId of the highest sublayer representation for which the level information is present in the i-th profile\_tier\_level( ) syntax structure in the VPS. The value of vps\_ptl\_max\_temporal\_id[ i ] shall be in the range of 0 to vps\_max\_sublayers\_minus1, inclusive. When not present, the value of vps\_ptl\_max\_temporal\_id[ i ] is inferred to be equal to vps\_max\_sublayers\_minus1.

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

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

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

– If vps\_num\_ptls\_minus1 is equal to 0, the value of vps\_ols\_ptl\_idx[ i ] is inferred to be equal to 0.

– Otherwise (vps\_num\_ptls\_minus1 is greater than 0 and vps\_num\_ptls\_minus1 + 1 is equal to TotalNumOlss), the value of vps\_ols\_ptl\_idx[ i ] is inferred to be equal to i.

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

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

**vps\_num\_dpb\_params\_minus1** plus 1, when present, specifies the number of dpb\_parameters( ) syntax strutcures in the VPS. The value of vps\_num\_dpb\_params\_minus1 shall be in the range of 0 to NumMultiLayerOlss − 1, inclusive.

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

if( vps\_each\_layer\_is\_an\_ols\_flag )  
 VpsNumDpbParams = 0 (43)  
else  
 VpsNumDpbParams = vps\_num\_dpb\_params\_minus1 + 1

**vps\_sublayer\_dpb\_params\_present\_flag** is used to control the presence of max\_dec\_pic\_buffering\_minus1[ ], max\_num\_reorder\_pics[ ], and max\_latency\_increase\_plus1[ ] syntax elements in the dpb\_parameters( ) syntax strucures in the VPS. When not present, vps\_sub\_dpb\_params\_info\_present\_flag is inferred to be equal to 0.

**vps\_dpb\_max\_temporal\_id**[ i ] specifies the TemporalId of the highest sublayer representation for which the DPB parameters may be present in the i-th dpb\_parameters( ) syntax strutcure in the VPS. The value of vps\_dpb\_max\_temporal\_id[ i ] shall be in the range of 0 to vps\_max\_sublayers\_minus1, inclusive. When not present, the value of vps\_dpb\_max\_temporal\_id[ i ] is inferred to be equal to vps\_max\_sublayers\_minus1.

**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\_bit\_depth\_minus8 for all SPSs that are referred to by CLVSs in the CVS for the i-th multi-layer OLS.

NOTE 2 – For decoding the i-th multi-layer OLS, the deoder can 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\_general\_hrd\_params\_present\_flag** equal to 1 specifies that the VPS contains a general\_hrd\_parameters( ) syntax structure and other HRD parameters. vps\_general\_hrd\_params\_present\_flag equal to 0 specifies that the VPS does not contain a general\_hrd\_parameters( ) syntax structure or other HRD parameters. When not present, the value of vps\_general\_hrd\_params\_present\_flag is inferred to be equal to 0.

When NumLayersInOls[ i ] is equal to 1, the general\_hrd\_parameters( ) syntax structure and the ols\_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\_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\_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\_hrd\_parameters syntax structure.

**vps\_num\_ols\_hrd\_params\_minus1** plus 1 specifies the number of ols\_hrd\_parameters( ) syntax structures present in the VPS when vps\_general\_hrd\_params\_present\_flag is equal to 1. The value of vps\_num\_ols\_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\_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.

**vps\_ols\_hrd\_idx**[ i ] specifies the index, to the list of ols\_hrd\_parameters( ) syntax structures in the VPS, of the ols\_hrd\_parameters( ) syntax structure that applies to the i-th multi-layer OLS. The value of vps\_ols\_hrd\_idx[ i ] shall be in the range of 0 to vps\_num\_ols\_hrd\_params\_minus1, inclusive.

When vps\_ols\_hrd\_idx[ i ] is not present, it is inferred as follows:

– If vps\_num\_ols\_hrd\_params\_minus1 is equal to 0, the value of vps\_ols\_hrd\_idx[[ i ] is inferred to be equal to 0.

– Otherwise (vps\_num\_ols\_hrd\_params\_minus1 + 1 is greater than 1 and equal to NumMultiLayerOlss), the value of vps\_ols\_hrd\_idx[ i ] is inferred to be equal to i.

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

Each ols\_hrd\_parameters( ) syntax structure in the VPS shall be referred to by at least one value of vps\_ols\_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 there are vps\_extension\_data\_flag syntax elements present in the VPS RBSP syntax structure.

**vps\_extension\_data\_flag** may have any value. Its presence and value do not affect decoder conformance to profiles specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore all vps\_extension\_data\_flag syntax elements.

#### Sequence parameter set RBSP semantics

An SPS RBSP shall be available to the decoding process prior to it being referenced, included in at least one AU with TemporalId equal to 0 or provided through external means.

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.

**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 value of vps\_max\_sublayers\_minus1 is inferred to be equal to 6.

– 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 inferred to be equal to 0.

– The value of vps\_independent\_layer\_flag[ GeneralLayerIdx[ nuh\_layer\_id ] ] is inferred to be equal to 1.

When vps\_independent\_layer\_flag[ GeneralLayerIdx[ nuh\_layer\_id ] ] is equal to 1, the SPS referred to by a CLVS with a particluar 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 may be present in each CLVS referring to the SPS. The value of sps\_max\_sublayers\_minus1 shall be in the range of 0 to vps\_max\_sublayers\_minus1, inclusive.

**sps\_reserved\_zero\_4bits** shall be equal to 0 in bitstreams conforming to this version of this Specification. Other values for sps\_reserved\_zero\_4bits are reserved for future use by ITU-T | ISO/IEC.

**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\_hrd\_parameters( ) syntax structure and an ols\_hrd\_parameters( ) syntax structure may 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 may be present in CLVS. sps\_gdr\_enabled\_flag equal to 0 specifies that GDR pictures are disabled and not present in CLVS. [Ed. (GJS): There should be a general review of the word "may", for wording consistency and clarity – e.g., whether it expresses permission.]

**sps\_chroma\_format\_idc** specifies the chroma sampling relative to the luma sampling as specified in clause 6.2.

When sps\_video\_parameter\_set\_id is greater than 0 and the SPS is referenced by a layer that is included in the i-th multi-layer OLS specified by the VPS for any i in the range of 0 to NumMultiLayerOlss − 1, inclusive, it is a requirement of bitstream conformance that the value of sps\_chroma\_format\_idc shall be less than or equal to the value of vps\_ols\_dpb\_chroma\_format[ i ].

**sps\_separate\_colour\_plane\_flag** equal to 1 specifies that the three colour components of the 4:4:4 chroma format are coded separately. sps\_separate\_colour\_plane\_flag equal to 0 specifies that the colour components are not coded separately. When sps\_separate\_colour\_plane\_flag is not present, it is inferred to be equal to 0. When sps\_separate\_colour\_plane\_flag is equal to 1, the coded picture consists of three separate components, each of which consists of coded samples of one colour plane (Y, Cb, or Cr) and uses the monochrome coding syntax. In this case, each colour plane is associated with a specific sh\_colour\_plane\_id value.

NOTE 1 – There is no dependency in decoding processes between the colour planes having different sh\_colour\_plane\_id values. For example, the decoding process of a monochrome picture with one value of sh\_colour\_plane\_id does not use any data from monochrome pictures having different values of sh\_colour\_plane\_id for inter prediction.

Depending on the value of sps\_separate\_colour\_plane\_flag, the value of the variable ChromaArrayType is assigned as follows:

– If sps\_separate\_colour\_plane\_flag is equal to 0, ChromaArrayType is set equal to sps\_chroma\_format\_idc.

– Otherwise (sps\_separate\_colour\_plane\_flag is equal to 1), ChromaArrayType is set equal to 0.

**sps\_ref\_pic\_resampling\_enabled\_flag** equal to 1 specifies that reference picture resampling is enabled and one or more slices of pictures in the CLVS may refer to a reference picture with a different spatial resolution in an active entry of a reference picture list. sps\_ref\_pic\_resampling\_enabled\_flag equal to 0 specifies that reference picture resampling is disabled and no slice of pictures in the CLVS refers to a reference picture with a different spatial resolution in an active entry of a reference picture list. [Ed. (YK): Check whether the phrase "refer to a reference picture with a different spatial resolution" should be changed to be "refer to a reference picture with a different scaling window".]

NOTE 2 – When sps\_ref\_pic\_resampling\_enabled\_flag is equal to 1, for a current picture the reference picture with a different spatial resolution may 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 may change within a CLVS referring to the SPS. sps\_res\_change\_in\_clvs\_allowed\_flag equal to 0 specifies that the picture spatial resolution does not change within any CLVS referring to the SPS. When not present, the value of sps\_res\_change\_in\_clvs\_allowed\_flag is inferred to be equal to 0.

**sps\_pic\_width\_max\_in\_luma\_samples** specifies the maximum width, in units of luma samples, of each decoded picture referring to the SPS. sps\_pic\_width\_max\_in\_luma\_samples shall not be equal to 0 and shall be an integer multiple of Max( 8, MinCbSizeY ).

When sps\_video\_parameter\_set\_id is greater than 0 and the SPS is referenced by a layer that is included in the i-th multi-layer OLS specified by the VPS for any i in the range of 0 to NumMultiLayerOlss − 1, inclusive, it is a requirement of bitstream conformance that the value of sps\_pic\_width\_max\_in\_luma\_samples shall be less than or equal to the value of vps\_ols\_dpb\_pic\_width[ i ].

**sps\_pic\_height\_max\_in\_luma\_samples** specifies the maximum height, in units of luma samples, of each decoded picture referring to the SPS. sps\_pic\_height\_max\_in\_luma\_samples shall not be equal to 0 and shall be an integer multiple of Max( 8, MinCbSizeY ).

When sps\_video\_parameter\_set\_id is greater than 0 and the SPS is referenced by a layer that is included in the i-th multi-layer OLS specified by the VPS for any i in the range of 0 to NumMultiLayerOlss − 1, inclusive, it is a requirement of bitstream conformance that the value of sps\_pic\_height\_max\_in\_luma\_samples shall be less than or equal to the value of vps\_ols\_dpb\_pic\_height[ i ].

**sps\_conformance\_window\_flag** equal to 1 indicates that the conformance cropping window offset parameters follow next in the SPS. sps\_conformance\_window\_flag equal to 0 indicates that the conformance cropping window offset parameters are not present in the SPS.

**sps\_conf\_win\_left\_offset**, **sps\_conf\_win\_right\_offset**, **sps\_conf\_win\_top\_offset**, and **sps\_conf\_win\_bottom\_offset** specify the cropping window that is applied to pictures with pps\_pic\_width\_in\_luma\_samples equal to sps\_pic\_width\_max\_in\_luma\_samples and pps\_pic\_height\_in\_luma\_samples equal to sps\_pic\_height\_max\_in\_luma\_samples. When sps\_conformance\_window\_flag is equal to 0, the values of sps\_conf\_win\_left\_offset, sps\_conf\_win\_right\_offset, sps\_conf\_win\_top\_offset, and sps\_conf\_win\_bottom\_offset are inferred to be equal to 0.

The conformance cropping window contains the luma samples with horizontal picture coordinates from SubWidthC \* sps\_conf\_win\_left\_offset to sps\_pic\_width\_max\_in\_luma\_samples − ( SubWidthC \* sps\_conf\_win\_right\_offset + 1 ) and vertical picture coordinates from SubHeightC \* sps\_conf\_win\_top\_offset to sps\_pic\_height\_max\_in\_luma\_samples − ( SubHeightC \* sps\_conf\_win\_bottom\_offset + 1 ), inclusive.

The value of SubWidthC \* ( sps\_conf\_win\_left\_offset + sps\_conf\_win\_right\_offset ) shall be less than sps\_pic\_width\_max\_in\_luma\_samples, and the value of SubHeightC \* ( sps\_conf\_win\_top\_offset + sps\_conf\_win\_bottom\_offset ) shall be less than sps\_pic\_height\_max\_in\_luma\_samples.

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

NOTE 3 – The conformance cropping window offset parameters are only applied at the output. All internal decoding processes are applied to the uncropped picture size.

**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.

The variables CtbLog2SizeY and CtbSizeY are derived as follows:

CtbLog2SizeY = sps\_log2\_ctu\_size\_minus5 + 5 (44)

CtbSizeY = 1  <<  CtbLog2SizeY (45)

**sps\_subpic\_info\_present\_flag** equal to 1 specifies that subpicture information is present for the CLVS and there may 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 4 – When a bitstream is the result of a sub-bitstream extraction process and contains only a subset of the subpictures of the input bitstream to the 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 Ceil( sps\_pic\_width\_max\_in\_luma\_samples ÷ CtbSizeY ) \* Ceil( sps\_pic\_height\_max\_in\_luma\_samples ÷ CtbSizeY ) − 1, inclusive. 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\_ctu\_top\_left\_x**[ i ] specifies horizontal position of top left CTU of i-th subpicture in unit of CtbSizeY. The length of the syntax element is Ceil( Log2( ( sps\_pic\_width\_max\_in\_luma\_samples + CtbSizeY  − 1 )  >>  CtbLog2SizeY ) ) bits. When not present, the value of sps\_subpic\_ctu\_top\_left\_x[ i ] is inferred to be equal to 0.

**sps\_subpic\_ctu\_top\_left\_y**[ i ] specifies vertical position of top left CTU of i-th subpicture in unit of CtbSizeY. The length of the syntax element is Ceil( Log2( ( sps\_pic\_height\_max\_in\_luma\_samples + CtbSizeY − 1 )  >>  CtbLog2SizeY ) ) bits. When not present, the value of sps\_subpic\_ctu\_top\_left\_y[ i ] is inferred to be equal to 0.

**sps\_subpic\_width\_minus1**[ i ] plus 1 specifies the width of the i-th subpicture in units of CtbSizeY. The length of the syntax element is Ceil( Log2( ( sps\_pic\_width\_max\_in\_luma\_samples + CtbSizeY − 1 )  >>  CtbLog2SizeY ) ) bits. When not present, the value of sps\_subpic\_width\_minus1[ i ] is inferred to be equal to ( ( sps\_pic\_width\_max\_in\_luma\_samples + CtbSizeY− 1 )  >>  CtbLog2SizeY ) − sps\_subpic\_ctu\_top\_left\_x[ i ] − 1.

**sps\_subpic\_height\_minus1**[ i ] plus 1 specifies the height of the i-th subpicture in units of CtbSizeY. The length of the syntax element is Ceil( Log2( ( sps\_pic\_height\_max\_in\_luma\_samples + CtbSizeY − 1 )  >>  CtbLog2SizeY ) ) bits. When not present, the value of sps\_subpic\_height\_minus1[ i ] is inferred to be equal to ( ( sps\_pic\_height\_max\_in\_luma\_samples + CtbSizeY− 1 )  >>  CtbLog2SizeY ) − sps\_subpic\_ctu\_top\_left\_y[ i ] − 1.

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

For each subpicture with subpicture index i in the range of 0 to sps\_num\_subpics\_minus1, inclusive, it is a requirement of bitstream conformance that all of the following conditions are true:

– The value of ( sps\_subpic\_ctu\_top\_left\_x[ i ] \* CtbSizeY ) shall be less than ( sps\_pic\_width\_max\_in\_luma\_samples − sps\_conf\_win\_right\_offset \* SubWidthC ).

– The value of ( ( sps\_subpic\_ctu\_top\_left\_x[ i ] + sps\_subpic\_width\_minus1[ i ] + 1 ) \* CtbSizeY ) shall be greater than ( sps\_conf\_win\_left\_offset \* SubWidthC ).

– The value of ( sps\_subpic\_ctu\_top\_left\_y[ i ] \* CtbSizeY ) shall be less than ( sps\_pic\_height\_max\_in\_luma\_samples − sps\_conf\_win\_bottom\_offset \* SubHeightC ).

– The value of ( ( sps\_subpic\_ctu\_top\_left\_y[ i ] + sps\_subpic\_height\_minus1[ i ] + 1 ) \* CtbSizeY ) shall be greater than ( sps\_conf\_win\_top\_offset \* SubHeightC ).

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

When sps\_num\_subpics\_minus1 is greater than 0 and sps\_subpic\_treated\_as\_pic\_flag[ i ] is equal to 1, for each CLVS of a current layer referring to the SPS, let targetAuSet be all the AUs starting from the AU containing the first picture of the CLVS in decoding order, to the AU containing the last picture of the CLVS in decoding order, inclusive, it is a requirement of bitstream conformance that all of the following conditions are true for the targetLayerSet that consists of the current layer and all the layers that have the current layer as a reference layer:

– For each AU in targetAuSet, all pictures of the layers in targetLayerSet shall have the same value of pps\_pic\_width\_in\_luma\_samples and the same value of pps\_pic\_height\_in\_luma\_samples.

– All the SPSs referred to by the layers in targetLayerSet shall have the same value of sps\_num\_subpics\_minus1 and shall have the same values of sps\_subpic\_ctu\_top\_left\_x[ j ], sps\_subpic\_ctu\_top\_left\_y[ j ], sps\_subpic\_width\_minus1[ j ], sps\_subpic\_height\_minus1[ j ], and sps\_subpic\_treated\_as\_pic\_flag[ j ], respectively, for each value of j in the range of 0 to sps\_num\_subpics\_minus1, inclusive.

– For each AU in targetAuSet, all pictures of the layers in targetLayerSet shall have the same value of SubpicIdVal[ j ] for each value of j in the range of 0 to sps\_num\_subpics\_minus1, inclusive.

**sps\_loop\_filter\_across\_subpic\_enabled\_flag**[ i ] equal to 1 specifies that in-loop filtering operations across subpicture boundaries is enabled and may be performed across the boundaries of the i-th subpicture in each coded picture in the CLVS. sps\_loop\_filter\_across\_subpic\_enabled\_flag[ i ] equal to 0 specifies that in-loop filtering operations across subpicture boundaries is disabled and are not performed across the boundaries of the i-th subpicture in each coded picture in the CLVS. When not present, the value of sps\_loop\_filter\_across\_subpic\_enabled\_pic\_flag[ i ] is inferred to be equal to 0.

**sps\_subpic\_id\_len\_minus1** plus 1 specifies the number of bits used to represent the syntax element sps\_subpic\_id[ i ], the syntax elements pps\_subpic\_id[ i ], when present, and the syntax element sh\_subpic\_id, when present. The value of sps\_subpic\_id\_len\_minus1 shall be in the range of 0 to 15, inclusive. The value of 1  <<  ( sps\_subpic\_id\_len\_minus1 + 1 ) shall be greater than or equal to sps\_num\_subpics\_minus1 + 1.

**sps\_subpic\_id\_mapping\_explicitly\_signalled\_flag** equal to 1 specifies that the subpicture ID mapping is explicitly signalled, either in the SPS or in the PPSs referred to by coded pictures of the CLVS. sps\_subpic\_id\_mapping\_explicitly\_signalled\_flag equal to 0 specifies that the subpicture ID mapping is not explicitly signalled for the CLVS. When not present, the value of sps\_subpic\_id\_mapping\_explicitly\_signalled\_flag is inferred to be equal to 0.

**sps\_subpic\_id\_mapping\_present\_flag** equal to 1 specifies that the subpicture ID mapping is signalled in the SPS when sps\_subpic\_id\_mapping\_explicitly\_signalled\_flag is equal to 1. sps\_subpic\_id\_mapping\_present\_flag equal to 0 specifies that subpicture ID mapping is signalled in the PPSs referred to by coded pictures of the CLVS when sps\_subpic\_id\_mapping\_explicitly\_signalled\_flag is equal to 1.

**sps\_subpic\_id**[ i ] specifies the subpicture ID of the i-th subpicture. The length of the sps\_subpic\_id[ i ] syntax element is sps\_subpic\_id\_len\_minus1 + 1 bits.

**sps\_bit\_depth\_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:

BitDepth = 8 + sps\_bit\_depth\_minus8 (46)

QpBdOffset = 6 \* sps\_bit\_depth\_minus8 (47)

sps\_bit\_depth\_minus8 shall be in the range of 0 to 8, inclusive.

When sps\_video\_parameter\_set\_id is greater than 0 and the SPS is referenced by a layer that is included in the i-th multi-layer OLS specified by the VPS for any i in the range of 0 to NumMultiLayerOlss − 1, inclusive, it is a requirement of bitstream conformance that the value of sps\_bit\_depth\_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 5 – 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 may 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:

MaxPicOrderCntLsb = 2( sps\_log2\_max\_pic\_order\_cnt\_lsb\_minus4 + 4 ) (48)

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 PHs 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 PHs 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 the PHs 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\_bits\_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\_bits\_bytes shall be equal to 0 in bitstreams conforming to this version of this Specification. Although the value of sps\_num\_extra\_ph\_bits\_bytes is required to be equal to 0 in this version of this Specification, decoder conforming to this version of this Specification shall allow the value of sps\_num\_extra\_ph\_bits\_bytes equal to 1 or 2 to appear in the syntax.

**sps\_num\_extra\_sh\_bits\_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\_bits\_bytes shall be equal to 0 in bitstreams conforming to this version of this Specification. Although the value of sps\_num\_extra\_sh\_bits\_bytes is required to be equal to 0 in this version of this Specification, decoder conforming to this version of this Specification shall allow the value of sps\_num\_extra\_sh\_bits\_bytes equal to 1 or 2 to appear in the syntax.

**sps\_sublayer\_dpb\_params\_flag** is used to control the presence of max\_dec\_pic\_buffering\_minus1[ i ], max\_num\_reorder\_pics[ i ], and max\_latency\_increase\_plus1[ i ] syntax elements in the dpb\_parameters( ) syntax strucure in the SPS. When not present, the value of sps\_sub\_dpb\_params\_info\_present\_flag is inferred to be equal to 0.

**sps\_qtbtt\_dual\_tree\_intra\_flag** equal to 1 specifies that, for I slices, each CTU is split into coding units with 64×64 luma samples using an implicit quadtree split, and these coding units are the root of two separate coding\_tree syntax structure for luma and chroma. sps\_qtbtt\_dual\_tree\_intra\_flag equal to 0 specifies separate coding\_tree syntax structure is not used for I slices. When sps\_qtbtt\_dual\_tree\_intra\_flag is not present, it is inferred to be equal to 0.

**sps\_log2\_min\_luma\_coding\_block\_size\_minus2** plus 2 specifies the minimum luma coding block size. The value range of sps\_log2\_min\_luma\_coding\_block\_size\_minus2 shall be in the range of 0 to Min( 4, sps\_log2\_ctu\_size\_minus5 + 3 ), inclusive.

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

MinCbLog2SizeY = sps\_log2\_min\_luma\_coding\_block\_size\_minus2 + 2 (49)

MinCbSizeY = 1  <<  MinCbLog2SizeY (50)

IbcBufWidthY = 256 \* 128 / CtbSizeY (51)

IbcBufWidthC = IbcBufWidthY / SubWidthC (52)

VSize = Min( 64, CtbSizeY ) (53)

The value of MinCbSizeY shall 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) or sps\_separate\_colour\_plane\_flag is equal to 1, CtbWidthC and CtbHeightC are both equal to 0.

– Otherwise, CtbWidthC and CtbHeightC are derived as follows:

CtbWidthC = CtbSizeY / SubWidthC (54)

CtbHeightC = CtbSizeY / SubHeightC (55)

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

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

**sps\_partition\_constraints\_override\_enabled\_flag** equal to 1 specifies the presence of ph\_partition\_constraints\_override\_flag in PHs referring to the SPS. sps\_partition\_constraints\_override\_enabled\_flag equal to 0 specifies the absence of ph\_partition\_constraints\_override\_flag in PHs 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\_luma present in PHs referring to the SPS. The value of sps\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_luma shall be in the range of 0 to Min( 6, CtbLog2SizeY ) − MinCbLog2SizeY, inclusive. The base 2 logarithm of the minimum size in luma samples of a luma leaf block resulting from quadtree splitting of a CTU is derived as follows:

MinQtLog2SizeIntraY = sps\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_luma + MinCbLog2SizeY (56)

**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 PHs referring to the SPS. The value of sps\_max\_mtt\_hierarchy\_depth\_intra\_slice\_luma shall be in the range of 0 to 2\*( CtbLog2SizeY − MinCbLog2SizeY ), inclusive.

**sps\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_luma** specifies the default difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a luma coding block that can be split using a binary split and the 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 PHs 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 ( sps\_qtbtt\_dual\_tree\_intra\_flag ? Min( 6, CtbLog2SizeY ) : CtbLog2SizeY ) − MinQtLog2SizeIntraY, inclusive. When sps\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_luma is not present, the value of sps\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_luma is inferred to be equal to 0.

**sps\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_luma** specifies the default difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a luma coding block that can be split using a ternary split and the 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 PHs referring to the SPS. The value of sps\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_luma shall be in the range of 0 to Min( 6, CtbLog2SizeY ) − MinQtLog2SizeIntraY, inclusive. When sps\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_luma is not present, the value of sps\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_luma is inferred to be equal to 0.

**sps\_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\_luma present in PHs referring to the SPS. The value of sps\_log2\_diff\_min\_qt\_min\_cb\_inter\_slice shall be in the range of 0 to Min( 6, CtbLog2SizeY ) − MinCbLog2SizeY, inclusive. The base 2 logarithm of the minimum size in luma samples of a luma leaf block resulting from quadtree splitting of a CTU is derived as follows:

MinQtLog2SizeInterY = sps\_log2\_diff\_min\_qt\_min\_cb\_inter\_slice + MinCbLog2SizeY (57)

**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 PHs referring to the SPS. The value of sps\_max\_mtt\_hierarchy\_depth\_inter\_slice shall be in the range of 0 to 2\*( CtbLog2SizeY − MinCbLog2SizeY ), inclusive.

**sps\_log2\_diff\_max\_bt\_min\_qt\_inter\_slice** specifies the default difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a luma coding block that can be split using a binary split and the 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 PHs referring to the SPS. The value of sps\_log2\_diff\_max\_bt\_min\_qt\_inter\_slice shall be in the range of 0 to CtbLog2SizeY − MinQtLog2SizeInterY, inclusive. When sps\_log2\_diff\_max\_bt\_min\_qt\_inter\_slice is not present, the value of sps\_log2\_diff\_max\_bt\_min\_qt\_inter\_slice is inferred to be equal to 0.

**sps\_log2\_diff\_max\_tt\_min\_qt\_inter\_slice** specifies the default difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a luma coding block that can be split using a ternary split and the 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 PHs referring to the SPS. The value of sps\_log2\_diff\_max\_tt\_min\_qt\_inter\_slice shall be in the range of 0 to Min( 6, CtbLog2SizeY ) − MinQtLog2SizeInterY, inclusive. When sps\_log2\_diff\_max\_tt\_min\_qt\_inter\_slice is not present, the value of sps\_log2\_diff\_max\_tt\_min\_qt\_inter\_slice is inferred to be equal to 0.

**sps\_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 PHs referring to the SPS. The value of sps\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_chroma shall be in the range of 0 to Min( 6, CtbLog2SizeY ) − MinCbLog2SizeY, inclusive. When not present, the value of sps\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_chroma is inferred to be equal to 0. The base 2 logarithm of the minimum size in luma samples of a chroma leaf block resulting from quadtree splitting of a CTU with treeType equal to DUAL\_TREE\_CHROMA is derived as follows:

MinQtLog2SizeIntraC = sps\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_chroma + MinCbLog2SizeY (58)

**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 PHs referring to the SPS. The value of sps\_max\_mtt\_hierarchy\_depth\_intra\_slice\_chroma shall be in the range of 0 to 2\*( CtbLog2SizeY − MinCbLog2SizeY ), inclusive. When not present, the value of sps\_max\_mtt\_hierarchy\_depth\_intra\_slice\_chroma is inferred to be equal to 0.

**sps\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_chroma** specifies the default difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a chroma coding block that can be split using a binary split and the 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 PHs referring to the SPS. The value of sps\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_chroma shall be in the range of 0 to Min( 6, CtbLog2SizeY ) − MinQtLog2SizeIntraC, inclusive. When sps\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_chroma is not present, the value of sps\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_chroma is inferred to be equal to 0.

**sps\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_chroma** specifies the default difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a chroma coding block that can be split using a ternary split and the 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 PHs referring to the SPS. The value of sps\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_chroma shall be in the range of 0 to Min( 6, CtbLog2SizeY ) − MinQtLog2SizeIntraC, inclusive. When sps\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_chroma is not present, the value of sps\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_chroma is inferred to be equal to 0.

**sps\_max\_luma\_transform\_size\_64\_flag** equal to 1 specifies that the maximum transform size in luma samples is equal to 64. sps\_max\_luma\_transform\_size\_64\_flag equal to 0 specifies that the maximum transform size in luma samples is equal to 32. When not present, the value of sps\_max\_luma\_transform\_size\_64\_flag is inferred to be equal to 0.

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

MinTbLog2SizeY = 2 (59)

MaxTbLog2SizeY = sps\_max\_luma\_transform\_size\_64\_flag ? 6 : 5 (60)

MinTbSizeY = 1  <<  MinTbLog2SizeY (61)

MaxTbSizeY = 1  <<  MaxTbLog2SizeY (62)

**sps\_joint\_cbcr\_enabled\_flag** equal to 0 specifies that the joint coding of chroma residuals is disabled and not used in decoding of pictures in the CLVS. sps\_joint\_cbcr\_enabled\_flag equal to 1 specifies that the joint coding of chroma residuals is enabled and may be used in decoding of pictures in the CLVS. When not present, the value of sps\_joint\_cbcr\_enabled\_flag is inferred to be equal to 0.

**sps\_same\_qp\_table\_for\_chroma\_flag** equal to 1 specifies that only one chroma QP mapping table is signalled and this table applies to Cb and Cr residuals and additionally to joint Cb-Cr residuals when sps\_joint\_cbcr\_enabled\_flag is equal to 1. sps\_same\_qp\_table\_for\_chroma\_flag equal to 0 specifies that chroma QP mapping tables, two for Cb and Cr, and one additional for joint Cb-Cr when sps\_joint\_cbcr\_enabled\_flag is equal to 1, are signalled in the SPS. When not present, the value of sps\_same\_qp\_table\_for\_chroma\_flag is inferred to be equal to 1.

**sps\_qp\_table\_start\_minus26**[ i ] plus 26 specifies the starting luma and chroma QP used to describe the i-th chroma QP mapping table. The value of sps\_qp\_table\_start\_minus26[ i ] shall be in the range of −26 − QpBdOffset to 36 inclusive. When not present, the value of sps\_qp\_table\_start\_minus26[ i ] is inferred to be equal to 0.

**sps\_num\_points\_in\_qp\_table\_minus1**[ i ] plus 1 specifies the number of points used to describe the i-th chroma QP mapping table. The value of sps\_num\_points\_in\_qp\_table\_minus1[ i ] shall be in the range of 0 to 63 + QpBdOffset, 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 ) (63)  
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 the sample adaptive offset process is enabled and may be applied to the reconstructed picture after the deblocking filter process for the CLVS. sps\_sao\_enabled\_flag equal to 0 specifies that the sample adaptive offset process is disabled and not applied to the reconstructed picture after the deblocking filter process for the CLVS.

**sps\_alf\_enabled\_flag** equal to 0 specifies that the adaptive loop filter is disabled and not applied in decoding of pictures in the CLVS. sps\_alf\_enabled\_flag equal to 1 specifies that the adaptive loop filter is enabled and may be applied in decoding of pictures in the CLVS.

**sps\_ccalf\_enabled\_flag** equal to 0 specifies that the cross-component adaptive loop filter is disabled and not applied in decoding of pictures in the CLVS. sps\_ccalf\_enabled\_flag equal to 1 specifies that the cross-component adaptive loop filter is enabled and may be applied in decoding of pictures in the CLVS. When not present, the value of sps\_ccalf\_enabled\_flag is inferred to be equal to 0.

**sps\_transform\_skip\_enabled\_flag** equa to 1 specifies that transform\_skip\_flag may be present in the transform unit syntax. sps\_transform\_skip\_enabled\_flag equal to 0 specifies that transform\_skip\_flag is not present in the transform unit syntax.

**sps\_log2\_transform\_skip\_max\_size\_minus2** specifies the maximum block size used for transform skip, and shall be in the range of 0 to 3, inclusive.

The variable MaxTsSize is set equal to 1  <<  ( sps\_log2\_transform\_skip\_max\_size\_minus2 + 2 ).

**sps\_bdpcm\_enabled\_flag** equal to 1 specifies that intra\_bdpcm\_luma\_flag and intra\_bdpcm\_chroma\_flag may 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\_weighted\_pred\_flag** equal to 1 specifies that weighted prediction may 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 may 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 may be used for inter prediction of one or more coded pictures in the CLVS.

**sps\_inter\_layer\_ref\_pics\_present\_flag** equal to 0 specifies that no ILRP is used for inter prediction of any coded picture in the CLVS. sps\_inter\_layer\_ref\_pics\_present\_flag equal to 1 specifies that ILRPs may be used for inter prediction of one or more coded pictures in the CLVS. When sps\_video\_parameter\_set\_id is equal to 0, the value of sps\_inter\_layer\_ref\_pics\_present\_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\_ref\_pics\_present\_flag shall be equal to 0. [Ed. (YK): Check whether there is a better name for this syntax element.]

**sps\_idr\_rpl\_present\_flag** equal to 1 specifies that reference picture list syntax elements are present in slice headers of IDR pictures. sps\_idr\_rpl\_present\_flag equal to 0 specifies that reference picture list syntax elements are not present in slice headers of IDR pictures.

**sps\_rpl1\_same\_as\_rpl0\_flag** equal to 1 specifies that the syntax element sps\_num\_ref\_pic\_lists[ 1 ] and the syntax structure ref\_pic\_list\_struct( 1, rplsIdx ) are not present and the following applies:

– The value of sps\_num\_ref\_pic\_lists[ 1 ] is inferred to be equal to the value of sps\_num\_ref\_pic\_lists[ 0 ].

– The value of each of syntax elements in ref\_pic\_list\_struct( 1, rplsIdx ) is inferred to be equal to the value of corresponding syntax element in ref\_pic\_list\_struct( 0, rplsIdx ) for rplsIdx ranging from 0 to sps\_num\_ref\_pic\_lists[ 0 ] − 1.

**sps\_num\_ref\_pic\_lists**[ i ] specifies the number of the ref\_pic\_list\_struct( listIdx, rplsIdx ) syntax structures with listIdx equal to i included in the SPS. The value of sps\_num\_ref\_pic\_lists[ i ] shall be in the range of 0 to 64, inclusive.

NOTE 6 – For each value of listIdx (equal to 0 or 1), a decoder should allocate memory for a total number of sps\_num\_ref\_pic\_lists[ i ] + 1 ref\_pic\_list\_struct( listIdx, rplsIdx ) syntax structures since there may be one ref\_pic\_list\_struct( listIdx, rplsIdx ) syntax structure directly signalled in the slice headers of a current picture.

**sps\_ref\_wraparound\_enabled\_flag** equal to 1 specifies that horizontal wrap-around motion compensation is enabled and may be applied in inter prediction when decoding pictures in the CLVS. sps\_ref\_wraparound\_enabled\_flag equal to 0 specifies that horizontal wrap-around motion compensation is disabled and not applied in inter prediction when decoding pictures in the CLVS.

It is a requirement of bitstream conformance that, when there is one or more values of i in the range of 0 to sps\_num\_subpics\_minus1, inclusive, for which sps\_subpic\_treated\_as\_pic\_flag[ i ] is equal to 1 and sps\_subpic\_width\_minus1[ i ] plus 1 is not equal to ( sps\_pic\_width\_max\_in\_luma\_samples + CtbSizeY− 1 )  >>  CtbLog2SizeY ), the value of sps\_ref\_wraparound\_enabled\_flag shall be equal to 0.

**sps\_temporal\_mvp\_enabled\_flag** equal to 1 specifies that temporal motion vector predictors are enabled and may be used in decoding of pictures in the CLVS. sps\_temporal\_mvp\_enabled\_flag equal to 0 specifies that temporal motion vector predictors are disabled and not used in decoding of pictures in the CLVS.

**sps\_sbtmvp\_enabled\_flag** equal to 1 specifies that subblock-based temporal motion vector predictors are enabled and may 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 and may be used in motion vector coding in decoding of pictures in the CVLS. amvr\_enabled\_flag equal to 0 specifies that adaptive motion vector difference resolution is disabled and not used in motion vector coding in decoding of pictures in the CLVS.

**sps\_bdof\_enabled\_flag** equal to 0 specifies that the bi-directional optical flow inter prediction is disabled and not used in decoding of pictures in the CLVS. sps\_bdof\_enabled\_flag equal to 1 specifies that the bi-directional optical flow inter prediction is enabled and may be used in decoding of pictures in the CLVS.

**sps\_bdof\_control\_present\_in\_ph\_flag** equal to 1 specifies that ph\_bdof\_disabled\_flag is present in PHs referring to the SPS. sps\_bdof\_control\_present\_in\_ph\_flag equal to 0 specifies that ph\_bdof\_disabled\_flag is not present in PHs referring to the SPS. When sps\_bdof\_control\_present\_in\_ph\_flag is 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 may be used in motion vector decoding in decoding of pictures in the CLVS. sps\_smvd\_enabled\_flag equal to 0 specifies that symmetric motion vector difference is disabled and not used in motion vector coding in decoding of pictures in the CLVS.

**sps\_dmvr\_enabled\_flag** equal to 1 specifies that decoder motion vector refinement based inter bi-prediction is enabled and may be used in decoding of pictures in the CLVS. sps\_dmvr\_enabled\_flag equal to 0 specifies that decoder motion vector refinement based inter bi-prediction is disabled and not used in decoding of pictures in the CLVS.

**sps\_dmvr\_control\_present\_in\_ph\_flag** equal to 1 specifies that ph\_dmvr\_disabled\_flag is present in PHs referring to the SPS. sps\_dmvr\_control\_present\_in\_ph\_flag equal to 0 specifies that ph\_dmvr\_disabled\_flag is not present in PHs referring to the SPS. When sps\_dmvr\_control\_present\_in\_ph\_flag is 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 and may be used in decoding of pictures in the CLVS. sps\_mmvd\_enabled\_flag equal to 0 specifies that merge mode with motion vector difference is disabled and not used in in decoding of pictures in the CLVS.

**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:

MaxNumMergeCand = 6 − sps\_six\_minus\_max\_num\_merge\_cand (64)

**sps\_sbt\_enabled\_flag** equal to 0 specifies that subblock transform for inter-predicted CUs is disabled and not used in decoding of pictures in the CLVS. sps\_sbt\_enabled\_flag equal to 1 specifies that subblock transform for inter-predicteds CU is enabled and may be used in decoding of pictures in the CLVS.

**sps\_affine\_enabled\_flag** equal to 0 specifies that affine model based motion compensation is disabled and not used in decoding of pictures in the CLVS and inter\_affine\_flag and cu\_affine\_type\_flag are not present in the coding unit syntax of the CLVS. sps\_affine\_enabled\_flag equal to 1 specifies that affine model based motion compensation is enabled and may be used in decoding of pictures in the CLVS and inter\_affine\_flag and cu\_affine\_type\_flag may be 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\_affine\_type\_flag** specifies whether 6-parameter affine model based motion compensation can be used for inter prediction. If sps\_affine\_type\_flag is equal to 0, the syntax shall be constrained such that no 6-parameter affine model based motion compensation is used in the CLVS, and cu\_affine\_type\_flag is not present in coding unit syntax in the CLVS. Otherwise (sps\_affine\_type\_flag is equal to 1), 6-parameter affine model based motion compensation can be used in the CLVS. When not present, the value of sps\_affine\_type\_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 and may be used in motion vector coding of affine inter mode in decoding of pictures in the CLVS. sps\_affine\_amvr\_enabled\_flag equal to 0 specifies that adaptive motion vector difference resolution is disabled and not used in motion vector coding of affine inter mode in decoding of pictures in 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 0 specifies that the affine motion compensation refined with optical flow is disabled and not used in decoding of pictures in the CLVS. sps\_affine\_prof\_enabled\_flag equal to 1 specifies that the affine motion compensation refined with optical flow is enabled and may be used in decoding of pictures in 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 is present in PHs referring to the SPS. sps\_prof\_control\_present\_in\_ph\_flag equal to 0 specifies that ph\_prof\_disabled\_flag is not present in PHs 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 0 specifies that bi-prediction with CU weights is disabled and not used in decoding of picturs in the CLVS and bcw\_idx is not present in the coding unit syntax of the CLVS. sps\_bcw\_enabled\_flag equal to 1 specifies that bi-prediction with CU weights is enabled and may be used in decoding of pictures in the CLVS and bcw\_idx may be present in the coding unit syntax of the CLVS.

**sps\_ciip\_enabled\_flag** specifies that ciip\_flag may 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\_mmvd\_fullpel\_only\_flag** equal to 1 specifies that merge mode with motion vector difference uses integer sample precision. sps\_mmvd\_fullpel\_only\_flag equal to 0 specifies that merge mode with motion vector difference may use fractional sample precision. When not present, the value of sps\_mmvd\_fullpel\_only\_flag is inferred to be equal to 0.

**sps\_gpm\_enabled\_flag** equal to 0 specifies that geometric partition based motion compensation is disabled and not used in decoding of picturs in 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. sps\_gpm\_enabled\_flag equal to 1 specifies that geometric partition based motion compensation is enabled and may be used in decoding of pictures in the CLVS and merge\_gpm\_partition\_idx, merge\_gpm\_idx0, and merge\_gpm\_idx1 may be 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 (65)  
else if( sps\_gpm\_enabled\_flag && MaxNumMergeCand = = 2 )  
 MaxNumGpmMergeCand = 2  
else  
 MaxNumGpmMergeCand = 0

**sps\_log2\_parallel\_merge\_level\_minus2** plus 2 specifies the value of the variable Log2ParMrgLevel, which is used in the derivation process for spatial merging candidates as specified in clause 8.5.2.3, the derivation process for motion vectors and reference indices in subblock merge mode as specified in clause 8.5.5.2, and to control the invocation of the updating process for the history-based motion vector predictor list in clause 8.5.2.1. The value of sps\_log2\_parallel\_merge\_level\_minus2 shall be in the range of 0 to CtbLog2SizeY − 2, inclusive. The variable Log2ParMrgLevel is derived as follows:

Log2ParMrgLevel = sps\_log2\_parallel\_merge\_level\_minus2 + 2 (66)

**sps\_isp\_enabled\_flag** equal to 1 specifies that intra prediction with subpartitions is enabled and may be used in decoding of pictures in the CLVS. sps\_isp\_enabled\_flag equal to 0 specifies that intra prediction with subpartitions is disabled and not used in decoding of pictures in the CLVS.

**sps\_mrl\_enabled\_flag** equal to 1 specifies that intra prediction with multiple reference lines is enabled and may be used in decoding of pictures in the CLVS. sps\_mrl\_enabled\_flag equal to 0 specifies that intra prediction with multiple reference lines is disabled and not used in decoding of pictures in the CLVS.

**sps\_mip\_enabled\_flag** equal to 1 specifies that matrix-based intra prediction is enabled and may be used in decoding of pictures in the CLVS. sps\_mip\_enabled\_flag equal to 0 specifies that matrix-based intra prediction is disabled and not used in decoding of pictures in 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 and not used in decoding of pictures in the CLVS. sps\_cclm\_enabled\_flag equal to 1 specifies that the cross-component linear model intra prediction from luma component to chroma componenent is enabled and may be used in decoding of pictures in 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\_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 may 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 may 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\_palette\_enabled\_flag** equal to 1 specifies that pred\_mode\_plt\_flag may be present in the coding unit syntax of the CLVS. sps\_palette\_enabled\_flag equal to 0 specifies that pred\_mode\_plt\_flag is not present in the coding unit syntax of 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 adaptive colour transform is enabled and may be used in decoding of pictures in the CLVS and the cu\_act\_enabled\_flag may be present in the coding unit syntax of the CLVS. sps\_act\_enabled\_flag equal to 0 speifies that adaptive colour transform is disabled and not used in decoding of pictures in 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\_internal\_bit\_depth\_minus\_input\_bit\_depth** specifies the minimum allowed quantization parameter for transform skip mode as follows:

QpPrimeTsMin = 4 + 6 \* sps\_internal\_bit\_depth\_minus\_input\_bit\_depth (67)

The value of sps\_internal\_bit\_depth\_minus\_input\_bit\_depth 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 and may be used in decoding of pictures in the CLVS. sps\_ibc\_enabled\_flag equal to 0 specifies that the IBC prediction mode is disabled and not used in decoding of pictures in 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 (68)  
else  
 MaxNumIbcMergeCand = 0

**sps\_lmcs\_enabled\_flag** equal to 1 specifies that luma mapping with chroma scaling is enabled and may be used in decoding of pictures in the CLVS. sps\_lmcs\_enabled\_flag equal to 0 specifies that luma mapping with chroma scaling is disabled and not used in decoding of pictures in the CLVS.

**sps\_lfnst\_enabled\_flag** equal to 1 specifies that lfnst\_idx may 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\_ladf\_enabled\_flag** equal to 1 specifies that sps\_num\_ladf\_intervals\_minus2, sps\_ladf\_lowest\_interval\_qp\_offset, sps\_ladf\_qp\_offset[ i ], and sps\_ladf\_delta\_threshold\_minus1[ i ] are present in the SPS. sps\_ladf\_enabled\_flag equal to 0 specifies that sps\_num\_ladf\_intervals\_minus2, sps\_ladf\_lowest\_interval\_qp\_offset, sps\_ladf\_qp\_offset[ i ], and sps\_ladf\_delta\_threshold\_minus1[ i ] are not present in the SPS.

**sps\_num\_ladf\_intervals\_minus2** plus 2 specifies the number of sps\_ladf\_delta\_threshold\_minus1[ i ] and sps\_ladf\_qp\_offset[ i ] syntax elements that are present in the SPS. The value of sps\_num\_ladf\_intervals\_minus2 shall be in the range of 0 to 3, inclusive.

**sps\_ladf\_lowest\_interval\_qp\_offset** specifies the offset used to derive the variable qP as specified in clause 8.8.3.6.1. The value of sps\_ladf\_lowest\_interval\_qp\_offset shall be in the range of −63 to 63, inclusive.

**sps\_ladf\_qp\_offset**[ i ] specifies the offset array used to derive the variable qP as specified in clause 8.8.3.6.1. 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 2BitDepth − 3, inclusive.

The value of SpsLadfIntervalLowerBound[ 0 ] is set equal to 0.

For each value of i in the range of 0 to sps\_num\_ladf\_intervals\_minus2, inclusive, the variable SpsLadfIntervalLowerBound[ i + 1 ] is derived as follows:

SpsLadfIntervalLowerBound[ i + 1 ] = SpsLadfIntervalLowerBound[ i ] (69)  
 + sps\_ladf\_delta\_threshold\_minus1[ i ] + 1

**sps\_explicit\_scaling\_list\_enabled\_flag** equal to 1 specifies that the use of an explicit scaling list, which is signalled in a scaling list APS, in the scaling process for transform coefficients when decoding a slice is enabled for the CLVS. sps\_explicit\_scaling\_list\_enabled\_flag equal to 0 specifies that the use of an explicit scaling list in the scaling process for transform coefficients when decoding a slice is disabled for the CLVS.

**sps\_scaling\_matrix\_for\_lfnst\_disabled\_flag** equal to 1 specifies that scaling matrices are disabled and not applied to blocks coded with LFNST for the CLVS. sps\_scaling\_matrix\_for\_lfnst\_disabled\_flag equal to 0 specifies that the scaling matrices is enabled and may be applied to blocks coded with LFNST for the CLVS. When not present, the value of sps\_scaling\_matrix\_for\_lfnst\_disabled\_flag is inferred to be equal to 1 [Ed.(JC): The inferred values of sps\_scaling\_matrix\_for\_lfnst\_disabled\_flag and sps\_scaling\_matrix\_for\_alternative\_colour\_space\_disabled\_flag are not consistent, would be good to make them consistent].

**sps\_scaling\_matrix\_for\_alternative\_colour\_space\_disabled\_flag** equal to 1 specifies that for the CLVS scaling matrices are disabled and not applied to blocks when the colour space of the blocks is not equal to the designated colour space of scaling matrix. sps\_scaling\_matrix\_for\_alternative\_colour\_space\_disabled\_flag equal to 0 specifies that for the CLVS scaling matrices are enabled and may be applied to the blocks when the colour space of the blocks is equal to the designated colour space of scaling matrices. When not present, the value of sps\_scaling\_matrix\_for\_alternative\_colour\_sapce\_disabled\_flag is inferred to be equal to 0.

**sps\_scaling\_matrix\_designated\_colour\_space\_flag** equal to 1 specifies that the designated colour space of scaling matrices is the original colour space. sps\_scaling\_matrix\_designated\_colour\_space\_flag equal to 0 specifies that the designated colour space of scaling matrices is the transformed colour space. When not present, the value of sps\_scaling\_matrix\_designated\_colour\_space\_flag is inferred to be equal to 1.

**sps\_dep\_quant\_enabled\_flag** equal to 0 specifies that dependent quantization is disabled and not used for pictures referring to the SPS. sps\_dep\_quant\_enabled\_flag equal to 1 specifies that dependent quantization is enabled and may be used for pictures referring to the SPS.

**sps\_sign\_data\_hiding\_enabled\_flag** equal to 0 specifies that sign bit hiding is disabled and not used for pictures referring to the SPS. sps\_sign\_data\_hiding\_enabled\_flag equal to 1 specifies that sign bit hiding is enabled and may be used for pictures referring to the SPS. When sps\_sign\_data\_hiding\_enabled\_flag is not present, it is inferred to be equal to 0.

**sps\_virtual\_boundaries\_enabled\_flag** equal to 1 specifies that disabling in-loop filtering across virtual boundaries is enabled and may be applied in the coded pictures in the CLVS. sps\_virtual\_boundaries\_enabled\_flag equal to 0 specifies that disabling in-loop filtering across virtual boundaries is disabled and not applied in the coded pictures in 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.

It is a requirement of bitstream conformance that when the value of sps\_res\_change\_in\_clvs\_allowed\_flag is equal to 1, the value of sps\_virtual\_boundaries\_present\_flag shall be equal to 0.

**sps\_num\_ver\_virtual\_boundaries** specifies the number of sps\_virtual\_boundary\_pos\_x[ i ] syntax elements that are present in the SPS. When sps\_num\_ver\_virtual\_boundaries is not present, it is inferred to be equal to 0.

**sps\_virtual\_boundary\_pos\_x**[ i ] 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[ i ] shall be in the range of 1 to Ceil( sps\_pic\_width\_max\_in\_luma\_samples ÷ 8 ) − 1, inclusive.

**sps\_num\_hor\_virtual\_boundaries** specifies the number of sps\_virtual\_boundary\_pos\_y[ i ] syntax elements that are present in the SPS. 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**[ i ] 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[ i ] shall be in the range of 1 to Ceil( sps\_pic\_height\_max\_in\_luma\_samples ÷ 8 ) − 1, inclusive.

**sps\_general\_hrd\_params\_present\_flag** equal to 1 specifies that the SPS contains a general\_hrd\_parameters( ) syntax structure and an ols\_hrd\_parameters( ) syntax structure. sps\_general\_hrd\_params\_present\_flag equal to 0 specifies that the SPS does not contain a general\_hrd\_parameters( ) syntax structure or an ols\_hrd\_parameters( ) syntax structure.

**sps\_sublayer\_cpb\_params\_present\_flag** equal to 1 specifies that the ols\_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\_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\_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 general\_frame\_only\_constraint\_flag is equal to 1, the value of sps\_field\_seq\_flag shall be equal to 0.

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 7 – 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\_parameters( ) is present in the SPS RBSP syntax structure. sps\_vui\_parameters\_present\_flag equal to 0 specifies that the syntax structure vui\_parameters( ) is not present in the SPS RBSP syntax structure.

**sps\_extension\_flag** equal to 0 specifies that no sps\_extension\_data\_flag syntax elements are present in the SPS RBSP syntax structure. sps\_extension\_flag equal to 1 specifies that there are sps\_extension\_data\_flag syntax elements present in the SPS RBSP syntax structure.

**sps\_extension\_data\_flag** may have any value. Its presence and value do not affect decoder conformance to profiles specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore all sps\_extension\_data\_flag syntax elements.

#### Picture parameter set RBSP semantics

A PPS RBSP shall be available to the decoding process prior to it being referenced, included in at least one AU with TemporalId less than or equal to the TemporalId of the PPS NAL unit or provided through external means**.**

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.

**pps\_seq\_parameter\_set\_id** specifies the value of sps\_seq\_parameter\_set\_id for the SPS. The value of pps\_seq\_parameter\_set\_id shall be in the range of 0 to 15, inclusive. The value of pps\_seq\_parameter\_set\_id shall be the same in all PPSs that are referred to by coded pictures in a CLVS.

**pps\_mixed\_nalu\_types\_in\_pic\_flag** equal to 1 specifies that each picture referring to the PPS has more than one VCL NAL unit and the VCL NAL units do not have the same value of nal\_unit\_type. pps\_mixed\_nalu\_types\_in\_pic\_flag equal to 0 specifies that each picture referring to the PPS has one or more VCL NAL units and the VCL NAL units of each picture refering to the PPS have the same value of nal\_unit\_type.

When no\_mixed\_nalu\_types\_in\_pic\_constraint\_flag is equal to 1, the value of pps\_mixed\_nalu\_types\_in\_pic\_flag shall be equal to 0.

NOTE 1– 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 cannot have slices with nal\_unit\_type equal to IDR\_W\_RADL or IDR\_N\_LP.

**pps\_pic\_width\_in\_luma\_samples** specifies the width of each decoded picture referring to the PPS in units of luma samples. pps\_pic\_width\_in\_luma\_samples shall not be equal to 0, shall be an integer multiple of Max( 8, MinCbSizeY ), and shall be less than or equal to sps\_pic\_width\_max\_in\_luma\_samples.

When sps\_res\_change\_in\_clvs\_allowed\_flag equal to 0, the value of pps\_pic\_width\_in\_luma\_samples shall be equal to sps\_pic\_width\_max\_in\_luma\_samples.

When sps\_ref\_wraparound\_enabled\_flag is equal to 1, the value of ( CtbSizeY / MinCbSizeY + 1 ) shall be less than or equal to the value of ( pps\_pic\_width\_in\_luma\_samples / MinCbSizeY − 1 ).

**pps\_pic\_height\_in\_luma\_samples** specifies the height of each decoded picture referring to the PPS in units of luma samples. pps\_pic\_height\_in\_luma\_samples shall not be equal to 0 and shall be an integer multiple of Max( 8, MinCbSizeY ), and shall be less than or equal to sps\_pic\_height\_max\_in\_luma\_samples.

When sps\_res\_change\_in\_clvs\_allowed\_flag equal to 0, the value of pps\_pic\_height\_in\_luma\_samples shall be equal to sps\_pic\_height\_max\_in\_luma\_samples.

The variables PicWidthInCtbsY, PicHeightInCtbsY, PicSizeInCtbsY, PicWidthInMinCbsY, PicHeightInMinCbsY, PicSizeInMinCbsY, PicSizeInSamplesY, PicWidthInSamplesC and PicHeightInSamplesC are derived as follows:

PicWidthInCtbsY = Ceil( pps\_pic\_width\_in\_luma\_samples ÷ CtbSizeY ) (70)

PicHeightInCtbsY = Ceil( pps\_pic\_height\_in\_luma\_samples ÷ CtbSizeY ) (71)

PicSizeInCtbsY = PicWidthInCtbsY \* PicHeightInCtbsY (72)

PicWidthInMinCbsY = pps\_pic\_width\_in\_luma\_samples / MinCbSizeY (73)

PicHeightInMinCbsY = pps\_pic\_height\_in\_luma\_samples / MinCbSizeY (74)

PicSizeInMinCbsY = PicWidthInMinCbsY \* PicHeightInMinCbsY (75)

PicSizeInSamplesY = pps\_pic\_width\_in\_luma\_samples \* pps\_pic\_height\_in\_luma\_samples (76)

PicWidthInSamplesC = pps\_pic\_width\_in\_luma\_samples / SubWidthC (77)

PicHeightInSamplesC = pps\_pic\_height\_in\_luma\_samples / SubHeightC (78)

**pps\_conformance\_window\_flag** equal to 1 specifies that the conformance cropping window offset parameters follow next in the PPS. pps\_conformance\_window\_flag equal to 0 specifies that the conformance cropping window offset parameters are not present in the PPS.

When pps\_pic\_width\_in\_luma\_samples is equal to sps\_pic\_width\_max\_in\_luma\_samples and pps\_pic\_height\_in\_luma\_samples is equal to sps\_pic\_height\_max\_in\_luma\_samples, the value of pps\_conformance\_window\_flag shall be equal to 0.

**pps\_conf\_win\_left\_offset**, **pps\_conf\_win\_right\_offset**, **pps\_conf\_win\_top\_offset**, and **pps\_conf\_win\_bottom\_offset** specify the samples of the pictures in the CLVS that are output from the decoding process, in terms of a rectangular region specified in picture coordinates for output.

When pps\_conformance\_window\_flag is equal to 0, the following applies:

– If pps\_pic\_width\_in\_luma\_samples is equal to sps\_pic\_width\_max\_in\_luma\_samples and pps\_pic\_height\_in\_luma\_samples is equal to sps\_pic\_height\_max\_in\_luma\_samples, the values of pps\_conf\_win\_left\_offset, pps\_conf\_win\_right\_offset, pps\_conf\_win\_top\_offset, and pps\_conf\_win\_bottom\_offset are inferred to be equal to sps\_conf\_win\_left\_offset, sps\_conf\_win\_right\_offset, sps\_conf\_win\_top\_offset, and sps\_conf\_win\_bottom\_offset, respectively.

– Otherwise, the values of pps\_conf\_win\_left\_offset, pps\_conf\_win\_right\_offset, pps\_conf\_win\_top\_offset, and pps\_conf\_win\_bottom\_offset are inferred to be equal to 0.

The conformance cropping window contains the luma samples with horizontal picture coordinates from SubWidthC \* pps\_conf\_win\_left\_offset to pps\_pic\_width\_in\_luma\_samples − ( SubWidthC \* pps\_conf\_win\_right\_offset + 1 ) and vertical picture coordinates from SubHeightC \* pps\_conf\_win\_top\_offset to pps\_pic\_height\_in\_luma\_samples − ( SubHeightC \* pps\_conf\_win\_bottom\_offset + 1 ), inclusive.

The value of SubWidthC \* ( pps\_conf\_win\_left\_offset + pps\_conf\_win\_right\_offset ) shall be less than pps\_pic\_width\_in\_luma\_samples, and the value of SubHeightC \* ( pps\_conf\_win\_top\_offset + pps\_conf\_win\_bottom\_offset ) shall be less than pps\_pic\_height\_in\_luma\_samples.

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

NOTE 2 – 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 the 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.

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, it is a requirement of bitstream conformance that pps\_conf\_win\_left\_offset, pps\_conf\_win\_right\_offset, pps\_conf\_win\_top\_offset, and pps\_conf\_win\_bottom\_offset, are equal to sps\_conf\_win\_left\_offset, sps\_conf\_win\_right\_offset, sps\_conf\_win\_top\_offset, and sps\_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 value of SubWidthC \* ( Abs( pps\_scaling\_win\_left\_offset ) + Abs( pps\_scaling\_win\_right\_offset ) ) shall be less than pps\_pic\_width\_in\_luma\_samples, and the value of SubHeightC \* ( Abs( pps\_scaling\_win\_top\_offset ) + Abs( pps\_scaling\_win\_bottom\_offset ) ) shall be less than pps\_pic\_height\_in\_luma\_samples.

The variables CurrPicScalWinWidthL and CurrPicScalWinHeightL are derived as follows:

CurrPicScalWinWidthL = pps\_pic\_width\_in\_luma\_samples − (79)  
 SubWidthC \* ( pps\_scaling\_win\_right\_offset + pps\_scaling\_win\_left\_offset )

CurrPicScalWinHeightL = pps\_pic\_height\_in\_luma\_samples − (80) SubHeightC \* ( pps\_scaling\_win\_bottom\_offset + pps\_scaling\_win\_top\_offset )

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 is present in PHs 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 PHs 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 may be partitioned into more than one tile or slice.

It is a requirement of bitstream conformance that the value of pps\_no\_pic\_partition\_flag shall be the same for all PPSs that are referred to by coded pictures within a CLVS.

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 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 ] (81)  
 else  
 SubpicIdVal[ i ] = i

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

– For any two differenty 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 ].

– For each value of i in the range of 0 to sps\_num\_subpics\_minus1, inclusive, when the value of SubpicIdVal[ i ] of a current picture is not equal to the value of SubpicIdVal[ i ] of a reference picture, the active entries of the RPLs of the coded slices in the i-th subpicture of the current picture shall not include that reference picture.

**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 clause 6.5.1. The value of pps\_tile\_column\_width\_minus1[ i ] shall be in the range of 0 to PicWidthInCtbsY − 1, inclusive. When not present, the value of pps\_tile\_column\_width\_minus1[ 0 ] is inferred to be equal to PicWidthInCtbsY − 1.

**pps\_tile\_row\_height\_minus1**[ i ] plus 1 specifies the height of the i-th tile row in units of CTBs for i in the range of 0 to pps\_num\_exp\_tile\_rows\_minus1, inclusive. pps\_tile\_row\_height\_minus1[ pps\_num\_exp\_tile\_rows\_minus1 ] is also used to derive the heights of the tile rows with index greater than pps\_num\_exp\_tile\_rows\_minus1 as specified in clause 6.5.1. The value of pps\_tile\_row\_height\_minus1[ i ] shall be in the range of 0 to PicHeightInCtbsY − 1, inclusive. When not present, the value of pps\_tile\_row\_height\_minus1[ 0 ] is inferred to be equal to PicHeightInCtbsY − 1.

**pps\_loop\_filter\_across\_tiles\_enabled\_flag** equal to 1 specifies that in-loop filtering operations across tile boundaries are enabled and may be performed across tile boundaries in 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 and not performed across tile boundaries in pictures referring to the PPS. The in-loop filtering operations include the deblocking filter, sample adaptive offset filter, and adaptive loop filter operations. When not present, the value of pps\_loop\_filter\_across\_tiles\_enabled\_flag is inferred to be equal to 1.

**pps\_rect\_slice\_flag** equal to 0 specifies that tiles within each slice are in raster scan order and the slice information is not signalled in PPS. pps\_rect\_slice\_flag equal to 1 specifies that tiles within each slice cover a rectangular region of the picture and the slice information is signalled in the PPS. When not present, 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 may consist of one or more rectangular slices. When not present, the value of pps\_single\_slice\_per\_subpic\_flag is inferred to be equal to 1. [Ed. (GJS): Consider renaming this flag or clarifying in another manner to avoid an interpretation that this flag is only relevant when tehre are more than one subpictures in each 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 MaxSlicesPerPicture − 1, inclusive, where MaxSlicesPerPicture 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 may 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 NumTileColumns − 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 RowHeight[ 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 3 – 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 may or may not be split into multiple slices.

**pps\_exp\_slice\_height\_in\_ctus\_minus1**[ i ][ j ] plus 1 specifies the height of the j-th rectangular slice in the tile containing the i-th slice, in units of CTU rows, for j in the range of 0 to pps\_num\_exp\_slices\_in\_tile[ i ] − 1, inclusive, when pps\_num\_exp\_slices\_in\_tile[ i ] is greater than 0. pps\_exp\_slice\_height\_in\_ctus\_minus1[ i ][ pps\_num\_exp\_slices\_in\_tile[ i ] ] is also used to derive the heights of the rectangular slices in the tile containing the i-th slice with index greater than pps\_num\_exp\_slices\_in\_tile[ i ] − 1 as specified in clause 6.5.1. The value of pps\_exp\_slice\_height\_in\_ctus\_minus1[ i ][ j ] shall be in the range of 0 to RowHeight[ SliceTopLeftTileIdx[ i ] / NumTileColumns ] − 1, inclusive.

**pps\_tile\_idx\_delta\_val**[ i ] specifies the difference between the tile index of the tile containing the first CTU in the ( i + 1 )-th rectangular slice and the tile index of the tile containing the first CTU in the i-th rectangular slice. The value of pps\_tile\_idx\_delta\_val[ i ] shall be in the range of −NumTilesInPic + 1 to NumTilesInPic − 1, inclusive. When not present, the value of pps\_tile\_idx\_delta\_val[ i ] is inferred to be equal to 0. When present, the value of pps\_tile\_idx\_delta\_val[ i ] shall not be equal to 0.

When pps\_rect\_slice\_flag is equal to 1, it is a requirement of bitstrream conformance that, for any two slices, with picture-level slice indices idxA and idxB, that belong to the same picture and different subpictures, when SubpicIdxForSlice[ idxA ] is less than SubpicIdxForSlice[ idxB ], the value of idxA shall be less than idxB.

**pps\_loop\_filter\_across\_slices\_enabled\_flag** equal to 1 specifies that in-loop filtering operations across slice boundaries are enabled and may be performed across slice boundaries in 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 and not performed across slice boundaries in pictures referring to the PPS. The in-loop filtering operations include the deblocking filter, sample adaptive offset filter, and adaptive loop filter 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 ] may be present in the PH syntax structures or the slice headers for pictures referring to the PPS.

**pps\_init\_qp\_minus26** plus 26 specifies the initial value of SliceQpY for each slice referring to the PPS. The initial value of SliceQpY is modified at the picture level when a non-zero value of ph\_qp\_delta is decoded or at the slice level when a non-zero value of sh\_qp\_delta is decoded. The value of pps\_init\_qp\_minus26 shall be in the range of −( 26 + QpBdOffset ) to +37, inclusive.

**pps\_cu\_qp\_delta\_enabled\_flag** equal to 1 specifies that the ph\_cu\_qp\_delta\_subdiv\_intra\_slice and ph\_cu\_qp\_delta\_subdiv\_inter\_slice syntax elements are present in PHs referring to the PPS, and the cu\_qp\_delta\_abs and cu\_qp\_delta\_sign\_flag syntax elements may 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 PHs 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 tc and β offset syntax elements are present in the PHs 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 in the PPS RBSP syntax structure and the chroma deblocking tc and β offset syntax elements are not present in the PHs or the SHs of pictures referring to the PPS. When ChromaArrayType is equal to 0, the value of pps\_chroma\_tool\_offsets\_present\_flag shall be equal to 0.

**pps\_cb\_qp\_offset** and **pps\_cr\_qp\_offset** specify the offsets to the luma quantization parameter Qp′Y used for deriving Qp′Cb and Qp′Cr, respectively. The values of pps\_cb\_qp\_offset and pps\_cr\_qp\_offset shall be in the range of −12 to +12, inclusive. When ChromaArrayType 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 equalt to 0.

**pps\_joint\_cbcr\_qp\_offset\_present\_flag** equal to 1 specifies that pps\_joint\_cbcr\_qp\_offset\_value and pps\_joint\_cbcr\_qp\_offset\_list[ i ] are present in the PPS RBSP syntax structure. pps\_joint\_cbcr\_qp\_offset\_present\_flag equal to 0 specifies that pps\_joint\_cbcr\_qp\_offset\_value and pps\_joint\_cbcr\_qp\_offset\_list[ i ] are not present in the PPS RBSP syntax structure. When ChromaArrayType is equal to 0 or sps\_joint\_cbcr\_enabled\_flag is equal to 0, the value of pps\_joint\_cbcr\_qp\_offset\_present\_flag shall be equal to 0. When not present, the value of pps\_joint\_cbcr\_qp\_offset\_present\_flag is inferred to be equal to 0.

**pps\_joint\_cbcr\_qp\_offset\_value** specifies the offset to the luma quantization parameter Qp′Y used for deriving Qp′CbCr. The value of pps\_joint\_cbcr\_qp\_offset\_value shall be in the range of −12 to +12, inclusive. When ChromaArrayType is equal to 0 or sps\_joint\_cbcr\_enabled\_flag is equal to 0, pps\_joint\_cbcr\_qp\_offset\_value is not used in the decoding process and decoders shall ignore its value. When pps\_joint\_cbcr\_qp\_offset\_present\_flag is equal to 0, pps\_joint\_cbcr\_qp\_offset\_value is not present and is inferred to be equal to 0.

**pps\_slice\_chroma\_qp\_offsets\_present\_flag** equal to 1 specifies that the sh\_cb\_qp\_offset and sh\_cr\_qp\_offset syntax elements are present in the associated slice headers. pps\_slice\_chroma\_qp\_offsets\_present\_flag equal to 0 specifies that the sh\_cb\_qp\_offset and sh\_cr\_qp\_offset syntax elements are not present in the associated slice headers. When not present, the value of pps\_slice\_chroma\_qp\_offsets\_present\_flag is inferred to be equal to 0.

**pps\_cu\_chroma\_qp\_offset\_list\_enabled\_flag** equal to 1 specifies that the ph\_cu\_chroma\_qp\_offset\_subdiv\_intra\_slice and ph\_cu\_chroma\_qp\_offset\_subdiv\_inter\_slice syntax elements are present in PHs referring to the PPS and cu\_chroma\_qp\_offset\_flag may 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 PHs referring to the PPS and the cu\_chroma\_qp\_offset\_flag is not present in the transform unit syntax and the palette coding syntax. When not present, the value of pps\_cu\_chroma\_qp\_offset\_list\_enabled\_flag is inferred to be equal to 0.

**pps\_chroma\_qp\_offset\_list\_len\_minus1** plus 1 specifies the number of pps\_cb\_qp\_offset\_list[ i ], pps\_cr\_qp\_offset\_list[ i ], and pps\_joint\_cbcr\_qp\_offset\_list[ i ], syntax elements that are present in the PPS RBSP syntax structure. The value of pps\_chroma\_qp\_offset\_list\_len\_minus1 shall be in the range of 0 to 5, inclusive.

**pps\_cb\_qp\_offset\_list**[ i ], **pps\_cr\_qp\_offset\_list**[ i ], and **pps\_joint\_cbcr\_qp\_offset\_list**[ i ], specify offsets used in the derivation of Qp′Cb, Qp′Cr, and Qp′CbCr, respectively. The values of pps\_cb\_qp\_offset\_list[ i ], pps\_cr\_qp\_offset\_list[ i ], and pps\_joint\_cbcr\_qp\_offset\_list[ i ] shall be in the range of −12 to +12, inclusive. When pps\_joint\_cbcr\_qp\_offset\_present\_flag is equal to 0, pps\_joint\_cbcr\_qp\_offset\_list[ i ] is not present and it is inferred to be equal to 0.

**pps\_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\_deblocking\_filter\_control\_present\_flag** equal to 1 specifies the presence of deblocking filter control syntax elements in the PPS. pps\_deblocking\_filter\_control\_present\_flag equal to 0 specifies the absence of deblocking filter control syntax elements in the PPS and that the deblocking filter is applied for all slices referring to the PPS, using 0-valued deblocking β and tC offsets.

**pps\_deblocking\_filter\_override\_enabled\_flag** equal to 1 specifies the presence of ph\_deblocking\_filter\_override\_flag in the PHs referring to the PPS or sh\_deblocking\_filter\_override\_flag in the slice headers referring to the PPS. pps\_deblocking\_filter\_override\_enabled\_flag equal to 0 specifies the absence of ph\_deblocking\_filter\_override\_flag in PHs referring to the PPS or sh\_deblocking\_filter\_override\_flag in slice headers referring to the PPS. 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 operation of deblocking filter is not applied for slices referring to the PPS for which 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\_deblockig\_filter\_disabled\_flag or sh\_deblocking\_filter\_disabled\_flag is present and equal to 1, and also specifies that the operation of deblocking filter is applied for slices referring to the PPS for which 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.

pps\_deblocking\_filter\_disabled\_flag equal to 0 specifies that the operation of the deblocking filter is applied for slices referring to the PPS for which 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 also specifies that the operation of deblocking filter is not applied for slices referring to the PPS for which ph\_deblocking\_filter\_disabled\_flag or sh\_deblocking\_filter\_disabled\_flag is present and equal to 1.

When not present, the value of pps\_deblocking\_filter\_disabled\_flag is inferred to be equal to 0.

**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 may be present in slice headers referring to the PPS. When not present, the value of pps\_dbf\_info\_in\_ph\_flag is inferred to be equal to 0.

**pps\_luma\_beta\_offset\_div2** and **pps\_luma\_tc\_offset\_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the luma component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of pps\_luma\_beta\_offset\_div2 and pps\_luma\_tc\_offset\_div2 shall both be in the range of −12 to 12, inclusive. When not present, the values of pps\_luma\_beta\_offset\_div2 and pps\_luma\_tc\_offset\_div2 are both inferred to be equal to 0.

**pps\_cb\_beta\_offset\_div2** and **pps\_cb\_tc\_offset\_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of pps\_cb\_beta\_offset\_div2 and pps\_cb\_tc\_offset\_div2 shall both be in the range of −12 to 12, inclusive. When not present, the values of pps\_cb\_beta\_offset\_div2 and pps\_cb\_tc\_offset\_div2 are inferred to be equal to pps\_luma\_beta\_offset\_div2 and pps\_luma\_tc\_offset\_div2, respectively.

**pps\_cr\_beta\_offset\_div2** and **pps\_cr\_tc\_offset\_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of pps\_cr\_beta\_offset\_div2 and pps\_cr\_tc\_offset\_div2 shall both be in the range of −12 to 12, inclusive. When not present, the values of pps\_cr\_beta\_offset\_div2 and pps\_cr\_tc\_offset\_div2 are inferred to be equal to pps\_luma\_beta\_offset\_div2 and pps\_luma\_tc\_offset\_div2, respectively.

**pps\_rpl\_info\_in\_ph\_flag** equal to 1 specifies that reference picture list 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 reference picture list information is not present in the PH syntax structure and may 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 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\_sao\_info\_in\_ph\_flag equal to 0 specifies that SAO filter information is not present in the PH syntax structure and may 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 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\_alf\_info\_in\_ph\_flag equal to 0 specifies that ALF information is not present in the PH syntax structure and may 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 may 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 may 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\_ref\_wraparound\_enabled\_flag** equal to 1 specifies that horizontal wrap-around motion compensation is enabled and may be applied in inter prediction in decoding of pictures referring to the PPS. pps\_ref\_wraparound\_enabled\_flag equal to 0 specifies that horizontal wrap-around motion compensation is disabled and not applied in inter prediction in decoding of pictures referring to the PPS.

When sps\_ref\_wraparound\_enabled\_flag is equal to 0 or the value of CtbSizeY / MinCbSizeY + 1 is greater than pps\_pic\_width\_in\_luma\_samples / MinCbSizeY − 1, the value of pps\_ref\_wraparound\_enabled\_flag shall be equal to 0.

**pps\_pic\_width\_minus\_wraparound\_offset** specifies the difference between the picture width and the offset used for computing the horizontal wrap-around position in units of MinCbSizeY luma samples. The value of pps\_pic\_width\_minus\_wraparound\_offset shall be less than or equal to ( pps\_pic\_width\_in\_luma\_samples / MinCbSizeY ) − ( CtbSizeY / MinCbSizeY ) − 2.

The variable PpsRefWraparoundOffset is set equal to pps\_pic\_width\_in\_luma\_samples / MinCbSizeY − pps\_pic\_width\_minus\_wraparound\_offset.

**pps\_picture\_header\_extension\_present\_flag** equal to 0 specifies that no PH extension syntax elements are present in PHs referring to the PPS. pps\_picture\_header\_extension\_present\_flag equal to 1 specifies that PH extension syntax elements are present in PHs referring to the PPS. pps\_picture\_header\_extension\_present\_flag shall be equal to 0 in bitstreams conforming to this version of this Specification.

**pps\_slice\_header\_extension\_present\_flag** equal to 0 specifies that no slice header extension syntax elements are present in the slice headers for coded pictures referring to the PPS. pps\_slice\_header\_extension\_present\_flag equal to 1 specifies that slice header extension syntax elements are present in the slice headers for coded pictures referring to the PPS. pps\_slice\_header\_extension\_present\_flag shall be equal to 0 in bitstreams conforming to this version of this Specification.

**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 there are pps\_extension\_data\_flag syntax elements present in the PPS RBSP syntax structure.

**pps\_extension\_data\_flag** may have any value. Its presence and value do not affect decoder conformance to profiles specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore all pps\_extension\_data\_flag syntax elements.

#### Adaptation parameter set semantics

Each APS RBSP shall be available to the decoding process prior to it being referenced, included 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.

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.

**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.

**aps\_params\_type** specifies the type of APS parameters carried in the APS as specified in Table 6.

Table 6 – APS parameters type codes and types of APS parameters

|  |  |  |
| --- | --- | --- |
| **aps\_params\_type** | **Name of aps\_params\_type** | **Type of APS parameters** |
| 0 | ALF\_APS | ALF parameters |
| 1 | LMCS\_APS | LMCS parameters |
| 2 | SCALING\_APS | Scaling list parameters |
| 3..7 | Reserved | Reserved |

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. [Ed. (YK): With the above changed constraint that 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, shouldn't this sentence be changed to be as follows? All APS NAL units with a particular value of nal\_unit\_type and a particular value of aps\_params\_type, regardless of the nuh\_layer\_id values, 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.

NOTE 1 – 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) can be shared across pictures, and different slices within a picture can refer to different ALF APSs.

NOTE 2 – A suffix APS NAL unit associated with a particular VCL NAL unit (this VCL NAL unit precedes the suffix APS NAL unit in decoding order) is not for use by the particular VCL NAL unit, but for use by VCL NAL units following the suffix APS NAL unit in decoding order.

**aps\_chroma\_present\_flag** equal to 1 specifies that the APS NAL unit may 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 there are aps\_extension\_data\_flag syntax elements present in the APS RBSP syntax structure.

**aps\_extension\_data\_flag** may have any value. Its presence and value do not affect decoder conformance to profiles specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore all aps\_extension\_data\_flag syntax elements.

#### Picture header RBSP semantics

The PH RBSP contains a PH syntax structure, i.e., picture\_header\_structure( ).

#### Picture header structure semantics

The PH syntax structure contains information that is common for all slices of the coded picture associated with the PH syntax structure.

**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 may or may not be an IRAP picture.

**ph\_gdr\_pic\_flag** equal to 1 specifies the picture associated with the PH is a GDR picture. ph\_gdr\_pic\_flag equal to 0 specifies that the picture associated with the PH 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. [Ed. (YK): Should the semantics of ph\_gdr\_pic\_flag also be specified "one way"? That'd be consistent with the other flags herein, but then there is some issue for using it to condition the presence of ph\_recovery\_poc\_cnt.]

NOTE 1 – When ph\_gdr\_or\_irap\_pic\_flag is equal to 1 and ph\_gdr\_pic\_flag is equal to 0, the picture associated with the PH 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 may or may not be one or more coded slices in the picture that have sh\_slice\_type equal to 0 or 1. [Ed. (YK): Double check the need/correctness of the inference rules for those syntax elements conditioned out by this flag equal to 0.]

When 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 may or may 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. [Ed. (YK): Double check the need/correctness of the inference rules for those syntax elements conditioned out by this flag equal to 1.]

NOTE 2 – For bitstreams that are suppposed to work subpicure 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\_non\_reference\_picture\_flag** equal to 1 specifies the picture associated with the PH is never used as a reference picture. ph\_non\_reference\_picture\_flag equal to 0 specifies the picture associated with the PH may or may not be used as a reference picture.

**ph\_pic\_parameter\_set\_id** specifies the value of pps\_pic\_parameter\_set\_id for the PPS in use. The value of ph\_pic\_parameter\_set\_id shall be in the range of 0 to 63, inclusive.

It is a requirement of bitstream conformance that the value of TemporalId of the PH shall be greater than or equal to the value of TemporalId of the PPS that has pps\_pic\_parameter\_set\_id equal to ph\_pic\_parameter\_set\_id.

**ph\_pic\_order\_cnt\_lsb** specifies the picture order count modulo MaxPicOrderCntLsb for the current picture. The length of the ph\_pic\_order\_cnt\_lsb syntax element is sps\_log2\_max\_pic\_order\_cnt\_lsb\_minus4 + 4 bits. The value of the ph\_pic\_order\_cnt\_lsb shall be in the range of 0 to MaxPicOrderCntLsb − 1, inclusive.

**ph\_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, when present, the value of ph\_no\_output\_of\_prior\_pics\_flag shall be the same for all pictures in an AU.

When ph\_no\_output\_of\_prior\_pics\_flag is present in the PHs of the pictures in an AU, the ph\_no\_output\_of\_prior\_pics\_flag value of the AU is the ph\_no\_output\_of\_prior\_pics\_flag value of the pictures in the AU.

**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:

recoveryPointPocVal = PicOrderCntVal + ph\_recovery\_poc\_cnt (82)

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 ] may be equal to 1 or 0. Decoders conforming to this version of this Specification shall ignore the value of ph\_extra\_bit[ i ]. Its value does not affect decoder conformance to profiles specified in this version of specification.

**ph\_poc\_msb\_cycle\_present\_flag** equal to 1 specifies that the syntax element ph\_poc\_msb\_cycle\_val is present in the PH. 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. When vps\_independent\_layer\_flag[ GeneralLayerIdx[ nuh\_layer\_id ] ] is equal to 0 and there is a picture in the current AU in a reference layer of the current layer [Ed. (YK): Consider expressing this condition in a more explicit fashion.], the value of ph\_poc\_msb\_cycle\_present\_flag shall be equal to 0.

**ph\_poc\_msb\_cycle\_val** specifies the value of the POC MSB cycle of the current picture. The length of the syntax element ph\_poc\_msb\_cycle\_val is sps\_poc\_msb\_cycle\_len\_minus1 + 1 bits.

**ph\_alf\_enabled\_flag** equal to 1 specifies that adaptive loop filter is enabled and may be used for the current picture. ph\_alf\_enabled\_flag equal to 0 specifies that adaptive loop filter is disabled and not used for the current picture. When not present, 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 associated with the PH 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 associated with the PH 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 picture associated with the PH.
* When ChromaArrayType 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.

**ph\_alf\_cb\_flag** equal to 0 specifies that the adaptive loop filter is disabled and not applied to the Cb colour component for the current picture. ph\_alf\_cb\_flag equal to 1 specifies that the adaptive loop filter is enabled and may be applied to the Cb colour component for the current picture. When ph\_alf\_cb\_flag is not present, it is inferred to be equal to 0.

**ph\_alf\_cr\_flag** equal to 0 specifies that the adaptive loop filter is disabled and not applied to the Cr colour component for the current picture. ph\_alf\_cr\_flag equal to 1 specifies that the adaptive loop filter is enabled and may be applied to the Cr colour component for the current picture. When ph\_alf\_cr\_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 associated with the PH 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 picture associated with the PH.
* 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.

**ph\_cc\_alf\_cb\_enabled\_flag** equal to 1 specifies that cross-component adaptive loop filter for the Cb colour component is enabled and may be used for the current picture. ph\_cc\_alf\_cb\_enabled\_flag equal to 0 specifies that cross-component adaptive loop filter for the Cb colour component is disabled and not used for the current picture. When not present, ph\_cc\_alf\_cb\_enabled\_flag is inferred to be equal to 0.

**ph\_cc\_alf\_cb\_aps\_id** specifies the aps\_adaptation\_parameter\_set\_id of the ALF APS that the Cb colour component of the slices associated with the PH refers to.

When ph\_cc\_alf\_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\_cc\_alf\_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\_cc\_alf\_cb\_aps\_id shall be less than or equal to the TemporalId of the picture associated with the PH.

**ph\_cc\_alf\_cr\_enabled\_flag** equal to 1 specifies that cross-compoent adaptive loop filter for the Cr colour component is enabled and may be used for the current picture. ph\_cc\_alf\_cr\_enabled\_flag equal to 0 specifies that cross-component adaptive loop filter for the Cr colour component is disabled and not used for the current picture. When not present, ph\_cc\_alf\_cr\_enabled\_flag is inferred to be equal to 0.

**ph\_cc\_alf\_cr\_aps\_id** specifies the aps\_adaptation\_parameter\_set\_id of the ALF APS that the Cr colour component of the slices associated with the PH refers to.

When ph\_cc\_alf\_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\_cc\_alf\_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\_cc\_alf\_cr\_aps\_id shall be less than or equal to the TemporalId of the picture associated with the PH.

**ph\_lmcs\_enabled\_flag** equal to 1 specifies that luma mapping with chroma scaling is enabled and may be used for the current picture. ph\_lmcs\_enabled\_flag equal to 0 specifies that luma mapping with chroma scaling is disabled and not used 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 associated with the PH 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 ChromaArrayType 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 may 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 and may be used for the current picture in the scaling process for transform coefficients when decoding a slice. ph\_explicit\_scaling\_list\_enabled\_flag equal to 0 specifies that the explicit scaling list is disabled and not used 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 ChromaArrayType  = =  0 ? 0 : 1.

**ph\_virtual\_boundaries\_present\_flag** equal to 1 specifies that information of virtual boundaries is signalled in the PH. ph\_virtual\_boundaries\_present\_flag equal to 0 specifies that information of virtual boundaries is not signalled in the PH. When there is one or more than one virtual boundaries signalled in the PH, 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.

It is a requirement of bitstream conformance that, when sps\_subpic\_info\_present\_flag is equal to 1, the value of ph\_virtual\_boundaries\_present\_flag shall 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 (83)

**ph\_num\_ver\_virtual\_boundaries** specifies the number of ph\_virtual\_boundary\_pos\_x[ i ] syntax elements that are present in the PH. 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 (84)

**ph\_virtual\_boundary\_pos\_x**[ i ] 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[ i ] shall be in the range of 1 to Ceil( pps\_pic\_width\_in\_luma\_samples ÷ 8 ) − 1, 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[ i ] : ph\_virtual\_boundary\_pos\_x[ i ] ) \* 8 (85)

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[ i ] syntax elements that are present in the PH. 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 (86)

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**[ i ] 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[ i ] shall be in the range of 1 to Ceil( pps\_pic\_height\_in\_luma\_samples ÷ 8 ) − 1, 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[ i ] : ph\_virtual\_boundary\_pos\_y[ i ] ) \* 8 (87)

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.

NOTE 5 – There is no picture in the bitsteam that has ph\_non\_reference\_picture\_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. ph\_partition\_constraints\_override\_flag equal to 0 specifies that partition constraint parameters are not present in the PH. 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) associated with the PH. The value of ph\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_luma shall be in the range of 0 to Min( 6, CtbLog2SizeY ) − MinCbLog2SizeY, inclusive. When not present, the value of ph\_log2\_diff\_min\_qt\_min\_cb\_luma is inferred to be equal to sps\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_luma.

**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) associated with the PH. The value of ph\_max\_mtt\_hierarchy\_depth\_intra\_slice\_luma shall be in the range of 0 to 2\*( CtbLog2SizeY − MinCbLog2SizeY ), inclusive. When not present, the value of ph\_max\_mtt\_hierarchy\_depth\_intra\_slice\_luma is inferred to be equal to sps\_max\_mtt\_hierarchy\_depth\_intra\_slice\_luma.

**ph\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_luma** specifies the difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a luma coding block that can be split using a binary split and the 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) associated with the PH. The value of ph\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_luma shall be in the range of 0 to ( sps\_qtbtt\_dual\_tree\_intra\_flag ? Min( 6, CtbLog2SizeY ) : CtbLog2SizeY ) − MinQtLog2SizeIntraY, inclusive. When not present, the value of ph\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_luma is inferred to be equal to sps\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_luma.

**ph\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_luma** specifies the difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a luma coding block that can be split using a ternary split and the 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) associated with the PH. The value of ph\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_luma shall be in the range of 0 to Min( 6, CtbLog2SizeY ) − MinQtLog2SizeIntraY, inclusive. When not present, the value of ph\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_luma is inferred to be equal to sps\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_luma.

**ph\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_chroma** specifies the difference between the base 2 logarithm of the minimum size in luma samples of a chroma leaf block resulting from quadtree splitting of a chroma CTU with treeType equal to DUAL\_TREE\_CHROMA and the base 2 logarithm of the minimum coding block size in luma samples for chroma CUs with treeType equal to DUAL\_TREE\_CHROMA in slices with sh\_slice\_type equal to 2 (I) associated with the PH. The value of ph\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_chroma shall be in the range of 0 to Min( 6, CtbLog2SizeY ) − MinCbLog2SizeY, inclusive. When not present, the value of ph\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_chroma is inferred to be equal to sps\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_chroma.

**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) associated with the PH. The value of ph\_max\_mtt\_hierarchy\_depth\_intra\_slice\_chroma shall be in the range of 0 to 2\*( CtbLog2SizeY − MinCbLog2SizeY ), inclusive. When not present, the value of ph\_max\_mtt\_hierarchy\_depth\_intra\_slice\_chroma is inferred to be equal to sps\_max\_mtt\_hierarchy\_depth\_intra\_slice\_chroma.

**ph\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_chroma** specifies the difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a chroma coding block that can be split using a binary split and the 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) associated with the PH. The value of ph\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_chroma shall be in the range of 0 to Min( 6, CtbLog2SizeY ) − MinQtLog2SizeIntraC, inclusive. When not present, the value of ph\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_chroma is inferred to be equal to sps\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_chroma.

**ph\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_chroma** specifies the difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a chroma coding block that can be split using a ternary split and the 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) associated with the PH. The value of ph\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_chroma shall be in the range of 0 to Min( 6, CtbLog2SizeY ) − MinQtLog2SizeIntraC, inclusive. When not present, the value of ph\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_chroma is inferred to be equal to sps\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_chroma

**ph\_cu\_qp\_delta\_subdiv\_intra\_slice** specifies the maximum cbSubdiv value of coding units in intra slice that convey cu\_qp\_delta\_abs and cu\_qp\_delta\_sign\_flag. The value of ph\_cu\_qp\_delta\_subdiv\_intra\_slice shall be in the range of 0 to 2 \* ( CtbLog2SizeY − MinQtLog2SizeIntraY + ph\_max\_mtt\_hierarchy\_depth\_intra\_slice\_luma ), inclusive.

When not present, the value of ph\_cu\_qp\_delta\_subdiv\_intra\_slice is inferred to be equal to 0.

**ph\_cu\_chroma\_qp\_offset\_subdiv\_intra\_slice** specifies the maximum cbSubdiv value of coding units in intra slice that convey cu\_chroma\_qp\_offset\_flag. The value of ph\_cu\_chroma\_qp\_offset\_subdiv\_intra\_slice shall be in the range of 0 to 2 \* ( CtbLog2SizeY − MinQtLog2SizeIntraY + ph\_max\_mtt\_hierarchy\_depth\_intra\_slice\_luma ), inclusive.

When not present, the value of ph\_cu\_chroma\_qp\_offset\_subdiv\_intra\_slice is inferred to be equal to 0.

**ph\_log2\_diff\_min\_qt\_min\_cb\_inter\_slice** specifies the difference between the base 2 logarithm of the minimum size in luma samples of a luma leaf block resulting from quadtree splitting of a CTU and the base 2 logarithm of the minimum luma coding block size in luma samples for luma CUs in the slices with sh\_slice\_type equal to 0 (B) or 1 (P) associated with the PH. The value of ph\_log2\_diff\_min\_qt\_min\_cb\_inter\_slice shall be in the range of 0 to Min( 6, CtbLog2SizeY ) − MinCbLog2SizeY, inclusive. When not present, the value of ph\_log2\_diff\_min\_qt\_min\_cb\_luma is inferred to be equal to sps\_log2\_diff\_min\_qt\_min\_cb\_inter\_slice.

**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) associated with the PH. The value of ph\_max\_mtt\_hierarchy\_depth\_inter\_slice shall be in the range of 0 to 2\*( CtbLog2SizeY − MinCbLog2SizeY ), inclusive. When not present, the value of ph\_max\_mtt\_hierarchy\_depth\_inter\_slice is inferred to be equal to sps\_max\_mtt\_hierarchy\_depth\_inter\_slice.

**ph\_log2\_diff\_max\_bt\_min\_qt\_inter\_slice** specifies the difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a luma coding block that can be split using a binary split and the 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) associated with the PH. The value of ph\_log2\_diff\_max\_bt\_min\_qt\_inter\_slice shall be in the range of 0 to CtbLog2SizeY − MinQtLog2SizeInterY, inclusive. When not present, the value of ph\_log2\_diff\_max\_bt\_min\_qt\_inter\_slice is inferred to be equal to sps\_log2\_diff\_max\_bt\_min\_qt\_inter\_slice.

**ph\_log2\_diff\_max\_tt\_min\_qt\_inter\_slice** specifies the difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a luma coding block that can be split using a ternary split and the 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) associated with the PH. The value of ph\_log2\_diff\_max\_tt\_min\_qt\_inter\_slice shall be in the range of 0 to Min( 6, CtbLog2SizeY ) − MinQtLog2SizeInterY, inclusive. When not present, the value of ph\_log2\_diff\_max\_tt\_min\_qt\_inter\_slice is inferred to be equal to sps\_log2\_diff\_max\_tt\_min\_qt\_inter\_slice.

**ph\_cu\_qp\_delta\_subdiv\_inter\_slice** specifies the maximum cbSubdiv value of coding units that in inter slice convey cu\_qp\_delta\_abs and cu\_qp\_delta\_sign\_flag. The value of ph\_cu\_qp\_delta\_subdiv\_inter\_slice shall be in the range of 0 to 2 \* ( CtbLog2SizeY − MinQtLog2SizeInterY + ph\_max\_mtt\_hierarchy\_depth\_inter\_slice ), inclusive.

When not present, the value of ph\_cu\_qp\_delta\_subdiv\_inter\_slice is inferred to be equal to 0.

**ph\_cu\_chroma\_qp\_offset\_subdiv\_inter\_slice** specifies the maximum cbSubdiv value of coding units in inter slice that convey cu\_chroma\_qp\_offset\_flag. The value of ph\_cu\_chroma\_qp\_offset\_subdiv\_inter\_slice shall be in the range of 0 to 2 \* ( CtbLog2SizeY − MinQtLog2SizeInterY + ph\_max\_mtt\_hierarchy\_depth\_inter\_slice ), inclusive.

When not present, the value of ph\_cu\_chroma\_qp\_offset\_subdiv\_inter\_slice is inferred to be equal to 0.

**ph\_temporal\_mvp\_enabled\_flag** equal to 0 specifies that temporal motion vector predictor is disabled and not used in decoding of the slices in the current picture. ph\_temporal\_mvp\_enabled\_flag equal to 1 specifies that temporal motion vector predictors is enabled and may be used in decoding of the slices in 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 can 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 can only be equal to 0 that are not listed.

The maximum number of subblock-based merging MVP candidates, MaxNumSubblockMergeCand, is derived as follows:

if( sps\_affine\_enabled\_flag )  
 MaxNumSubblockMergeCand = 5 − sps\_five\_minus\_max\_num\_subblock\_merge\_cand (88)  
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 reference picture list 0. ph\_collocated\_from\_l0\_flag equal to 0 specifies that the collocated picture used for temporal motion vector prediction is derived from reference picture list 1. When ph\_temporal\_mvp\_enabled\_flag and pps\_rpl\_info\_in\_ph\_flag are both equal to 1 and num\_ref\_entries[ 1 ][ RplsIdx[ 1 ] ] is equal to 0, the value of ph\_collocated\_from\_l0\_flag is inferred to be equal to 1.

**ph\_collocated\_ref\_idx** specifies the reference index of the collocated picture used for temporal motion vector prediction.

When ph\_collocated\_from\_l0\_flag is equal to 1, ph\_collocated\_ref\_idx refers to an entry in reference picture list 0, and the value of ph\_collocated\_ref\_idx shall be in the range of 0 to num\_ref\_entries[ 0 ][ RplsIdx[ 0 ] ] − 1, inclusive.

When ph\_collocated\_from\_l0\_flag is equal to 0, ph\_collocated\_ref\_idx refers to an entry in reference picture list 1, and the value of ph\_collocated\_ref\_idx shall be in the range of 0 to num\_ref\_entries[ 1 ][ RplsIdx[ 1 ] ] − 1, inclusive.

When not present, the value of ph\_collocated\_ref\_idx is inferred to be equal to 0.

**ph\_mmvd\_fullpel\_only\_flag** equal to 1 specifies that merge mode with motion vector difference uses integer sample precision in the slices associated with the PH. ph\_mmvd\_fullpel\_only\_flag equal to 0 specifies that merge mode with motion vector difference may use fractional sample precision in the slices associated with the PH. When not present, the value of ph\_mmvd\_fullpel\_only\_flag is inferred to be 0.

**ph\_mvd\_l1\_zero\_flag** equal to 1 specifies that the mvd\_coding( x0, y0, 1, cpIdx ) syntax structure is not parsed and MvdL1[ x0 ][ y0 ][ compIdx ] and MvdCpL1[ x0 ][ y0 ][ cpIdx ][ compIdx ] are set equal to 0 for compIdx = 0..1 and cpIdx = 0..2. ph\_mvd\_l1\_zero\_flag equal to 0 specifies that the mvd\_coding( x0, y0, 1, cpIdx ) syntax structure is parsed. When not present, the value of ph\_mvd\_l1\_zero\_flag is inferred to be 1.

**ph\_bdof\_disabled\_flag** equal to 1 specifies that bi-directional optical flow inter prediction based inter bi-prediction is disabled and not used in the slices associated with the PH. ph\_bdof\_disabled\_flag equal to 0 specifies that bi-directional optical flow inter prediction based inter bi-prediction is enabled and may be used in the slices associated with the PH.

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 decoder motion vector refinement based inter bi-prediction is disabled and not used in the slices associated with the PH. ph\_dmvr\_disabled\_flag equal to 0 specifies that decoder motion vector refinement based inter bi-prediction is enabled and may be used in the slices associated with the PH.

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 and not used in the slices associated with the PH. ph\_prof\_disabled\_flag equal to 0 specifies that prediction refinement with optical flow is enabled and may be used in the slices associated with the PH.

When ph\_prof\_disabled\_flag is not present, the following applies:

* 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 QpY 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 QpY quantization parameter for all slices of the picture, SliceQpY, is derived as follows:

SliceQpY = 26 + pps\_init\_qp\_minus26 + ph\_qp\_delta (89)

The value of SliceQpY shall be in the range of −QpBdOffset to +63, inclusive.

**ph\_joint\_cbcr\_sign\_flag** specifies whether, in transform units with tu\_joint\_cbcr\_residual\_flag[ x0 ][ y0 ] equal to 1, the collocated residual samples of both chroma components have inverted signs. When tu\_joint\_cbcr\_residual\_flag[ x0 ][ y0 ] equal to 1 for a transform unit, ph\_joint\_cbcr\_sign\_flag equal to 0 specifies that the sign of each residual sample of the Cr (or Cb) component is identical to the sign of the collocated Cb (or Cr) residual sample and ph\_joint\_cbcr\_sign\_flag equal to 1 specifies that the sign of each residual sample of the Cr (or Cb) component is given by the inverted sign of the collocated Cb (or Cr) residual sample.

**ph\_sao\_luma\_enabled\_flag** equal to 1 specifies that SAO is enabled and may be used for the luma component of the current picture. ph\_sao\_luma\_enabled\_flag equal to 0 specifies that SAO is disabled and not used 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.

**ph\_sao\_chroma\_enabled\_flag** equal to 1 specifies that SAO is enabled and may be used for the chroma component of the current picture. ph\_sao\_chroma\_enabled\_flag equal to 0 specifies that SAO is disabled and not used 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\_filter\_override\_flag** equal to 1 specifies that deblocking parameters are present in the PH. ph\_deblocking\_filter\_override\_flag equal to 0 specifies that deblocking parameters are not present in the PH. When not present, the value of ph\_deblocking\_filter\_override\_flag is inferred to be equal to 0.

**ph\_deblocking\_filter\_disabled\_flag** equal to 1 specifies that the operation of the deblocking filter is not applied for the slices associated with the PH 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 also specifies that the operation of the deblocking filter is applied for the slices associated with the PH 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.

ph\_deblocking\_filter\_disabled\_flag equal to 0 specifies that the operation of the deblocking filter is applied for the slices associated with the PH 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 also specifies that the operation of the deblocking filter is not applied for the slices associated with the PH 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.

[Ed. (YK): It's probably better to replace the above two paragraphs with the following:

When pps\_dbf\_info\_in\_ph\_flag is equal to 1, **ph\_deblocking\_filter\_disabled\_flag** equal to 1 specifies that the operation of the deblocking filter is not applied for the slices associated with the PH, and ph\_deblocking\_filter\_disabled\_flag equal to 0 specifies that the operation of the deblocking filter is applied for the slices associated with the PH.]

When ph\_deblocking\_filter\_disabled\_flag is not present, it is inferred as follows:

* If pps\_deblocking\_filter\_disabled\_flag and ph\_deblocking\_filter\_override\_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\_filter\_override\_flag is equal to 0), the value of ph\_deblocking\_filter\_disabled\_flag is inferred to be equal to pps\_deblocking\_filter\_disabled\_flag.

[Ed. Consider the naming of the syntax elements for deblocking parameter override in PH and SH to improve clarity.]

**ph\_luma\_beta\_offset\_div2** and **ph\_luma\_tc\_offset\_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the luma component for the slices associated with the PH. The values of ph\_luma\_beta\_offset\_div2 and ph\_luma\_tc\_offset\_div2 shall both be in the range of −12 to 12, inclusive. When not present, the values of ph\_luma\_beta\_offset\_div2 and ph\_luma\_tc\_offset\_div2 are inferred to be equal to pps\_luma\_beta\_offset\_div2 and pps\_luma\_tc\_offset\_div2, respectively.

**ph\_cb\_beta\_offset\_div2** and **ph\_cb\_tc\_offset\_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for the slices associated with the PH. 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 to be equal to ph\_luma\_beta\_offset\_div2 and ph\_luma\_tc\_offset\_div2, respectively.

**ph\_cr\_beta\_offset\_div2** and **ph\_cr\_tc\_offset\_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for the slices associated with the PH. 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 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. The value of ph\_extension\_length shall be in the range of 0 to 256, inclusive. When not present, the value of ph\_extension\_length is inferred to be equal to 0.

**ph\_extension\_data\_byte** may have any value. Decoders conforming to this version of this Specification shall ignore the value of ph\_extension\_data\_byte. Its value does not affect decoder conformance to profiles specified in this version of specification.[Ed. (RSS): When profiles have been defined in Annex A, consider changing the last sentence to "Its value does not affect decoder conformance to profiles specified in Annex A".]

#### 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.

#### 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\_au\_flag** equal to 1 specifies that the AU containing the AU delimiter is an IRAP or GDR AU. aud\_irap\_or\_gdr\_au\_flag equal to 0 specifies that the AU containing the AU delimiter is not an IRAP or GDR AU.

**aud\_pic\_type** indicates that the sh\_slice\_type values for all slices of the coded pictures in the AU containing the AU delimiter NAL unit are members of the set listed in Table 7 for the given value of aud\_pic\_type. The value of aud\_pic\_type shall be equal to 0, 1 or 2 in bitstreams conforming to this version of this Specification. Other values of aud\_pic\_type are reserved for future use by ITU‑T | ISO/IEC. Decoders conforming to this version of this Specification shall ignore reserved values of aud\_pic\_type.

Table 7 – Interpretation of aud\_pic\_type

|  |  |
| --- | --- |
| **aud\_pic\_type** | **sh\_slice\_type values that may be present in the AU** |
| 0 | I |
| 1 | P, I |
| 2 | B, P, I |

#### End of sequence RBSP semantics

When present, the EOS RBSP specifies that the current PU is the last PU in the CLVS in decoding order and the next subsequent PU 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.

#### 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.

#### 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.

#### Slice layer RBSP semantics

The slice layer RBSP consists of a slice header and slice data.

#### RBSP slice trailing bits semantics

**cabac\_zero\_word** is a byte-aligned sequence of two bytes equal to 0x0000.

Let the variable NumBytesInPicVclNalUnits be the sum of the values of NumBytesInNalUnit for all VCL NAL units of a coded picture.

Let the variable BinCountsInPicNalUnits be the number of times that the parsing process function DecodeBin( ), specified in clause 9.3.4.3.1, is invoked to decode the contents of all VCL NAL units of a coded picture.

Let the variable RawMinCuBits be derived as follows:

RawMinCuBits = MinCbSizeY \* MinCbSizeY \*  
 ( BitDepth + 2 \* BitDepth / ( SubWidthC \* SubHeightC ) ) (90)

Let the variable vclByteScaleFactor be derived to be equal to ( 32 + 4 \* general\_tier\_flag ) ÷ 3.

The value of BinCountsInPicNalUnits shall be less than or equal to vclByteScaleFactor \* NumBytesInPicVclNalUnits + ( RawMinCuBits \* PicSizeInMinCbsY ) ÷ 32.

NOTE – The constraint on the maximum number of bins resulting from decoding the contents of the coded slice NAL units can be met by inserting a number of cabac\_zero\_word syntax elements to increase the value of NumBytesInPicVclNalUnits. Each cabac\_zero\_word is represented in a NAL unit by the three-byte sequence 0x000003 (as a result of the constraints on NAL unit contents that result in requiring inclusion of an emulation\_prevention\_three\_byte for each cabac\_zero\_word).

Let the variable NumBytesInSubpicVclNalUnits be the sum of the values NumBytesInNalUnit for all VCL NAL units of a subpicture with subpicture index subpicIdxA.

Let the variable BinCountsInSubpicNalUnits be the number of times that the parsing process function DecodeBin( ), specified in clause clause 9.3.4.3.1, is invoked to decode the contents of all VCL NAL units of a subpicture with subpicture index subpicIdxA.

The variable subpicSizeInMinCbsY for the subpicture with subpicture index subpicIdxA is derived to be equal to ( ( sps\_subpic\_width\_minus1[ subpicIdxA ] + 1) \* CtbSizeY / MinCbSizeY \* ( sps\_subpic\_height\_minus1[ subpicIdxA ] + 1) \* CtbSizeY / MinCbSizeY ).

For each subpicture with subpicture index subpicIdxA for which sps\_subpic\_treated\_as\_pic\_flag[ subpicIdxA ] is equal to 1, the value of BinCountsInSubpicNalUnits shall be less than or equal to vclByteScaleFactor \* NumBytesInSubpicVclNalUnits + ( RawMinCuBits \* subpicSizeInMinCbsY ) ÷ 32.

#### RBSP trailing bits semantics

**rbsp\_stop\_one\_bit** shall be equal to 1.

**rbsp\_alignment\_zero\_bit** shall be equal to 0.

#### Byte alignment semantics

**alignment\_bit\_equal\_to\_one** shall be equal to 1.

**alignment\_bit\_equal\_to\_zero** shall be equal to 0.

#### Extra picture header bits structure semantics

**extra\_ph\_bit\_present\_flag**[ i ] equal to 1 specifies that the i-th extra bit is present in the PH syntax structures referring to the SPS. extra\_ph\_bit\_present\_flag[ i ] equal to 0 specifies that the i-th extra bit is not present in the PH syntax structures referring to the SPS.

The variable NumExtraPhBits is derived as follows:

NumExtraPhBits = 0  
for( i = 0; i < ( sps\_num\_extra\_ph\_bits\_bytes \* 8 ); i++ )  
 if( extra\_ph\_bit\_present\_flag[ i ] ) (91)  
 NumExtraPhBits++

#### Extra slice header bits structure semantics

**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. 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\_bits\_bytes \* 8 ); i++ )  
 if( extra\_sh\_bit\_present\_flag[ i ] ) (92)  
 NumExtraShBits++

#### 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 on luma component. alf\_luma\_clip\_flag equal to 1 specifies that non-linear adaptive loop filtering may be applied on luma component.

**alf\_luma\_num\_filters\_signalled\_minus1** plus 1 specifies the number of adpative loop filter classes for which luma coefficients can be signalled. The value of alf\_luma\_num\_filters\_signalled\_minus1 shall be in the range of 0 to NumAlfFilters − 1, inclusive.

**alf\_luma\_coeff\_delta\_idx**[ filtIdx ] specifies the indices of the signalled adaptive loop filter luma coefficient deltas for the filter class indicated by filtIdx ranging from 0 to NumAlfFilters − 1. When alf\_luma\_coeff\_delta\_idx[ filtIdx ] is not present, it is inferred to be equal to 0. The length of alf\_luma\_coeff\_delta\_idx[ filtIdx ] is Ceil( Log2( alf\_luma\_num\_filters\_signalled\_minus1 + 1 ) ) bits. The value of alf\_luma\_coeff\_delta\_idx[ filtIdx ] shall be in the range of 0 to alf\_luma\_num\_filters\_signalled\_minus1, inclusive.

**alf\_luma\_coeff\_abs**[ sfIdx ][ j ] specifies the absolute value of the j-th coefficient of the signalled luma filter indicated by sfIdx. When alf\_luma\_coeff\_abs[ sfIdx ][ j ] is not present, it is inferred to be equal 0. The value of alf\_luma\_coeff\_abs[ sfIdx ][ j ] shall be in the range of 0 to 128, inclusive.

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

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

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

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

filtCoeff[ sfIdx ][ j ] = alf\_luma\_coeff\_abs[ sfIdx ][ j ] \* (93)  
 ( 1 − 2 \* alf\_luma\_coeff\_sign[ sfIdx ][ j ] )

The luma filter coefficients AlfCoeffL[ aps\_adaptation\_parameter\_set\_id ] with elements AlfCoeffL[ aps\_adaptation\_parameter\_set\_id ][ filtIdx ][ j ], with filtIdx = 0..NumAlfFilters − 1 and j = 0..11 are derived as follows:

AlfCoeffL[ aps\_adaptation\_parameter\_set\_id ][ filtIdx ][ j ] = filtCoeff[ alf\_luma\_coeff\_delta\_idx[ filtIdx ] ][ j ] (94)

The fixed filter coefficients AlfFixFiltCoeff[ i ][ j ] with i = 0..64, j = 0..11 and the class to filter mapping AlfClassToFiltMap[ m ][ n ] with m = 0..15 and n = 0..24 are derived as follows:

AlfFixFiltCoeff = (95)

{

{ 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 = (96)

{

{ 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 AlfCoeffL[ aps\_adaptation\_parameter\_set\_id ][ filtIdx ][ j ] with filtIdx = 0..NumAlfFilters − 1, j = 0..11 shall be in the range of −27 to 27 − 1, inclusive.

**alf\_luma\_clip\_idx**[ sfIdx ][ j ] specifies the clipping index of the clipping value to use before multiplying by the j-th coefficient of the signalled luma filter indicated by sfIdx. It is a requirement of bitstream conformance that the values of alf\_luma\_clip\_idx[ sfIdx ][ j ] with sfIdx = 0..alf\_luma\_num\_filters\_signalled\_minus1 and j = 0..11 shall be in the range of 0 to 3, inclusive.

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

**alf\_chroma\_clip\_flag** equal to 0 specifies that linear adaptive loop filtering is applied on chroma components; alf\_chroma\_clip\_flag equal to 1 specifies that non-linear adaptive loop filtering is applied on chroma components. When not present, alf\_chroma\_clip\_flag is inferred to be equal to 0.

**alf\_chroma\_num\_alt\_filters\_minus1** plus 1 specifies the number of alternative filters for chroma components. The value of alf\_chroma\_num\_alt\_filters\_minus1 shall be in the range of 0 to 7, inclusive.

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

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

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

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

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

AlfCoeffC[ aps\_adaptation\_parameter\_set\_id ][ altIdx ][ j ] = alf\_chroma\_coeff\_abs[ altIdx ][ j ] \* (97)  
 ( 1 − 2 \* alf\_chroma\_coeff\_sign[ altIdx ][ j ] )

It is a requirement of bitstream conformance that the values of AlfCoeffC[ aps\_adaptation\_parameter\_set\_id ][ altIdx ][ j ] with altIdx = 0..alf\_chroma\_num\_alt\_filters\_minus1, j = 0..5 shall be in the range of −27 to 27 − 1, inclusive.

**alf\_chroma\_clip\_idx**[ altIdx ][ j ] specifies the clipping index of the clipping value to use before multiplying by the j-th coefficient of the alternative chroma filter with index altIdx. It is a requirement of bitstream conformance that the values of alf\_chroma\_clip\_idx[ altIdx ][ j ] with altIdx = 0..alf\_chroma\_num\_alt\_filters\_minus1, j = 0..5 shall be in the range of 0 to 3, inclusive.

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

Table 8 – Specification AlfClip depending on BitDepth and clipIdx

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **BitDepth** | **clipIdx** | | | |
| **0** | **1** | **2** | **3** |
| **8** | 28 | 25 | 23 | 21 |
| **9** | 29 | 26 | 24 | 22 |
| **10** | 210 | 27 | 25 | 23 |
| **11** | 211 | 28 | 26 | 24 |
| **12** | 212 | 29 | 27 | 25 |
| **13** | 213 | 210 | 28 | 26 |
| **14** | 214 | 211 | 29 | 27 |
| **15** | 215 | 212 | 210 | 28 |
| **16** | 216 | 213 | 211 | 29 |

**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 \* alf\_cc\_cb\_coeff\_sign[ k ][ j ] ) \* 2alf\_cc\_cb\_mapped\_coeff\_abs[ k ][ j ] − 1.

**alf\_cc\_cr\_filters\_signalled\_minus1** plus 1 specifies the number of cross-component filters for the Cr colour component signalled in the current ALF APS. The value of alf\_cc\_cr\_filters\_signalled\_minus1 shall be in the range of 0 to 3, inclusive.

**alf\_cc\_cr\_mapped coeff\_abs**[ k ][ j ]specifies the absolute value of the j-th mapped coefficient of the signalled k-th cross-component filter for the Cr colour component. When alf\_cc\_cr\_mapped coeff\_abs[ k ][ j ] is not present, it is inferred to be equal to 0.

**alf\_cc\_cr\_coeff\_sign[ k ][ j ]** specifies the sign of the j-th coefficient of the signalled k-th cross-component filter for the Cr colour component as follows:

* If alf\_cc\_cr\_coeff\_sign[ k ][ j ] is equal to 0, the corresponding cross-component filter coefficient has a positive value.
* Otherwise (alf\_cc\_cr\_sign[ k ][ j ] is equal to 1), the corresponding cross-component filter coefficient has a negative value.

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

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

* If alf\_cc\_cr\_mapped\_coeff\_abs[ k ][ j ] is equal to 0, CcAlfApsCoeffCr[ aps\_adaptation\_parameter\_set\_id ][ k ][ j ] is set equal to 0.
* Otherwise, CcAlfApsCoeffCr[ aps\_adaptation\_parameter\_set\_id ][ k ][ j ] is set equal to ( 1 − 2 \* alf\_cc\_cr\_coeff\_sign[ k ][ j ] ) \* 2alf\_cc\_cr\_mapped\_coeff\_abs[ k ][ j ] − 1.

#### Luma mapping with chroma scaling data semantics

**lmcs\_min\_bin\_idx** specifies the minimum bin index used in the luma mapping with chroma scaling construction process. The value of lmcs\_min\_bin\_idx shall be in the range of 0 to 15, inclusive.

**lmcs\_delta\_max\_bin\_idx** specifies the delta value between 15 and the maximum bin index LmcsMaxBinIdx used in the luma mapping with chroma scaling construction process. The value of lmcs\_delta\_max\_bin\_idx shall be in the range of 0 to 15, inclusive. The value of LmcsMaxBinIdx is set equal to 15 − lmcs\_delta\_max\_bin\_idx. The value of LmcsMaxBinIdx shall be greater than or equal to lmcs\_min\_bin\_idx.

**lmcs\_delta\_cw\_prec\_minus1** plus 1 specifies the number of bits used for the representation of the syntax lmcs\_delta\_abs\_cw[ i ]. The value of lmcs\_delta\_cw\_prec\_minus1 shall be in the range of 0 to 14, inclusive.

**lmcs\_delta\_abs\_cw**[ i ] specifies the absolute delta codeword value for the ith bin.

**lmcs\_delta\_sign\_cw\_flag**[ i ] specifies the sign of the variable lmcsDeltaCW[ i ] as follows:

* If lmcs\_delta\_sign\_cw\_flag[ i ] is equal to 0, lmcsDeltaCW[ i ] is a positive value.
* Otherwise ( lmcs\_delta\_sign\_cw\_flag[ i ] is not equal to 0 ), lmcsDeltaCW[ i ] is a negative value.

When lmcs\_delta\_sign\_cw\_flag[ i ] is not present, it is inferred to be equal to 0.

The variable OrgCW is derived as follows:

OrgCW = ( 1  <<  BitDepth ) / 16 (98)

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

lmcsDeltaCW[ i ] = ( 1 − 2 \* lmcs\_delta\_sign\_cw\_flag[ i ] ) \* lmcs\_delta\_abs\_cw[ i ] (99)

The variable lmcsCW[ i ] is derived as follows:

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

lmcsCW[ i ] = OrgCW + lmcsDeltaCW[ i ] (100)

The value of lmcsCW[ i ] shall be in the range of OrgCW  >>  3 to ( OrgCW  <<  3 ) − 1, inclusive.

* For i = LmcsMaxBinIdx + 1..15, lmcsCW[ i ] is set equal 0.

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

<= ( 1  <<  BitDepth ) − 1 (101)

The variable InputPivot[ i ], with i = 0..16, is derived as follows:

InputPivot[ i ] = i \* OrgCW (102)

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 ) (103)  
 InvScaleCoeff[ i ] = 0  
 else  
 InvScaleCoeff[ i ] = OrgCW \* ( 1 << 11 ) / lmcsCW[ i ]  
}

It is a requirement of bitstream conformance that, for i = lmcs\_min\_bin\_idx..LmcsMaxBinIdx, when the value of LmcsPivot[ i ] is not a multiple of 1  <<  ( BitDepth − 5 ), the value of ( LmcsPivot[ i ] >> ( BitDepth − 5 ) ) shall not be equal to the value of ( LmcsPivot[ i + 1 ] >> ( BitDepth − 5 ) ).

**lmcs\_delta\_abs\_crs** specifies the absolute codeword value of the variable lmcsDeltaCrs. The value of lmcs\_delta\_abs\_crs shall be in the range of 0 and 7, inclusive. When not present, lmcs\_delta\_abs\_crs is inferred to be equal to 0.

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

The variable lmcsDeltaCrs is derived as follows:

lmcsDeltaCrs = ( 1 − 2 \* lmcs\_delta\_sign\_crs\_flag ) \* lmcs\_delta\_abs\_crs (104)

It is a requirement of bitstream conformance that, when lmcsCW[ i ] is not equal to 0, ( lmcsCW[ i ] + lmcsDeltaCrs ) shall be in the range of ( OrgCW >> 3 ) to ( ( OrgCW << 3 ) − 1 ), inclusive.

The variable ChromaScaleCoeff[ i ], with i = 0…15, is derived as follows:

if( lmcsCW[ i ] = = 0 )  
 ChromaScaleCoeff[ i ] = ( 1 << 11 )  
else (105)  
 ChromaScaleCoeff[ i ] = OrgCW \* ( 1  <<  11 ) / ( lmcsCW[ i ] + lmcsDeltaCrs )

#### 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[ id ]. 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 ) ) (106)

The variables refId and matrixSize are derived as follows:

refId = id − scaling\_list\_pred\_id\_delta[ id ] (107)

matrixSize = ( id < 2 ) ? 2 : ( ( id < 8 ) ? 4 : 8 ) (108)

The (matrixSize)x(matrixSize) array ScalingMatrixPred[ x ][ y ] with x = 0..matrixSize − 1, y = 0..matrixSize − 1 and the variable ScalingMatrixDCPred are derived as follows:

* When both scaling\_list\_copy\_mode\_flag[ id ] and scaling\_list\_pred\_mode\_flag[ id ] are equal to 0, all elements of ScalingMatrixPred are set equal to 8, and the value of ScalingMatrixDCPred is set equal to 8.
* Otherwise, when scaling\_list\_pred\_id\_delta[ id ] is equal to 0, all elements of ScalingMatrixPred are set equal to 16, and ScalingMatrixDCPred is set equal to 16.
* Otherwise (either scaling\_list\_copy\_mode\_flag[ id ] or scaling\_list\_pred\_mode\_flag[ id ] is equal to 1 and scaling\_list\_pred\_id\_delta[ id ] is greater than 0), ScalingMatrixPred is set equal to ScalingMatrixRec[ refId ], and the following applies for ScalingMatrixDCPred:
* If refId is greater than 13, ScalingMatrixDCPred is set equal to ScalingMatrixDCRec[ refId − 14 ].
* Otherwise (refId is less than or equal to 13), ScalingMatrixDCPred is set equal to ScalingMatrixPred[ 0 ][ 0 ].

**scaling\_list\_dc\_coef**[ id − 14 ] is used to derive the value of the variable ScalingMatrixDC[ id − 14 ] when id is greater than 13 as follows:

ScalingMatrixDCRec[ id − 14 ] = ( ScalingMatrixDCPred + scaling\_list\_dc\_coef[ id − 14 ] ) & 255 (109)

When not present,the value of scaling\_list\_dc\_coef[ id − 14 ] is inferred to be equal to 0. The value of scaling\_list\_dc\_coef[ id – 14 ] shall be in the range of −128 to 127, inclusive. The value of ScalingMatrixDCRec[ id − 14 ] shall be greater than 0.

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

The (matrixSize)x(matrixSize) array ScalingMatrixRec[ id ] is derived as follows:

ScalingMatrixRec[ id ][ x ][ y ] = ( ScalingMatrixPred[ x ][ y ] + ScalingList[ id ][ k ] ) & 255 (110)  
 with k = 0..( matrixSize \* matrixSize − 1 ),  
 x = DiagScanOrder[ Log2( matrixSize ) ][ Log2( matrixSize ) ][ k ][ 0 ], and  
 y = DiagScanOrder[ Log2( matrixSize ) ][ Log2( matrixSize ) ][ k ][ 1 ]

The value of ScalingMatrixRec[ id ][ x ][ y ] shall be greater than 0.

### Profile, tier, and level semantics

#### 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.

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.

**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 may be higher than but cannot 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 should indicate the profile that provides the preferred decoded result or the preferred bitstream identification, as determined by the encoder (in a manner not specified in this Specification).

NOTE 3 – When the CVSs of OlsInScope conform to different profiles, multiple profile\_tier\_level( ) syntax structures may 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.

**num\_sub\_profiles** specifies the number of the general\_sub\_profile\_idc[ i ] syntax elements.

**general\_sub\_profile\_idc**[ i ] specifies the indicator of the i-th interoperability metadata registered as specified by Rec. ITU-T T.35, the contents of which are not specified in this Specification. [Ed. (YK): Consider improving the wording of "interoperability metadata".]

**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. 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\_alignment\_zero\_bits** shall be equal to 0.

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:

– 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 ].

#### General constraint information semantics

**general\_non\_packed\_constraint\_flag** equal to 1 specifies that there shall not be any frame packing arrangement SEI messages present in the bitstream of the OlsInScope. general\_non\_packed\_constraint\_flag equal to 0 does not impose such a constraint.

NOTE 1 – Decoders may ignore the value of general\_non\_packed\_constraint\_flag, as there are no decoding process requirements associated with the presence or interpretation of frame packing arrangement SEI messages.

**general\_frame\_only\_constraint\_flag** equal to 1 specifies that OlsInScope conveys pictures that represent frames. general\_frame\_only\_constraint\_flag equal to 0 specifies that OlsInScope conveys pictures that may or may not represent frames.

NOTE 2 – Decoders may ignore the value of general\_frame\_only\_constraint\_flag, as there are no decoding process requirements associated with it.

**general\_non\_projected\_constraint\_flag** equal to 1 specifies that there shall not be any equirectangular projection SEI messages or generalized cubemap projection SEI messages present in the bitstream of the OlsInScope. general\_non\_projected\_constraint\_flag equal to 0 does not impose such a constraint.

NOTE 3 – Decoders may ignore the value of general\_non\_projected\_constraint\_flag, as there are no decoding process requirements associated with the presence or interpretation of equirectangular projection SEI messages and generalized cubemap projection SEI messages.

**general\_one\_picture\_only\_constraint\_flag** equal to 1 specifies that there is only one coded picture in the bitstream. general\_one\_picture\_only\_constraint\_flag equal to 0 does not impose such a constraint.

**intra\_only\_constraint\_flag** equal to 1 specifies that sh\_slice\_type shall be equal to I. intra\_only\_constraint\_flag equal to 0 does not impose such a constraint. When general\_one\_picture\_only\_constraint\_flag is equal to 1, the value of intra\_only\_constraint\_flag shall be equal to 1.

**max\_bitdepth\_constraint\_idc** specifies that sps\_bit\_depth\_minus8 shall be in the range of 0 to max\_bitdepth\_constraint\_idc, inclusive.

**max\_chroma\_format\_constraint\_idc** specifies that sps\_chroma\_format\_idc shall be in the range of 0 to max\_chroma\_format\_constraint\_idc, inclusive.

**single\_layer\_constraint\_flag** equal to 1 specifies that sps\_video\_parameter\_set\_id shall be equal to 0. single\_layer\_constraint\_flag equal to 0 does not impose such a constraint. When general\_one\_picture\_only\_constraint\_flag is equal to 1, the value of single\_layer\_constraint\_flag shall be equal to 1.

**all\_layers\_independent\_constraint\_flag** equal to 1 specifies that vps\_all\_independent\_layers\_flag shall be equal to 1. all\_layers\_independent\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_ref\_pic\_resampling\_constraint\_flag** equal to 1 specifies that sps\_ref\_pic\_resampling\_enabled\_flag shall be equal to 0. no\_ref\_pic\_resampling\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_res\_change\_in\_clvs\_constraint\_flag** equal to 1 specifies that sps\_res\_change\_in\_clvs\_allowed\_flag shall be equal to 0. no\_res\_change\_in\_clvs\_constraint\_flag equal to 0 does not impose such a constraint.

**one\_tile\_per\_pic\_constraint\_flag** equal to 1 specifies that each picture shall contain only one tile. one\_tile\_per\_pic\_constraint\_flag equal to 0 does not impose such a constraint.

**pic\_header\_in\_slice\_header\_constraint\_flag** equal to 1 specifies that each picture shall contain only one slice and the value of sh\_picture\_header\_in\_slice\_header\_flag in each slice shall be equal to 1. pic\_header\_in\_slice\_header\_constraint\_flag equal to 0 does not impose such a constraint.

**one\_slice\_per\_pic\_constraint\_flag** equal to 1 specifies that each picture shall contain only one slice. one\_slice\_per\_pic\_constraint\_flag equal to 0 does not impose such a constraint. When pic\_header\_in\_slice\_header\_constraint\_flag is equal to 1, the value of one\_slice\_per\_pic\_constraint\_flag shall be equal to 1.

**one\_subpic\_per\_pic\_constraint\_flag** equal to 1 specifies that each picture shall contain only one subpicture. one\_subpic\_per\_pic\_constraint\_flag equal to 0 does not impose such a constraint. When one\_slice\_per\_pic\_constraint\_flag is equal to 1, the value of one\_subpic\_per\_pic\_constraint\_flag shall be equal to 1.

**no\_qtbtt\_dual\_tree\_intra\_constraint\_flag** equal to 1 specifies that sps\_qtbtt\_dual\_tree\_intra\_flag shall be equal to 0. no\_qtbtt\_dual\_tree\_intra\_constraint\_flag equal to 0 does not impose such a constraint. When max\_chroma\_format\_constraint\_idc is equal to 0, the value of no\_qtbtt\_dual\_tree\_intra\_constraint\_flag shall be equal to 1.

**no\_partition\_constraints\_override\_constraint\_flag** equal to 1 specifies that sps\_partition\_constraints\_override\_enabled\_flag shall be equal to 0. no\_partition\_constraints\_override\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_sao\_constraint\_flag** equal to 1 specifies that sps\_sao\_enabled\_flag shall be equal to 0. no\_sao\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_alf\_constraint\_flag** equal to 1 specifies that sps\_alf\_enabled\_flag shall be equal to 0. no\_alf\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_ccalf\_constraint\_flag** equal to 1 specifies that sps\_ccalf\_enabled\_flag shall be equal to 0. no\_ccalf\_constraint\_flag equal to 0 does not impose such a constraint. When max\_chroma\_format\_constraint\_idc is equal to 0 or no\_alf\_constraint\_flag is equal 1, the value of no\_ccalf\_constraint\_flag shall be equal to 1.

**no\_joint\_cbcr\_constraint\_flag** equal to 1 specifies that sps\_joint\_cbcr\_enabled\_flag shall be equal to 0. no\_joint\_cbcr\_constraint\_flag equal to 0 does not impose such a constraint. When max\_chroma\_format\_constraint\_idc is equal to 0, the value of no\_joint\_cbcr\_constraint\_flag shall be equal to 1.

**no\_mrl\_constraint\_flag** equal to 1 specifies that sps\_mrl\_enabled\_flag shall be equal to 0. no\_mrl\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_isp\_constraint\_flag** equal to 1 specifies that sps\_isp\_enabled\_flag shall be equal to 0. no\_isp\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_mip\_constraint\_flag** equal to 1 specifies that sps\_mip\_enabled\_flag shall be equal to 0. no\_mip\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_ref\_wraparound\_constraint\_flag** equal to 1 specifies that sps\_ref\_wraparound\_enabled\_flag shall be equal to 0. no\_ref\_wraparound\_constraint\_flag equal to 0 does not impose such a constraint. When intra\_only\_constraint\_flag is equal to 1, the value of no\_ref\_wraparound\_constraint\_flag shall be equal to 1.

**no\_temporal\_mvp\_constraint\_flag** equal to 1 specifies that sps\_temporal\_mvp\_enabled\_flag shall be equal to 0. no\_temporal\_mvp\_constraint\_flag equal to 0 does not impose such a constraint. When intra\_only\_constraint\_flag is equal to 1, the value of no\_temporal\_mvp\_constraint\_flag shall be equal to 1.

**no\_sbtmvp\_constraint\_flag** equal to 1 specifies that sps\_sbtmvp\_enabled\_flag shall be equal to 0. no\_sbtmvp\_constraint\_flag equal to 0 does not impose such a constraint. When no\_temporal\_mvp\_constraint\_flag is equal to 1, the value of no\_sbtmvp\_constraint\_flag shall be equal to 1.

**no\_amvr\_constraint\_flag** equal to 1 specifies that sps\_amvr\_enabled\_flag shall be equal to 0. no\_amvr\_constraint\_flag equal to 0 does not impose such a constraint. When intra\_only\_constraint\_flag is equal to 1, the value of no\_amvr\_constraint\_flag shall be equal to 1.

**no\_bdof\_constraint\_flag** equal to 1 specifies that sps\_bdof\_enabled\_flag shall be equal to 0. no\_bdof\_constraint\_flag equal to 0 does not impose such a constraint. When intra\_only\_constraint\_flag is equal to 1, the value of no\_bdof\_constraint\_flag shall be equal to 1.

**no\_dmvr\_constraint\_flag** equal to 1 specifies that sps\_dmvr\_enabled\_flag shall be equal to 0. no\_dmvr\_constraint\_flag equal to 0 does not impose such a constraint. When intra\_only\_constraint\_flag is equal to 1, the value of no\_dmvr\_constraint\_flag shall be equal to 1.

**no\_cclm\_constraint\_flag** equal to 1 specifies that sps\_cclm\_enabled\_flag shall be equal to 0. no\_cclm\_constraint\_flag equal to 0 does not impose such a constraint. When max\_chroma\_format\_constraint\_idc is equal to 0, the value of no\_cclm\_constraint\_flag shall be equal to 1.

**no\_mts\_constraint\_flag** equal to 1 specifies that sps\_mts\_enabled\_flag shall be equal to 0. no\_mts\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_sbt\_constraint\_flag** equal to 1 specifies that sps\_sbt\_enabled\_flag shall be equal to 0. no\_sbt\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_lfnst\_constraint\_flag** equal to 1 specifies that sps\_lfnst\_enabled\_flag shall be equal to 0. no\_lfnst\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_affine\_motion\_constraint\_flag** equal to 1 specifies that sps\_affine\_enabled\_flag  shall be equal to 0. no\_affine\_motion\_constraint\_flag equal to 0 does not impose such a constraint. When intra\_only\_constraint\_flag is equal to 1, the value of no\_affine\_motion\_constraint\_flag shall be equal to 1.

**no\_mmvd\_constraint\_flag** equal to 1 specifies that sps\_mmvd\_enabled\_flag shall be equal to 0. no\_mmvd\_constraint\_flag equal to 0 does not impose such a constraint. When intra\_only\_constraint\_flag is equal to 1, the value of no\_mmvd\_constraint\_flag shall be equal to 1.

**no\_smvd\_constraint\_flag** equal to 1 specifies that sps\_smvd\_enabled\_flag shall be equal to 0. no\_smvd\_constraint\_flag equal to 0 does not impose such a constraint. When intra\_only\_constraint\_flag is equal to 1, the value of no\_smvd\_constraint\_flag shall be equal to 1.

**no\_prof\_constraint\_flag** equal to 1 specifies that sps\_affine\_prof\_enabled\_flag shall be equal to 0. no\_prof\_constraint\_flag equal to 0 does not impose such a constraint. When intra\_only\_constraint\_flag is equal to 1, the value of no\_prof\_constraint\_flag shall be equal to 1.

**no\_bcw\_constraint\_flag** equal to 1 specifies that sps\_bcw\_enabled\_flag shall be equal to 0. no\_bcw\_constraint\_flag equal to 0 does not impose such a constraint. When intra\_only\_constraint\_flag is equal to 1, the value of no\_bcw\_constraint\_flag shall be equal to 1.

**no\_ibc\_constraint\_flag** equal to 1 specifies that sps\_ibc\_enabled\_flag shall be equal to 0. no\_ibc\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_ciip\_constraint\_flag** equal to 1 specifies that sps\_ciip\_enabled\_flag shall be equal to 0. no\_cipp\_constraint\_flag equal to 0 does not impose such a constraint. When intra\_only\_constraint\_flag is equal to 1, the value of no\_cipp\_constraint\_flag shall be equal to 1.

**no\_gpm\_constraint\_flag** equal to 1 specifies that sps\_gpm\_enabled\_flag shall be equal to 0. no\_gpm\_constraint\_flag equal to 0 does not impose such a constraint. When intra\_only\_constraint\_flag is equal to 1, the value of no\_gpm\_constraint\_flag shall be equal to 1.

**no\_ladf\_constraint\_flag** equal to 1 specifies that sps\_ladf\_enabled\_flag shall be equal to 0. no\_ladf\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_transform\_skip\_constraint\_flag** equal to 1 specifies that sps\_transform\_skip\_enabled\_flag shall be equal to 0. no\_transform\_skip\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_bdpcm\_constraint\_flag** equal to 1 specifies that sps\_bdpcm\_enabled\_flag shall be equal to 0. no\_bdpcm\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_palette\_constraint\_flag** equal to 1 specifies that sps\_palette\_enabled\_flag shall be equal to 0. no\_palette\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_act\_constraint\_flag** equal to 1 specifies that sps\_act\_enabled\_flag shall be equal to 0. no\_act\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_lmcs\_constraint\_flag** equal to 1 specifies that sps\_lmcs\_enabled\_flag shall be equal to 0. no\_lmcs\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_cu\_qp\_delta\_constraint\_flag** equal to 1 specifies that pps\_cu\_qp\_delta\_enabled\_flag shall be equal to 0. no\_cu\_qp\_delta\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_chroma\_qp\_offset\_constraint\_flag** equal to 1 specifies that pps\_cu\_chroma\_qp\_offset\_list\_enabled\_flag shall be equal to 0. no\_chroma\_qp\_offset\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_dep\_quant\_constraint\_flag** equal to 1 specifies that sps\_dep\_quant\_enabled\_flag shall be equal to 0. no\_dep\_quant\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_sign\_data\_hiding\_constraint\_flag** equal to 1 specifies that sps\_sign\_data\_hiding\_enabled\_flag shall be equal to 0. no\_sign\_data\_hiding\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_tsrc\_constraint\_flag** equal to 1 specifies that sh\_ts\_residual\_coding\_disabled\_flag shall be equal to 0. no\_tsrc\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_mixed\_nalu\_types\_in\_pic\_constraint\_flag** equal to 1 specifies that it is a requirement of bitstream conformance that pps\_mixed\_nalu\_types\_in\_pic\_flag shall be equal to 0. no\_mixed\_nalu\_types\_in\_pic\_constraint\_flag equal to 0 does not impose such a constraint.

**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. no\_trail\_constraint\_flag equal to 0 does not impose such a constraint.

**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. no\_stsa\_constraint\_flag equal to 0 does not impose such a constraint.

**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. no\_rasl\_constraint\_flag equal to 0 does not impose such a constraint.

**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. no\_radl\_constraint\_flag equal to 0 does not impose such a constraint.

**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. no\_idr\_constraint\_flag equal to 0 does not impose such a constraint.

**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. no\_cra\_constraint\_flag equal to 0 does not impose such a constraint.

**no\_gdr\_constraint\_flag** equal to 1 specifies that sps\_gdr\_enabled\_flag shall be equal to 0. no\_gdr\_constraint\_flag equal to 0 does not impose such a constraint.

**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, and the sps\_lmcs\_enabled\_flag and sps\_scaling\_list\_enabled\_flag shall both be equal to 0. no\_aps\_constraint\_flag equal to 0 does not impose such a constraint.

**gci\_alignment\_zero\_bits** shall be equal to 0.

**gci\_num\_reserved\_bytes** specifies the number of the reserved constraint bytes. The value of gci\_num\_reserved\_bytes shall be 0. Other values of gci\_num\_reserved\_bytes are reserved for future use by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification.

**gci\_reserved\_byte**[ i ]may have any value. Its presence and value do not affect decoder conformance to profiles specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore the values of all the gci\_reserved\_byte[ i ] syntax elements.

### 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.

**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 max\_dec\_pic\_buffering\_minus1[ i ] shall be in the range of 0 to MaxDpbSize − 1, inclusive, where MaxDpbSize is as specified in clause A.4.2. When i is greater than 0, max\_dec\_pic\_buffering\_minus1[ i ] shall be greater than or equal to max\_dec\_pic\_buffering\_minus1[ i − 1 ]. When 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 max\_dec\_pic\_buffering\_minus1[ maxSubLayersMinus1 ].

**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 max\_num\_reorder\_pics[ i ] shall be in the range of 0 to max\_dec\_pic\_buffering\_minus1[ i ], inclusive. When i is greater than 0, max\_num\_reorder\_pics[ i ] shall be greater than or equal to max\_num\_reorder\_pics[ i − 1 ]. When 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 max\_num\_reorder\_pics[ maxSubLayersMinus1 ].

**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 max\_latency\_increase\_plus1[ i ] is not equal to 0, the value of MaxLatencyPictures[ i ] is specified as follows:

MaxLatencyPictures[ i ] = max\_num\_reorder\_pics[ i ] + max\_latency\_increase\_plus1[ i ] − 1 (7‑111)

When max\_latency\_increase\_plus1[ i ] is equal to 0, no corresponding limit is expressed.

The value of max\_latency\_increase\_plus1[ i ] shall be in the range of 0 to 232 − 2, inclusive. When 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 max\_latency\_increase\_plus1[ maxSubLayersMinus1 ].

### HRD parameters semantics

#### General HRD parameters semantics

The general\_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\_hrd\_parameters( ) syntax structure present in any VPSs or SPSs in the bitstream shall be identical.

When included in a VPS, the general\_hrd\_parameters( ) syntax structure applies to all OLSs specified by the VPS. When included in an SPS, the general\_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 may be equal to 27 000 000 and num\_units\_in\_tick may be equal to 1 080 000, and consequently a clock tick may 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\_hrd\_parameters( ) syntax structure. general\_nal\_hrd\_params\_present\_flag equal to 0 specifies that NAL HRD parameters are not present in the general\_hrd\_parameters( ) syntax structure.

NOTE 1 – When general\_nal\_hrd\_params\_present\_flag is equal to 0, the conformance of the bitstream cannot 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 DU information SEI messages, by some means not specified in this Specification.

The variable NalHrdBpPresentFlag is derived as follows:

– If one or more of the following conditions are true, the value of NalHrdBpPresentFlag is set equal to 1:

– general\_nal\_hrd\_params\_present\_flag is present in the bitstream and is equal to 1.

– The need for presence of BPs for NAL HRD operation to be present in the bitstream in BP SEI messages is determined by the application, by some means not specified in this Specification.

– Otherwise, the value of NalHrdBpPresentFlag is set equal to 0.

**general\_vcl\_hrd\_params\_present\_flag** equal to 1 specifies that VCL HRD parameters (pertaining to Type I bitstream conformance point) are present in the general\_hrd\_parameters( ) syntax structure. general\_vcl\_hrd\_params\_present\_flag equal to 0 specifies that VCL HRD parameters are not present in the general\_hrd\_parameters( ) syntax structure.

NOTE 2 – When general\_vcl\_hrd\_params\_present\_flag is equal to 0, the conformance of the bitstream cannot 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 DU information SEI messages, by some means not specified in this Specification.

The variable VclHrdBpPresentFlag is derived as follows:

– If one or more of the following conditions are true, the value of VclHrdBpPresentFlag is set equal to 1:

– general\_vcl\_hrd\_params\_present\_flag is present in the bitstream and is equal to 1.

– The need for presence of BPs for VCL HRD operation to be present in the bitstream in BP SEI messages is determined by the application, by some means not specified in this Specification.

– Otherwise, the value of VclHrdBpPresentFlag is set equal to 0.

The variable CpbDpbDelaysPresentFlag is derived as follows:

– If one or more of the following conditions are true, the value of CpbDpbDelaysPresentFlag is set equal to 1:

– general\_nal\_hrd\_params\_present\_flag is present in the bitstream and is equal to 1.

– general\_vcl\_hrd\_params\_present\_flag is present in the bitstream and is equal to 1.

– The need for presence of CPB and DPB output delays to be present in the bitstream in PT SEI messages is determined by the application, by some means not specified in this Specification.

– Otherwise, the value of CpbDpbDelaysPresentFlag is set equal to 0.

It is a requirement of bitstream conformance that the values of general\_nal\_hrd\_params\_present\_flag and general\_vcl\_hrd\_params\_present\_flag in each general\_hrd\_parameters( ) syntax structure shall not be both 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 may or may not apply to the AU for any OLS in the bitstream and scalable-nested PT SEI messages may be present.

**general\_decoding\_unit\_hrd\_params\_present\_flag** equal to 1 specifies that DU level HRD parameters are present and the HRD may operate at AU level or DU level. general\_decoding\_unit\_hrd\_params\_present\_flag equal to 0 specifies that DU level HRD parameters are not present and the HRD operates at AU level. When general\_decoding\_unit\_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\_decoding\_unit\_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 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.

#### OLS HRD parameters semantics

When an ols\_hrd\_parameters( ) syntax structure is included in a VPS, the OLSs to which the ols\_hrd\_parameters( ) syntax structure applies are specified by the VPS. When an ols\_hrd\_parameters( ) syntax structure is included in an SPS, the ols\_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 below. fixed\_pic\_rate\_general\_flag[ i ] equal to 0 indicates that this constraint may 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 below. fixed\_pic\_rate\_within\_cvs\_flag[ i ] equal to 0 indicates that this constraint may 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.

**elemental\_duration\_in\_tc\_minus1**[ i ] plus 1 (when present) 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 as specified below. 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\_general\_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:

DpbOutputElementalInterval[ n ] = DpbOutputInterval[ n ]  elementalOutputPeriods (112)

where DpbOutputInterval[ n ] is specified in Equation C.16 and elementalOutputPeriods is specified as follows:

– If a PT SEI message is present for picture n, elementalOutputPeriods is equal to the value of pt\_display\_elemental\_periods\_minus1 + 1.

– Otherwise, elementalOutputPeriods is equal to 1.

When Htid is equal to i and fixed\_pic\_rate\_general\_flag[ i ] is equal to 1 for a CVS containing picture n, and picture n is a picture that is output and is not the last picture in the bitstream (in output order) that is output, the value computed for DpbOutputElementalInterval[ n ] shall be equal to ClockTick \* ( elemental\_duration\_in\_tc\_minus1[ i ] + 1 ), wherein ClockTick is as specified in Equation C.1 (using the value of ClockTick for the CVS containing picture n) when one of the following conditions is true for the following picture in output order nextPicInOutputOrder that is specified for use in Equation C.16:

– 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.

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 CVS (in output order) that is output, the value computed for DpbOutputElementalInterval[ n ] shall be equal to ClockTick \* ( elemental\_duration\_in\_tc\_minus1[ i ] + 1 ), wherein ClockTick is as specified in Equation C.1 (using the value of ClockTick for the CVS containing picture n) when the following picture in output order nextPicInOutputOrder that is specified for use in Equation C.16 is in the same CVS as picture n. [Ed. (GJS): Should this paragraph be an "otherwise" clause that only needs to be considered when Htid is equal to i and fixed\_pic\_rate\_general\_flag[ i ] is equal to 0?]

**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.

#### Sublayer HRD parameters semantics

When a sublayer\_hrd\_parameters( ) syntax structure is included in the i-th ols\_hrd\_parameters( ) syntax structure in a VPS, the value of maxSublayersMinus1 is set equal to vps\_hrd\_max\_tid[ i ]. When a sublayer\_hrd\_parameters( ) syntax structure is included in the ols\_hrd\_parameters( ) syntax structure in an SPS, the value of maxSublayersMinus1 is set equal to sps\_max\_sublayers\_minus1.

**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 232 − 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 ) (113)

– When the bit\_rate\_value\_minus1[ i ][ j ] syntax element is not present, it is inferred as follows:

– If general\_hrd\_params\_present\_flag is equal to 1, bit\_rate\_value\_minus1[ i ][ j ] is inferred to be equal to bit\_rate\_value\_minus1[ maxSublayersMinus1 ][ j ].

– Otherwise (general\_hrd\_params\_present\_flag is equal to 0), the value of BitRate[ i ][ j ] is inferred to be equal to CpbBrVclFactor \* MaxBR for VCL HRD parameters and to be equal to CpbBrNalFactor \* MaxBR for NAL HRD parameters, where MaxBR, CpbBrVclFactor and CpbBrNalFactor 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 232 − 2, inclusive. For any j greater than 0 and any particular value of i, cpb\_size\_value\_minus1[ i ][ j ] shall be less than or equal to cpb\_size\_value\_minus1[ i ][ j − 1 ].

When DecodingUnitHrdFlag is equal to 0, the following apples:

– The CPB size in bits is given by:

CpbSize[ i ][ j ] = ( cpb\_size\_value\_minus1[ i ][ j ] + 1 ) \* 2( 4 + cpb\_size\_scale ) (114)

– When the cpb\_size\_value\_minus1[ i ][ j ] syntax element is not present, it is inferred as follows:

– If general\_hrd\_params\_present\_flag is equal to 1, cpb\_size\_value\_minus1[ i ][ j ] is inferred to be equal to cpb\_size\_value\_minus1[ maxSublayersMinus1 ][ j ].

– Otherwise (general\_hrd\_params\_present\_flag is equal to 0), the value of CpbSize[ i ][ j ] is inferred to be equal to CpbBrVclFactor \* MaxCPB for VCL HRD parameters and to be equal to CpbBrNalFactor \* MaxCPB for NAL HRD parameters, where MaxCPB, CpbBrVclFactor and CpbBrNalFactor are specified in Annex A.

**cpb\_size\_du\_value\_minus1**[ i ][ j ] is used together with cpb\_size\_du\_scale to specify the i-th CPB size with Htid equal to i when the CPB operates at DU level. cpb\_size\_du\_value\_minus1[ i ][ j ] shall be in the range of 0 to 232 − 2, inclusive. For any j greater than 0 and any particular value of i, cpb\_size\_du\_value\_minus1[ i ][ j ] shall be less than or equal to cpb\_size\_du\_value\_minus1[ i ][ j − 1 ].

When DecodingUnitHrdFlag is equal to 1, the following applies:

– The CPB size in bits is given by:

CpbSize[ i ][ j ] = ( cpb\_size\_du\_value\_minus1[ i ][ j ] + 1 ) \* 2( 4 + cpb\_size\_du\_scale ) (115)

– When the cpb\_size\_du\_value\_minus1[ i ][ j ] syntax element is not present, it is inferred as follows:

– If general\_hrd\_params\_present\_flag is equal to 1, cpb\_size\_du\_value\_minus1[ i ][ j ] is inferred to be equal to cpb\_size\_du\_value\_minus1[ maxSublayersMinus1 ][ j ].

– Otherwise (general\_hrd\_params\_present\_flag is equal to 0), the value of CpbSize[ i ][ j ] is inferred to be equal to CpbVclFactor \* MaxCPB for VCL HRD parameters and to be equal to CpbNalFactor \* MaxCPB for NAL HRD parameters, where MaxCPB, CpbVclFactor and CpbNalFactor are specified in Annex A.

**bit\_rate\_du\_value\_minus1**[ i ][ j ] (together with bit\_rate\_scale) specifies the maximum input bit rate for the j-th CPB with Htid equal to i when the CPB operates at the DU level. bit\_rate\_du\_value\_minus1[ i ][ j ] shall be in the range of 0 to 232 − 2, inclusive. For any j greater than 0 and any particular value of i, bit\_rate\_du\_value\_minus1[ i ][ j ] shall be greater than bit\_rate\_du\_value\_minus1[ i ][ j − 1 ].

When DecodingUnitHrdFlag is equal to 1, the folowing applies:

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

BitRate[ i ][ j ] = ( bit\_rate\_du\_value\_minus1[ i ][ j ] + 1 ) \* 2( 6 + bit\_rate\_scale ) (116)

– When the bit\_rate\_du\_value\_minus1[ i ][ j ] syntax element is not present, it is inferred as follows:

– If general\_hrd\_params\_present\_flag is equal to 1, bit\_rate\_du\_value\_minus1[ i ][ j ] is inferred to be equal to bit\_rate\_du\_value\_minus1[ maxSublayersMinus1 ][ j ].

– Otherwise (general\_hrd\_params\_present\_flag is equal to 0), the value of BitRate[ i ][ j ] is inferred to be equal to BrVclFactor \* MaxBR for VCL HRD parameters and to be equal to BrNalFactor \* MaxBR for NAL HRD parameters, where MaxBR, BrVclFactor and BrNalFactor are specified in Annex A.

**cbr\_flag**[ i ][ j ] equal to 0 specifies that to decode this bitstream by the HRD using the j-th CPB specification, the hypothetical stream scheduler (HSS) operates in an intermittent bit rate mode. cbr\_flag[ i ][ j ] equal to 1 specifies that the HSS operates in a constant bit rate (CBR) mode.

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

– When the cbr\_flag[ i ][ j ] syntax element is not present, it is inferred as follows:

– If general\_hrd\_params\_present\_flag is equal to 1, cbr\_flag[ i ][ j ] is inferred to be equal to cbr\_flag[ maxSublayersMinus1 ][ j ].

– Otherwise (general\_hrd\_params\_present\_flag is equal to 0), the value of cbr\_flag[ i ][ j ] is inferred to be equal to 0.

### 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.

### Slice header semantics

#### General 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 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 sps\_separate\_colour\_plane\_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\_wp\_info\_in\_ph\_flag, or pps\_qp\_delta\_info\_in\_ph\_flag is equal to 1.

**sh\_subpic\_id** specifies the subpicture ID of the subpicture that contains the slice. If sh\_subpic\_id is present, the value of the variable CurrSubpicIdx is derived to be such that SubpicIdVal[ CurrSubpicIdx ] is equal to sh\_subpic\_id. Otherwise (sh\_subpic\_id is not present), CurrSubpicIdx is derived to be equal to 0. The length of sh\_subpic\_id is sps\_subpic\_id\_len\_minus1 + 1 bits.

**sh\_slice\_address** specifies the slice address of the slice. When not present, the value of sh\_slice\_address is inferred to be equal to 0.

If pps\_rect\_slice\_flag is equal to 0, the following applies:

* The slice address is the raster scan tile index of the first tile in the slice.
* The length of sh\_slice\_address is Ceil( Log2 ( NumTilesInPic ) ) bits.
* The value of sh\_slice\_address shall be in the range of 0 to NumTilesInPic − 1, inclusive.

Otherwise (pps\_rect\_slice\_flag is equal to 1), the following applies:

* The slice address is the subpicture-level slice index of the current slice, i.e., SubpicLevelSliceIdx[ j ], where j is the picture-level slice index of the current slice.
* The length of sh\_slice\_address is Ceil( Log2( NumSlicesInSubpic[ CurrSubpicIdx ] ) ) bits.
* The value of sh\_slice\_address shall be in the range of 0 to NumSlicesInSubpic[ CurrSubpicIdx ] − 1, inclusive.

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

* If pps\_rect\_slice\_flag is equal to 0 or sps\_subpic\_info\_present\_flag is equal to 0, the value of sh\_slice\_address shall not be equal to the value of sh\_slice\_address of any other coded slice NAL unit of the same coded picture.
* Otherwise, the pair of sh\_subpic\_id and sh\_slice\_address values shall not be equal to the pair of sh\_subpic\_id and sh\_slice\_address values of any other coded slice NAL unit of the same coded picture.
* The shapes of the slices of a picture shall be such that each CTU, when decoded, shall have its entire left boundary and entire top boundary consisting of a picture boundary or consisting of boundaries of previously decoded CTU(s).

**sh\_extra\_bit**[ i ] may be equal to 1 or 0. Decoders conforming to this version of this Specification shall ignore the value of sh\_extra\_bit[ i ]. Its value does not affect decoder conformance to profiles specified in this version of specification.

**sh\_num\_tiles\_in\_slice\_minus1** plus 1, when present, specifies the number of tiles in the slice. The value of sh\_num\_tiles\_in\_slice\_minus1 shall be in the range of 0 to NumTilesInPic − 1, inclusive. When not present, the value of sh\_num\_tiles\_in\_slice\_minus1 shall be inferred to be equal to 0.

The variable NumCtusInCurrSlice, which specifies the number of CTUs in the current slice, and the list CtbAddrInCurrSlice[ i ], for i ranging from 0 to NumCtusInCurrSlice − 1, inclusive, specifying the picture raster scan address of the i-th CTB within the slice, are derived as follows:

if( pps\_rect\_slice\_flag ) {  
 picLevelSliceIdx = sh\_slice\_address  
 for( j = 0; j < CurrSubpicIdx; j++ )  
 picLevelSliceIdx += NumSlicesInSubpic[ j ]  
 NumCtusInCurrSlice = NumCtusInSlice[ picLevelSliceIdx ]  
 for( i = 0; i < NumCtusInCurrSlice; i++ )  
 CtbAddrInCurrSlice[ i ] = CtbAddrInSlice[ picLevelSliceIdx ][ i ] (117)  
} 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 = tileRowBd[ tileY ]; ctbY < tileRowBd[ tileY + 1 ]; ctbY++ ) {  
 for( ctbX = tileColBd[ tileX ]; ctbX < tileColBd[ tileX + 1 ]; ctbX++ ) {  
 CtbAddrInCurrSlice[ NumCtusInCurrSlice ] = ctbY \* PicWidthInCtb + ctbX  
 NumCtusInCurrSlice++  
 }  
 }  
 }  
}

The variables SubpicLeftBoundaryPos, SubpicTopBoundaryPos, SubpicRightBoundaryPos, and SubpicBotBoundaryPos are derived as follows:

if( sps\_subpic\_treated\_as\_pic\_flag[ CurrSubpicIdx ] ) {  
 SubpicLeftBoundaryPos = sps\_subpic\_ctu\_top\_left\_x[ CurrSubpicIdx ] \* CtbSizeY  
 SubpicRightBoundaryPos = Min( sps\_pic\_width\_max\_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 (118)  
 SubpicBotBoundaryPos = Min( sps\_pic\_height\_max\_in\_luma\_samples − 1,  
 ( sps\_subpic\_ctu\_top\_left\_y[ CurrSubpicIdx ] +  
 sps\_subpic\_height\_minus1[ CurrSubpicIdx ] + 1 ) \* CtbSizeY − 1 )  
}

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

Table 9 – Name association to sh\_slice\_type

|  |  |
| --- | --- |
| sh\_slice\_type | Name of sh\_slice\_type |
| 0 | B (B slice) |
| 1 | P (P slice) |
| 2 | I (I slice) |

When not present, the value of sh\_slice\_type is inferred to be equal to 2.

When ph\_intra\_slice\_allowed\_flag is equal to 0, the value of sh\_slice\_type shall be equal to 0 or 1.

When both of the following conditions are true, the value of sh\_slice\_type shall be equal to 2:

* The value of nal\_unit\_type is in the range of IDR\_W\_RADL to CRA\_NUT, inclusive.
* The value of vps\_independent\_layer\_flag[ GeneralLayerIdx[ nuh\_layer\_id ] ] is equal to 1 or the current picture is the first picture in the current AU.

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

* If sh\_slice\_type equal to 2 (I), the following applies:

MinQtLog2SizeY = MinCbLog2SizeY + ph\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_luma (119)

MinQtLog2SizeC = MinCbLog2SizeY + ph\_log2\_diff\_min\_qt\_min\_cb\_intra\_slice\_chroma (120)

MaxBtSizeY = 1 << ( MinQtLog2SizeY + ph\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_luma ) (121)

MaxBtSizeC = 1 << ( MinQtLog2SizeC + ph\_log2\_diff\_max\_bt\_min\_qt\_intra\_slice\_chroma ) (122)

MaxTtSizeY = 1 << ( MinQtLog2SizeY + ph\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_luma ) (123)

MaxTtSizeC = 1 << ( MinQtLog2SizeC + ph\_log2\_diff\_max\_tt\_min\_qt\_intra\_slice\_chroma ) (124)

MaxMttDepthY = ph\_max\_mtt\_hierarchy\_depth\_intra\_slice\_luma (125)

MaxMttDepthC = ph\_max\_mtt\_hierarchy\_depth\_intra\_slice\_chroma (126)

CuQpDeltaSubdiv = ph\_cu\_qp\_delta\_subdiv\_intra\_slice (127)

CuChromaQpOffsetSubdiv = ph\_cu\_chroma\_qp\_offset\_subdiv\_intra\_slice (128)

* Otherwise (sh\_slice\_type equal to 0 (B) or 1 (P)), the following applies:

MinQtLog2SizeY = MinCbLog2SizeY + ph\_log2\_diff\_min\_qt\_min\_cb\_inter\_slice (129)

MinQtLog2SizeC = MinCbLog2SizeY + ph\_log2\_diff\_min\_qt\_min\_cb\_inter\_slice (130)

MaxBtSizeY = 1 << ( MinQtLog2SizeY + ph\_log2\_diff\_max\_bt\_min\_qt\_inter\_slice) (131)

MaxBtSizeC = 1 << ( MinQtLog2SizeC + ph\_log2\_diff\_max\_bt\_min\_qt\_inter\_slice) (132)

MaxTtSizeY = 1 << ( MinQtLog2SizeY + ph\_log2\_diff\_max\_tt\_min\_qt\_inter\_slice) (133)

MaxTtSizeC = 1 << ( MinQtLog2SizeC + ph\_log2\_diff\_max\_tt\_min\_qt\_inter\_slice) (134)

MaxMttDepthY = ph\_max\_mtt\_hierarchy\_depth\_inter\_slice (135)

MaxMttDepthC = ph\_max\_mtt\_hierarchy\_depth\_inter\_slice (136)

CuQpDeltaSubdiv = ph\_cu\_qp\_delta\_subdiv\_inter\_slice (137)

CuChromaQpOffsetSubdiv =ph\_cu\_chroma\_qp\_offset\_subdiv\_inter\_slice (138)

* The following applies:

MinQtSizeY = 1 << MinQtLog2SizeY (139)

MinQtSizeC = 1 << MinQtLog2SizeC (140)

MinBtSizeY = 1 << MinCbLog2SizeY (141)

MinTtSizeY = 1 << MinCbLog2SizeY (142)

**sh\_alf\_enabled\_flag** equal to 1 specifies that adaptive loop filter is enabled and may be applied to Y, Cb, or Cr colour component in a slice. sh\_alf\_enabled\_flag equal to 0 specifies that adaptive loop filter is disabled and not applied for all colour components in a 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 ChromaArrayType 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\_flag** equal to 0 specifies that the adaptive loop filter is not applied to the Cb colour component of the current slice. sh\_alf\_cb\_flag equal to 1 specifies that the adaptive loop filter is applied to the Cb colour component of the current slice. When sh\_alf\_cb\_flag is not present, it is inferred to be equal to ph\_alf\_cb\_flag.

**sh\_alf\_cr\_flag** equal to 0 specifies that the adaptive loop filter is not applied to the Cr colour component of the current slice. sh\_alf\_cr\_flag equal to 1 specifies that the adaptive loop filter is applied to the Cr colour component of the current slice. When sh\_alf\_cr\_flag is not present, it is inferred to be equal to ph\_alf\_cr\_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\_cc\_alf\_cb\_enabled\_flag** equal to 0 specifies that the cross-component filter is disabled and not applied to the Cb colour component. sh\_cc\_alf\_cb\_enabled\_flag equal to 1 specifies that the cross-component filter is enabled and may be applied to the Cb colour component. When sh\_cc\_alf\_cb\_enabled\_flag is not present, it is inferred to be equal to ph\_cc\_alf\_cb\_enabled\_flag.

**sh\_cc\_alf\_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\_cc\_alf\_cb\_aps\_id shall be less than or equal to the TemporalId of the coded slice NAL unit. When sh\_cc\_alf\_cb\_enabled\_flag is equal to 1 and sh\_cc\_alf\_cb\_aps\_id is not present, the value of sh\_cc\_alf\_cb\_aps\_id is inferred to be equal to the value of ph\_cc\_alf\_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\_cc\_alf\_cb\_aps\_id shall be equal to 1.

**sh\_cc\_alf\_cr\_enabled\_flag** equal to 0 specifies that the cross-component filter is disabled and not applied to the Cr colour component. sh\_cc\_alf\_cr\_enabled\_flag equal to 1 specifies that the cross-component adaptive loop filter is enabled and may be applied to the Cr colour component. When sh\_cc\_alf\_cr\_enabled\_flag is not present, it is inferred to be equal to ph\_cc\_alf\_cr\_enabled\_flag.

**sh\_cc\_alf\_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\_cc\_alf\_cr\_aps\_id shall be less than or equal to the TemporalId of the coded slice NAL unit. When sh\_cc\_alf\_cr\_enabled\_flag is equal to 1 and sh\_cc\_alf\_cr\_aps\_id is not present, the value of sh\_cc\_alf\_cr\_aps\_id is inferred to be equal to the value of ph\_cc\_alf\_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\_cc\_alf\_cr\_aps\_id shall be equal to 1.

**sh\_lmcs\_enabled\_flag** equal to 1 specifies that luma mapping is enabled for the current slice [Ed. (YK): When this flag is equal to 1, which is more accurate between "luma mapping is enabled for the current slice " and "luma mapping is used for the current slice"?] and chroma scaling may be enabled for the current slice (depending on the value of ph\_chroma\_residual\_scale\_flag). sh\_lmcs\_enabled\_flag equal to 0 specifies that luma mapping with chroma scaling is disabled and not used for the current slice. When sh\_lmcs\_enabled\_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, which is signalled in the referenced scaling list APS with aps\_params\_type equal to SCALING\_APS and aps\_adaptation\_parameter\_set\_id equal to ph\_scaling\_list\_aps\_id, is used in the scaling process for transform coefficients when decoding of the current slice. [Ed. (YK): When this flag is equal to 1, which is more accurate, "is enabled for the current slice" or "is used for 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 of 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\_colour\_plane\_id** identifies the colour plane associated with the current slice when sps\_separate\_colour\_plane\_flag is equal to 1. The value of sh\_colour\_plane\_id shall be in the range of 0 to 2, inclusive. sh\_colour\_plane\_id values 0, 1 and 2 correspond to the Y, Cb and Cr planes, respectively. The value 3 of sh\_colour\_plane\_id is reserved for future use by ITU-T | ISO/IEC.

NOTE 1 – There is no dependency between the decoding processes of different colour planes of one picture.

**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 and the syntax element sh\_num\_ref\_idx\_active\_minus1[ 1 ] is present for B slices. sh\_num\_ref\_idx\_active\_override\_flag equal to 0 specifies that the syntax elements sh\_num\_ref\_idx\_active\_minus1[ 0 ] and sh\_num\_ref\_idx\_active\_minus1[ 1 ] are not present. When not present, the value of sh\_num\_ref\_idx\_active\_override\_flag is inferred to be equal to 1.

**sh\_num\_ref\_idx\_active\_minus1**[ i ] is used for the derivation of the variable NumRefIdxActive[ i ] as specified by Equation 143. 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 (143)  
 else {  
 if( num\_ref\_entries[ i ][ RplsIdx[ i ] ] >= pps\_num\_ref\_idx\_default\_active\_minus1[ i ] + 1 )  
 NumRefIdxActive[ i ] = pps\_num\_ref\_idx\_default\_active\_minus1[ i ] + 1  
 else  
 NumRefIdxActive[ i ] = num\_ref\_entries[ i ][ RplsIdx[ i ] ]  
 }  
 } else /\* sh\_slice\_type = = I | | ( sh\_slice\_type = = P && i = = 1 ) \*/  
 NumRefIdxActive[ i ] = 0  
}

The value of NumRefIdxActive[ i ] − 1 specifies the maximum reference index for reference picture list i that may be used to decode the slice. When the value of NumRefIdxActive[ i ] is equal to 0, no reference index for reference picture list i may be 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 reference picture list 0. sh\_collocated\_from\_l0\_flag equal to 0 specifies that the collocated picture used for temporal motion vector prediction is derived from reference picture list 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 reference picture list 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 reference picture list 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.

It is a requirement of bitstream conformance that the picture referred to by sh\_collocated\_ref\_idx shall be the same for all slices of a coded picture and RprConstraintsActive[ sh\_collocated\_from\_l0\_flag ? 0 : 1 ][ sh\_collocated\_ref\_idx ] shall be equal to 0.

NOTE – The above constraint requires the collocated picture to have the same spatial resolution and the same scaling window offsets as the current picture.

**sh\_qp\_delta** specifies the initial value of QpY 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 QpY quantization parameter for the slice, SliceQpY, is derived as follows:

SliceQpY = 26 + pps\_init\_qp\_minus26 + sh\_qp\_delta (144)

The value of SliceQpY shall be in the range of −QpBdOffset to +63, inclusive.

When either of the following conditions is true, the value of NumRefIdxActive[ 0 ] shall be less than or equal to the value of NumWeightsL0:

* The value of pps\_wp\_info\_in\_ph\_flag is equal to 1, pps\_weighted\_pred\_flag is equal to 1, and sh\_slice\_type is equal to P.
* The value of pps\_wp\_info\_in\_ph\_flag is equal to 1, pps\_weighted\_bipred\_flag is equal to 1, and sh\_slice\_type is equal to B.

When pps\_wp\_info\_in\_ph\_flag is equal to 1, pps\_weighted\_bipred\_flag is equal to 1, and sh\_slice\_type is equal to B, the value of NumRefIdxActive[ 1 ] shall be less than or equal to the value of NumWeightsL1.

When either of the following conditions is true, for each value of i in the range of 0 to NumRefIdxActive[ 0 ] − 1, inclusive, the values of luma\_weight\_l0\_flag[ i ] and chroma\_weight\_l0\_flag[ i ] are both inferred to be equal to 0:

* The value of pps\_wp\_info\_in\_ph\_flag is equal to 1, pps\_weighted\_pred\_flag is equal to 0, and sh\_slice\_type is equal to P.
* The value of pps\_wp\_info\_in\_ph\_flag is equal to 1, pps\_weighted\_bipred\_flag is equal to 0, and sh\_slice\_type is equal to B.

**sh\_cb\_qp\_offset** specifies a difference to be added to the value of pps\_cb\_qp\_offset when determining the value of the Qp′Cb quantization parameter. The value of sh\_cb\_qp\_offset shall be in the range of −12 to +12, inclusive. When sh\_cb\_qp\_offset is not present, it is inferred to be equal to 0. The value of pps\_cb\_qp\_offset + sh\_cb\_qp\_offset shall be in the range of −12 to +12, inclusive.

**sh\_cr\_qp\_offset** specifies a difference to be added to the value of pps\_cr\_qp\_offset when determining the value of the Qp′Cr quantization parameter. The value of sh\_cr\_qp\_offset shall be in the range of −12 to +12, inclusive. When sh\_cr\_qp\_offset is not present, it is inferred to be equal to 0. The value of pps\_cr\_qp\_offset + sh\_cr\_qp\_offset shall be in the range of −12 to +12, inclusive.

**sh\_joint\_cbcr\_qp\_offset** specifies a difference to be added to the value of pps\_joint\_cbcr\_qp\_offset\_value when determining the value of the Qp′CbCr. The value of sh\_joint\_cbcr\_qp\_offset shall be in the range of −12 to +12, inclusive. When sh\_joint\_cbcr\_qp\_offset is not present, it is inferred to be equal to 0. The value of pps\_joint\_cbcr\_qp\_offset\_value + sh\_joint\_cbcr\_qp\_offset shall be in the range of −12 to +12, inclusive.

**sh\_cu\_chroma\_qp\_offset\_enabled\_flag** equal to 1 specifies that the cu\_chroma\_qp\_offset\_flag may 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\_flag** equal to 1 specifies that SAO is enabled for the luma component in the current slice. [Ed. (YK): When this flag is equal to 1, which is more accurate, "is enabled for the current slice" or "is used for the current slice"?] sh\_sao\_luma\_flag equal to 0 specifies that SAO is disabled for the luma component in the current slice. When sh\_sao\_luma\_flag is not present, it is inferred to be equal to ph\_sao\_luma\_enabled\_flag.

**sh\_sao\_chroma\_flag** equal to 1 specifies that SAO is enabled for the chroma component in the current slice. [Ed. (YK): When this flag is equal to 1, which is more accurate, "is enabled for the current slice" or "is used for the current slice"?] sh\_sao\_chroma\_flag equal to 0 specifies that SAO is disabled for the chroma component in the current slice. When sh\_sao\_chroma\_flag is not present, it is inferred to be equal to ph\_sao\_chroma\_enabled\_flag.

**sh\_deblocking\_filter\_override\_flag** equal to 1 specifies that deblocking parameters are present in the slice header. sh\_deblocking\_filter\_override\_flag equal to 0 specifies that deblocking parameters are not present in the slice header. When not present, the value of sh\_deblocking\_filter\_override\_flag is inferred to be equal to 0.

**sh\_deblocking\_filter\_disabled\_flag** equal to 1 specifies that the operation of the deblocking filter is not applied for the current slice. sh\_deblocking\_filter\_disabled\_flag equal to 0 specifies that the operation of the deblocking filter is applied 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\_filter\_override\_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\_filter\_override\_flag is equal to 0), the value of sh\_deblocking\_filter\_disabled\_flag is inferred to be equal to ph\_deblocking\_filter\_disabled\_flag.

[Ed. Consider the naming of the syntax elements for deblocking parameter override in PH and SH to improve clarity.]

**sh\_luma\_beta\_offset\_div2** and **sh\_luma\_tc\_offset\_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the luma component for the current slice. The values of sh\_luma\_beta\_offset\_div2 and sh\_luma\_tc\_offset\_div2 shall both be in the range of −12 to 12, inclusive. When not present, the values of sh\_luma\_beta\_offset\_div2 and sh\_luma\_tc\_offset\_div2 are inferred to be equal to ph\_luma\_beta\_offset\_div2 and ph\_luma\_tc\_offset\_div2, respectively.

**sh\_cb\_beta\_offset\_div2** and **sh\_cb\_tc\_offset\_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for the current slice. The values of sh\_cb\_beta\_offset\_div2 and sh\_cb\_tc\_offset\_div2 shall both be in the range of −12 to 12, inclusive. When not present, the values of sh\_cb\_beta\_offset\_div2 and sh\_cb\_tc\_offset\_div2 are inferred to be equal to sh\_luma\_beta\_offset\_div2 and sh\_luma\_tc\_offset\_div2, respectively.

**sh\_cr\_beta\_offset\_div2** and **sh\_cr\_tc\_offset\_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for the current slice. The values of sh\_cr\_beta\_offset\_div2 and sh\_cr\_tc\_offset\_div2 shall both be in the range of −12 to 12, inclusive. When not present, the values of sh\_cr\_beta\_offset\_div2 and sh\_cr\_tc\_offset\_div2 are inferred to be equal to sh\_luma\_beta\_offset\_div2 and sh\_luma\_tc\_offset\_div2, respectively.

**sh\_dep\_quant\_enabled\_flag** equal to 0 specifies that dependent quantization is disabled for the current slice. sh\_dep\_quant\_enabled\_flag equal to 1 specifies that dependent quantization is enabled for the current slice. [Ed. (YK): When this flag is equal to 1, which is more accurate, "is enabled for the current slice" or "is used for the current slice"?] When sh\_dep\_quant\_enabled\_flag is not present, it is inferred to be equal to 0.

**sh\_sign\_data\_hiding\_enabled\_flag** equal to 0 specifies that sign bit hiding is disabled for the current slice. sh\_sign\_data\_hiding\_enabled\_flag equal to 1 specifies that sign bit hiding is enabled for the current slice. [Ed. (YK): When this flag is equal to 1, which is more accurate, "is enabled for the current slice" or "is used for the current slice"?] When sh\_sign\_data\_hiding\_enabled\_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 infered 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. The value of sh\_slice\_header\_extension\_length shall be in the range of 0 to 256, inclusive. When not present, the value of sh\_slice\_header\_extension\_length is inferred to be equal to 0.

**sh\_slice\_header\_extension\_data\_byte**[ i ] may have any value. Decoders conforming to this version of this Specification shall ignore the values of all the sh\_slice\_header\_extension\_data\_byte[ i ] syntax elements. Its value does not affect decoder conformance to profiles specified in this version of specification.[Ed. (RSS): When profiles have been defined in Annex A, consider changing the last sentence to "Its value does not affect decoder conformance to profiles specified in Annex A".]

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 (145) prevCtbAddrX = CtbAddrInCurrSlice[ i − 1 ] % PicWidthInCtbsY  
 prevCtbAddrY = CtbAddrInCurrSlice[ i − 1 ] / PicWidthInCtbsY  
 if( CtbToTileRowBd[ ctbAddrY ] != CtbToTileRowBd[ prevCtbAddrY ] | |  
 CtbToTileColBd[ ctbAddrX ] != CtbToTileColBd[ prevCtbAddrX ] | |  
 ( ctbAddrY != prevCtbAddrY && sps\_entropy\_coding\_sync\_enabled\_flag ) )  
 NumEntryPoints++  
 }

**sh\_entry\_offset\_len\_minus1** plus 1 specifies the length, in bits, of the sh\_entry\_point\_offset\_minus1[ i ] syntax elements. The value of sh\_entry\_offset\_len\_minus1 shall be in the range of 0 to 31, inclusive.

**sh\_entry\_point\_offset\_minus1**[ i ] plus 1 specifies the i-th entry point offset in bytes, and is represented by sh\_entry\_offset\_len\_minus1 plus 1 bits. The slice data that follow the slice header consists of NumEntryPoints + 1 subsets, with subset index values ranging from 0 to NumEntryPoints, inclusive. The first byte of the slice data is considered byte 0. When present, emulation prevention bytes that appear in the slice data portion of the coded slice NAL unit are counted as part of the slice data for purposes of subset identification. Subset 0 consists of bytes 0 to sh\_entry\_point\_offset\_minus1[ 0 ], inclusive, of the coded slice data, subset k, with k in the range of 1 to NumEntryPoints − 1, inclusive, consists of bytes firstByte[ k ] to lastByte[ k ], inclusive, of the coded slice data with firstByte[ k ] and lastByte[ k ] derived as follows:

(146)

lastByte[ k ] = firstByte[ k ] + sh\_entry\_point\_offset\_minus1[ k ] (147)

The last subset (with subset index equal to NumEntryPoints) consists of the remaining bytes of the coded slice data.

When sps\_entropy\_coding\_sync\_enabled\_flag is equal to 0 and the slice contains one or more complete tiles, each subset shall consist of all coded bits of all CTUs in the slice that are within the same tile, and the number of subsets (i.e., the value of NumEntryPoints + 1) shall be equal to the number of tiles in the slice.

When sps\_entropy\_coding\_sync\_enabled\_flag is equal to 0 and the slice contains a subset of CTU rows from a single tile, the NumEntryPoints shall be 0, and the number of subsets shall be 1. The subset shall consist of all coded bits of all CTUs in the slice.

When sps\_entropy\_coding\_sync\_enabled\_flag is equal to 1, each subset k with k in the range of 0 to NumEntryPoints, inclusive, shall consist of all coded bits of all CTUs in a CTU row within a tile, and the number of subsets ( i.e., the value of NumEntryPoints + 1 ) shall be equal to the total number of tile-specific CTU rows in the slice.

#### 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 reference picture list 0 when pps\_wp\_info\_in\_ph\_flag is equal to 1. The value of num\_l0\_weights shall be in the range of 0 to Min( 15, num\_ref\_entries[ 0 ][ RplsIdx[ 0 ] ] ), inclusive.

If pps\_wp\_info\_in\_ph\_flag is equal to 1, the variable NumWeightsL0 is set equal to num\_l0\_weights. Otherwise (pps\_wp\_info\_in\_ph\_flag is equal to 0), NumWeightsL0 is set equal to NumRefIdxActive[ 0 ].

**luma\_weight\_l0\_flag**[ i ] equal to 1 specifies that weighting factors for the luma component of list 0 prediction using RefPicList[ 0 ][ i ] are present. luma\_weight\_l0\_flag[ i ] equal to 0 specifies that these weighting factors are not present.

**chroma\_weight\_l0\_flag**[ i ] equal to 1 specifies that weighting factors for the chroma prediction values of list 0 prediction using RefPicList[ 0 ][ i ] are present. chroma\_weight\_l0\_flag[ i ] equal to 0 specifies that these weighting factors are not present. When chroma\_weight\_l0\_flag[ i ] is not present, it is inferred to be equal to 0.

**delta\_luma\_weight\_l0**[ i ] is the difference of the weighting factor applied to the luma prediction value for list 0 prediction using RefPicList[ 0 ][ i ].

The variable LumaWeightL0[ i ] is derived to be equal to ( 1  <<  luma\_log2\_weight\_denom ) + delta\_luma\_weight\_l0[ i ]. When luma\_weight\_l0\_flag[ i ] is equal to 1, the value of delta\_luma\_weight\_l0[ i ] shall be in the range of −128 to 127, inclusive. When luma\_weight\_l0\_flag[ i ]is equal to 0, LumaWeightL0[ i ] is inferred to be equal to 2luma\_log2\_weight\_denom.

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

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

The variable ChromaWeightL0[ i ][ j ] is derived to be equal to ( 1  <<  ChromaLog2WeightDenom ) + delta\_chroma\_weight\_l0[ i ][ j ]. When chroma\_weight\_l0\_flag[ i ] is equal to 1, the value of delta\_chroma\_weight\_l0[ i ][ j ] shall be in the range of −128 to 127, inclusive. When chroma\_weight\_l0\_flag[ i ] is equal to 0**,** ChromaWeightL0[ i ][ j ] is inferred to be equal to 2ChromaLog2WeightDenom.

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

The variable ChromaOffsetL0[ i ][ j ] is derived as follows:

ChromaOffsetL0[ i ][ j ] = Clip3( −128, 127,  
 ( 128 + delta\_chroma\_offset\_l0[ i ][ j ] − (148)  
 ( ( 128 \* ChromaWeightL0[ i ][ j ] )  >>  ChromaLog2WeightDenom ) ) )

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

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

The variable NumWeightsL1 is derived as follows:

if( !pps\_weighted\_bipred\_flag | |  
 ( pps\_wp\_info\_in\_ph\_flag && num\_ref\_entries[ 1 ][ RplsIdx[ 1 ] ] = = 0 ) )  
 NumWeightsL1 = 0  
else if( pps\_wp\_info\_in\_ph\_flag ) (149)  
 NumWeightsL1 = num\_l1\_weights  
else  
 NumWeightsL1 = NumRefIdxActive[ 1 ]

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

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

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

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

It is a requirement of bitstream conformance that, when sh\_slice\_type is equal to P, sumWeightL0Flags shall be less than or equal to 24 and when sh\_slice\_type is equal to B, the sum of sumWeightL0Flags and sumWeightL1Flags shall be less than or equal to 24.

### Reference picture lists semantics

The ref\_pic\_lists( ) syntax structure may be present in the PH syntax structure or the slice header.

**rpl\_sps\_flag**[ i ] equal to 1 specifies that reference picture list i in ref\_pic\_lists( ) is derived based on one of the ref\_pic\_list\_struct( listIdx, rplsIdx ) syntax structures with listIdx equal to i in the SPS. rpl\_sps\_flag[ i ] equal to 0 specifies that reference picture list i of the picture is derived based on the ref\_pic\_list\_struct( listIdx, rplsIdx ) syntax structure with listIdx equal to i that is directly included in ref\_pic\_lists( ).

When rpl\_sps\_flag[ i ] is not present, the following applies:

* If sps\_num\_ref\_pic\_lists[ i ] is equal to 0, the value of rpl\_sps\_flag[ i ] is inferred to be equal to 0.
* Otherwise (sps\_num\_ref\_pic\_lists[ i ] is greater than 0), when pps\_rpl1\_idx\_present\_flag is equal to 0 and i is equal to 1, the value of rpl\_sps\_flag[ 1 ] is inferred to be equal to rpl\_sps\_flag[ 0 ].

**rpl\_idx**[ i ] specifies the index, into the list of the ref\_pic\_list\_struct( listIdx, rplsIdx ) syntax structures with listIdx equal to i included in the SPS, of the ref\_pic\_list\_struct( listIdx, rplsIdx ) syntax structure with listIdx equal to i that is used for derivation of reference picture list i of the current picture. The syntax element rpl\_idx[ i ] is represented by Ceil( Log2( sps\_num\_ref\_pic\_lists[ i ] ) ) bits. The value of rpl\_idx[ i ] shall be in the range of 0 to sps\_num\_ref\_pic\_lists[ i ] − 1, inclusive. When not present, if rpl\_sps\_flag[ i ] is equal to 1 and pps\_rpl1\_idx\_present\_flag is equal to 0, the value of rpl\_idx[ 1 ] is inferred to be equal to rpl\_idx[ 0 ], otherwise the value of rpl\_idx[ 1 ] is inferred to be equal to 0.

The variable RplsIdx[ i ] is derived as follows:

RplsIdx[ i ] = rpl\_sps\_flag[ i ] ? rpl\_idx[ i ] : sps\_num\_ref\_pic\_lists[ i ] (150)

**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 reference picture list in the ref\_pic\_lists( ) syntax structure. The length of the poc\_lsb\_lt[ i ][ j ] syntax element is sps\_log2\_max\_pic\_order\_cnt\_lsb\_minus4 + 4 bits.

The variable PocLsbLt[ i ][ j ] is derived as follows:

PocLsbLt[ i ][ j ] = ltrp\_in\_header\_flag[ i ][ RplsIdx[ i ] ] ? (151)  
 poc\_lsb\_lt[ i ][ j ] : rpls\_poc\_lsb\_lt[ listIdx ][ RplsIdx[ i ] ][ j ]

**delta\_poc\_msb\_cycle\_present\_flag**[ i ][ j ] equal to 1 specifies that delta\_poc\_msb\_cycle\_lt[ i ][ j ] is present. delta\_poc\_msb\_cycle\_present\_flag[ i ][ j ]equal to 0 specifies that delta\_poc\_msb\_cycle\_lt[ i ][ j ] is not present.

Let prevTid0Pic be the previous picture in decoding order that has nuh\_layer\_id the same as the slice or picture header referring to the ref\_pic\_lists( ) syntax structure, has TemporalId 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 (152)  
 deltaPocMsbCycleLt[ i ][ j ] = delta\_poc\_msb\_cycle\_lt[ i ][ j ] + deltaPocMsbCycleLt[ i ][ j − 1 ]  
FullPocLt[ i ][ j ] = PicOrderCntVal – deltaPocMsbCycleLt[ i ][ j ] \* MaxPicOrderCntLsb −  
 ( PicOrderCntVal & ( MaxPicOrderCntLsb − 1 ) ) + PocLsbLt[ i ][ j ]

The value of delta\_poc\_msb\_cycle\_lt[ i ][ j ] shall be in the range of 0 to 2(32 − 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.

### Reference picture list structure semantics

The ref\_pic\_list\_struct( listIdx, rplsIdx ) syntax structure may be present in an SPS, in a PH syntax structure, or in a slice header. Depending on whether the syntax structure is included in an SPS, a PH syntax structure, or a slice header, the following applies:

– If present in a PH syntax structure or slice header, the ref\_pic\_list\_struct( listIdx, rplsIdx ) syntax structure specifies reference picture list listIdx of the current picture (i.e., the coded picture containing the PH syntax structure or slice header).

– Otherwise (present in an SPS), the ref\_pic\_list\_struct( listIdx, rplsIdx ) syntax structure specifies a candidate for reference picture list listIdx, and the term "the current picture" in the semantics specified in the remainder of this clause refers to each picture that 1) has a PH syntax structure or one or more slices containing rpl\_idx[ listIdx ] equal to an index into the list of the ref\_pic\_list\_struct( listIdx, rplsIdx ) syntax structures included in the SPS, and 2) is in a CLVS that refers to the SPS.

**num\_ref\_entries**[ listIdx ][ rplsIdx ] specifies the number of entries in the ref\_pic\_list\_struct( listIdx, rplsIdx ) syntax structure. The value of num\_ref\_entries[ listIdx ][ rplsIdx ] shall be in the range of 0 to MaxDpbSize + 13, inclusive, where MaxDpbSize is as specified in clause A.4.2.

**ltrp\_in\_header\_flag**[ listIdx ][ rplsIdx ] equal to 0 specifies that the POC LSBs of the LTRP entries indicated in the ref\_pic\_list\_struct( listIdx, rplsIdx ) syntax structure are present in the same syntax structure. ltrp\_in\_header\_flag[ listIdx ][ rplsIdx ] equal to 1 specifies that the POC LSBs of the LTRP entries indicated in the ref\_pic\_list\_struct( listIdx, rplsIdx ) syntax structure are not present in the same syntax structure. When sps\_long\_term\_ref\_pics\_flag is equal to 1 and rplsIdx is equal to sps\_num\_ref\_pic\_lists[ listIdx ], the vlaue of ltrp\_in\_header\_flag[ listIdx ][ rplsIdx ] is inferred to be equal to 1.

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

**st\_ref\_pic\_flag**[ listIdx ][ rplsIdx ][ i ] equal to 1 specifies that the i-th entry in the ref\_pic\_list\_struct( listIdx, rplsIdx ) syntax structure is an STRP entry. st\_ref\_pic\_flag[ listIdx ][ rplsIdx ][ i ] equal to 0 specifies that the i-th entry in the ref\_pic\_list\_struct( listIdx, rplsIdx ) syntax structure is an LTRP entry. When inter\_layer\_ref\_pic\_flag[ listIdx ][ rplsIdx ][ i ] is equal to 0 and st\_ref\_pic\_flag[ listIdx ][ rplsIdx ][ i ] is not present, the value of st\_ref\_pic\_flag[ listIdx ][ rplsIdx ][ i ] is inferred to be equal to 1.

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

for( i = 0, NumLtrpEntries[ listIdx ][ rplsIdx ] = 0; i < num\_ref\_entries[ listIdx ][ rplsIdx ]; i++ )  
 if( !inter\_layer\_ref\_pic\_flag[ listIdx ][ rplsIdx ][ i ]  &&  !st\_ref\_pic\_flag[ listIdx ][ rplsIdx ][ i ] ) (153)  
 NumLtrpEntries[ listIdx ][ rplsIdx ]++

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

if( ( sps\_weighted\_pred\_flag | | sps\_weighted\_bipred\_flag ) && i != 0 )  
 AbsDeltaPocSt[ listIdx ][ rplsIdx ][ i ] = abs\_delta\_poc\_st[ listIdx ][ rplsIdx ][ i ] (154)  
else  
 AbsDeltaPocSt[ listIdx ][ rplsIdx ][ i ] = abs\_delta\_poc\_st[ listIdx ][ rplsIdx ][ i ] + 1

The value of abs\_delta\_poc\_st[ listIdx ][ rplsIdx ][ i ] shall be in the range of 0 to 215 − 1, inclusive.

**strp\_entry\_sign\_flag**[ listIdx ][ rplsIdx ][ i ] equal to 1 specifies that i-th entry in the syntax structure ref\_pic\_list\_struct( listIdx, rplsIdx ) has a value greater than or equal to 0. strp\_entry\_sign\_flag[ listIdx ][ rplsIdx ][ i ] equal to 0 specifies that the i-th entry in the syntax structure ref\_pic\_list\_struct( listIdx, rplsIdx ) has a value less than 0. When not present, the value of strp\_entry\_sign\_flag[ listIdx ][ rplsIdx ][ i ] is inferred to be equal to 1.

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

for( i = 0; i < num\_ref\_entries[ listIdx ][ rplsIdx ]; i++ )  
 if( !inter\_layer\_ref\_pic\_flag[ listIdx ][ rplsIdx ][ i ]  &&  st\_ref\_pic\_flag[ listIdx ][ rplsIdx ][ i ] ) (155)  
 DeltaPocValSt[ listIdx ][ rplsIdx ][ i ] = ( strp\_entry\_sign\_flag[ listIdx ][ rplsIdx ][ i ] ) ?  
 AbsDeltaPocSt[ listIdx ][ rplsIdx ][ i ] : 0 − AbsDeltaPocSt[ listIdx ][ rplsIdx ][ i ]

**rpls\_poc\_lsb\_lt**[ listIdx ][ rplsIdx ][ i ] specifies the value of the picture order count modulo MaxPicOrderCntLsb of the picture referred to by the i-th entry in the ref\_pic\_list\_struct( listIdx, rplsIdx ) syntax structure. The length of the rpls\_poc\_lsb\_lt[ listIdx ][ rplsIdx ][ i ] syntax element is sps\_log2\_max\_pic\_order\_cnt\_lsb\_minus4 + 4 bits.

**ilrp\_idx**[ listIdx ][ rplsIdx ][ i ] specifies the index, to the list of the direct reference layers, of the ILRP of the i-th entry in the ref\_pic\_list\_struct( listIdx, rplsIdx ) syntax structure. The value of ilrp\_idx[ listIdx ][ rplsIdx ][ i ] shall be in the range of 0 to NumDirectRefLayers[ GeneralLayerIdx[ nuh\_layer\_id ] ] − 1, inclusive.

### Slice data semantics

#### 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.

#### Coding tree unit semantics

The CTU is the root node of the coding tree structure.

The array IsAvailable[ cIdx ][ x ][ y ] specifying whether the sample at ( x, y ) is available for use in the derivation process for neighbouring block availability as specified in clause 6.4.4 is initialized as follows for cIdx = 0..2, x = x0..x0 + CtbSizeY − 1, and y = y0..y0 + CtbSizeY − 1:

IsAvailable[ cIdx ][ x ][ y ] = FALSE (156)

**alf\_ctb\_flag**[ cIdx ][ xCtb >> CtbLog2SizeY ][ yCtb >> CtbLog2SizeY ]equal to 1 specifies that the adaptive loop filter is applied to the coding tree block of the colour component indicated by cIdx of the coding tree unit at luma location ( xCtb, yCtb ). alf\_ctb\_flag[ cIdx ][ xCtb >> CtbLog2SizeY ][ yCtb >> CtbLog2SizeY ] equal to 0 specifies that the adaptive loop filter is not applied to the coding tree block of the colour component indicated by cIdx of the coding tree unit at luma location ( xCtb, yCtb ).

When alf\_ctb\_flag[ cIdx ][ xCtb >> CtbLog2SizeY ][ yCtb >> CtbLog2SizeY ] is not present, it is inferred to be equal to 0.

**alf\_use\_aps\_flag** equal to 0 specifies that one of the fixed filter sets is applied to the luma CTB. alf\_use\_aps\_flag equal to 1 specifies that a filter set from an APS is applied to the luma CTB. When alf\_use\_aps\_flag is not present, it is inferred to be equal to 0.

**alf\_luma\_prev\_filter\_idx** specifies the previous filter that is applied to the luma CTB. The value of alf\_luma\_prev\_filter\_idx shall be in a range of 0 to sh\_num\_alf\_aps\_ids\_luma − 1, inclusive. When alf\_luma\_prev\_filter\_idx is not present, it is inferred to be equal to 0.

The variable AlfCtbFiltSetIdxY[ xCtb >> CtbLog2SizeY ][ yCtb >> CtbLog2SizeY ] specifying the filter set index for the luma CTB at location ( xCtb, yCtb ) is derived as follows:

* If alf\_use\_aps\_flag is equal to 0, AlfCtbFiltSetIdxY[ xCtb >> CtbLog2SizeY ][ yCtb >> CtbLog2SizeY ] is set equal to alf\_luma\_fixed\_filter\_idx.
* Otherwise, AlfCtbFiltSetIdxY[ xCtb >> CtbLog2SizeY ][ yCtb >> CtbLog2SizeY ] is set equal to 16 + alf\_luma\_prev\_filter\_idx.

**alf\_luma\_fixed\_filter\_idx** specifies the fixed filter that is applied to the luma CTB. The value of alf\_luma\_fixed\_filter\_idx shall be in a range of 0 to 15, inclusive.

**alf\_ctb\_filter\_alt\_idx**[ chromaIdx ][ xCtb >> CtbLog2SizeY ][ yCtb >> CtbLog2SizeY ] specifies the index of the alternative chroma filter applied to the coding tree block of the chroma component, with chromaIdx equal to 0 for Cb and chromaIdx equal 1 for Cr, of the coding tree unit at luma location ( xCtb, yCtb ). When alf\_ctb\_filter\_alt\_idx[ chromaIdx ][ xCtb >> CtbLog2SizeY ][ yCtb >> CtbLog2SizeY ] is not present, it is infered 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\_cc\_cb\_idc[ xCtb >> CtbLog2SizeY ][ yCtb >> CtbLog2SizeY ] not equal to 0, alf\_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\_cc\_cr\_idc[ xCtb >> CtbLog2SizeY ][ yCtb >> CtbLog2SizeY ] not equal to 0, alf\_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.

#### Sample adaptive offset semantics

**sao\_merge\_left\_flag** equal to 1 specifies that the syntax elements sao\_type\_idx\_luma, sao\_type\_idx\_chroma, sao\_band\_position, sao\_eo\_class\_luma, sao\_eo\_class\_chroma, sao\_offset\_abs and sao\_offset\_sign\_flag are derived from the corresponding syntax elements of the left CTB. sao\_merge\_left\_flag equal to 0 specifies that these syntax elements are not derived from the corresponding syntax elements of the left CTB. When sao\_merge\_left\_flag is not present, it is inferred to be equal to 0.

**sao\_merge\_up\_flag** equal to 1 specifies that the syntax elements sao\_type\_idx\_luma, sao\_type\_idx\_chroma, sao\_band\_position, sao\_eo\_class\_luma, sao\_eo\_class\_chroma, sao\_offset\_abs and sao\_offset\_sign\_flag are derived from the corresponding syntax elements of the above CTB. sao\_merge\_up\_flag equal to 0 specifies that these syntax elements are not derived from the corresponding syntax elements of the above CTB. When sao\_merge\_up\_flag is not present, it is inferred to be equal to 0.

**sao\_type\_idx\_luma** specifies the offset type for the luma component. The array SaoTypeIdx[ cIdx ][ rx ][ ry ] specifies the offset type as specified in Table 10 for the CTB at the location ( rx, ry ) for the colour component cIdx. The value of SaoTypeIdx[ 0 ][ rx ][ ry ] is derived as follows:

* If sao\_type\_idx\_luma is present, SaoTypeIdx[ 0 ][ rx ][ ry ] is set equal to sao\_type\_idx\_luma.
* Otherwise (sao\_type\_idx\_luma is not present), SaoTypeIdx[ 0 ][ rx ][ ry ] is derived as follows:
* If sao\_merge\_left\_flag is equal to 1, SaoTypeIdx[ 0 ][ rx ][ ry ] is set equal to SaoTypeIdx[ 0 ][ rx − 1 ][ ry ].
* Otherwise, if sao\_merge\_up\_flag is equal to 1, SaoTypeIdx[ 0 ][ rx ][ ry ] is set equal to SaoTypeIdx[ 0 ][ rx ][ ry − 1 ].
* Otherwise, SaoTypeIdx[ 0 ][ rx ][ ry ] is set equal to 0.

**sao\_type\_idx\_chroma** specifies the offset type for the chroma components. The values of SaoTypeIdx[ cIdx ][ rx ][ ry ] are derived as follows for cIdx equal to 1..2:

* If sao\_type\_idx\_chroma is present, SaoTypeIdx[ cIdx ][ rx ][ ry ] is set equal to sao\_type\_idx\_chroma.
* Otherwise (sao\_type\_idx\_chroma is not present), SaoTypeIdx[ cIdx ][ rx ][ ry ] is derived as follows:
* If sao\_merge\_left\_flag is equal to 1, SaoTypeIdx[ cIdx ][ rx ][ ry ] is set equal to SaoTypeIdx[ cIdx ][ rx − 1 ][ ry ].
* Otherwise, if sao\_merge\_up\_flag is equal to 1, SaoTypeIdx[ cIdx ][ rx ][ ry ] is set equal to SaoTypeIdx[ cIdx ][ rx ][ ry − 1 ].
* Otherwise, SaoTypeIdx[ cIdx ][ rx ][ ry ] is set equal to 0.

Table 10 – Specification of the SAO type

|  |  |
| --- | --- |
| **SaoTypeIdx[ cIdx ][ rx ][ ry ]** | **SAO type (informative)** |
| 0 | Not applied |
| 1 | Band offset |
| 2 | Edge offset |

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

When sao\_offset\_abs[ cIdx ][ rx ][ ry ][ i ] is not present, it is inferred as follows:

* If sao\_merge\_left\_flag is equal to 1, sao\_offset\_abs[ cIdx ][ rx ][ ry ][ i ] is inferred to be equal to sao\_offset\_abs[ cIdx ][ rx − 1 ][ ry ][ i ].
* Otherwise, if sao\_merge\_up\_flag is equal to 1, sao\_offset\_abs[ cIdx ][ rx ][ ry ][ i ] is inferred to be equal to sao\_offset\_abs[ cIdx ][ rx ][ ry − 1 ][ i ].
* Otherwise, sao\_offset\_abs[ cIdx ][ rx ][ ry ][ i ] is inferred to be equal to 0.

**sao\_offset\_sign\_flag**[ cIdx ][ rx ][ ry ][ i ] specifies the sign of the offset value of i-th category for the CTB at the location ( rx, ry ) for the colour component cIdx.

When sao\_offset\_sign\_flag[ cIdx ][ rx ][ ry ][ i ] is not present, it is inferred as follows:

* If sao\_merge\_left\_flag is equal to 1, sao\_offset\_sign\_flag[ cIdx ][ rx ][ ry ][ i ] is inferred to be equal to sao\_offset\_sign\_flag[ cIdx ][ rx − 1 ][ ry ][ i ].
* Otherwise, if sao\_merge\_up\_flag is equal to 1, sao\_offset\_sign\_flag[ cIdx ][ rx ][ ry ][ i ] is inferred to be equal to sao\_offset\_sign\_flag[ cIdx ][ rx ][ ry − 1 ][ i ].
* Otherwise, if SaoTypeIdx[ cIdx ][ rx ][ ry ] is equal to 2, the following applies:
* If i is equal to 0 or 1, sao\_offset\_sign\_flag[ cIdx ][ rx ][ ry ][ i ] is inferred to be equal 0.
* Otherwise (i is equal to 2 or 3), sao\_offset\_sign\_flag[ cIdx ][ rx ][ ry ][ i ] is inferred to be equal 1.
* Otherwise, sao\_offset\_sign\_flag[ cIdx ][ rx ][ ry ][ i ] is inferred to be equal 0.

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

SaoOffsetVal[ cIdx ][ rx ][ ry ][ 0 ] = 0  
for( i = 0; i < 4; i++ )  
SaoOffsetVal[ cIdx ][ rx ][ ry ][ i + 1 ] = ( 1 − 2 \* sao\_offset\_sign\_flag[ cIdx ][ rx ][ ry ][ i ] ) \* (157)  
 ( sao\_offset\_abs[ cIdx ][ rx ][ ry ][ i ] << ( BitDepth − Min( 10, BitDepth ) ) )

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

When sao\_band\_position[ cIdx ][ rx ][ ry ] is not present, it is inferred as follows:

* If sao\_merge\_left\_flag is equal to 1, sao\_band\_position[ cIdx ][ rx ][ ry ] is inferred to be equal to sao\_band\_position[ cIdx ][ rx − 1 ][ ry ].
* Otherwise, if sao\_merge\_up\_flag is equal to 1, sao\_band\_position[ cIdx ][ rx ][ ry ] is inferred to be equal to sao\_band\_position[ cIdx ][ rx ][ ry − 1 ].
* Otherwise, sao\_band\_position[ cIdx ][ rx ][ ry ] is inferred to be equal to 0.

**sao\_eo\_class**\_**luma** specifies the edge offset class for the luma component. The array SaoEoClass[ cIdx ][ rx ][ ry ] specifies the offset type as specified in Table 11 for the CTB at the location ( rx, ry ) for the colour component cIdx. The value of SaoEoClass[ 0 ][ rx ][ ry ] is derived as follows:

* If sao\_eo\_class\_luma is present, SaoEoClass[ 0 ][ rx ][ ry ] is set equal to sao\_eo\_class\_luma.
* Otherwise (sao\_eo\_class\_luma is not present), SaoEoClass[ 0 ][ rx ][ ry ] is derived as follows:
* If sao\_merge\_left\_flag is equal to 1, SaoEoClass[ 0 ][ rx ][ ry ] is set equal to SaoEoClass[ 0 ][ rx − 1 ][ ry  ].
* Otherwise, if sao\_merge\_up\_flag is equal to 1, SaoEoClass[ 0 ][ rx ][ ry ] is set equal to SaoEoClass[ 0 ][ rx ][ ry − 1 ].
* Otherwise, SaoEoClass[ 0 ][ rx ][ ry ] is set equal to 0.

**sao\_eo\_class\_chroma** specifies the edge offset class for the chroma components. The values of SaoEoClass[ cIdx ][ rx ][ ry ] are derived as follows for cIdx equal to 1..2:

* If sao\_eo\_class\_chroma is present, SaoEoClass[ cIdx ][ rx ][ ry ] is set equal to sao\_eo\_class\_chroma.
* Otherwise (sao\_eo\_class\_chroma is not present), SaoEoClass[ cIdx ][ rx ][ ry ] is derived as follows:
* If sao\_merge\_left\_flag is equal to 1, SaoEoClass[ cIdx ][ rx ][ ry ] is set equal to SaoEoClass[ cIdx ][ rx − 1 ][ ry  ].
* Otherwise, if sao\_merge\_up\_flag is equal to 1, SaoEoClass[ cIdx ][ rx ][ ry ] is set equal to SaoEoClass[ cIdx ][ rx ][ ry − 1 ].
* Otherwise, SaoEoClass[ cIdx ][ rx ][ ry ] is set equal to 0.

Table 11 – Specification of the SAO edge offset class

|  |  |
| --- | --- |
| **SaoEoClass[ cIdx ][ rx ][ ry ]** | **SAO edge offset class (informative)** |
| 0 | 1D 0-degree edge offset |
| 1 | 1D 90-degree edge offset |
| 2 | 1D 135-degree edge offset |
| 3 | 1D 45-degree edge offset |

#### Coding tree semantics

The variables allowSplitQt, allowSplitBtVer, allowSplitBtHor, allowSplitTtVer, and allowSplitTtHor are derived as follows:

* The allowed quad split process as specified in clause 6.4.1 is invoked with the coding block size cbSize set equal to cbWidth, the current multi-type tree depth mttDepth, treeTypeCurr and modeTypeCurr as inputs, and the output is assigned to allowSplitQt.
* The variables minQtSize, maxBtSize, maxTtSize and maxMttDepth are derived as follows:
* If treeType is equal to DUAL\_TREE\_CHROMA, minQtSize, maxBtSize, maxTtSize and maxMttDepth are set equal to MinQtSizeC, MaxBtSizeC, MaxTtSizeC and MaxMttDepthC + depthOffset, respectively.
* Otherwise, minQtSize, maxBtSize, maxTtSize and maxMttDepth are set equal to MinQtSizeY, MaxBtSizeY, MaxTtSizeY and MaxMttDepthY + depthOffset, respectively.
* The allowed binary split process as specified in clause 6.4.2 is invoked with the binary split mode SPLIT\_BT\_VER, the coding block width cbWidth, the coding block height cbHeight, the location ( x0, y0 ), the current multi-type tree depth mttDepth, the maximum multi-type tree depth with offset maxMttDepth, the maximum binary tree size maxBtSize, the minimum quadtree size minQtSize, the current partition index partIdx, treeTypeCurr and modeTypeCurr as inputs, and the output is assigned to allowSplitBtVer.
* The allowed binary split process as specified in clause 6.4.2 is invoked with the binary split mode SPLIT\_BT\_HOR, the coding block height cbHeight, the coding block width cbWidth, the location ( x0, y0 ), the current multi-type tree depth mttDepth, the maximum multi-type tree depth with offset maxMttDepth, the maximum binary tree size maxBtSize, the minimum quadtree size minQtSize, the current partition index partIdx, treeTypeCurr and modeTypeCurr as inputs, and the output is assigned to allowSplitBtHor.
* The allowed ternary split process as specified in clause 6.4.3 is invoked with the ternary split mode SPLIT\_TT\_VER, the coding block width cbWidth, the coding block height cbHeight, the location ( x0, y0 ), the current multi-type tree depth mttDepth, the maximum multi-type tree depth with offset maxMttDepth, the maximum ternary tree size maxTtSize, treeTypeCurr and modeTypeCurr as inputs, and the output is assigned to allowSplitTtVer.
* The allowed ternary split process as specified in clause 6.4.3 is invoked with the ternary split mode SPLIT\_TT\_HOR, the coding block height cbHeight, the coding block width cbWidth, the location ( x0, y0 ), the current multi-type tree depth mttDepth, the maximum multi-type tree depth with offset maxMttDepth, the maximum ternary tree size maxTtSize, treeTypeCurr and modeTypeCurr as inputs, and the output is assigned to allowSplitTtHor.

**split\_cu\_flag** equal to 0 specifies that a coding unit is not split. split\_cu\_flag equal to 1 specifies that a coding unit is split into four coding units using a quad split as indicated by the syntax element split\_qt\_flag, or into two coding units using a binary split or into three coding units using a ternary split as indicated by the syntax element mtt\_split\_cu\_binary\_flag. The binary or ternary split can be either vertical or horizontal as indicated by the syntax element mtt\_split\_cu\_vertical\_flag.

When split\_cu\_flag is not present, the value of split\_cu\_flag is inferred as follows:

* If one or more of the following conditions are true, the value of split\_cu\_flag is inferred to be equal to 1:
* x0 + cbWidth is greater than pps\_pic\_width\_in\_luma\_samples.
* y0 + cbHeight is greater than pps\_pic\_height\_in\_luma\_samples.
* Otherwise, the value of split\_cu\_flag is inferred to be equal to 0.

**split\_qt\_flag** specifies whether a coding unit is split into coding units with half horizontal and vertical size.

When split\_qt\_flag is not present, the following applies:

* If all of the following conditions are true, split\_qt\_flag is inferred to be equal to 1:
* split\_cu\_flag is equal to 1.
* allowSplitQt, allowSplitBtHor, allowSplitBtVer, allowSplitTtHor and allowSplitTtVer are eqaul 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 as 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 spliting modes indicated by MttSplitMode (informative)

MttSplitMode[ x0 ][ y0 ][ mttDepth ] represents horizontal and vertical binary and ternary splittings of a coding unit within the multi-type tree as illustrated in Figure 8. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

Table 12 – Specification of MttSplitMode[ x ][ y ][ mttDepth ] for x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1

|  |  |  |
| --- | --- | --- |
| **MttSplitMode[ x0 ][ y0 ][ mttDepth ]** | **mtt\_split\_cu\_vertical\_flag** | **mtt\_split\_cu\_binary\_flag** |
| SPLIT\_TT\_HOR | 0 | 0 |
| SPLIT\_BT\_HOR | 0 | 1 |
| SPLIT\_TT\_VER | 1 | 0 |
| SPLIT\_BT\_VER | 1 | 1 |

The variable modeTypeCondition is derived as follows:

* If one or more of the following conditions are true, modeTypeCondition is set equal to 0:
* sh\_slice\_type is equal to I and sps\_qtbtt\_dual\_tree\_intra\_flag is equal to 1.
* modeTypeCurr is not equal to MODE\_TYPE\_ALL.
* sps\_chroma\_format\_idc is equal to 0.
* sps\_chroma\_format\_idc is equal to 3.
* Otherwise, if one 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 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 MttSplitMode[ x0 ][ y0 ][ mttDepth ] is equal to SPLIT\_TT\_VER.
* Otherwise, modeTypeCondition is set equal to 0.

**mode\_constraint\_flag** equal to 0 specifies that coding units inside the current coding tree node can only use inter prediction coding modes. mode\_constraint\_flag equal to 1 specifies that coding units inside the current coding tree node cannot use inter prediction coding modes.

#### 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 (158)

CbPosY[ chType ][ x ][ y ] = y0 (159)

CbWidth[ chType ][ x ][ y ] = cbWidth (160)

CbHeight[ chType ][ x ][ y ] = cbHeight (161)

CqtDepth[ chType ][ x ][ y ] = cqtDepth (162)

The variable MvdLX[ x0 ][ y0 ][ compIdx ], with X = 0..1 and compIdx = 0..1, is set equal to 0.

The variable MvdCpLX[ x0 ][ y0 ][ cpIdx ][ compIdx ], with X = 0..1, cpIdx = 0..2 and compIdx = 0..1, is set equal to 0.

The variable CclmEnabled is derived by invoking the cross-component chroma intra prediction mode checking process specified in clause 8.4.4 with the luma location ( xCb, yCb ) set equal to ( x0, y0 ) as input.

**cu\_skip\_flag**[ x0 ][ y0 ] equal to 1 specifies that for the current coding unit, when decoding a P or B slice, no more syntax elements except one or more of the following are parsed after cu\_skip\_flag[ x0 ][ y0 ]: the IBC mode flag pred\_mode\_ibc\_flag [ x0 ][ y0 ], and the merge\_data( ) syntax structure; when decoding an I slice, no more syntax elements except merge\_idx[ x0 ][ y0 ] are parsed after cu\_skip\_flag[ x0 ][ y0 ]. cu\_skip\_flag[ x0 ][ y0 ] equal to 0 specifies that the coding unit is not skipped. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When cu\_skip\_flag[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

**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 x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1:

* If pred\_mode\_flag is equal to 0, CuPredMode[ chType ][ x ][ y ] is set equal to MODE\_INTER.
* Otherwise (pred\_mode\_flag is equal to 1), CuPredMode[ chType ][ 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 infered 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[ chType ][ x ][ y ] is set to be equal to MODE\_IBC for x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1.

**pred\_mode\_plt\_flag** specifies the use of palette mode in the current coding unit. pred\_mode\_plt\_flag equal to 1 indicates that palette mode is applied in the current coding unit. pred\_mode\_plt\_flag equal to 0 indicates that palette mode is not applied in the current coding unit. When pred\_mode\_plt\_flag is not present, it is inferred to be equal to 0.

When pred\_mode\_plt\_flag is equal to 1, the variable CuPredMode[ chType ][ x ][ y ] is set to be equal to MODE\_PLT for x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1.

**cu\_act\_enabled\_flag** equal to 1 specifies that the residuals of the current coding unit are coded in YCgCo colour space. cu\_act\_enabled\_flag equal to 0 specifies that the residuals of the current coding unit are coded in original colour space. 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 intra luma prediction mode is specified by intra\_bdpcm\_luma\_dir\_flag. intra\_bdpcm\_luma\_flag equal to 0 specifies that BDPCM is not applied to the current luma coding block at the location ( x0, y0 ).

When intra\_bdpcm\_luma\_flag is not present it is inferred to be equal to 0.

The variable BdpcmFlag[ x ][ y ][ cIdx ] is set equal to intra\_bdpcm\_luma\_flag for x = x0..x0 + cbWidth − 1, y = y0..y0 + cbHeight − 1 and cIdx = 0.

**intra\_bdpcm\_luma\_dir\_flag** equal to 0 specifies that the BDPCM prediction direction is horizontal. intra\_bdpcm\_luma\_dir\_flag equal to 1 specifies that the BDPCM prediction direction is vertical.

The variable BdpcmDir[ x ][ y ][ cIdx ] is set equal to intra\_bdpcm\_luma\_dir\_flag for x = x0..x0 + cbWidth − 1, y = y0..y0 + cbHeight − 1 and cIdx = 0.

**intra\_mip\_flag**[ x0 ][ y0 ] equal to 1 specifies that the intra prediction type for luma samples is matrix-based intra prediction. intra\_mip\_flag[ x0 ][ y0 ] equal to 0 specifies that the intra prediction type for luma samples is not matrix-based intra prediction.

When intra\_mip\_flag[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

**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**[ x0 ][ y0 ] specifies the intra prediction reference line index.

When intra\_luma\_ref\_idx[ x0 ][ y0 ] is not present it is inferred to be equal to 0.

The variable IntraLumaRefLineIdx[ x ][ y ] is set equal to intra\_luma\_ref\_idx[ x0 ][ y0 ] for x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1.

**intra\_subpartitions\_mode\_flag**[ x0 ][ y0 ] equal to 1 specifies that the current intra coding unit is partitioned into NumIntraSubPartitions[ x0 ][ y0 ] rectangular transform block subpartitions. intra\_subpartitions\_mode\_flag[ x0 ][ y0 ] equal to 0 specifies that the current intra coding unit is not partitioned into rectangular transform block subpartitions.

When intra\_subpartitions\_mode\_flag[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

**intra\_subpartitions\_split\_flag**[ x0 ][ y0 ] specifies whether the intra subpartitions split type is horizontal or vertical. When intra\_subpartitions\_split\_flag[ x0 ][ y0 ] is not present, it is inferred as follows:

* If cbHeight is greater than MaxTbSizeY, intra\_subpartitions\_split\_flag[ x0 ][ y0 ] is inferred to be equal to 0.
* Otherwise (cbWidth is greater than MaxTbSizeY), intra\_subpartitions\_split\_flag[ x0 ][ y0 ] is inferred to be equal to 1.

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[ x0 ][ y0 ] is equal to 0, IntraSubPartitionsSplitType is set equal to 0.
* Otherwise, the IntraSubPartitionsSplitType is set equal to 1 + intra\_subpartitions\_split\_flag[ x0 ][ y0 ].

**Table 13 – Name association to IntraSubPartitionsSplitType**

|  |  |
| --- | --- |
| IntraSubPartitionsSplitType | Name of IntraSubPartitionsSplitType |
| 0 | ISP\_NO\_SPLIT |
| 1 | ISP\_HOR\_SPLIT |
| 2 | ISP\_VER\_SPLIT |

The variable NumIntraSubPartitions specifies the number of transform block subpartitions into which an intra luma coding block is divided. NumIntraSubPartitions is derived as follows:

* If IntraSubPartitionsSplitType is equal to ISP\_NO\_SPLIT, NumIntraSubPartitions is set equal to 1.
* Otherwise, if one of the following conditions is true, NumIntraSubPartitions is set equal to 2:
  + cbWidth is equal to 4 and cbHeight is equal to 8,
  + cbWidth is equal to 8 and cbHeight is equal to 4.
* Otherwise, NumIntraSubPartitions is set equal to 4.

The syntax elements **intra\_luma\_mpm\_flag**[ x0 ][ y0 ], **intra\_luma\_not\_planar\_flag**[ x0 ][ y0 ], **intra\_luma\_mpm\_idx**[ x0 ][ y0 ] and **intra\_luma\_mpm\_remainder**[ x0 ][ y0 ] specify the intra prediction mode for luma samples. The array indices x0, y0 specify the location ( x0 , y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture. When intra\_luma\_mpm\_flag[ x0 ][ y0 ] is equal to 1, the intra prediction mode is inferred from a neighbouring intra-predicted coding unit according to clause 8.4.2.

When intra\_luma\_mpm\_flag[ x0 ][ y0 ] is not present, it is inferred to be equal to 1.

When intra\_luma\_not\_planar\_flag[ x0 ][ y0 ] is not present, it is inferred to be equal to 1.

**intra\_bdpcm\_chroma\_flag** equal to 1 specifies that BDPCM is applied to the current chroma coding blocks at the location ( x0, y0 ), i.e. the transform is skipped, the intra chroma prediction mode is specified by intra\_bdpcm\_chroma\_dir\_flag. intra\_bdpcm\_chroma\_flag equal to 0 specifies that BDPCM is not applied to the current chroma coding blocks at the location ( x0, y0 ).

When intra\_bdpcm\_chroma\_flag is not present it is inferred to be equal to 0.

The variable BdpcmFlag[ x ][ y ][ cIdx ] is set equal to intra\_bdpcm\_chroma\_flag for x = x0..x0 + cbWidth − 1, y = y0..y0 + cbHeight − 1 and cIdx = 1..2.

**intra\_bdpcm\_chroma\_dir\_flag** equal to 0 specifies that the BDPCM prediction direction is horizontal. intra\_bdpcm\_chroma\_dir\_flag equal to 1 specifies that the BDPCM prediction direction is vertical.

The variable BdpcmDir[ x ][ y ][ cIdx ] is set equal to intra\_bdpcm\_chroma\_dir\_flag for x = x0..x0 + cbWidth − 1, y = y0..y0 + cbHeight − 1 and cIdx = 1..2.

**cclm\_mode\_flag** equal to 1 specifies that one of the INTRA\_LT\_CCLM, INTRA\_L\_CCLM and INTRA\_T\_CCLM intra chroma 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 intra chroma 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 intra chroma prediction modes is applied.

**intra\_chroma\_pred\_mode** specifies the intra prediction mode for chroma samples. When intra\_chroma\_pred\_mode is not present, it is inferred to be equal to 0.

**general\_merge\_flag**[ x0 ][ y0 ] specifies whether the inter prediction parameters for the current coding unit are inferred from a neighbouring inter-predicted partition. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When general\_merge\_flag[ x0 ][ y0 ] is not present, it is inferred as follows:

* If cu\_skip\_flag[ x0 ][ y0 ] is equal to 1, general\_merge\_flag[ x0 ][ y0 ] is inferred to be equal to 1.
* Otherwise, general\_merge\_flag[ x0 ][ y0 ] is inferred to be equal to 0.

**mvp\_l0\_flag**[ x0 ][ y0 ] specifies the motion vector predictor index of list 0 where x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When mvp\_l0\_flag[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

**mvp\_l1\_flag**[ x0 ][ y0 ] has the same semantics as mvp\_l0\_flag, with l0 and list 0 replaced by l1 and list 1, respectively.

**inter\_pred\_idc**[ x0 ][ y0 ] specifies whether list0, list1, or bi-prediction is used for the current coding unit according to Table 14. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

Table 14 – Name association to inter prediction mode

|  |  |  |  |
| --- | --- | --- | --- |
| **inter\_pred\_idc** | **Name of inter\_pred\_idc** | | |
| ( cbWidth + cbHeight )  >  12 | ( cbWidth + cbHeight )  = =  12 | ( cbWidth + cbHeight )  = =  8 |
| 0 | PRED\_L0 | PRED\_L0 | n.a. |
| 1 | PRED\_L1 | PRED\_L1 | n.a. |
| 2 | PRED\_BI | n.a. | n.a. |

When inter\_pred\_idc[ x0 ][ y0 ] is not present, it is inferred to be equal to PRED\_L0.

**sym\_mvd\_flag**[ x0 ][ y0 ] equal to 1 specifies that the syntax elements ref\_idx\_l0[ x0 ][ y0 ] and ref\_idx\_l1[ x0 ][ y0 ], and the mvd\_coding( x0, y0, refList ,cpIdx ) syntax structure for refList equal to 1 are not present. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When sym\_mvd\_flag[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

**ref\_idx\_l0**[ x0 ][ y0 ] specifies the list 0 reference picture index for the current coding unit. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When ref\_idx\_l0[ x0 ][ y0 ] is not present it is inferred as follows:

* If sym\_mvd\_flag[ x0 ][ y0 ] is equal to 1, ref\_idx\_l0[ x0 ][ y0 ] is inferred to be equal to RefIdxSymL0.
* Otherwise (sym\_mvd\_flag[ x0 ][ y0 ] is equal to 0), ref\_idx\_l0[ x0 ][ y0 ] is inferred to be equal to 0.

**ref\_idx\_l1**[ x0 ][ y0 ] has the same semantics as ref\_idx\_l0, with l0, L0 and list 0 replaced by l1, L1 and list 1, respectively.

**inter\_affine\_flag**[ x0 ][ y0 ] equal to 1 specifies that for the current coding unit, when decoding a P or B slice, affine model based motion compensation is used to generate the prediction samples of the current coding unit. inter\_affine\_flag[ x0 ][ y0 ] equal to 0 specifies that the coding unit is not predicted by affine model based motion compensation. When inter\_affine\_flag[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

**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.

MotionModelIdc[ x ][ y ] represents motion model of a coding unit as illustrated in Table 15. The array indices x, y specify the luma sample location ( x, y ) relative to the top-left luma sample of the picture.

The variable MotionModelIdc[ x ][ y ] is derived as follows for x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1:

* If general\_merge\_flag[ x0 ][ y0 ] is equal to 1, the following applies:

MotionModelIdc[ x ][ y ] = merge\_subblock\_flag[ x0 ][ y0 ] (163)

* Otherwise (general\_merge\_flag[ x0 ][ y0 ] is equal to 0), the following applies:

MotionModelIdc[ x ][ y ] = inter\_affine\_flag[ x0 ][ y0 ] + cu\_affine\_type\_flag[ x0 ][ y0 ] (164)

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 motion vector difference. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture. amvr\_flag[ x0 ][ y0 ] equal to 0 specifies that the resolution of the motion vector difference is 1/4 of a luma sample. amvr\_flag[ x0 ][ y0 ] equal to 1 specifies that the resolution of the motion vector difference is further specified by amvr\_precision\_idx[ x0 ][ y0 ].

When amvr\_flag[ x0 ][ y0 ] is not present, it is inferred as follows:

* If CuPredMode[ chType ][ x0 ][ y0 ] is equal to MODE\_IBC, amvr\_flag[ x0 ][ y0 ] is inferred to be equal to 1.
* Otherwise ( CuPredMode[ chType ][ x0 ][ y0 ] is not equal to MODE\_IBC ), amvr\_flag[ x0 ][ y0 ] is inferred to be equal to 0.

**amvr\_precision\_idx**[ x0 ][ y0 ] specifies that the resolution of the motion vector difference with AmvrShift is defined in Table 16. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When amvr\_precision\_idx[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

The motion vector differences are modified as follows:

* If inter\_affine\_flag[ x0 ][ y0 ] is equal to 0, the variables MvdL0[ x0 ][ y0 ][ 0 ], MvdL0[ x0 ][ y0 ][ 1 ], MvdL1[ x0 ][ y0 ][ 0 ], MvdL1[ x0 ][ y0 ][ 1 ] are modified as follows:

MvdL0[ x0 ][ y0 ][ 0 ] = MvdL0[ x0 ][ y0 ][ 0 ]  << AmvrShift (165)

MvdL0[ x0 ][ y0 ][ 1 ] = MvdL0[ x0 ][ y0 ][ 1 ]  <<  AmvrShift (166)

MvdL1[ x0 ][ y0 ][ 0 ] = MvdL1[ x0 ][ y0 ][ 0 ]  <<  AmvrShift (167)

MvdL1[ x0 ][ y0 ][ 1 ] = MvdL1[ x0 ][ y0 ][ 1 ]  <<  AmvrShift (168)

* 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:

MvdCpL0[ x0 ][ y0 ][ 0 ][ 0 ] = MvdCpL0[ x0 ][ y0 ][ 0 ][ 0 ]  <<  AmvrShift (169)

MvdCpL1[ x0 ][ y0 ] [ 0 ][ 1 ] = MvdCpL1[ x0 ][ y0 ][ 0 ][ 1 ]  <<  AmvrShift (170)

MvdCpL0[ x0 ][ y0 ][ 1 ][ 0 ] = MvdCpL0[ x0 ][ y0 ][ 1 ][ 0 ]  <<  AmvrShift (171)

MvdCpL1[ x0 ][ y0 ] [ 1 ][ 1 ] = MvdCpL1[ x0 ][ y0 ][ 1 ][ 1 ]  <<  AmvrShift (172)

MvdCpL0[ x0 ][ y0 ][ 2 ][ 0 ] = MvdCpL0[ x0 ][ y0 ][ 2 ][ 0 ]  <<  AmvrShift (173)

MvdCpL1[ x0 ][ y0 ] [ 2 ][ 1 ] = MvdCpL1[ x0 ][ y0 ][ 2 ][ 1 ]  <<  AmvrShift (174)

Table 16 – Specification of AmvrShift

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| amvr\_flag | amvr\_precision\_idx | AmvrShift | | |
| inter\_affine\_flag = =1 | CuPredMode[ chType ][ x0 ][ y0 ] = = MODE\_IBC ) | inter\_affine\_flag = =0 && CuPredMode[ chType ][ x0 ][ y0 ] != MODE\_IBC |
| 0 | - | 2 (1/4 luma sample) | - | 2 (1/4 luma sample) |
| 1 | 0 | 0 (1/16 luma sample) | 4 (1 luma sample) | 3 (1/2 luma sample) |
| 1 | 1 | 4 (1 luma sample) | 6 (4 luma samples) | 4 (1 luma sample) |
| 1 | 2 | - | - | 6 (4 luma samples) |

**bcw\_idx**[ x0 ][ y0 ] specifies the weight index of bi-prediction with CU weights. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When bcw\_idx[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

**cu\_coded\_flag** equal to 1 specifies that the transform\_tree( ) syntax structure is present for the current coding unit. cu\_coded\_flag equal to 0 specifies that the transform\_tree( ) syntax structure is not present for the current coding unit.

When cu\_coded\_flag is not present, it is inferred as follows:

* If cu\_skip\_flag[ x0 ][ y0 ] is equal to 1 or pred\_mode\_plt\_flag is equal to 1, cu\_coded\_flag is inferred to be equal to 0.
* Otherwise, cu\_coded\_flag is inferred to be equal to 1.

**cu\_sbt\_flag** equal to 1 specifies that for the current coding unit, subblock transform is used. cu\_sbt\_flag equal to 0 specifies that for the current coding unit, subblock transform is not used.

When cu\_sbt\_flag is not present, its value is inferred to be equal to 0.

NOTE – : When subblock transform is used, a coding unit is split into two transform units; one transform unit has residual data, the other does not have residual data.

**cu\_sbt\_quad\_flag** equal to 1 specifies that for the current coding unit, the subblock transform includes a transform unit of 1/4 size of the current coding unit. cu\_sbt\_quad\_flag equal to 0 specifies that for the current coding unit the subblock transform includes a transform unit of 1/2 size of the current coding unit.

When cu\_sbt\_quad\_flag is not present, its value is inferred to be equal to 0.

**cu\_sbt\_horizontal\_flag** equal to 1 specifies that the current coding unit is split horizontally into 2 transform units. cu\_sbt\_horizontal\_flag[ x0 ][ y0 ] equal to 0 specifies that the current coding unit is split vertically into 2 transform units.

When cu\_sbt\_horizontal\_flag is not present, its value is derived as follows:

* If cu\_sbt\_quad\_flag is equal to 1, cu\_sbt\_horizontal\_flag is set to be equal to allowSbtHorQ.
* Otherwise (cu\_sbt\_quad\_flag is equal to 0), cu\_sbt\_horizontal\_flag is set to be equal to allowSbtHorH.

**cu\_sbt\_pos\_flag** equal to 1 specifies that the tu\_y\_coded\_flag, tu\_cb\_coded\_flag and tu\_cr\_coded\_flag of the first transform unit in the current coding unit are not present. cu\_sbt\_pos\_flag equal to 0 specifies that the tu\_y\_coded\_flag, tu\_cb\_coded\_flag and tu\_cr\_coded\_flag of the second transform unit in the current coding unit are not present.

The variable SbtNumFourthsTb0 is derived as follows:

sbtMinNumFourths = cu\_sbt\_quad\_flag ? 1 : 2 (175)

SbtNumFourthsTb0 = cu\_sbt\_pos\_flag ? ( 4 − sbtMinNumFourths ) : sbtMinNumFourths (176)

**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 is derived as follows:

* If treeType is equal to SINGLE\_TREE, the following applies:

ApplyLfnstFlag = ( lfnst\_idx > 0 && cIdx = = 0 ) ? 1 : 0 (177)

* Otherwise, the following applies:

ApplyLfnstFlag = ( lfnst\_idx > 0 ) ? 1 : 0 (178)

**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:

IbcVirBuf[ 0 ][ x ][ y ] = −1 (179)

* 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 + VSize − 1 and y = y0..y0 + VSize − 1:

IbcVirBuf[ 0 ][ ( x + ( IbcBufWidthY >> 1 ) ) % IbcBufWidthY ][ y % CtbSizeY ] = −1 (180)

#### Palette coding semantics

In the following semantics, the array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture. The array indices xC, yC specify the location ( xC, yC ) of the sample relative to the top-left luma sample of the picture, when treeType is equal to SINGLE\_TREE or DUAL\_TREE\_LUMA; and relative to the top-left chroma sample of the picture, when treeType is equal to DUAL\_TREE\_CHROMA. The array index startComp specifies the first colour component of the current palette table. startComp equal to 0 indicates the Y component; startComp equal to 1 indicates the Cb component; startComp equal to 2 indicates the Cr component. numComps specifies the number of colour components in the current palette table.

The predictor palette consists of palette entries from previous coding units that are used to predict the entries in the current palette.

PredictorPaletteSize[ startComp ] specifies the size of the predictor palette for the first colour component of the current palette table startComp. PredictorPaletteSize[ startComp ] is derived as specified in clause 8.4.5.3.

PalettePredictorEntryReuseFlags[ i ] equal to 1specifies that the i-th entry in the predictor palette is reused in the current palette. PalettePredictorEntryReuseFlags[ i ] equal to 0specifies 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:

CurrentPaletteSize[ startComp ] = NumPredictedPaletteEntries + num\_signalled\_palette\_entries (181)

The value of CurrentPaletteSize[ startComp ] shall be in the range of 0 to maxNumPaletteEntries, inclusive.

**new\_palette\_entries**[ cIdx ][ i ] specifies the value for the i-th signalled palette entry for the colour component cIdx.

The variable LocalDualTreeFlag is derived as follows:

LocalDualTreeFlag = ( treeType != SINGLE\_TREE &&  
 ( sh\_slice\_type != I | | ( sh\_slice\_type = = I && sps\_qtbtt\_dual\_tree\_intra\_flag = = 0 ) ) ) ? 1 : 0 (182)

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++) (183)  
 for( i = 0; i < num\_signalled\_palette\_entries[startComp]; i++ )  
 CurrentPaletteEntries[ cIdx ][ numPredictedPaletteEntries + i ] = new\_palette\_entries[ cIdx ][ i ]

**palette\_escape\_val\_present\_flag** equal to 1 specifies that the current coding unit contains at least one escape coded sample. palette\_escape\_val\_present\_flag equal to 0 specifies that there are no escape coded samples in the current coding unit. When not present, the value of palette\_escape\_val\_present\_flag is inferred to be equal to 1.

The variable MaxPaletteIndex specifies the maximum possible value for a palette index for the current coding unit. The value of MaxPaletteIndex is set equal to CurrentPaletteSize[ startComp ] − 1 + palette\_escape\_val\_present\_flag.

**palette\_idx\_idc** is an indication of an index to the palette table, CurrentPaletteEntries. The value of palette\_idx\_idc shall be in the range of 0 to MaxPaletteIndex, inclusive, for the first index in the block and in the range of 0 to ( MaxPaletteIndex − 1 ), inclusive, for the remaining indices in the block.

When palette\_idx\_idc is not present, it is inferred to be equal to 0.

**palette\_transpose\_flag** equal to 1 specifies that vertical traverse scan is applied for scanning the indices for samples in the current coding unit. palette\_transpose\_flag equal to 0 specifies that horizontal traverse scan is applied for scanning the indices for samples in the current coding unit. When not present, the value of palette\_transpose\_flag is inferred to be equal to 0.

The array TraverseScanOrder specifies the scan order array for palette coding. TraverseScanOrder is assigned the horizontal scan order HorTravScanOrder if palette\_transpose\_flag is equal to 0 and TraverseScanOrder is assigned the vertical scan order VerTravScanOrder if if palette\_transpose\_flag is equal to 1.

**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 run index is the same as the index at the previous scanned position if copy\_above\_palette\_indices\_flag 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 ] { (184) else {  
 if( !palette\_transpose\_flag )  
 adjustedRefPaletteIndex = PaletteIndexMap[ xC ][ yC − 1 ]  
 else  
 adjustedRefPaletteIndex = PaletteIndexMap[ xC − 1 ][ yC ]  
 }  
}

When CopyAboveIndicesFlag[ xC ][ yC ] is equal to 0, the variable CurrPaletteIndex is derived as follows:

if( CurrPaletteIndex >= adjustedRefPaletteIndex )  
 CurrPaletteIndex++ (185)

**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 ) ) − 1, inclusive, for cIdx equal to 0, and in the range of 0 to (1  <<  ( BitDepth + 1 ) ) − 1, inclusive, for cIdx not equal to 0.

#### 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.

**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 paramters. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When mmvd\_merge\_flag[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

**mmvd\_cand\_flag**[ x0 ][ y0 ] specifies whether the first (0) or the second (1) candidate in the merging candidate list is used with the motion vector difference derived from mmvd\_distance\_idx[ x0 ][ y0 ] and mmvd\_direction\_idx[ x0 ][ y0 ]. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When mmvd\_cand\_flag[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

**mmvd\_distance\_idx**[ x0 ][ y0 ] specifies the index used to derive MmvdDistance[ x0 ][ y0 ] as specified in Table 17. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

Table 17 – Specification of MmvdDistance[ x0 ][ y0 ] based on mmvd\_distance\_idx[ x0 ][ y0 ]

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

**mmvd\_direction\_idx**[ x0 ][ y0 ] specifies index used to derive MmvdSign[ x0 ][ y0 ] as specified in Table 18. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

Table 18 – Specification of MmvdSign[ x0 ][ y0 ] based on mmvd\_direction\_idx[ x0 ][ y0 ]

|  |  |  |
| --- | --- | --- |
| mmvd\_direction\_idx[ x0 ][ y0 ] | MmvdSign[ x0 ][ y0 ][0] | MmvdSign[ x0 ][ y0 ][1] |
| 0 | +1 | 0 |
| 1 | −1 | 0 |
| 2 | 0 | +1 |
| 3 | 0 | −1 |

Both components of the merge plus MVD offset MmvdOffset[ x0 ][ y0 ] are derived as follows:

MmvdOffset[ x0 ][ y0 ][ 0 ] = ( MmvdDistance[ x0 ][ y0 ] << 2 ) \* MmvdSign[ x0 ][ y0 ][0] (186)

MmvdOffset[ x0 ][ y0 ][ 1 ] = ( MmvdDistance[ x0 ][ y0 ] << 2 ) \* MmvdSign[ x0 ][ y0 ][1] (187)

**ciip\_flag**[ x0 ][ y0 ] specifies whether the combined inter-picture merge and intra-picture prediction is applied for the current coding unit. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When ciip\_flag[ x0 ][ y0 ] is not present, it is inferred as follows:

* If all the following conditions are true, ciip\_flag[ x0 ][ y0 ] is inferred to be equal to 1:
* sps\_ciip\_enabled\_flag is equal to 1.
* general\_merge\_flag[ x0 ][ y0 ] is equal to 1.
* merge\_subblock\_flag[ x0 ][ y0 ] is equal to 0.
* regular\_merge\_flag[ x0 ][ y0 ] is equal to 0.
* cu\_skip\_flag[ x0 ][ y0 ] is equal to 0.
* cbWidth is less than 128.
* cbHeight is less than 128.
* cbWidth \* cbHeight is greater than or equal to 64.
* Otherwise, ciip\_flag[ x0 ][ y0 ] is inferred to be equal to 0.

When ciip\_flag[ x0 ][ y0 ] is equal to 1, the variable IntraPredModeY[ x ][ y ] with x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1 is set to be equal to INTRA\_PLANAR.

The variable MergeGpmFlag[ x0 ][ y0 ], which specifies whether geometric partitioning based motion compensation is used to generate the prediction samples of the current coding unit, when decoding a B slice, is derived as follows:

* If all the following conditions are true, MergeGpmFlag[ x0 ][ y0 ] is set equal to 1:
* sps\_gpm\_enabled\_flag is equal to 1.
* sh\_slice\_type is equal to B.
* general\_merge\_flag[ x0 ][ y0 ] is equal to 1.
* cbWidth is greater than or equal to 8.
* cbHeight is greater than or equal to 8.
* cbWidth is less than 8 \* cbHeight.
* cbHeight is less than 8 \* cbWidth.
* regular\_merge\_flag[ x0 ][ y0 ] is equal to 0.
* merge\_subblock\_flag[ x0 ][ y0 ] is equal to 0.
* ciip\_flag[ x0 ][ y0 ] is equal to 0.
* Otherwise, MergeGpmFlag[ x0 ][ y0 ] is set equal to 0.

**merge\_gpm\_partition\_idx**[ x0 ][ y0 ] specifies the partitioning shape of the geometric partitioning merge mode. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When merge\_gpm\_partition\_idx[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

**merge\_gpm\_idx0**[ x0 ][ y0 ] specifies the first merging candidate index of the geometric partitioning based motion compensation candidate list where x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When merge\_gpm\_idx0[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

**merge\_gpm\_idx1**[ x0 ][ y0 ] specifies the second merging candidate index of the geometric partitioning based motion compensation candidate list where x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When merge\_gpm\_idx1[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

#### 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:

lMvd[ compIdx ] = abs\_mvd\_greater0\_flag[ compIdx ] \*  
 ( abs\_mvd\_minus2[ compIdx ] + 2 ) \* ( 1 − 2 \* mvd\_sign\_flag[ compIdx ] ) (188)

The value of lMvd[ compIdx ] shall be in the range of −217 to 217 − 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 variable MvdLX[ x0 ][ y0 ][ compIdx ], with X being 0 or 1, specifies 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 horizontal motion vector component difference is assigned compIdx = 0 and the vertical motion vector component is assigned compIdx = 1.
* 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 variable MvdCpLX[ x0 ][ y0 ][ cpIdx ][ compIdx ], with X being 0 or 1, specifies 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. The horizontal motion vector component difference is assigned compIdx = 0 and the vertical motion vector component is assigned compIdx = 1.
* 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 MvdL1[ x0 ][ y0 ][ compIdx] shall be in the range of −217 to 217 − 1, inclusive.

#### Transform tree semantics

[Ed. (BB): The transform scheme does not have any syntax for spliting a CU into TUs. However, if the height or width of a CU is larger than the current maximum transform length of 64 luma samples or the corresponding chroma sample length, the CU will be implicitly split to divide it into TUs.]

#### Transform unit semantics

The transform coefficient levels are represented by the arrays TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ]. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered transform block relative to the top-left luma sample of the picture. The array index cIdx specifies an indicator for the colour component; it is equal to 0 for Y, 1 for Cb, and 2 for Cr. The array indices xC and yC specify the transform coefficient location ( xC, yC ) within the current transform block. When the value of TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] is not specified in clause 7.3.10.11, 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 top-left location ( x0, y0 ) of the considered transform block.

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 top-left location ( x0, y0 ) of the considered transform block.

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, 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, if treeType is equal to DUAL\_TREE\_CHROMA, tu\_y\_coded\_flag[ x0 ][ y0 ] is inferred to be equal to 0.
* Otherwise, tu\_y\_coded\_flag[ x0 ][ y0 ] is inferred to be equal to 1.

**tu\_joint\_cbcr\_residual\_flag**[ x0 ][ y0 ] specifies whether the residual samples for both chroma components Cb and Cr are coded as a single transform block. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered transform block relative to the top-left luma sample of the picture.

tu\_joint\_cbcr\_residual\_flag[ x0 ][ y0 ] equal to 1 specifies that the transform unit syntax includes the transform coefficient levels for a single transform block from which the residual samples for both Cb and Cr are derived. tu\_joint\_cbcr\_residual\_flag[ x0 ][ y0 ] equal to 0 specifies that the transform coefficient levels of the chroma components are coded as indicated by the syntax elements tu\_cb\_coded\_flag[ x0 ][ y0 ] and tu\_cr\_coded\_flag[ x0 ][ y0 ].

When tu\_joint\_cbcr\_residual\_flag[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

Depending on tu\_joint\_cbcr\_residual\_flag[ x0 ][ y0 ], tu\_cb\_coded\_flag[ x0 ][ y0 ], and tu\_cr\_coded\_flag[ x0 ][ y0 ], the variable TuCResMode[ x0 ][ y0 ] is derived as follows:

* If tu\_joint\_cbcr\_residual\_flag[ x0 ][ y0 ] is equal to 0, the variable TuCResMode[ x0 ][ y0 ] is set equal to 0.
* Otherwise, if tu\_cb\_coded\_flag[ x0 ][ y0 ] is equal to 1 and tu\_cr\_coded\_flag[ x0 ][ y0 ] is equal to 0, the variable TuCResMode[ x0 ][ y0 ] is set equal to 1.
* Otherwise, if tu\_cb\_coded\_flag[ x0 ][ y0 ] is equal to 1, the variable TuCResMode[ x0 ][ y0 ] is set equal to 2.
* Otherwise, the variable TuCResMode[ x0 ][ y0 ] is set equal to 3.

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

**cu\_qp\_delta\_sign\_flag** specifies the sign of CuQpDeltaVal as follows:

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

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

When cu\_qp\_delta\_abs is present, the variables IsCuQpDeltaCoded and CuQpDeltaVal are derived as follows:

IsCuQpDeltaCoded = 1 (189)

CuQpDeltaVal = cu\_qp\_delta\_abs \* ( 1 − 2 \* cu\_qp\_delta\_sign\_flag ) (190)

The value of CuQpDeltaVal shall be in the range of −( 32 + QpBdOffset / 2 ) to +( 31 + QpBdOffset / 2 ), inclusive.

**cu\_chroma\_qp\_offset\_flag** when present and equal to 1, specifies that an entry in the pps\_cb\_qp\_offset\_list[ ] is used to determine the value of CuQpOffsetCb, a corresponding entry in the pps\_cr\_qp\_offset\_list[ ] is used to determine the value of CuQpOffsetCr, and a corresponding entry in the pps\_joint\_cbcr\_qp\_offset\_list[ ] is used to determine the value of CuQpOffsetCbCr. cu\_chroma\_qp\_offset\_flag equal to 0 specifies that these lists are not used to determine the values of CuQpOffsetCb, CuQpOffsetCr, and CuQpOffsetCbCr.

**cu\_chroma\_qp\_offset\_idx**, when present, specifies the index into the pps\_cb\_qp\_offset\_list[ ], pps\_cr\_qp\_offset\_list[ ], and pps\_joint\_cbcr\_qp\_offset\_list[ ] that is used to determine the value of CuQpOffsetCb, CuQpOffsetCr, and CuQpOffsetCbCr. When present, the value of cu\_chroma\_qp\_offset\_idx shall be in the range of 0 to pps\_chroma\_qp\_offset\_list\_len\_minus1, inclusive. When not present, the value of cu\_chroma\_qp\_offset\_idx is inferred to be equal to 0.

When cu\_chroma\_qp\_offset\_flag is present, the following applies:

* The variable IsCuChromaQpOffsetCoded is set equal to 1.
* The variables CuQpOffsetCb, CuQpOffsetCr, and CuQpOffsetCbCr are derived as follows:
* If cu\_chroma\_qp\_offset\_flag is equal to 1, the following applies:

CuQpOffsetCb = pps\_cb\_qp\_offset\_list[ cu\_chroma\_qp\_offset\_idx ] (191)

CuQpOffsetCr = pps\_cr\_qp\_offset\_list[ cu\_chroma\_qp\_offset\_idx ] (192)

CuQpOffsetCbCr = pps\_joint\_cbcr\_qp\_offset\_list[ cu\_chroma\_qp\_offset\_idx ] (193)

* Otherwise (cu\_chroma\_qp\_offset\_flag is equal to 0), CuQpOffsetCb, CuQpOffsetCr, and CuQpOffsetCbCr are all set equal to 0.

**transform\_skip\_flag**[ x0 ][ y0 ][ cIdx ] specifies whether a transform is applied to the associated transform block or not. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered transform block relative to the top-left luma sample of the picture. The array index cIdx specifies an indicator for the colour component; it is equal to 0 for Y, 1 for Cb, and 2 for Cr. transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] equal to 1 specifies that no transform is applied to the associated transform block. transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] equal to 0 specifies that the decision whether transform is applied to the associated transform block or not depends on other syntax elements.

When transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] is not present, it is inferred as follows:

* If BdpcmFlag[ x0 ][ y0 ][ cIdx ] is equal to 1, transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] is inferred to be equal to 1.
* Otherwise (BdpcmFlag[ x0 ][ y0 ][ cIdx ] is equal to 0), transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] is inferred to be equal to 0.

#### Residual coding semantics

The array AbsLevel[ xC ][ yC ] represents an array of absolute values of transform coefficient levels for the current transform block and the array AbsLevelPass1[ xC ][ yC ] represents an array of partially reconstructed absolute values of transform coefficient levels for the current transform block. The array indices xC and yC specify the transform coefficient location ( xC, yC ) within the current transform block. When the value of AbsLevel[ xC ][ yC ] is not specified in clause 7.3.10.11, it is inferred to be equal to 0. When the value of AbsLevelPass1[ xC ][ yC ] is not specified in clause 7.3.10.11, 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:

CoeffMin = −( 1 << 15 ) (194)

CoeffMax = ( 1 << 15 ) − 1 (195)

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

QStateTransTable[ ][ ] = { { 0, 2 }, { 2, 0 }, { 1, 3 }, { 3, 1 } } (196)

**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:

LastSignificantCoeffX = last\_sig\_coeff\_x\_prefix (197)

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

LastSignificantCoeffX = ( 1  <<  ( (last\_sig\_coeff\_x\_prefix  >>  1 ) − 1 ) ) \* (198)  
 ( 2 + (last\_sig\_coeff\_x\_prefix & 1 ) ) + last\_sig\_coeff\_x\_suffix

**last\_sig\_coeff\_y\_suffix** specifies the suffix of the row position of the last significant coefficient in scanning order within a transform block. The values of last\_sig\_coeff\_y\_suffix shall be in the range of 0 to ( 1  <<  ( ( last\_sig\_coeff\_y\_prefix  >>  1 ) − 1 ) ) − 1, inclusive.

The row position of the last significant coefficient in scanning order within a transform block LastSignificantCoeffY is derived as follows:

* If last\_sig\_coeff\_y\_suffix is not present, the following applies:

LastSignificantCoeffY = last\_sig\_coeff\_y\_prefix (199)

* Otherwise (last\_sig\_coeff\_y\_suffix is present), the following applies:

LastSignificantCoeffY = ( 1  <<  ( ( last\_sig\_coeff\_y\_prefix  >>  1 ) − 1 ) ) \* (200)  
 ( 2 + ( last\_sig\_coeff\_y\_prefix & 1 ) ) + last\_sig\_coeff\_y\_suffix

**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:

* If sb\_coded\_flag[ xS ][ yS ] is equal to 0, the 16 transform coefficient levels of the subblock at location ( xS, yS ) are inferred to be equal to 0.
* Otherwise (sb\_coded\_flag[ xS ][ yS ] is equal to 1), the following applies:
* If ( xS, yS ) is equal to ( 0, 0 ) and ( LastSignificantCoeffX, LastSignificantCoeffY ) is not equal to ( 0, 0 ), at least one of the 16 sig\_coeff\_flag syntax elements is present for the subblock at location ( xS, yS ).
* Otherwise, at least one of the 16 transform coefficient levels of the subblock at location ( xS, yS ) has a non-zero value.

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 ( xC, yC ) is the last significant location ( LastSignificantCoeffX, LastSignificantCoeffY ) in scan order or all of the following conditions are true, sig\_coeff\_flag[ xC ][ yC ] is inferred to be equal to 1:
* ( xC & ( (1 << log2SbW ) − 1 ), yC & ( (1 << log2SbH ) − 1 ) ) is equal to ( 0, 0 ).
* inferSbDcSigCoeffFlag is equal to 1.
* sb\_coded\_flag[ xS ][ yS ] is equal to 1.
* Otherwise, sig\_coeff\_flag[ xC ][ yC ] is inferred to be equal to 0.

**abs\_level\_gtx\_flag**[ n ][ j ] specifies whether the absolute value of the transform coefficient level (at scanning position n) is greater than ( j << 1 ) + 1. When abs\_level\_gtx\_flag[ n ][ j ] is not present, it is inferred to be equal to 0.

**par\_level\_flag**[ n ] specifies the parity of the transform coefficient level at scanning position n. When par\_level\_flag[ n ] is not present, it is inferred to be equal to 0.

**abs\_remainder**[ n ] is the remaining absolute value of a transform coefficient level that is coded with Golomb-Rice code at the scanning position n. When abs\_remainder[ n ] is not present, it is inferred to be equal to 0.

It is a requirement of bitstream conformance that the value of abs\_remainder[ n ] shall be constrained such that the corresponding value of TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] is in the range of CoeffMin to CoeffMax, inclusive.

**dec\_abs\_level**[ n ] is an intermediate value that is coded with Golomb-Rice code at the scanning position n. Given ZeroPos[ n ] that is derived in clause 9.3.3.2 during the parsing of dec\_abs\_level[ n ], the absolute value of a transform coefficient level at location ( xC, yC ) AbsLevel[ xC ][ yC ] is derived as follows:

* If dec\_abs\_level[ n ] is not present or equal to ZeroPos[ n ], AbsLevel[ xC ][ yC ] is set equal to 0.
* Otherwise, if dec\_abs\_level[ n ] is less than ZeroPos[ n ], AbsLevel[ xC ][ yC ] is set equal to dec\_abs\_level[ n ] + 1;
* Otherwise (dec\_abs\_level[ n ] is greater than ZeroPos[ n ]), AbsLevel[ xC ][ yC ] is set equal to dec\_abs\_level[ n ].

It is a requirement of bitstream conformance that the value of dec\_abs\_level[ n ] shall be constrained such that the corresponding value of TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] is in the range of CoeffMin to CoeffMax, inclusive.

**coeff\_sign\_flag**[ n ] specifies the sign of a transform coefficient level for the scanning position n as follows:

* If coeff\_sign\_flag[ n ] is equal to 0, the corresponding transform coefficient level has a positive value.
* Otherwise (coeff\_sign\_flag[ n ] is equal to 1), the corresponding transform coefficient level has a negative value.

When coeff\_sign\_flag[ n ] is not present, it is inferred to be equal to 0.

The value of CoeffSignLevel[ xC ][ yC ] specifies the sign of a transform coefficient level at the location ( xC, yC ) as follows:

* If CoeffSignLevel[ xC ][ yC ] is equal to 0, the corresponding transform coefficient level is equal to zero
* Otherwise, if CoeffSignLevel[ xC ][ yC ] is equal to 1, the corresponding transform coefficient level has a positive value.
* Otherwise (CoeffSignLevel[ xC ][ yC ] is equal to −1), the corresponding transform coefficient level has a negative value.

# Decoding process

## General decoding process

* + 1. **General**

Input to this process is a bitstream BitstreamToDecode. Output of this process is a list of decoded pictures.

The decoding process is specified such that all decoders that conform to a specified profile and level will produce numerically identical cropped decoded output pictures when invoking the decoding process associated with that profile for a bitstream conforming to that profile and level. Any decoding process that produces identical cropped decoded output pictures to those produced by the process described herein (with the correct output order or output timing, as specified) conforms to the decoding process requirements of this Specification.

For each IRAP picture in the bitstream, the following applies:

– If the picture is the first picture of a layer in the bitstream in decoding order, an IDR picture, or the first picture of a layer that follows an EOS NAL unit of the layer in decoding order, the variable NoOutputBeforeRecoveryFlag for the picture is set equal to 1.

– Otherwise, when the picture is a CRA picture, the following applies:

– If some external means not specified in this Specification is available to set the variable HandleCraAsCvsStartFlag for the picture to a value , HandleCraAsCvsStartFlag for the picture is set equal to the value provided by the external means and NoOutputBeforeRecoveryFlag is set equal to HandleCraAsCvsStartFlag.

– Otherwise, HandleCraAsCvsStartFlag and NoOutputBeforeRecoveryFlag are both set equal to 0 for the picture.

For each GDR picture in the bitstream, the following applies:

– If the picture is the first picture of a layer in the bitstream in decoding order or the first picture of a layer that follows an EOS NAL unit of the layer in decoding order, the variable NoOutputBeforeRecoveryFlag for the picture is set equal to 1.

– Otherwise, if some external means not specified in this Specification is available to set the variable HandleGdrAsCvsStartFlag for the picture to a value , HandleGdrAsCvsStartFlag is set equal to the value provided by the external means and NoOutputBeforeRecoveryFlag is set equal to HandleGdrAsCvsStartFlag for the picture.

– Otherwise, HandleGdrAsCvsStartFlag 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 – The above operations, for both IRAP pictures and GDR pictures, are needed for identification of the CLVSS pictures and CLVSs of each layer and consequently the CVSS AUs and the CVSs in the bitstream.

The variables TargetOlsIdx, which identifies the OLS index of the target OLS to be decoded, and the variable Htid, which identifies the highest temporal sublayer to be decoded, are set by some external means not specified in this Specification. The bitstream BitstreamToDecode does not contain any other layers than those included in the target OLS and does not include any NAL unit with TemporalId greater than Htid.

Clause 8.1.2 is repeatedly invoked for each coded picture in BitstreamToDecode in decoding order.

* + 1. **Decoding process for a coded picture**

The decoding processes specified in this clause apply to each coded picture, referred to as the current picture and denoted by the variable CurrPic, 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 SL.

– Otherwise (sps\_chroma\_format\_idc is not equal to 0), the current picture consists of 3 sample arrays SL, SCb, SCr.

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).

Depending on the value of sps\_separate\_colour\_plane\_flag, the decoding process is structured as follows:

– If sps\_separate\_colour\_plane\_flag is equal to 0, the decoding process is invoked a single time with the current picture being the output.

– Otherwise (sps\_separate\_colour\_plane\_flag is equal to 1), the decoding process is invoked three times. Inputs to the decoding process are all NAL units of the coded picture with identical value of sh\_colour\_plane\_id. The decoding process of NAL units with a particular value of sh\_colour\_plane\_id is specified as if only a CVS with monochrome colour format with that particular value of sh\_colour\_plane\_id would be present in the bitstream. The output of each of the three decoding processes is assigned to one of the 3 sample arrays of the current picture, with the NAL units with sh\_colour\_plane\_id equal to 0, 1 and 2 being assigned to SL, SCb and SCr, respectively.

NOTE – The variable ChromaArrayType is derived as equal to 0 when sps\_separate\_colour\_plane\_flag is equal to 1 and sps\_chroma\_format\_idc is equal to 3. In the decoding process, the value of this variable is evaluated resulting in operations identical to that of monochrome pictures (when sps\_chroma\_format\_idc is equal to 0).

The decoding process operates as follows for the current picture CurrPic:

1. The decoding of NAL units is specified in clause 8.2.
2. The processes in clause 8.3 specify the following decoding processes using syntax elements in the slice header layer and above:

– Variables and functions relating to picture order count are derived as specified in clause 8.3.1. This needs to be invoked only for the first slice of a picture.

– At the beginning of the decoding process for each slice of a picture, the decoding process for reference picture lists construction specified in clause 8.3.2 is invoked for derivation of reference picture list 0 (RefPicList[ 0 ]) and reference picture list 1 (RefPicList[ 1 ]).

– The decoding process for reference picture marking in clause 8.3.3 is invoked, wherein reference pictures may 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 a CRA picture with NoOutputBeforeRecoveryFlag equal to 1 or 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.

– 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.

1. The processes in clauses 0, 8.5, 8.6, 8.7, and 8.8 specify decoding processes using syntax elements in all syntax structure layers. 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.
2. After all slices of the current picture have been decoded, the current decoded picture is marked as "used for short-term reference", and each ILRP entry in RefPicList[ 0 ] or RefPicList[ 1 ] is marked as "used for short-term reference".

## 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.

## Slice decoding process

### 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.

If vps\_independent\_layer\_flag[ GeneralLayerIdx[ nuh\_layer\_id ] ] is equal to 0 and there is a picture picA in the current AU in a reference layer of the current layer[Ed. (YK): Consider expressing this condition in a more explicit fashion.], 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 below.

When ph\_poc\_msb\_cycle\_present\_flag is equal to 0 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 equal to 0, and is not a RASL or RADL picture.
* The variable prevPicOrderCntLsb is set equal to ph\_pic\_order\_cnt\_lsb of prevTid0Pic.
* The variable prevPicOrderCntMsb is set equal to PicOrderCntMsb of prevTid0Pic.

The variable PicOrderCntMsb of the current picture is derived as follows:

* If ph\_poc\_msb\_cycle\_present\_flag is equal to 1, PicOrderCntMsb is set equal to ph\_poc\_msb\_cycle\_val \* MaxPicOrderCntLsb.
* Otherwise (ph\_poc\_msb\_cycle\_present\_flag is equal to 0), if the current picture is a CLVSS picture, PicOrderCntMsb is set equal to 0.
* Otherwise, PicOrderCntMsb is derived as follows:

if( ( ph\_pic\_order\_cnt\_lsb < prevPicOrderCntLsb ) &&  
 ( ( prevPicOrderCntLsb − ph\_pic\_order\_cnt\_lsb ) >= ( MaxPicOrderCntLsb / 2 ) ) )  
 PicOrderCntMsb = prevPicOrderCntMsb + MaxPicOrderCntLsb (201)  
else if( ( ph\_pic\_order\_cnt\_lsb > prevPicOrderCntLsb ) &&  
 ( ( ph\_pic\_order\_cnt\_lsb − prevPicOrderCntLsb ) > ( MaxPicOrderCntLsb / 2 ) ) )  
 PicOrderCntMsb = prevPicOrderCntMsb − MaxPicOrderCntLsb  
else  
 PicOrderCntMsb = prevPicOrderCntMsb

PicOrderCntVal is derived as follows:

PicOrderCntVal = PicOrderCntMsb + ph\_pic\_order\_cnt\_lsb (202)

NOTE 1 – All CLVSS pictures for which ph\_poc\_msb\_cycle\_val is not present will have PicOrderCntVal equal to ph\_pic\_order\_cnt\_lsb since for those pictures PicOrderCntMsb is set equal to 0.

The value of PicOrderCntVal shall be in the range of −231 to 231 − 1, inclusive.

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

All pictures in any particular AU shall have the same value of PicOrderCntVal.

The function PicOrderCnt( picX ) is specified as follows:

PicOrderCnt( picX ) = PicOrderCntVal of the picture picX (203)

The function DiffPicOrderCnt( picA, picB ) is specified as follows:

DiffPicOrderCnt( picA, picB ) = PicOrderCnt( picA ) − PicOrderCnt( picB ) (204)

The bitstream shall not contain data that result in values of DiffPicOrderCnt( picA, picB ) used in the decoding process that are not in the range of −215 to 215 − 1, inclusive.

NOTE 2 – Let X be the current picture and Y and Z be two other pictures in the same CVS, Y and Z are considered to be in the same output order direction from X when both DiffPicOrderCnt( X, Y ) and DiffPicOrderCnt( X, Z ) are positive or both are negative.

### 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 a reference picture list. When decoding an I slice, no reference picture list is used in decoding of the slice data. When decoding a P slice, only reference picture list 0 (i.e., RefPicList[ 0 ]), is used in decoding of the slice data. When decoding a B slice, both reference picture list 0 and reference picture list 1 (i.e., RefPicList[ 1 ]) are used in decoding of the slice data.

At the beginning of the decoding process for each slice of a picture, the reference picture lists RefPicList[ 0 ] and RefPicList[ 1 ] are derived. The reference picture lists are used in marking of reference pictures as specified in clause 8.3.3 or in decoding of the slice data.

NOTE 1 – For an I slice of a picture, RefPicList[ 0 ] and RefPicList[ 1 ] may be derived for bitstream conformance checking purpose, but their derivation is not necessary for decoding of the current picture or pictures following the current picture in decoding order. For a P slice of a picture, RefPicList[ 1 ] may be derived for bitstream conformance checking purpose, but its derivation is not necessary for decoding of the current picture or pictures following the current picture in decoding order.

If sps\_idr\_rpl\_present\_flag is equal to 0 and nal\_unit\_type is equal to IDR\_W\_RADL or IDR\_N\_LP, the reference picture lists RefPicList[ 0 ] and RefPicList[ 1 ] are both derived to be empty, i.e., to contain 0 entries, and the following applies for each i equal to 0 or 1:

* The value of RplsIdx[ i ] is inferred to be equal to sps\_num\_ref\_pic\_lists[ i ].
* The value of num\_ref\_entries[ i ][ RplsIdx[ i ] ] is inferred to be equal to 0.
* The value of NumRefIdxActive[ i ] is inferred to be equal to 0.

Otherwise, the reference picture lists RefPicList[ 0 ] and RefPicList[ 1 ], the reference picture scaling ratios RefPicScale[ i ][ j ][ 0 ] and RefPicScale[ i ][ j ][ 1 ], and the reference picture scaled flags RprConstraintsActive[ 0 ][ j ] and RprConstraintsActive[ 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" (205)  
 pocBase = RefPicPocList[ i ][ j ]  
 } else {  
 if( !delta\_poc\_msb\_cycle\_present\_flag[ i ][ k ] ) {  
 if( there is a reference picA in the DPB with the same nuh\_layer\_id as the current picture and  
 PicOrderCntVal & ( MaxPicOrderCntLsb − 1 ) equal to PocLsbLt[ i ][ k ] )  
 RefPicList[ i ][ j ] = picA  
 else  
 RefPicList[ i ][ j ] = "no reference picture"  
 RefPicLtPocList[ i ][ j ] = PocLsbLt[ i ][ k ]  
 } else {  
 if( there is a reference picA in the DPB with the same nuh\_layer\_id as the current picture and  
 PicOrderCntVal equal to FullPocLt[ i ][ k ] )  
 RefPicList[ i ][ j ] = picA  
 else  
 RefPicList[ i ][ j ] = "no reference picture"  
 RefPicLtPocList[ i ][ j ] = FullPocLt[ i ][ k ]  
 }  
 k++  
 }  
 } else {  
 layerIdx = DirectRefLayerIdx[ GeneralLayerIdx[ nuh\_layer\_id ] ][ ilrp\_idx[ i ][ RplsIdx ][ 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 ]  
  
 RefPicScale[ i ][ j ][ 0 ] = ( ( fRefWidth  <<  14 ) + ( CurrPicScalWinWidthL  >>  1 ) ) /  
 CurrPicScalWinWidthL  
 RefPicScale[ i ][ j ][ 1 ] = ( ( fRefHeight  <<  14 ) + ( CurrPicScalWinHeightL  >>  1 ) ) /  
 CurrPicScalWinHeightL  
 RprConstraintsActive[ 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 )  
 }  
}

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 may be used for inter prediction of the current picture and one or more pictures 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 may be used in inter prediction for one or more pictures that follow the current picture in decoding order.

NOTE 4 – There may 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[ 0 ] that is equal to "no reference picture" should be ignored. An unintentional picture loss should 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:

* For each i equal to 0 or 1, num\_ref\_entries[ i ][ RplsIdx[ i ] ] shall not be less than NumRefIdxActive[ i ].
* The picture referred to by each active entry in RefPicList[ 0 ] or RefPicList[ 1 ] shall be present in the DPB and shall have TemporalId less than or equal to that of the current picture.
* The picture referred to by each entry in RefPicList[ 0 ] or RefPicList[ 1 ] shall not be the current picture and shall have ph\_non\_reference\_picture\_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 224.
* Let setOfRefPics be the set of unique pictures referred to by all entries in RefPicList[ 0 ] that have the same nuh\_layer\_id as the current picture and all entries in RefPicList[ 1 ] that have the same nuh\_layer\_id as the current picture. The number of pictures in setOfRefPics shall be less than or equal to MaxDpbSize − 1, inclusive, where MaxDpbSize is as specified in clause A.4.2, and setOfRefPics shall be the same for all slices of a picture.
* When the current slice has nal\_unit\_type equal to STSA\_NUT, there shall be no active entry in RefPicList[ 0 ] or RefPicList[ 1 ] that has TemporalId equal to that of the current picture and nuh\_layer\_id equal to that of the current picture.
* When the current picture is a picture that follows, in decoding order, an STSA picture that has TemporalId equal to that of the current picture and nuh\_layer\_id equal to that of the current picture, there shall be no picture that precedes the STSA picture in decoding order, has TemporalId equal to that of the current picture, and has nuh\_layer\_id equal to that of the current picture included as an active entry in RefPicList[ 0 ] or RefPicList[ 1 ].
* When the current subpicture, with TemporalId equal to a particular value tId, nuh\_layer\_id equal to a particular value layerId, and subpicture index equal to a particular value subpicIdx, is a subpicture that follows, in decoding order, an STSA subpicture with TemporalId equal to tId, nuh\_layer\_id equal to layerId, and subpicture index equal to subpicIdx, there shall be no picture with TemporalId equal to tId and nuh\_layer\_id equal to layerId that precedes the picture containing the STSA subpicture in decoding order included as an active entry in RefPicList[ 0 ] or RefPicList[ 1 ].
* When the current picture, with nuh\_layer\_id equal to a particular value layerId, is an IRAP picture, there shall be no picture referred to by an entry in RefPicList[ 0 ] or RefPicList[ 1 ] that precedes, in output order or decoding order, any preceding IRAP picture with nuh\_layer\_id equal to layerId in decoding order (when present).
* When the current subpicture, with nuh\_layer\_id equal to a particular value layerId and subpicture index equal to a particular value subpicIdx, is an IRAP subpicture, there shall be no picture referred to by an entry in RefPicList[ 0 ] or RefPicList[ 1 ] that precedes, in output order or decoding order, any preceding picture, in decoding order (when present), containing an IRAP subpicture with nuh\_layer\_id equal to layerId and subpicture index equal to subpicIdx.
* When the current picture is not a RASL picture associated with a CRA picture with NoOutputBeforeRecoveryFlag equal to 1, there shall be no picture referred to by an active entry in RefPicList[ 0 ] or RefPicList[ 1 ] that was generated by the decoding process for generating unavailable reference pictures for the CRA picture associated with the current picture.
* When the current subpicture is not a RASL subpicture associated with a CRA subpicture in a CRA picture with NoOutputBeforeRecoveryFlag equal to 1, there shall be no picture referred to by an active entry in RefPicList[ 0 ] or RefPicList[ 1 ] that was generated by the decoding process for generating unavailable reference pictures for the CRA picture containing the CRA subpicture associated with the current subpicture.
* When the current picture, with nuh\_layer\_id equal to a particular value layerId, is not any of the following, there shall be no picture referred to by an entry in RefPicList[ 0 ] or RefPicList[ 1 ] that was generated by the decoding process for generating unavailable reference pictures for the IRAP or GDR picture associated with the current picture:
  + A CRA picture with NoOutputBeforeRecoveryFlag equal to 1
  + A picture, associated with a CRA picture with NoOutputBeforeRecoveryFlag equal to 1, that precedes, in decoding order, 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:
  + A CRA subpicture in a CRA picture with NoOutputBeforeRecoveryFlag equal to 1
  + A subpicture, associated with a CRA subpicture in a CRA picture with NoOutputBeforeRecoveryFlag equal to 1, that precedes, in decoding order, 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 subpicure 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 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 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
  + A picture that precedes the associated IRAP 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 with nuh\_layer\_id equal to layerId containing a RASL subpicture with subpicture index equal to subpicIdx
  + A picture that precedes the picture containing the associated IRAP subpicture in decoding order
* The picture referred to by each ILRP entry in RefPicList[ 0 ] or RefPicList[ 1 ] of a slice of the current picture shall be in the same AU as the current picture.
* The picture referred to by each ILRP entry in RefPicList[ 0 ] or RefPicList[ 1 ] of a slice of the current picture shall be present in the DPB, shall have nuh\_layer\_id refPicLayerId less than the nuh\_layer\_id of the current picture, and shall either be an IRAP picture or have TemporalId less than or equal to Max( 0, vps\_max\_tid\_il\_ref\_pics\_plus1[ currLayerIdx ][ refLayerIdx ] − 1 ), where currLayerIdx and refLayerIdx are equal to GeneralLayerIdx[ nuh\_layer\_id ] and GeneralLayerIdx[ refpicLayerId ], respectively.
* Each ILRP entry in RefPicList[ 0 ] or RefPicList[ 1 ] of a slice shall be an active entry.
* When vps\_independent\_layer\_flag[ GeneralLayerIdx[ nuh\_layer\_id ] ] is equal to 0 and sps\_num\_subpics\_minus1 is greater than 0, either of the following two conditions (but not both) shall be true:
  + The picture referred to by each active entry in RefPicList[ 0 ] or RefPicList[ 1 ] has the same subpicture layout as the current picture (i.e., the SPSs referred to by that picture and the current picture have the same value of sps\_num\_subpics\_minus1 and the same values of sps\_subpic\_ctu\_top\_left\_x[ j ], sps\_subpic\_ctu\_top\_left\_y[ j ], sps\_subpic\_width\_minus1[ j ], and sps\_subpic\_height\_minus1[ j ], respectively, for each value of j in the range of 0 to sps\_num\_subpics\_minus1, inclusive).
  + The picture referred to by each active entry in RefPicList[ 0 ] or RefPicList[ 1 ] is an ILRP for which the value of sps\_num\_subpics\_minus1 is equal to 0.

### Decoding process for reference picture marking

This process is invoked once per picture, after decoding of a slice header and the decoding process for reference picture list construction for the slice as specified in clause 8.3.2, but prior to the decoding of the slice data. This process may 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 the Log2( MaxLtPicOrderCntLsb ) LSBs of 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 an STRP with 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".

### Decoding process for generating unavailable reference pictures

#### General decoding process for generating unavailable reference pictures

This process is invoked once per coded picture when the current picture is a CRA picture with NoOutputBeforeRecoveryFlag equal to 1 or a GDR picture with NoOutputBeforeRecoveryFlag equal to 1.

When this process is invoked, the following applies:

* For each RefPicList[ i ][ j ], with i in the range of 0 to 1, inclusive, and j in the range of 0 to num\_ref\_entries[ i ][ RplsIdx[ i ] ] − 1, inclusive, that is equal to "no reference picture", a picture is generated as specified in 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 ][ RplsIdx[ i ] ][ j ] is equal to 1 and inter\_layer\_ref\_pic\_flag[ i ][ RplsIdx[ i ] ][ j ] is equal to 0, the value of PicOrderCntVal for the generated picture is set equal to RefPicPocList[ i ][ j ] and the generated picture is marked as "used for short-term reference".
* Otherwise, when st\_ref\_pic\_flag[ i ][ RplsIdx[ i ] ][ j ] is equal to 0 and inter\_layer\_ref\_pic\_flag[ i ][ RplsIdx[ i ] ][ j ] is equal to 0, the value of PicOrderCntVal for the generated picture is set equal to RefPicLtPocList[ i ][ j ], the value of ph\_pic\_order\_cnt\_lsb for the generated picture is inferred to be equal to ( RefPicLtPocList[ i ][ j ] & ( MaxPicOrderCntLsb – 1 ) ), and the generated picture is marked as "used for long-term reference".
* The value of PicOutputFlag for the generated reference picture is set equal to 0.
* RefPicList[ i ][ j ] is set to be the generated reference picture.

#### 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 SL for the picture is set equal to 1 << ( BitDepth − 1 ).
* When ChromaArrayType is not equal to 0, the value of each element in the sample arrays SCb and SCr for the picture is set equal to 1 << ( BitDepth − 1 ).
* The prediction mode CuPredMode[ 0 ][ x ][ y ] is set equal to MODE\_INTRA for x ranging from 0 to pps\_pic\_width\_in\_luma\_samples − 1, inclusive, and y ranging from 0 to pps\_pic\_height\_in\_luma\_samples − 1, inclusive.

NOTE – The output of the recovery point picture following a GDR picture with NoOutputBeforeRecoveryFlag equal to 1 and the pictures following that recovery point picture in output order and decoding order is independent of the values set for the elements of SL, SCb, SCr and CuPredMode[ 0 ][ x ][ y ].

### Decoding process for symmetric motion vector difference reference indices

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

The variable RefIdxSymLX with X being 0 and 1 is derived as follows:

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

### 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 reference picture list construction for the slice as specified in clause 8.3.2, but prior to the decoding of any coding unit.

When ph\_temporal\_mvp\_enabled\_flag is equal to 1, the variable ColPic is derived as follows:

* If sh\_slice\_type is equal to B and sh\_collocated\_from\_l0\_flag is equal to 0, ColPic is set equal to RefPicList1[ sh\_collocated\_ref\_idx ].
* Otherwise (sh\_slice\_type is equal to B and sh\_collocated\_from\_l0\_flag is equal to 1, or sh\_slice\_type is equal to P), ColPic is set equal to RefPicList0[ sh\_collocated\_ref\_idx ].

The variable NoBackwardPredFlag is derived as follows:

* If DiffPicOrderCnt( aPic, CurrPic ) is less than or equal to 0 for each active picture aPic in RefPicList0 or RefPicList1 of the current slice, NoBackwardPredFlag is set equal to 1.
* Otherwise, NoBackwardPredFlag is set equal to 0.

## Decoding process for coding units coded in intra prediction mode

### General decoding process for coding units coded in intra prediction mode

Inputs to this process are:

* a luma location ( xCb, yCb ) specifying the top-left sample of the current coding block relative to the top‑left luma sample of the current picture,
* a variable cbWidth specifying the width of the current coding block in luma samples,
* a variable cbHeight specifying the height of the current coding block in luma samples,
* a variable treeType specifying whether a single or a dual tree is used and if a dual tree is used, it specifies whether the current tree corresponds to the luma or chroma components.

Output of this process is a modified reconstructed picture before in-loop filtering.

The derivation process for quantization parameters as specified in clause 8.7.1 is invoked with the luma location ( xCb, yCb ), the width of the current coding block in luma samples cbWidth and the height of the current coding block in luma samples cbHeight, and the variable treeType as inputs.

When treeType is equal to SINGLE\_TREE or treeType is equal to DUAL\_TREE\_LUMA, the decoding process for luma samples is specified as follows:

* If pred\_mode\_plt\_flag is equal to 1, the general decoding process for palette blocks as specified in clause 8.4.5.3 is invoked with ( xCbComp, yCbComp ) set equal to the luma location ( xCb, yCb ), the variable treeType, the variable cIdx set equal to 0, the variable nCbW set equal to cbWidth, the variable nCbH set equal to cbHeight.
* Otherwise (pred\_mode\_plt\_flag is equal to 0), the following applies:

1. The luma intra prediction mode is derived as follows:

* If intra\_mip\_flag[ xCb ][ yCb ] is equal to 1, IntraPredModeY[ x ][ y ] with x = xCb..xCb + cbWidth − 1 and y = yCb..yCb + cbHeight − 1 is set to be equal to intra\_mip\_mode[ xCb ][ yCb ].
* Otherwise, the derivation process for the luma intra prediction mode as specified in clause 8.4.2 is invoked with the luma location ( xCb, yCb ), the width of the current coding block in luma samples cbWidth and the height of the current coding block in luma samples cbHeight as input.

1. The variable predModeIntra is set equal to IntraPredModeY[ xCb ][ yCb ].
2. When cu\_act\_enabled\_flag[ xCb ][ yCb ] is equal to 1, the following applies:

* The general decoding process for intra blocks as specified in clause 8.4.5.1 is invoked with the sample location ( xTb0, yTb0 ) set equal to the luma location ( xCb, yCb ), the variable nTbW set equal to cbWidth, the variable nTbH set equal to cbHeight, predModeIntra, and the variable cIdx set equal to 0 and controlPara set equal to 1 as inputs, and the output is a residual sample array resSamplesL.
* The general decoding process for intra blocks as specified in clause 8.4.5.1 is invoked with the sample location ( xTb0, yTb0 ) set equal to the luma location ( xCb, yCb ), the variable nTbW set equal to cbWidth, the variable nTbH set equal to cbHeight, predModeIntra, and the variable cIdx set equal to 1 and controlPara set equal to 1 as inputs, and the output is a residual sample array resSamplesCb.
* The general decoding process for intra blocks as specified in clause 8.4.5.1 is invoked with the sample location ( xTb0, yTb0 ) set equal to the luma location ( xCb, yCb ), the variable nTbW set equal to cbWidth, the variable nTbH set equal to cbHeight, predModeIntra, and the variable cIdx set equal to 2 and controlPara set equal to 1 as inputs, and the output is a residual sample array resSamplesCr.
* The residual modification process for residual blocks using colour space conversion as specified in clause 8.7.4.6 is invoked with the variable nTbW set equal to cbWidth, the variable nTbH set equal to cbHeight, the array 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.

1. The general decoding process for intra blocks as specified in clause 8.4.5.1 is invoked with the sample location ( xTb0, yTb0 ) set equal to the luma location ( xCb, yCb ), the variable nTbW set equal to cbWidth, the variable nTbH set equal to cbHeight, predModeIntra, the variable cIdx set equal to 0, and controlPara set equal to ( cu\_act\_enabled\_flag[ xCb ][ yCb ] ? 2 : 3 ) 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 ChromaArrayType is not equal to 0, the decoding process for chroma samples is specified as follows:

* If pred\_mode\_plt\_flag is equal to 1, the following applies:
* The general decoding process for palette blocks as specified in clause 8.4.5.3 is invoked with ( xCbComp, yCbComp ) set equal to the chroma location ( xCb / SubWidthC , yCb / SubHeightC ), the variable treeType, the variable cIdx set equal to 1, the variable nCbW set equal to ( cbWidth / SubWidthC ), the variable nCbH set equal to ( cbHeight / SubHeightC ).
* The general decoding process for palette blocks as specified in clause 8.4.5.3 is invoked with ( xCbComp, yCbComp ) set equal to the chroma location ( xCb / SubWidthC , yCb / SubHeightC ), the variable treeType, the variable cIdx set equal to 2, the variable nCbW set equal to ( cbWidth / SubWidthC ), the variable nCbH set equal to ( cbHeight / SubHeightC ).
* Otherwise (pred\_mode\_plt\_flag is equal to 0), the following applies:

1. The derivation process for the chroma intra prediction mode as specified in clause 8.4.3 is invoked with the luma location ( xCb, yCb ), the width of the current coding block in luma samples cbWidth, the height of the current coding block in luma samples cbHeight, and the tree type treeType as inputs.
2. The general decoding process for intra blocks as specified in clause 8.4.5.1 is invoked with the sample location ( xTb0, yTb0 ) set equal to the chroma location ( xCb / SubWidthC, yCb / SubHeightC ), the variable 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, and controlPara set equal to ( cu\_act\_enabled\_flag[ xCb ][ yCb ] ? 2 : 3 ), and the output is a modified reconstructed picture before in-loop filtering.
3. The general decoding process for intra blocks as specified in clause 8.4.5.1 is invoked with the sample location ( xTb0, yTb0 ) set equal to the chroma location ( xCb / SubWidthC, yCb / SubHeightC ), the variable 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 the output is a modified reconstructed picture before in-loop filtering.

### 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), the following ordered steps apply:

1. The neighbouring locations ( xNbA, yNbA ) and ( xNbB, yNbB ) are set equal to ( xCb − 1, yCb + cbHeight − 1 ) and ( xCb + cbWidth − 1, yCb − 1 ), respectively.
2. For X being replaced by either A or B, the variables candIntraPredModeX are derived as follows:

* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring location ( xNbY, yNbY ) set equal to ( xNbX, yNbX ), checkPredModeY set equal to FALSE, and cIdx set equal to 0 as inputs, and the output is assigned to availableX.
* The candidate intra prediction mode candIntraPredModeX is derived as follows:
* If one or more of the following conditions are true, candIntraPredModeX is set equal to INTRA\_PLANAR.
* The variable availableX is equal to FALSE.
* CuPredMode[ 0 ][ xNbX ][ yNbX ] is not equal to MODE\_INTRA.
* intra\_mip\_flag[ 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 ].

1. 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:

candModeList[ 0 ] = candIntraPredModeA (206)

candModeList[ 1 ] = 2 + ( ( candIntraPredModeA + 61 ) % 64 ) (207)

candModeList[ 2 ] = 2 + ( ( candIntraPredModeA − 1 ) % 64 ) (208)

candModeList[ 3 ] = 2 + ( ( candIntraPredModeA + 60 ) % 64 ) (209)

candModeList[ 4 ] = 2 + ( candIntraPredModeA % 64 ) (210)

* Otherwise, if candIntraPredModeB is not equal to candIntraPredModeA and candIntraPredModeA or candIntraPredModeB is greater than INTRA\_DC, the following applies:
  + The variables minAB and maxAB are derived as follows:

minAB = Min( candIntraPredModeA, candIntraPredModeB ) (211)

maxAB = Max( candIntraPredModeA, candIntraPredModeB ) (212)

* + If candIntraPredModeA and candIntraPredModeB are both greater than INTRA\_DC, candModeList[ x ] with x = 0..4 is derived as follows:

candModeList[ 0 ] = candIntraPredModeA (213)

candModeList[ 1 ] = candIntraPredModeB (214)

* + If maxAB − minAB is equal to 1, inclusive, the following applies:

candModeList[ 2 ] = 2 + ( ( minAB + 61 ) % 64 ) (215)

candModeList[ 3 ] = 2 + ( ( maxAB − 1 ) % 64 ) (216)

candModeList[ 4 ] = 2 + ( ( minAB + 60 ) % 64 ) (217)

* + Otherwise, if maxAB − minAB is greater than or equal to 62, the following applies:

candModeList[ 2 ] = 2 + ( ( minAB − 1 ) % 64 ) (218)

candModeList[ 3 ] = 2 + ( ( maxAB + 61 ) % 64 ) (219)

candModeList[ 4 ] = 2 + ( minAB % 64 ) (220)

* + Otherwise, if maxAB − minAB is equal to 2, the following applies:

candModeList[ 2 ] = 2 + ( ( minAB − 1 ) % 64 ) (221)

candModeList[ 3 ] = 2 + ( ( minAB + 61 ) % 64 ) (222)

candModeList[ 4 ] = 2 + ( ( maxAB − 1 ) % 64 ) (223)

* + Otherwise, the following applies:

candModeList[ 2 ] = 2 + ( ( minAB + 61 ) % 64 ) (224)

candModeList[ 3 ] = 2 + ( ( minAB − 1 ) % 64 ) (225)

candModeList[ 4 ] = 2 + ( ( maxAB + 61 ) % 64 ) (226)

* + Otherwise (candIntraPredModeA or candIntraPredModeB is greater than INTRA\_DC), candModeList[ x ] with x = 0..4 is derived as follows:

candModeList[ 0 ] = maxAB (227)

candModeList[ 1 ] = 2 + ( ( maxAB + 61 ) % 64 ) (228)

candModeList[ 2 ] = 2 + ( ( maxAB − 1 ) % 64 ) (229)

candModeList[ 3 ] = 2 + ( ( maxAB + 60 ) % 64 ) (230)

candModeList[ 4 ] = 2 + ( maxAB % 64 ) (231)

* Otherwise, the following applies:

candModeList[ 0 ] = INTRA\_DC (232)

candModeList[ 1 ] = INTRA\_ANGULAR50 (233)

candModeList[ 2 ] = INTRA\_ANGULAR18 (234)

candModeList[ 3 ] = INTRA\_ANGULAR46 (235)

candModeList[ 4 ] = INTRA\_ANGULAR54 (236)

1. IntraPredModeY[ xCb ][ yCb ] is derived by applying the following procedure:

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

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

( candModeList[ i ], candModeList[ j ] ) = Swap( candModeList[ i ], candModeList[ j ] ) (237)

1. IntraPredModeY[ xCb ][ yCb ] is derived by the following ordered steps:
   1. IntraPredModeY[ xCb ][ yCb ] is set equal to intra\_luma\_mpm\_remainder[ xCb ][ yCb ].
   2. The value of IntraPredModeY[ xCb ][ yCb ] is incremented by one.
   3. For i equal to 0 to 4, inclusive, when IntraPredModeY[ xCb ][ yCb ] is greater than or equal to candModeList[ i ], the value of IntraPredModeY[ xCb ][ yCb ] is incremented by one.

The variable IntraPredModeY[ x ][ y ] with x = xCb..xCb + cbWidth − 1 and y = yCb..yCb + cbHeight − 1 is set to be equal to IntraPredModeY[ xCb ][ yCb ].

### Derivation process for chroma intra prediction mode

Input to this process are:

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

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

If treeType is equal to SINGLE\_TREE, ChromaArrayType is equal to 3, intra\_chroma\_pred\_mode is equal to 4, and intra\_mip\_flag[ xCb + cbWidth / 2 ][ yCb + cbHeight / 2 ] [Ed. (SL): to clarify whether or why "+ cbWidth / 2" and "+ cbHeight / 2" are needed.] is equal to 1, the following applies:

* The MIP chroma direct mode flag MipChromaDirectFlag[ xCb ][ yCb ] is set equal to 1.
* The chroma intra prediction mode IntraPredModeC[ xCb ][ yCb ] is set equal to IntraPredModeY[ xCb + cbWidth / 2 ][ yCb + cbHeight / 2 ] [Ed. (SL): to clarify whether or why "+ cbWidth / 2" and "+ cbHeight / 2" are needed.].

Otherwise, the following applies:

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

Table 20 – Specification of IntraPredModeC[ xCb ][ yCb ] depending on cclm\_mode\_flag, cclm\_mode\_idx, intra\_chroma\_pred\_mode and lumaIntraPredMode

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| cclm\_mode\_flag | cclm\_mode\_idx | intra\_chroma\_pred\_mode | lumaIntraPredMode | | | | |
| 0 | 50 | 18 | 1 | X ( 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 sps\_chroma\_format\_idc is equal to 2, the chroma intra prediction mode Y is derived using the chroma intra prediction mode X in Table 20 as specified in Table 21, and the chroma intra prediction mode X is set equal to the chroma intra prediction mode Y afterwards.

Table 21 – Specification of the 4:2:2 mapping process from chroma intra prediction mode X to mode Y when sps\_chroma\_format\_idc is equal to 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **mode X** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| **mode Y** | 0 | 1 | 61 | 62 | 63 | 64 | 65 | 66 | 2 | 3 | 5 | 6 | 8 | 10 | 12 | 13 | 14 | 16 |
| **mode X** | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 |
| **mode Y** | 18 | 20 | 22 | 23 | 24 | 26 | 28 | 30 | 31 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 |
| **mode X** | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 |
| **mode Y** | 41 | 42 | 43 | 43 | 44 | 44 | 45 | 45 | 46 | 47 | 48 | 48 | 49 | 49 | 50 | 51 | 51 | 52 |
| **mode X** | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 |  |  |  |  |  |
| **mode Y** | 52 | 53 | 54 | 55 | 55 | 56 | 56 | 57 | 57 | 58 | 59 | 59 | 60 |  |  |  |  |  |

### 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 to this process is:

* a variable CclmEnabled specifying if a cross-component chroma intra prediction mode is enabled (TRUE) or not enabled (FALSE) for the current chroma coding block.

The variable CclmEnabled is derived as follows:

* If sps\_cclm\_enabled\_flag is equal to 0, CclmEnabled is set equal to 0.
* Otherwise, if one or more of the following conditions are true, CclmEnabled is set equal to 1:
  + sps\_qtbtt\_dual\_tree\_intra\_flag is equal to 0.
  + sh\_slice\_type is not equal to I.
  + CtbLog2SizeY is less than 6.
* Otherwise the following applies:
  + The variables xCb64, yCb64, yCb32 are derived as follows:

xCb64 = ( xCb >> 6) << 6 (238)

yCb64 = ( yCb >> 6 ) << 6 (239)

yCb32 = ( yCb >> 5 ) << 5 (240)

* + If one or more of the following conditions are true, the variable CclmEnabled is set equal to 1:
  + CbWidth[ 1 ][ xCb64 ][ yCb64 ] is equal to 64 and CbHeight[ 1 ][ xCb64 ][ yCb64 ] is equal to 64.
  + CqtDepth[ 1 ][ xCb64 ][ yCb64 ] is equal to CtbLog2SizeY − 6, MttSplitMode[ xCb64 ][ yCb64 ][ 0 ] is equal to SPLIT\_BT\_HOR, CbWidth[ 1 ][ xCb64 ][ yCb32 ] is equal to 64 and CbHeight[ 1 ][ xCb64 ][ yCb32 ] is equal to 32.
  + CqtDepth[ 1 ][ xCb64 ][ yCb64 ] is greater than CtbLog2SizeY − 6.
  + CqtDepth[ 1 ][ xCb64 ][ yCb64 ] is equal to CtbLog2SizeY − 6, MttSplitMode[ xCb64 ][ yCb64 ][ 0 ] is equal to SPLIT\_BT\_HOR, and MttSplitMode[ xCb64 ][ yCb32 ][ 1 ] is equal to SPLIT\_BT\_VER.
  + Otherwise, the variable CclmEnabled is set equal to 0.

When CclmEnabled is equal to 1 and one of the following conditions is true, CclmEnabled is set equal to 0:

* CbWidth[ 0 ][ xCb64 ][ yCb64 ] and CbHeight[ 0 ][ xCb64 ][ yCb64 ] are both equal to 64, and intra\_subpartitions\_mode\_flag[ xCb64 ][ yCb64 ] is equal to 1.
* CbWidth[ 0 ][ xCb64 ][ yCb64 ] or CbHeight[ 0 ][ xCb64 ][ yCb64 ] is less than 64, and CqtDepth[ 0 ][ xCb64 ][ yCb64 ] is equal to CtbLog2SizeY − 6.

### Decoding process for intra blocks

#### General decoding process for intra blocks

Inputs to this process are:

* a sample location ( xTb0, yTb0 ) specifying the top-left sample of the current transform block relative to the top‑left sample of the current picture,
* a variable nTbW specifying the width of the current transform block,
* a variable nTbH specifying the height of the current transform block,
* a variable predModeIntra specifying the intra prediction mode,
* a variable cIdx specifying the colour component of the current block,
* a variable controlPara specifying the output of the process.

Output of this process is a modified reconstructed picture before in-loop filtering when controlPara is not equal to 1 or a residual sample array when controlPara is equal to 1.

The maximum transform block width maxTbWidth and height maxTbHeight are derived as follows:

maxTbWidth = ( cIdx  = =  0 ) ? MaxTbSizeY : MaxTbSizeY / SubWidthC (241)

maxTbHeight = ( cIdx  = =  0 ) ? MaxTbSizeY : MaxTbSizeY / SubHeightC (242)

The luma sample location is derived as follows:

( xTbY, yTbY ) = ( cIdx  = =  0 ) ? ( xTb0, yTb0 ) : ( xTb0 \* SubWidthC, yTb0 \* SubHeightC ) (243)

Depending on maxTbSize, the following applies:

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

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

verSplitFirst = ( nTbW \* ( cIdx = = 0 ? 1 : SubWidthC ) > nTbH \*( cIdx = = 0 ? 1 : SubHeightC ) ) (244)  
 && ( nTbW > maxTbWidth )

newTbW = verSplitFirst ? ( nTbW / 2 ) : nTbW (245)

newTbH = !verSplitFirst ? ( nTbH / 2 ) : nTbH (246)

1. The general decoding process for intra blocks as specified in this clause is invoked with the location ( xTb0, yTb0 ), 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, and the variable controlPara as inputs, and the output is a modified reconstructed picture before in-loop filtering.
2. The following applies:

* If verSplitFirst is equal to TRUE, the general decoding process for intra blocks as specified in this clause is invoked with the location ( xTb0, yTb0 ) set equal to ( xTb0 + newTbW, yTb0 ), the transform block width nTbW set equal to newTbW and the height nTbH set equal to newTbH, the intra prediction mode predModeIntra, the variable cIdx, and the variable controlPara as inputs, and the output is a modified reconstructed picture before in-loop filtering.
* Otherwise (verSplitFirst is equal to FALSE), the general decoding process for intra blocks as specified in this clause is invoked with the location ( xTb0, yTb0 ) set equal to ( xTb0, yTb0 + newTbH ), the transform block width nTbW set equal to newTbW and the height nTbH set equal to newTbH, the intra prediction mode predModeIntra, the variable cIdx, and the variable controlPara 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:

nW = IntraSubPartitionsSplitType = = ISP\_VER\_SPLIT ? nTbW / NumIntraSubPartitions : nTbW (247)

nH = IntraSubPartitionsSplitType = = ISP\_HOR\_SPLIT ? nTbH / NumIntraSubPartitions : nTbH (248)

xPartInc = ISP\_VER\_SPLIT ? 1 : 0 (249)

yPartInc = ISP\_HOR\_SPLIT ? 1 : 0 (250)

nPbW = Max( 4 , nW ) (251)

pbFactor = nPbW / nW (252)

* For i = 0..NumIntraSubPartitions − 1, the following applies:

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

xPartIdx = i \* xPartInc (253)

yPartIdx = i \* yPartInc (254)

xPartPbIdx = xPartIdx % pbFactor (255)

1. When controlPara is not equal to 1 and xPartPbIdx is equal to 0, the intra sample prediction process as specified in clause 8.4.5.2 is invoked with the location ( xTbCmp, yTbCmp ) set equal to ( xTb0 + nW \* xPartIdx, yTb0 + nH \* yPartIdx ), the intra prediction mode predModeIntra, the transform block width nTbW and height nTbH set equal to nPbW and nH, the coding block width nCbW and height nCbH set equal to nTbW and nTbH, and the variable cIdx as inputs, and the output is an (nPbW)x(nH) array predSamples.
2. When controlPara is not equal to 2, the scaling and transformation process as specified in clause 8.7.2 is invoked with the luma location ( xTbY, yTbY ) set equal to ( xTbY + nW \* xPartIdx, yTbY + nH \* yPartIdx ), the variable cIdx, the variable predMode set equal to MODE\_INTRA 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.
3. When controlPara is not equal to 1, the picture reconstruction process for a colour component as specified in clause 8.7.5 is invoked with the transform block location ( xTbComp, yTbComp ) set equal to ( xTb0 + nW \* xPartIdx, yTb0 + nH \* yPartIdx ), the transform block width nTbW, the transform block height nTbH set equal to nW and nH, the variable cIdx, the (nW)x(nH) array predSamples[ x ][ y ] with x = xPartPbIdx \* nW..( xPartPbIdx +1 ) \* nW − 1, y = 0..nH − 1, and the (nW)x(nH) array resSamples as inputs, and the output is a modified reconstructed picture before in-loop filtering.

#### Intra sample prediction

Inputs to this process are:

* a sample location ( xTbCmp, yTbCmp ) specifying the top-left sample of the current transform block relative to the top‑left sample of the current picture,
* a variable predModeIntra specifying the intra prediction mode,
* a variable nTbW specifying the transform block width,
* a variable nTbH specifying the transform block height,
* a variable nCbW specifying the coding block width,
* a variable nCbH specifying the coding block height,
* a variable cIdx specifying the colour component of the current block.

Outputs of this process are the predicted samples predSamples[ x ][ y ], with x = 0..nTbW − 1, y = 0..nTbH − 1.

The predicted samples predSamples[ x ][ y ] are derived as follows:

* If intra\_mip\_flag[ xTbComp ][ yTbComp ] is equal to 1 and cIdx is equal to 0, or if MipChromaDirectFlag[ xTbComp ][ yTbComp ] is equal to 1 and cIdx is not equal to 0, the matrix-based intra sample prediction process as specified in clause 8.4.5.2.1 is invoked with the location ( xTbCmp, yTbCmp ), the intra prediction mode predModeIntra, the transform block width nTbW and height nTbH, and the variable cIdx as inputs, and the output is predSamples.
* Otherwise, the general intra sample prediction process as specified in clause 8.4.5.2.5 is invoked with the location ( xTbCmp, yTbCmp ), the intra prediction mode predModeIntra, the transform block width nTbW and height nTbH, the coding block width nCbW and height nCbH, and the variable cIdx as inputs, and the output is predSamples.

##### Matrix-based intra sample prediction

Inputs to this process are:

* a sample location ( xTbCmp, yTbCmp ) specifying the top-left sample of the current transform block relative to the top‑left sample of the current picture,
* a variable predModeIntra specifying the intra prediction mode,
* a variable nTbW specifying the transform block width,
* a variable nTbH specifying the transform block height,
* a variable cIdx specifying the colour component of the current block.

Outputs of this process are the predicted samples predSamples[ x ][ y ], with x = 0..nTbW − 1, y = 0..nTbH − 1.

The variable mipSizeId is derived as follows:

* If both nTbW and nTbH are equal to 4, mipSizeId is set equal to 0.
* Otherwise, if either cbWidth or cbHeight is equal to 4, or both cbWidth and cbHeight 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[ xTbComp ][ yTbComp ] (256)

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

inSize = ( 2 \* boundarySize ) − ( mipSizeId  = =  2 ) ? 1 : 0 (257)

refW = nTbW + 1 (258)

refH = nTbH + 1 (259)

For the generation of the reference samples refT[ x ] with x = 0..refW − 1 and refL[ y ] with y = 0..refH − 1, the following applies:

* The reference sample availability marking process as specified in clause 8.4.5.2.7 is invoked with the sample location ( xTbCmp, yTbCmp ), reference line index equal to 0, the reference sample width refW, the reference sample height refH, and the colour component index cIdx as inputs, and the reference samples refUnfilt[ x ][ y ] with x = −1, y = −1..ref − 1 and x = 0..ref − 1, y = −1 as output.
* When at least one sample refUnfilt[ x ][ y ] with x = −1, y = −1..refH − 1 and x = 0..refW − 1, y = −1 is marked as "not available for intra prediction", the reference sample substitution process as specified in clause 8.4.5.2.8 is invoked with reference line index 0, the reference sample width refW, the reference sample height refH, the reference samples refUnfilt[ x ][ y ] with x = −1, y = −1..refH − 1 and x = 0..refW − 1, y = −1, and the colour component index cIdx as inputs, and the modified reference samples refUnfilt[ x ][ y ] with x = −1, y = −1..refH − 1 and x = 0..refW − 1, y = −1 as output.
* The reference samples refT[ x ] with x = 0..nTbW − 1 and refL[ y ] with y = 0..nTbH − 1 are assigned as follows:

refT[ x ] = refUnfilt[ x ][ −1 ] (260)

refL[ y ] = refUnfilt[ −1 ][ y ] (261)

For the generation of the input samples p[ x ] with x = 0..inSize − 1, the following applies:

* The MIP boundary downsampling process as specified in clause 8.4.5.2.2 is invoked for the top reference samples with the block size nTbW, the reference samples refT[ x ] with x = 0..nTbW − 1, and the boundary size boundarySize as inputs, and reduced boundary samples redT[ x ] with x = 0..boundarySize − 1 as outputs.
* The MIP boundary downsampling process as specified in clause 8.4.5.2.2 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 ] (262)

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

p[ 0 ] = ( 1  <<  ( BitDepth − 1 ) ) − pTemp[ 0 ]  
p[ x ] = pTemp[ x ] − pTemp[ 0 ] for x = 1..inSize − 1 (263)

For the intra sample prediction process according to predModeIntra, the following ordered steps apply:

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

* The variable modeId is set equal to predModeIntra.
* The weight matrix mWeight[ x ][ y ] with x = 0..inSize − 1, y = 0..predSize \* predSize − 1 is derived by invoking the MIP weight matrix derivation process as specified in clause 8.4.5.2.3 with mipSizeId and modeId as inputs.
* The matrix-based intra prediction samples predMip[ x ][ y ], with x = 0..predSize − 1, y = 0..predSize − 1 are derived as follows:

oW = 32 − 32 \* () (264)

predMip[ x ][ y ] = ( ( ( ) +   
 oW ) >> 6 ) + pTemp[ 0 ] (265)

1. The matrix-based intra prediction samples predMip[ x ][ y ], with x = 0..predSize − 1, y = 0..predSize − 1 are clipped as follows:

predMip[ x ][ y ] = Clip1( predMip[ x ][ y ] ) (266)

1. When isTransposed is equal to TRUE, the predSize x predSize array predMip[ x ][ y ] with x = 0..predSize − 1, y = 0..predSize − 1 is transposed as follows:

predTemp[ y ][ x ] = predMip[ x ][ y ] (267)

predMip = predTemp (268)

1. The predicted samples predSamples[ x ][ y ], with x = 0..nTbW − 1, y = 0..nTbH − 1 are derived as follows:

* If nTbW is greater than predSize or nTbH is greater than predSize, the MIP prediction upsampling process as specified in clause 8.4.5.2.4 is invoked with the input block size predSize, matrix-based intra prediction samples predMip[ x ][ y ] with x = 0..predSize − 1, y = 0..predSize − 1, the transform block width nTbW, the transform block height nTbH, the top reference samples refT[ x ] with x = 0..nTbW − 1, and the left reference samples refL[ y ] with y = 0..nTbH − 1 as inputs, and the output is the predicted sample array predSamples.
* Otherwise, predSamples[ x ][ y ], with x = 0..nTbW − 1, y = 0..nTbH − 1 is set equal to predMip[ x ][ y ].

##### MIP boundary sample downsampling process

Inputs to this process are:

* a variable nTbS specifying the transform block size,
* reference samples refS[ x ] with x = 0..nTbS − 1,
* a variable boundarySize specifying the downsampled boundary size.

Outputs of this process are the reduced boundary samples redS[ x ] with x = 0..boundarySize − 1.

The reduced boundary samples redS[ x ] with x = 0..boundarySize − 1 are derived as follows:

* If boundarySize is less than nTbs, the following applies:

bDwn = nTbs / boundarySize (269)

redS[ x ] = ( + ( 1  <<  ( Log2( bDwn ) − 1 ) ) )  >>  Log2( bDwn ) (270)

* Otherwise (boundarySize is equal to nTbs), redS[ x ] is set equal to refS[ x ].

##### MIP weight matrix derivation process

Inputs to this process are:

* a variable mipSizeId,
* a variable modeId.

Output of this process is the MIP weight matrix mWeight[ x ][ y ].

The MIP weight matrix mWeight[ x ][ y ] is derived depending on mipSizeId and modeId as follows:

* If mipSizeId is equal to 0 and modeId is equal to 0, the following applies:

mWeight[ x ][ y ] = (271)

{

{ 32, 30, 90, 28}, { 32, 32, 72, 28}, { 34, 77, 53, 30}, { 51, 124, 36, 37},

{ 31, 31, 95, 37}, { 33, 31, 70, 50}, { 52, 80, 25, 60}, { 78, 107, 1, 65},

{ 31, 29, 37, 95}, { 38, 34, 19, 101}, { 73, 85, 0, 81}, { 92, 99, 0, 65},

{ 34, 29, 14, 111}, { 48, 48, 7, 100}, { 80, 91, 0, 74}, { 89, 97, 0, 64}

},

* Otherwise, if mipSizeId is equal to 0 and modeId is equal to 1, the following applies:

mWeight[ x ][ y ] = (272)

{

{ 31, 23, 34, 29}, { 31, 43, 34, 31}, { 30, 95, 34, 32}, { 29, 100, 35, 33},

{ 31, 23, 34, 29}, { 31, 43, 34, 31}, { 30, 95, 34, 32}, { 29, 99, 35, 33},

{ 31, 24, 35, 29}, { 31, 44, 34, 31}, { 30, 95, 35, 32}, { 29, 99, 35, 33},

{ 31, 24, 35, 30}, { 31, 44, 35, 31}, { 30, 95, 35, 32}, { 29, 99, 35, 33}

},

* Otherwise, if mipSizeId is equal to 0 and modeId is equal to 2, the following applies:

mWeight[ x ][ y ] = (273)

{

{ 32, 32, 36, 58}, { 32, 29, 26, 66}, { 36, 37, 23, 61}, { 79, 84, 3, 37},

{ 32, 32, 30, 69}, { 33, 29, 24, 71}, { 44, 16, 21, 70}, { 96, 18, 0, 57},

{ 32, 31, 24, 74}, { 33, 30, 23, 71}, { 36, 24, 24, 71}, { 59, 9, 16, 68},

{ 32, 32, 23, 75}, { 33, 30, 24, 70}, { 32, 30, 25, 71}, { 36, 26, 25, 70}

},

* Otherwise, if mipSizeId is equal to 0 and modeId is equal to 3, the following applies:

mWeight[ x ][ y ] = (274)

{

{ 32, 33, 34, 32}, { 32, 30, 22, 38}, { 29, 46, 25, 38}, { 53, 123, 28, 22},

{ 32, 33, 30, 37}, { 32, 30, 21, 38}, { 32, 40, 24, 38}, { 64, 116, 26, 17},

{ 32, 32, 23, 49}, { 32, 30, 21, 39}, { 34, 39, 24, 37}, { 72, 109, 23, 16},

{ 33, 31, 17, 60}, { 32, 31, 21, 39}, { 35, 41, 24, 37}, { 72, 106, 22, 18}

},

* Otherwise, if mipSizeId is equal to 0 and modeId is equal to 4, the following applies:

mWeight[ x ][ y ] = (275)

{

{ 34, 25, 89, 20}, { 38, 32, 47, 24}, { 40, 86, 29, 27}, { 38, 98, 32, 29},

{ 34, 31, 94, 40}, { 44, 25, 83, 27}, { 54, 72, 43, 16}, { 47, 94, 33, 22},

{ 33, 31, 36, 94}, { 43, 23, 51, 76}, { 62, 55, 64, 25}, { 57, 89, 38, 15},

{ 32, 32, 28, 101}, { 38, 26, 33, 94}, { 55, 38, 68, 47}, { 59, 80, 52, 16}

},

* Otherwise, if mipSizeId is equal to 0 and modeId is equal to 5, the following applies:

mWeight[ x ][ y ] = (276)

{

{ 28, 30, 68, 29}, { 23, 48, 23, 48}, { 39, 98, 16, 42}, { 84, 86, 20, 17},

{ 25, 31, 52, 74}, { 38, 68, 5, 70}, { 95, 78, 7, 21}, {127, 54, 12, 0},

{ 30, 47, 14, 107}, { 79, 76, 0, 53}, {127, 59, 7, 1}, {127, 51, 9, 0},

{ 50, 71, 1, 96}, {109, 69, 7, 25}, {127, 56, 9, 0}, {123, 53, 13, 0}

},

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

mWeight[ x ][ y ] = (277)

{

{ 40, 20, 72, 18}, { 48, 29, 44, 18}, { 53, 81, 35, 18}, { 48, 96, 33, 22},

{ 45, 23, 79, 49}, { 61, 21, 56, 49}, { 72, 52, 32, 48}, { 65, 69, 20, 50},

{ 41, 27, 29, 96}, { 49, 22, 28, 94}, { 52, 22, 28, 93}, { 49, 27, 27, 92},

{ 37, 29, 26, 98}, { 39, 28, 28, 97}, { 38, 28, 30, 97}, { 38, 29, 30, 95}

},

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

mWeight[ x ][ y ] = (278)

{

{ 33, 27, 43, 27}, { 32, 29, 31, 31}, { 31, 73, 33, 31}, { 35, 104, 34, 28},

{ 32, 30, 63, 22}, { 33, 26, 33, 29}, { 33, 57, 33, 30}, { 37, 100, 35, 27},

{ 32, 31, 85, 25}, { 34, 25, 39, 25}, { 35, 39, 32, 28}, { 40, 91, 35, 25},

{ 32, 30, 77, 50}, { 34, 26, 54, 22}, { 37, 31, 34, 27}, { 45, 75, 34, 23}

},

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

mWeight[ x ][ y ] = (279)

{

{ 34, 25, 77, 19}, { 36, 34, 56, 24}, { 41, 83, 39, 30}, { 47, 96, 28, 35},

{ 34, 31, 70, 65}, { 38, 29, 53, 77}, { 43, 36, 37, 83}, { 48, 39, 28, 83},

{ 33, 31, 31, 98}, { 33, 31, 30, 99}, { 34, 30, 31, 98}, { 36, 29, 31, 96},

{ 32, 32, 30, 97}, { 32, 32, 31, 96}, { 31, 33, 33, 96}, { 32, 33, 34, 94}

},

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

mWeight[ x ][ y ] = (280)

{

{ 30, 30, 93, 19}, { 31, 59, 67, 34}, { 31, 79, 36, 59}, { 30, 67, 17, 79},

{ 30, 38, 68, 69}, { 29, 40, 43, 91}, { 26, 35, 32, 101}, { 23, 32, 30, 101},

{ 26, 34, 30, 101}, { 23, 33, 30, 102}, { 20, 32, 31, 102}, { 18, 33, 32, 102},

{ 23, 33, 31, 100}, { 20, 34, 32, 100}, { 18, 35, 33, 100}, { 18, 35, 33, 100}

},

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

mWeight[ x ][ y ] = (281)

{

{ 31, 54, 90, 26}, { 32, 60, 53, 61}, { 34, 49, 37, 84}, { 34, 39, 35, 89},

{ 35, 38, 41, 88}, { 35, 35, 32, 96}, { 35, 31, 33, 96}, { 35, 32, 35, 94},

{ 34, 34, 30, 97}, { 35, 32, 33, 95}, { 35, 32, 34, 94}, { 35, 34, 34, 93},

{ 34, 34, 34, 93}, { 35, 34, 34, 93}, { 35, 34, 34, 92}, { 36, 34, 35, 91}

},

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

mWeight[ x ][ y ] = (282)

{

{ 32, 29, 54, 24}, { 31, 32, 34, 29}, { 31, 43, 34, 29}, { 32, 67, 36, 28},

{ 31, 34, 69, 37}, { 31, 35, 46, 33}, { 30, 35, 39, 33}, { 30, 42, 39, 36},

{ 31, 35, 39, 88}, { 30, 38, 41, 84}, { 30, 39, 40, 81}, { 39, 46, 38, 78},

{ 31, 36, 34, 96}, { 34, 38, 37, 93}, { 55, 42, 38, 82}, { 89, 53, 38, 65}

},

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

mWeight[ x ][ y ] = (283)

{

{ 32, 33, 43, 29}, { 32, 30, 29, 33}, { 31, 47, 31, 33}, { 33, 100, 31, 31},

{ 32, 33, 74, 25}, { 32, 32, 34, 31}, { 32, 33, 30, 33}, { 32, 68, 30, 32},

{ 32, 31, 91, 40}, { 32, 32, 58, 26}, { 31, 31, 30, 32}, { 31, 42, 30, 33},

{ 32, 31, 49, 85}, { 32, 31, 83, 35}, { 31, 33, 48, 29}, { 31, 36, 32, 33}

},

* Otherwise, if mipSizeId is equal to 0 and modeId is equal to 13, the following applies:

mWeight[ x ][ y ] = (284)

{

{ 31, 29, 81, 35}, { 32, 28, 34, 50}, { 31, 75, 16, 43}, { 34, 103, 29, 32},

{ 32, 32, 53, 78}, { 31, 28, 36, 88}, { 30, 52, 18, 73}, { 52, 88, 17, 35},

{ 32, 32, 35, 94}, { 30, 31, 35, 95}, { 36, 29, 31, 92}, {100, 43, 16, 40},

{ 32, 32, 35, 93}, { 30, 32, 38, 93}, { 55, 18, 37, 83}, {127, 0, 30, 40}

},

* Otherwise, if mipSizeId is equal to 0 and modeId is equal to 14, the following applies:

mWeight[ x ][ y ] = (285)

{

{ 31, 22, 47, 30}, { 31, 48, 25, 34}, { 30, 95, 31, 32}, { 32, 103, 33, 32},

{ 30, 24, 57, 31}, { 30, 47, 26, 34}, { 31, 95, 31, 32}, { 43, 97, 35, 25},

{ 29, 26, 44, 63}, { 37, 38, 24, 47}, { 74, 63, 28, 20}, {110, 58, 34, 3},

{ 46, 22, 5, 108}, { 93, 5, 9, 77}, {127, 0, 17, 52}, {127, 0, 15, 50}

},

* Otherwise, if mipSizeId is equal to 0 and modeId is equal to 15, the follow applies:

mWeight[ x ][ y ] = (286)

{

{ 32, 27, 68, 24}, { 35, 23, 35, 28}, { 35, 64, 29, 29}, { 37, 104, 33, 28},

{ 32, 32, 91, 40}, { 36, 23, 67, 36}, { 49, 23, 39, 28}, { 60, 67, 30, 20},

{ 32, 32, 36, 95}, { 35, 29, 38, 93}, { 50, 16, 30, 84}, { 72, 16, 15, 65},

{ 32, 32, 27, 100}, { 33, 32, 29, 100}, { 37, 29, 30, 98}, { 48, 21, 29, 90}

},

* Otherwise, if mipSizeId is equal to 1 and modeId is equal to 0, the following applies:

mWeight[ x ][ y ] = (287)

{

{ 30, 63, 46, 37, 25, 33, 33, 34}, { 30, 60, 66, 38, 32, 31, 32, 33},

{ 29, 45, 74, 42, 32, 32, 32, 33}, { 30, 39, 62, 58, 32, 33, 32, 33},

{ 30, 66, 55, 39, 32, 30, 30, 36}, { 29, 54, 69, 40, 33, 31, 31, 33},

{ 28, 48, 71, 43, 32, 33, 32, 33}, { 28, 41, 72, 46, 32, 34, 32, 33},

{ 30, 66, 56, 40, 32, 33, 28, 33}, { 29, 55, 69, 39, 33, 33, 30, 32},

{ 27, 46, 72, 43, 33, 33, 32, 33}, { 27, 42, 69, 48, 32, 34, 32, 33},

{ 30, 63, 55, 40, 32, 33, 35, 30}, { 29, 56, 66, 40, 33, 33, 33, 30},

{ 27, 47, 69, 44, 33, 33, 33, 32}, { 27, 42, 65, 50, 32, 34, 32, 33}

},

* Otherwise, if mipSizeId is equal to 1 and modeId is equal to 1, the following applies:

mWeight[ x ][ y ] = (288)

{

{ 32, 33, 30, 31, 74, 30, 31, 32}, { 33, 56, 28, 30, 41, 29, 32, 32},

{ 33, 77, 52, 26, 29, 34, 30, 32}, { 33, 37, 80, 41, 31, 34, 30, 32},

{ 32, 32, 33, 31, 59, 76, 28, 31}, { 33, 31, 31, 30, 78, 40, 28, 32},

{ 33, 47, 28, 29, 53, 27, 31, 31}, { 33, 61, 44, 28, 34, 32, 31, 31},

{ 32, 31, 34, 30, 26, 64, 76, 27}, { 32, 31, 34, 29, 45, 86, 36, 29},

{ 33, 27, 34, 29, 73, 55, 25, 32}, { 33, 33, 34, 30, 62, 33, 30, 31},

{ 32, 31, 34, 30, 30, 29, 58, 74}, { 32, 31, 35, 29, 27, 53, 77, 35},

{ 32, 30, 36, 29, 40, 80, 44, 31}, { 33, 28, 37, 30, 58, 60, 31, 33}

},

* Otherwise, if mipSizeId is equal to 1 and modeId is equal to 2, the following applies:

mWeight[ x ][ y ] = (289)

{

{ 32, 51, 27, 32, 27, 50, 29, 32}, { 32, 95, 42, 29, 29, 42, 30, 32},

{ 32, 27, 99, 34, 31, 41, 29, 32}, { 32, 34, 21, 104, 31, 42, 30, 32},

{ 32, 45, 30, 32, 9, 88, 40, 30}, { 32, 77, 38, 30, 9, 76, 38, 30},

{ 32, 38, 78, 33, 14, 67, 37, 30}, { 32, 30, 30, 87, 20, 59, 38, 31},

{ 33, 37, 32, 32, 27, 18, 106, 34}, { 34, 44, 34, 31, 25, 17, 108, 31},

{ 36, 39, 45, 31, 24, 15, 108, 30}, { 37, 31, 31, 54, 25, 14, 101, 32},

{ 36, 33, 32, 30, 29, 37, 13, 110}, { 39, 32, 32, 29, 27, 37, 15, 108},

{ 44, 33, 31, 27, 25, 37, 16, 106}, { 47, 30, 31, 32, 25, 34, 19, 102}

},

* Otherwise, if mipSizeId is equal to 1 and modeId is equal to 3, the following applies:

mWeight[ x ][ y ] = (290)

{

{ 32, 48, 35, 35, 47, 68, 31, 31}, { 32, 33, 59, 40, 27, 71, 33, 30},

{ 32, 29, 47, 65, 24, 62, 37, 30}, { 33, 33, 31, 81, 26, 50, 42, 32},

{ 32, 30, 40, 38, 30, 70, 55, 31}, { 32, 20, 46, 50, 26, 55, 64, 31},

{ 33, 30, 29, 66, 25, 41, 72, 33}, { 36, 34, 27, 69, 26, 31, 67, 39},

{ 33, 28, 36, 40, 30, 26, 85, 47}, { 36, 27, 33, 50, 31, 20, 79, 53},

{ 43, 30, 26, 57, 28, 17, 67, 62}, { 51, 27, 28, 55, 22, 23, 49, 70},

{ 38, 29, 32, 39, 28, 30, 22, 104}, { 51, 31, 28, 43, 24, 31, 17, 102},

{ 69, 23, 30, 40, 15, 38, 10, 95}, { 77, 13, 35, 38, 8, 43, 8, 90}

},

* Otherwise, if mipSizeId is equal to 1 and modeId is equal to 4, the following applies:

mWeight[ x ][ y ] = (291)

{

{ 32, 38, 32, 33, 101, 40, 29, 32}, { 32, 40, 37, 32, 100, 36, 30, 32},

{ 32, 37, 46, 35, 94, 33, 30, 31}, { 33, 34, 30, 62, 81, 35, 30, 31},

{ 32, 32, 33, 32, 22, 102, 39, 29}, { 32, 31, 33, 33, 26, 104, 34, 28},

{ 33, 33, 33, 33, 31, 103, 32, 28}, { 33, 32, 34, 36, 37, 94, 33, 28},

{ 32, 33, 32, 32, 34, 24, 99, 36}, { 32, 34, 33, 33, 33, 30, 98, 32},

{ 33, 33, 34, 33, 31, 37, 95, 29}, { 33, 33, 33, 36, 30, 46, 85, 31},

{ 32, 33, 32, 33, 30, 34, 23, 104}, { 32, 34, 33, 33, 31, 32, 30, 98},

{ 32, 33, 34, 34, 31, 29, 39, 91}, { 33, 33, 32, 37, 32, 30, 47, 82}

},

* Otherwise, if mipSizeId is equal to 1 and modeId is equal to 5, the following applies:

mWeight[ x ][ y ] = (292)

{

{ 32, 52, 48, 31, 38, 76, 26, 32}, { 33, 19, 62, 50, 25, 50, 51, 31},

{ 33, 30, 20, 74, 29, 29, 54, 51}, { 34, 35, 23, 56, 31, 25, 41, 76},

{ 33, 25, 38, 39, 28, 39, 83, 35}, { 35, 28, 25, 47, 31, 23, 57, 74},

{ 37, 35, 22, 38, 31, 27, 30, 101}, { 38, 32, 33, 29, 30, 31, 27, 103},

{ 34, 32, 27, 37, 32, 25, 41, 92}, { 38, 33, 28, 32, 30, 31, 18, 111},

{ 40, 32, 33, 27, 29, 33, 18, 111}, { 40, 32, 34, 27, 28, 33, 23, 105},

{ 35, 32, 30, 33, 31, 33, 20, 107}, { 38, 31, 33, 30, 29, 33, 21, 106},

{ 40, 32, 33, 29, 29, 34, 22, 105}, { 40, 32, 33, 30, 29, 34, 24, 101}

},

* Otherwise, if mipSizeId is equal to 1 and modeId is equal to 6, the following applies:

mWeight[ x ][ y ] = (293)

{

{ 32, 28, 31, 33, 92, 33, 30, 31}, { 33, 30, 28, 33, 71, 26, 32, 30},

{ 33, 60, 26, 33, 47, 28, 33, 30}, { 33, 63, 44, 36, 37, 31, 33, 30},

{ 33, 30, 31, 33, 43, 90, 33, 29}, { 33, 28, 29, 34, 71, 71, 26, 30},

{ 33, 30, 26, 33, 86, 45, 28, 30}, { 33, 38, 29, 32, 74, 32, 33, 29},

{ 33, 32, 30, 32, 29, 41, 95, 27}, { 34, 31, 29, 33, 26, 71, 73, 22},

{ 34, 31, 29, 33, 37, 88, 46, 25}, { 33, 32, 28, 34, 55, 75, 36, 28},

{ 34, 31, 30, 32, 33, 27, 43, 89}, { 35, 32, 28, 33, 33, 23, 77, 59},

{ 34, 33, 28, 33, 30, 35, 91, 37}, { 34, 34, 28, 34, 33, 53, 74, 31}

},

* Otherwise, if mipSizeId is equal to 1 and modeId is equal to 7, the following applies:

mWeight[ x ][ y ] = (294)

{

{ 33, 49, 26, 32, 26, 52, 28, 31}, { 33, 71, 72, 24, 30, 32, 34, 31},

{ 32, 23, 70, 68, 32, 32, 32, 32}, { 31, 33, 21, 106, 33, 32, 32, 33},

{ 34, 47, 32, 29, 5, 86, 44, 26}, { 34, 44, 89, 28, 28, 37, 33, 30},

{ 32, 27, 46, 89, 33, 31, 31, 32}, { 30, 33, 20, 107, 33, 33, 32, 33},

{ 35, 39, 42, 27, 26, 24, 92, 35}, { 34, 27, 87, 43, 30, 34, 38, 31},

{ 31, 31, 32, 100, 32, 33, 30, 32}, { 29, 32, 22, 106, 33, 33, 32, 33},

{ 35, 29, 47, 32, 32, 32, 17, 100}, { 34, 24, 69, 60, 34, 33, 28, 44},

{ 31, 33, 31, 99, 32, 33, 32, 31}, { 29, 33, 25, 103, 33, 33, 32, 35}

},

* Otherwise, if mipSizeId is equal to 2 and modeId is equal to 0, the following applies:

mWeight[ x ][ y ] = (295)

{

{ 42, 37, 33, 27, 44, 33, 35}, { 71, 39, 34, 24, 36, 35, 36},

{ 77, 46, 35, 33, 30, 34, 36}, { 64, 60, 35, 33, 31, 32, 36},

{ 49, 71, 38, 32, 32, 31, 36}, { 42, 66, 50, 33, 31, 32, 36},

{ 40, 52, 67, 33, 31, 32, 35}, { 38, 43, 75, 33, 32, 32, 35},

{ 56, 40, 33, 26, 43, 38, 36}, { 70, 49, 34, 30, 28, 38, 38},

{ 65, 57, 36, 34, 28, 33, 39}, { 59, 60, 39, 33, 30, 31, 38},

{ 55, 60, 43, 33, 30, 31, 38}, { 51, 61, 47, 33, 30, 32, 37},

{ 46, 62, 51, 34, 30, 32, 37}, { 42, 60, 55, 33, 31, 32, 37},

{ 60, 42, 34, 30, 37, 43, 38}, { 68, 52, 35, 35, 22, 37, 40},

{ 62, 58, 37, 34, 28, 31, 40}, { 58, 59, 41, 33, 30, 30, 39},

{ 56, 59, 44, 34, 30, 31, 38}, { 53, 60, 45, 33, 30, 31, 38},

{ 49, 65, 45, 33, 30, 31, 38}, { 45, 64, 47, 33, 31, 32, 38},

{ 59, 44, 35, 31, 34, 43, 41}, { 66, 53, 36, 35, 25, 31, 43},

{ 61, 58, 38, 34, 29, 30, 40}, { 59, 57, 41, 33, 30, 31, 39},

{ 57, 58, 43, 33, 30, 31, 39}, { 54, 61, 43, 33, 31, 31, 39},

{ 51, 64, 43, 33, 31, 31, 39}, { 48, 64, 45, 33, 32, 31, 39},

{ 57, 45, 35, 30, 35, 40, 44}, { 65, 54, 37, 33, 33, 24, 44},

{ 63, 56, 38, 34, 30, 29, 39}, { 61, 56, 41, 34, 30, 32, 39},

{ 58, 58, 42, 33, 31, 31, 39}, { 54, 62, 41, 33, 31, 31, 39},

{ 51, 65, 42, 33, 31, 31, 39}, { 48, 63, 43, 33, 32, 31, 39},

{ 55, 46, 35, 30, 36, 38, 47}, { 65, 53, 37, 32, 36, 26, 40},

{ 65, 54, 38, 33, 31, 30, 38}, { 63, 55, 39, 33, 30, 32, 38},

{ 59, 58, 40, 33, 31, 31, 39}, { 54, 64, 40, 33, 31, 30, 40},

{ 49, 66, 40, 32, 32, 30, 41}, { 48, 64, 42, 32, 32, 30, 41},

{ 54, 46, 35, 30, 34, 39, 49}, { 64, 52, 36, 32, 34, 34, 35},

{ 65, 53, 37, 33, 32, 32, 37}, { 63, 55, 38, 33, 31, 31, 39},

{ 59, 60, 38, 33, 31, 31, 40}, { 54, 64, 38, 33, 32, 30, 40},

{ 49, 66, 39, 33, 32, 29, 41}, { 47, 64, 42, 32, 33, 29, 42},

{ 51, 46, 35, 31, 33, 37, 54}, { 61, 51, 36, 32, 33, 38, 36},

{ 63, 53, 37, 32, 32, 34, 37}, { 62, 55, 37, 33, 32, 32, 39},

{ 58, 59, 37, 33, 32, 31, 40}, { 53, 63, 38, 33, 32, 31, 40},

{ 49, 64, 40, 33, 33, 30, 41}, { 46, 62, 42, 33, 33, 30, 42}

},

* Otherwise, if mipSizeId is equal to 2 and modeId is equal to 1, the following applies:

mWeight[ x ][ y ] = (296)

{

{ 39, 34, 33, 58, 44, 31, 32}, { 60, 38, 32, 40, 51, 30, 31},

{ 73, 49, 31, 39, 48, 32, 31}, { 60, 73, 30, 39, 46, 33, 32},

{ 43, 87, 35, 38, 45, 33, 32}, { 35, 78, 54, 36, 45, 33, 32},

{ 33, 47, 86, 35, 44, 33, 32}, { 31, 17, 114, 34, 44, 34, 33},

{ 43, 37, 32, 53, 70, 30, 31}, { 53, 50, 30, 42, 72, 31, 30},

{ 52, 66, 30, 39, 70, 32, 30}, { 46, 78, 35, 37, 68, 34, 30},

{ 43, 75, 48, 37, 66, 34, 30}, { 40, 62, 68, 35, 65, 35, 30},

{ 33, 37, 97, 33, 62, 37, 31}, { 26, 14, 122, 32, 59, 38, 33},

{ 40, 39, 33, 34, 87, 37, 30}, { 45, 54, 32, 34, 84, 41, 29},

{ 41, 70, 35, 33, 83, 40, 29}, { 37, 73, 44, 32, 82, 40, 30},

{ 37, 65, 60, 31, 81, 41, 29}, { 35, 48, 82, 30, 79, 43, 29},

{ 28, 27, 108, 28, 76, 45, 30}, { 19, 11, 127, 27, 70, 46, 32},

{ 38, 40, 34, 27, 73, 62, 28}, { 39, 54, 35, 30, 73, 62, 28},

{ 33, 65, 41, 29, 75, 59, 28}, { 30, 65, 53, 27, 76, 58, 29},

{ 29, 53, 72, 26, 77, 58, 29}, { 27, 35, 95, 24, 77, 60, 28},

{ 19, 19, 117, 23, 74, 61, 30}, { 9, 16, 127, 23, 68, 60, 34},

{ 35, 40, 35, 29, 44, 89, 30}, { 33, 51, 39, 29, 49, 86, 30},

{ 28, 57, 49, 28, 53, 83, 30}, { 24, 52, 65, 26, 56, 82, 30},

{ 22, 39, 86, 24, 58, 82, 30}, { 18, 22, 108, 23, 59, 82, 31},

{ 10, 13, 125, 22, 58, 80, 33}, { 0, 19, 127, 22, 56, 74, 40},

{ 33, 40, 36, 31, 28, 90, 45}, { 29, 46, 44, 29, 31, 92, 43},

{ 24, 45, 58, 28, 34, 91, 43}, { 19, 37, 78, 26, 37, 91, 43},

{ 15, 22, 99, 25, 38, 91, 42}, { 11, 11, 118, 24, 39, 90, 44},

{ 2, 11, 127, 23, 41, 85, 48}, { 0, 17, 127, 23, 43, 75, 55},

{ 31, 37, 39, 30, 28, 54, 82}, { 27, 37, 52, 28, 30, 58, 79},

{ 22, 30, 70, 27, 32, 58, 79}, { 15, 19, 91, 26, 33, 58, 79},

{ 10, 8, 111, 25, 34, 58, 79}, { 5, 2, 125, 25, 35, 57, 80},

{ 0, 9, 127, 25, 36, 53, 84}, { 0, 13, 127, 25, 39, 47, 88},

{ 28, 29, 46, 28, 39, 2, 123}, { 24, 24, 62, 27, 41, 1, 125},

{ 19, 14, 81, 25, 43, 0, 126}, { 13, 4, 101, 24, 44, 0, 127},

{ 6, 0, 116, 23, 45, 0, 127}, { 0, 0, 126, 23, 45, 1, 127},

{ 0, 4, 127, 25, 44, 2, 127}, { 0, 9, 127, 25, 44, 3, 127}

},

* Otherwise, if mipSizeId is equal to 2 and modeId is equal to 2, the following applies:

mWeight[ x ][ y ] = (297)

{

{ 30, 32, 32, 42, 34, 32, 32}, { 63, 26, 34, 16, 38, 32, 32},

{ 98, 26, 34, 25, 34, 33, 32}, { 75, 61, 30, 31, 32, 33, 32},

{ 36, 94, 32, 30, 33, 32, 32}, { 26, 76, 58, 30, 33, 32, 32},

{ 30, 39, 91, 31, 32, 33, 31}, { 32, 23, 105, 32, 32, 32, 32},

{ 34, 30, 33, 31, 52, 29, 32}, { 66, 24, 34, 11, 41, 33, 32},

{ 97, 28, 34, 24, 34, 33, 32}, { 71, 65, 30, 30, 32, 33, 32},

{ 34, 92, 35, 30, 33, 32, 32}, { 26, 70, 64, 29, 34, 32, 32},

{ 30, 37, 94, 30, 33, 32, 31}, { 32, 23, 105, 31, 33, 33, 31},

{ 37, 29, 33, 8, 79, 27, 32}, { 71, 22, 35, 5, 50, 32, 32},

{ 98, 29, 34, 23, 34, 34, 32}, { 66, 70, 30, 31, 31, 33, 32},

{ 31, 92, 38, 30, 33, 32, 32}, { 26, 66, 68, 29, 34, 32, 31},

{ 30, 34, 97, 30, 34, 33, 31}, { 31, 22, 106, 30, 34, 33, 31},

{ 40, 28, 34, 0, 76, 46, 28}, { 76, 21, 35, 0, 55, 35, 32},

{ 97, 32, 34, 21, 37, 33, 33}, { 61, 75, 29, 30, 32, 32, 32},

{ 29, 92, 40, 29, 33, 32, 32}, { 26, 62, 73, 29, 34, 32, 31},

{ 29, 32, 99, 30, 34, 33, 30}, { 31, 22, 107, 30, 34, 33, 31},

{ 42, 27, 34, 1, 48, 79, 25}, { 80, 20, 35, 0, 48, 47, 31},

{ 94, 36, 32, 17, 40, 33, 33}, { 55, 80, 29, 27, 35, 31, 32},

{ 27, 90, 43, 28, 34, 32, 31}, { 26, 58, 76, 29, 33, 33, 30},

{ 29, 30, 101, 29, 34, 34, 30}, { 31, 21, 108, 29, 35, 34, 30},

{ 44, 26, 34, 6, 30, 80, 40}, { 81, 21, 35, 0, 41, 52, 35},

{ 90, 41, 31, 14, 41, 35, 33}, { 51, 82, 29, 24, 37, 32, 32},

{ 27, 87, 47, 27, 35, 32, 31}, { 26, 54, 79, 29, 34, 33, 30},

{ 29, 29, 102, 28, 34, 33, 30}, { 31, 21, 108, 28, 35, 33, 31},

{ 47, 26, 34, 7, 34, 44, 75}, { 80, 24, 34, 0, 41, 41, 50},

{ 84, 45, 31, 12, 40, 36, 36}, { 49, 81, 31, 22, 37, 33, 32},

{ 28, 81, 51, 26, 35, 33, 31}, { 28, 51, 81, 28, 34, 33, 30},

{ 29, 30, 101, 28, 35, 33, 31}, { 31, 22, 107, 28, 35, 33, 32},

{ 48, 27, 34, 10, 40, 16, 97}, { 75, 27, 34, 3, 42, 26, 66},

{ 77, 47, 33, 12, 40, 32, 43}, { 49, 75, 36, 21, 37, 33, 35},

{ 32, 72, 55, 25, 36, 33, 32}, { 30, 49, 81, 27, 35, 33, 31},

{ 30, 32, 98, 28, 35, 32, 32}, { 31, 24, 104, 28, 35, 32, 33}

},

* Otherwise, if mipSizeId is equal to 2 and modeId is equal to 3, the following applies:

mWeight[ x ][ y ] = (298)

{

{ 36, 29, 33, 43, 47, 29, 31}, { 74, 20, 35, 19, 47, 34, 32},

{ 92, 35, 32, 29, 31, 40, 34}, { 53, 80, 26, 33, 28, 36, 37},

{ 24, 91, 41, 31, 31, 31, 38}, { 25, 57, 74, 31, 32, 30, 37},

{ 32, 28, 99, 32, 32, 29, 36}, { 34, 20, 105, 33, 32, 30, 35},

{ 50, 26, 34, 33, 74, 30, 31}, { 75, 28, 33, 23, 46, 47, 33},

{ 64, 58, 29, 30, 26, 46, 40}, { 31, 85, 37, 31, 27, 33, 44},

{ 22, 67, 64, 30, 31, 28, 42}, { 29, 35, 93, 31, 32, 27, 40},

{ 33, 20, 105, 32, 33, 27, 37}, { 34, 19, 106, 33, 32, 29, 36},

{ 51, 29, 33, 25, 72, 51, 30}, { 61, 42, 31, 30, 31, 60, 39},

{ 40, 70, 34, 32, 24, 41, 50}, { 22, 72, 54, 30, 31, 27, 50},

{ 25, 44, 83, 30, 33, 25, 44}, { 32, 23, 102, 32, 33, 26, 40},

{ 34, 18, 107, 32, 33, 28, 37}, { 34, 19, 105, 33, 32, 30, 35},

{ 45, 35, 32, 30, 39, 79, 33}, { 43, 53, 33, 35, 24, 53, 55},

{ 27, 67, 45, 32, 29, 27, 61}, { 22, 53, 72, 30, 33, 22, 52},

{ 28, 31, 95, 31, 33, 25, 43}, { 32, 20, 105, 32, 33, 27, 38},

{ 34, 18, 107, 32, 32, 29, 36}, { 34, 20, 105, 33, 31, 31, 35},

{ 38, 40, 32, 35, 23, 72, 54}, { 31, 55, 39, 34, 29, 32, 73},

{ 22, 57, 60, 31, 35, 18, 64}, { 25, 39, 86, 31, 35, 22, 49},

{ 30, 24, 101, 32, 33, 27, 40}, { 33, 19, 106, 32, 32, 30, 36},

{ 34, 18, 107, 33, 31, 31, 35}, { 34, 20, 104, 33, 31, 32, 34},

{ 33, 42, 35, 34, 28, 39, 82}, { 26, 51, 50, 33, 34, 18, 80},

{ 23, 46, 74, 31, 35, 20, 59}, { 27, 32, 93, 32, 34, 26, 44},

{ 31, 22, 103, 32, 32, 30, 37}, { 33, 19, 106, 33, 31, 31, 35},

{ 34, 19, 106, 33, 31, 32, 34}, { 35, 21, 103, 34, 31, 32, 34},

{ 29, 41, 41, 33, 34, 20, 92}, { 24, 44, 62, 34, 35, 18, 73},

{ 24, 37, 83, 34, 33, 25, 52}, { 28, 28, 97, 33, 32, 30, 40},

{ 32, 23, 103, 33, 31, 32, 36}, { 34, 20, 105, 34, 30, 33, 34},

{ 35, 20, 104, 34, 30, 33, 33}, { 35, 22, 102, 34, 30, 33, 34},

{ 27, 38, 51, 34, 34, 20, 86}, { 26, 37, 71, 35, 34, 24, 64},

{ 27, 33, 87, 35, 32, 30, 47}, { 30, 28, 96, 34, 31, 32, 39},

{ 32, 24, 100, 35, 30, 32, 36}, { 34, 23, 101, 34, 30, 33, 34},

{ 35, 23, 101, 34, 30, 32, 34}, { 34, 24, 99, 35, 30, 33, 34}

},

* Otherwise, if mipSizeId is equal to 2 and modeId is equal to 4, the following applies:

mWeight[ x ][ y ] = (299)

{

{ 39, 30, 31, 67, 33, 34, 31}, { 72, 21, 32, 43, 39, 33, 31},

{100, 23, 32, 35, 39, 34, 31}, { 75, 63, 24, 32, 38, 34, 32},

{ 32, 98, 26, 29, 37, 35, 32}, { 22, 77, 55, 29, 36, 35, 31},

{ 31, 37, 90, 31, 35, 35, 32}, { 35, 22, 100, 33, 33, 36, 33},

{ 47, 29, 32, 74, 54, 32, 31}, { 71, 24, 32, 60, 50, 36, 30},

{ 86, 31, 30, 46, 48, 37, 30}, { 65, 63, 25, 34, 46, 39, 30},

{ 33, 85, 32, 28, 43, 40, 30}, { 26, 64, 60, 27, 39, 41, 30},

{ 33, 33, 87, 29, 35, 41, 31}, { 37, 23, 93, 32, 33, 41, 32},

{ 41, 32, 32, 45, 84, 32, 32}, { 55, 31, 32, 50, 70, 40, 30},

{ 62, 37, 31, 45, 61, 45, 29}, { 53, 55, 31, 36, 55, 48, 29},

{ 38, 63, 40, 29, 48, 50, 28}, { 34, 49, 60, 27, 43, 51, 29},

{ 38, 30, 78, 28, 38, 50, 31}, { 40, 24, 83, 30, 36, 48, 33},

{ 35, 33, 33, 29, 75, 58, 29}, { 39, 35, 33, 34, 68, 59, 29},

{ 41, 39, 34, 36, 61, 62, 29}, { 41, 43, 37, 33, 54, 64, 28},

{ 41, 43, 45, 30, 48, 65, 29}, { 42, 36, 56, 27, 44, 63, 30},

{ 42, 30, 65, 27, 41, 60, 33}, { 42, 28, 68, 28, 37, 56, 36},

{ 33, 34, 33, 31, 42, 88, 30}, { 31, 36, 34, 31, 44, 84, 31},

{ 31, 37, 35, 32, 43, 83, 31}, { 35, 35, 39, 32, 40, 82, 31},

{ 40, 32, 44, 31, 38, 81, 31}, { 44, 30, 48, 30, 37, 78, 33},

{ 44, 30, 52, 28, 37, 72, 36}, { 43, 30, 55, 29, 35, 66, 40},

{ 32, 33, 33, 34, 25, 85, 48}, { 30, 34, 34, 33, 25, 88, 44},

{ 30, 34, 36, 34, 25, 90, 41}, { 33, 32, 38, 34, 25, 90, 40},

{ 38, 29, 41, 34, 26, 88, 40}, { 42, 29, 41, 33, 27, 85, 41},

{ 43, 30, 42, 31, 28, 80, 43}, { 42, 31, 45, 31, 30, 72, 47},

{ 32, 33, 33, 33, 26, 54, 79}, { 31, 32, 34, 35, 20, 68, 68},

{ 32, 32, 35, 36, 17, 76, 62}, { 34, 31, 36, 36, 17, 79, 59},

{ 37, 29, 37, 36, 18, 78, 58}, { 39, 29, 37, 35, 20, 77, 58},

{ 41, 30, 37, 34, 22, 74, 58}, { 40, 31, 40, 32, 26, 68, 59},

{ 33, 31, 34, 33, 29, 31, 98}, { 34, 30, 34, 35, 23, 45, 88},

{ 34, 31, 34, 36, 20, 54, 82}, { 35, 31, 34, 36, 18, 59, 78},

{ 36, 31, 34, 37, 19, 60, 76}, { 38, 30, 34, 36, 20, 61, 74},

{ 39, 31, 35, 35, 22, 60, 73}, { 39, 31, 37, 34, 24, 59, 71}

},

* Otherwise (mipSizeId is equal to 2 and modeId is equal to 5), the following applies:

mWeight[ x ][ y ] = (300)

{

{ 30, 33, 32, 55, 32, 32, 32}, { 47, 30, 31, 29, 36, 32, 32},

{ 81, 28, 32, 28, 34, 32, 32}, { 85, 46, 29, 32, 32, 33, 32},

{ 54, 82, 26, 32, 32, 33, 32}, { 30, 90, 38, 31, 32, 33, 32},

{ 30, 56, 73, 31, 33, 32, 32}, { 37, 21, 102, 32, 32, 32, 32},

{ 33, 32, 31, 68, 39, 31, 31}, { 38, 32, 31, 43, 34, 33, 31},

{ 63, 30, 31, 29, 34, 32, 32}, { 82, 37, 30, 29, 33, 32, 32},

{ 71, 63, 27, 31, 32, 33, 32}, { 44, 86, 30, 30, 33, 33, 32},

{ 33, 72, 55, 30, 32, 32, 31}, { 37, 37, 86, 31, 32, 33, 31},

{ 34, 33, 32, 60, 61, 29, 32}, { 36, 33, 31, 56, 38, 32, 31},

{ 51, 30, 31, 38, 33, 33, 32}, { 75, 31, 31, 30, 33, 33, 32},

{ 80, 47, 29, 30, 32, 33, 31}, { 60, 73, 27, 30, 33, 33, 31},

{ 41, 78, 41, 30, 33, 32, 31}, { 38, 53, 68, 30, 32, 33, 31},

{ 33, 33, 32, 43, 77, 35, 30}, { 35, 33, 31, 55, 54, 29, 32},

{ 43, 32, 31, 46, 39, 31, 32}, { 64, 30, 31, 35, 34, 33, 32},

{ 79, 37, 30, 31, 32, 33, 31}, { 73, 57, 28, 30, 32, 33, 31},

{ 54, 73, 33, 30, 32, 33, 31}, { 43, 64, 52, 30, 32, 33, 31},

{ 33, 33, 32, 34, 68, 58, 28}, { 34, 33, 31, 45, 70, 33, 31},

{ 38, 33, 31, 48, 52, 29, 32}, { 54, 31, 31, 40, 39, 31, 32},

{ 73, 32, 31, 34, 34, 33, 31}, { 77, 45, 29, 31, 32, 32, 32},

{ 65, 63, 30, 31, 31, 33, 31}, { 51, 66, 42, 30, 32, 33, 31},

{ 33, 32, 32, 34, 44, 81, 31}, { 34, 33, 31, 38, 66, 52, 28},

{ 36, 33, 30, 44, 62, 34, 31}, { 47, 31, 31, 43, 48, 30, 32},

{ 64, 31, 31, 38, 38, 32, 32}, { 75, 38, 30, 33, 34, 32, 32},

{ 71, 53, 30, 31, 32, 33, 32}, { 59, 61, 37, 30, 32, 33, 32},

{ 33, 32, 31, 35, 31, 71, 54}, { 34, 33, 31, 37, 49, 70, 33},

{ 36, 33, 31, 41, 60, 48, 30}, { 43, 32, 31, 43, 54, 35, 31},

{ 56, 31, 31, 40, 44, 32, 32}, { 68, 35, 30, 36, 37, 32, 32},

{ 70, 45, 30, 33, 34, 33, 32}, { 63, 55, 35, 31, 33, 33, 32},

{ 33, 32, 31, 33, 34, 36, 87}, { 34, 32, 31, 36, 38, 62, 52},

{ 36, 33, 31, 39, 50, 57, 36}, { 41, 33, 31, 41, 53, 43, 33},

{ 50, 33, 31, 41, 48, 36, 32}, { 59, 35, 31, 37, 41, 34, 32},

{ 65, 42, 31, 35, 36, 33, 32}, { 62, 49, 35, 33, 34, 34, 33}

}

##### 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 (301)

upVer = nTbH / predSize (302)

predSamples[ ( x + 1 ) \* upHor − 1 ][ ( y + 1 ) \* upVer − 1 ] = predMip[ x ][ y ] (303)

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 ] (304)

predSamples[ xHor + dX ][ yHor ] = ( sum + upHor /2 ) / upHor (305)

* + - 1. When upVer is greater than 1, vertical upsampling for all sparse positions ( xVer, yVer ) = ( m, n \* upVer − 1 ) with m = 0..predSize − 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 ] (306)

predSamples[ xVer ][ yVer + dY ] = ( sum + upVer / 2 ) / upVer (307)

##### General intra sample prediction

Inputs to this process are:

* a sample location ( xTbCmp, yTbCmp ) specifying the top-left sample of the current transform block relative to the top‑left sample of the current picture,
* a variable predModeIntra specifying the intra prediction mode,
* a variable nTbW specifying the transform block width,
* a variable nTbH specifying the transform block height,
* a variable nCbW specifying the coding block width,
* a variable nCbH specifying the coding block height,
* a variable cIdx specifying the colour component of the current block.

Outputs of this process are the predicted samples predSamples[ x ][ y ], with x = 0..nTbW − 1, y = 0..nTbH − 1.

The variables refW and refH are derived as follows:

* If IntraSubPartitionsSplitType is equal to ISP\_NO\_SPLIT or cIdx is not equal to 0, the following applies:

refW = nTbW \* 2 (308)

refH = nTbH \* 2 (309)

* Otherwise ( IntraSubPartitionsSplitType is not equal to ISP\_NO\_SPLIT and cIdx is equal to 0 ), the following applies:

refW = nCbW + nTbW (310)

refH = nCbH + nTbH (311)

The variable refIdx specifying the intra prediction reference line index is derived as follows:

refIdx = ( cIdx  = =  0 ) ? IntraLumaRefLineIdx[ xTbCmp ][ yTbCmp ] : 0 (312)

The wide angle intra prediction mode mapping process as specified in clause 8.4.5.2.6 is invoked with predModeIntra, nTbW, nTbH and cIdx as inputs, and the modified predModeIntra as output.

The variable refFilterFlag is derived as follows:

* If predModeIntra is equal to 0, −14, −12, −10, −6, 2, 34, 66, 72, 76, 78, or 80, refFilterFlag is set equal to 1.
* Otherwise, refFilterFlag is set equal to 0.

For the generation of the reference samples p[ x ][ y ] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx, the following ordered steps apply:

1. The reference sample availability marking process as specified in clause 8.4.5.2.7 is invoked with the sample location ( xTbCmp, yTbCmp ), the intra prediction reference line index refIdx, the reference sample width refW, the reference sample height refH, the colour component index cIdx as inputs, and the reference samples refUnfilt[ x ][ y ] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = − refIdx..refW − 1, y = −1 − refIdx as output.
2. When at least one sample refUnfilt[ x ][ y ] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx is marked as "not available for intra prediction", the reference sample substitution process as specified in clause 8.4.5.2.8 is invoked with the intra prediction reference line index refIdx, the reference sample width refW, the reference sample height refH, the reference samples refUnfilt[ x ][ y ] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx, and the colour component index cIdx as inputs, and the modified reference samples refUnfilt[ x ][ y ] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx as output.
3. The reference sample filtering process as specified in clause 8.4.5.2.9 is invoked with the intra prediction reference line index refIdx, the transform block width nTbW and height nTbH, the reference sample width refW, the reference sample height refH, the reference filter flag refFilterFlag, the unfiltered samples refUnfilt[ x ][ y ] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx, and the colour component index cIdx as inputs, and the reference samples p[ x ][ y ] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx as output.

The intra sample prediction process according to predModeIntra applies as follows:

* If predModeIntra is equal to INTRA\_PLANAR, the corresponding intra prediction mode process specified in clause 8.4.5.2.10 is invoked with the transform block width nTbW, and the transform block height nTbH, and the reference sample array p as inputs, and the output is the predicted sample array predSamples.
* Otherwise, if predModeIntra is equal to INTRA\_DC, the corresponding intra prediction mode process specified in clause 8.4.5.2.11 is invoked with the transform block width nTbW, the transform block height nTbH, the intra prediction reference line index refIdx, and the reference sample array p as inputs, and the output is the predicted sample array predSamples.
* Otherwise, if predModeIntra is equal to INTRA\_LT\_CCLM, INTRA\_L\_CCLM or INTRA\_T\_CCLM, the corresponding intra prediction mode process specified in clause 8.4.5.2.13 is invoked with the intra prediction mode predModeIntra, the sample location ( xTbC, yTbC ) set equal to ( xTbCmp, yTbCmp ), the transform block width nTbW and height nTbH, the colour component index cIdx, and the reference sample array p as inputs, and the output is the predicted sample array predSamples.
* Otherwise, the corresponding intra prediction mode process specified in clause 8.4.5.2.12 is invoked with the intra prediction mode predModeIntra, the intra prediction reference line index refIdx, the transform block width nTbW, the transform block height nTbH, the reference sample width refW, the reference sample height refH, the coding block width nCbW and height nCbH, the reference filter flag refFilterFlag, the colour component index cIdx, and the reference sample array p as inputs, and the predicted sample array predSamples as outputs.

When all of the following conditions are true, the position-dependent prediction sample filtering process specified in clause 8.4.5.2.14 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 or cIdx is not equal to 0
* BdpcmFlag[ xTbCmp ][ yTbCmp ][ 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

##### Wide angle intra prediction mode mapping process

Inputs to this process are:

* a variable predModeIntra specifying the intra prediction mode,
* a variable nTbW specifying the transform block width,
* a variable nTbH specifying the transform block height,
* a variable cIdx specifying the colour component of the current block.

Output of this process is the modified intra prediction mode predModeIntra.

The variables nW and nH are derived as follows:

* If IntraSubPartitionsSplitType is equal to ISP\_NO\_SPLIT or cIdx is not equal to 0, the following applies:

nW = nTbW (313)

nH = nTbH (314)

* Otherwise ( IntraSubPartitionsSplitType is not equal to ISP\_NO\_SPLIT and cIdx is equal to 0 ), the following applies:

nW = nCbW (315)

nH = nCbH (316)

The variable whRatio is set equal to Abs( Log2( nW / 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

##### Reference sample availability marking process

Inputs to this process are:

* a sample location ( xTbCmp, yTbCmp ) specifying the top-left sample of the current transform block relative to the top‑left sample of the current picture,
* a variable refIdx specifying the intra prediction reference line index,
* a variable refW specifying the width of the reference area in units of samples,
* a variable refH specifying the height of the reference area in units of samples,
* a variable cIdx specifying the colour component of the current block.

Outputs of this process are the reference samples refUnfilt[ x ][ y ] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx for intra sample prediction.

The refW + refH + 1 + ( 2 \* refIdx ) neighbouring samples refUnfilt[ x ][ y ] that are constructed samples prior to the in-loop filter process, with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx, are derived as follows:

* The neighbouring location (xNbCmp, yNbCmp ) is specified by:

( xNbCmp, yNbCmp ) = ( xTbCmp + x, yTbCmp + y ) (317)

* The current luma location ( xTbY, yTbY ) and the neighbouring luma location (xNbY, yNbY ) are derived as follows:

( xTbY, yTbY ) = ( cIdx  = =  0 ) ? ( xTbCmp, yTbCmp ) :  (318)  
 ( xTbCmp \* SubWidthC, yTbCmp \* SubHeightC )

( xNbY, yNbY ) = ( cIdx  = =  0 ) ? ( xNbCmp, yNbCmp ) :  (319)  
 ( xNbCmp \* SubWidthC, yNbCmp \* SubHeightC )

* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xTbY, yTbY ), the neighbouring luma location ( xNbY, yNbY ), checkPredModeY set equal to FALSE, and cIdx as inputs, and the output is assigned to availableN.
* Each sample refUnfilt[ x ][ y ] is derived as follows:
* If availableN is equal to FALSE, the sample refUnfilt[ x ][ y ] is marked as "not available for intra prediction".
* Otherwise, the sample refUnfilt[ x ][ y ] is marked as "available for intra prediction" and the sample at the location ( xNbCmp, yNbCmp ) is assigned to refUnfilt[ x ][ y ].

##### 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,
* a variable cIdx specifying the colour component of the current block.

Outputs of this process are the modified reference samples refUnfilt[ x ][ y ] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx for intra sample prediction.

The values of the samples refUnfilt[ x ][ y ] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx are modified as follows:

* If all samples refUnfilt[ x ][ y ] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx are marked as "not available for intra prediction", all values of refUnfilt[ x ][ y ] are set equal to 1  <<  ( BitDepth − 1 ).
* Otherwise (at least one but not all samples refUnfilt[ x ][ y ] are marked as "not available for intra prediction"), the following ordered steps apply:

1. When refUnfilt[ −1 − refIdx ][ refH − 1 ] is marked as "not available for intra prediction", search sequentially starting from x = −1 − refIdx, y = refH − 1 to x = −1 − refIdx, y = −1 − refIdx, then from x = −refIdx, y = −1 − refIdx to x = refW − 1, y = −1 − refIdx, for a sample refUnfilt[ x ][ y ] that is marked as "available for intra prediction". Once a sample refUnfilt[ x ][ y ] marked as "available for intra prediction" is found, the search is terminated and the value of refUnfilt[ −1 − refIdx ][ refH − 1 ] is set equal to the value of refUnfilt[ x ][ y ].
2. For x = −1 − refIdx, y = refH − 2..−1 − refIdx, when refUnfilt[ x ][ y ] is marked as "not available for intra prediction", the value of refUnfilt[ x ][ y ] is set equal to the value of refUnfilt[ x ][ y + 1 ].
3. For x = −refIdx..refW − 1, y = −1 − refIdx, when refUnfilt[ x ][ y ] is marked as "not available for intra prediction", the value of refUnfilt[ x ][ y ] is set equal to the value of refUnfilt[ x − 1 ][ y ].

All samples refUnfilt[ x ][ y ] with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx are marked as "available for intra prediction".

##### Reference sample filtering process

Inputs to this process are:

* a variable refIdx specifying the intra prediction reference line index,
* a variable nTbW specifying the transform block width,
* a variable nTbH specifying the transform block height,
* a variable refW specifying the reference samples width,
* a variable refH specifying the reference samples height,
* a variable refFilterFlag specifying the value of reference filter flag,
* the (unfiltered) neighbouring samples refUnfilt[ x ][ y ], with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx,
* a variable cIdx specifying the colour component of the current block.

Outputs of this process are the reference samples p[ x ][ y ], with x = −1 − refIdx, y = −1 − refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1 − refIdx.

The variable filterFlag is derived as follows:

* If all of the following conditions are true, filterFlag is set equal to 1:
* refIdx is equal to 0
* nTbW \* nTbH is greater than 32
* cIdx is equal to 0
* IntraSubPartitionsSplitType is equal to ISP\_NO\_SPLIT
* refFilterFlag is equal to 1
* Otherwise, filterFlag is set equal to 0.

For the derivation of the reference samples p[ x ][ y ] the following applies:

* If filterFlag is equal to 1, the filtered sample values p[ x ][ y ] with x = −1, y = −1..refH − 1 and x = 0..refW − 1, y = −1 are derived as follows:

p[ −1 ][ −1 ] = ( refUnfilt[ −1 ][ 0 ] + 2 \* refUnfilt[ −1 ][ −1 ] + refUnfilt[ 0 ][ −1 ] + 2 )  >>  2 (320)

p[ −1 ][ y ] = ( refUnfilt[ −1 ][ y + 1 ] + 2 \* refUnfilt[ −1 ][ y ] + refUnfilt[ −1 ][ y − 1 ] + 2 )  >>  2  
 for y = 0..refH − 2 (321)

p[ −1 ][ refH − 1 ] = refUnfilt[ −1 ][ refH − 1 ] (322)

p[ x ][ −1 ] = ( refUnfilt[ x − 1 ][ −1 ] + 2 \* refUnfilt[ x ][ −1 ] + refUnfilt[ x + 1 ][ −1 ] + 2 )  >>  2  
 for x = 0..refW − 2  (323)

p[ refW − 1 ][ −1 ] = refUnfilt[ refW − 1 ][ −1 ] (324)

* Otherwise, the reference samples values p[ x ][ y ] are set equal to the unfiltered sample values refUnfilt[ x ][ y ] with x = −1− refIdx, y = −1− refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1− refIdx.

##### Specification of INTRA\_PLANAR intra prediction mode

Inputs to this process are:

* a variable nTbW specifying the transform block width,
* a variable nTbH specifying the transform block height,
* the neighbouring samples p[ x ][ y ], with x = −1, y = −1..nTbH and x = 0..nTbW, y = −1.

Outputs of this process are the predicted samples predSamples[ x ][ y ], with x = 0..nTbW − 1, y = 0..nTbH − 1.

The values of the prediction samples predSamples[ x ][ y ], with x = 0..nTbW − 1 and y = 0..nTbH − 1, are derived as follows:

predV[ x ][ y ] = ( ( nTbH − 1 − y ) \* p[ x ][ −1 ] + ( y + 1 ) \* p[ −1 ][ nTbH ] ) << Log2 ( nTbW )  (325)

predH[ x ][ y ] = ( ( nTbW − 1 − x ) \* p[ −1 ][ y ] + ( x + 1 ) \* p[ nTbW ][ −1 ] ) << Log2 ( nTbH )  (326)

predSamples[ x ][ y ] = ( predV[ x ][ y ] + predH[ x ][ y ] + nTbW \* nTbH ) >>   
 ( Log2 ( nTbW ) + Log2 ( nTbH ) + 1 ) (327)

##### Specification of INTRA\_DC intra prediction mode

Inputs to this process are:

* a variable nTbW specifying the transform block width,
* a variable nTbH specifying the transform block height,
* a variable refIdx specifying the intra prediction reference line index,
* the neighbouring samples p[ x ][ y ], with x = −1 − refIdx, y = 0..nTbH − 1 and x = 0..nTbW − 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 values of the prediction samples predSamples[ x ][ y ], with x = 0..nTbW − 1, y = 0..nTbH − 1, are derived by the following ordered steps:

1. A variable dcVal is derived as follows:

* When nTbW is equal to nTbH:

dcVal = ( + (328)   
 + nTbW )  >>  ( Log2( nTbW ) + 1 )

* When nTbW is greater than nTbH:

dcVal = ( + ( nTbW >> 1 ) )  >>  Log2( nTbW ) (329)

* When nTbW is less than nTbH:

dcVal = ( + ( nTbH >> 1 ) )  >>  Log2( nTbH ) (330)

1. The prediction samples predSamples[ x ][ y ] are derived as follows:

predSamples[ x ][ y ] = dcVal, with x = 0..nTbW − 1, y = 0.. nTbH − 1 (331)

##### Specification of INTRA\_ANGULAR2..INTRA\_ANGULAR66 intra prediction modes

Inputs to this process are:

* the intra prediction mode predModeIntra,
* a variable refIdx specifying the intra prediction reference line index,
* a variable nTbW specifying the transform block width,
* a variable nTbH specifying the transform block height,
* a variable refW specifying the reference samples width,
* a variable refH specifying the reference samples height,
* a variable nCbW specifying the coding block width,
* a variable nCbH specifying the coding block height,
* a variable refFilterFlag specifying the value of reference filter flag,
* a variable cIdx specifying the colour component of the current block,
* the neighbouring samples p[ x ][ y ], with x = −1− refIdx, y = −1− refIdx..refH − 1 and x = −refIdx..refW − 1, y = −1− refIdx.

Outputs of this process are the predicted samples predSamples[ x ][ y ], with x = 0..nTbW − 1, y = 0..nTbH − 1.

The variable nTbS is set equal to ( Log2 ( nTbW ) + Log2 ( nTbH ) )  >>  1.

The variable filterFlag is derived as follows:

* If one or more of the following conditions are true, filterFlag is set equal to 0:
* refFilterFlag is equal to 1
* refIdx is not equal to 0
* IntraSubPartitionsSplitType is not equal to ISP\_NO\_SPLIT
* Otherwise, the following applies:
* The variable minDistVerHor is set equal to Min( Abs( predModeIntra − 50 ), Abs( predModeIntra − 18 ) ).
* The variable intraHorVerDistThres[ nTbS ] is specified in Table 23.
* The variable filterFlag is derived as follows:
* If minDistVerHor is greater than intraHorVerDistThres[ nTbS ], filterFlag is set equal to 1.
* Otherwise, filterFlag is set equal to 0.

Table 23 – Specification of intraHorVerDistThres[ nTbS ] for various transform block sizes nTbS

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **nTbS = 2** | **nTbS = 3** | **nTbS = 4** | **nTbS = 5** | **nTbS = 6** | **nTbS = 7** |
| **intraHorVerDistThres[ nTbS ]** | 24 | 14 | 2 | 0 | 0 | 0 |

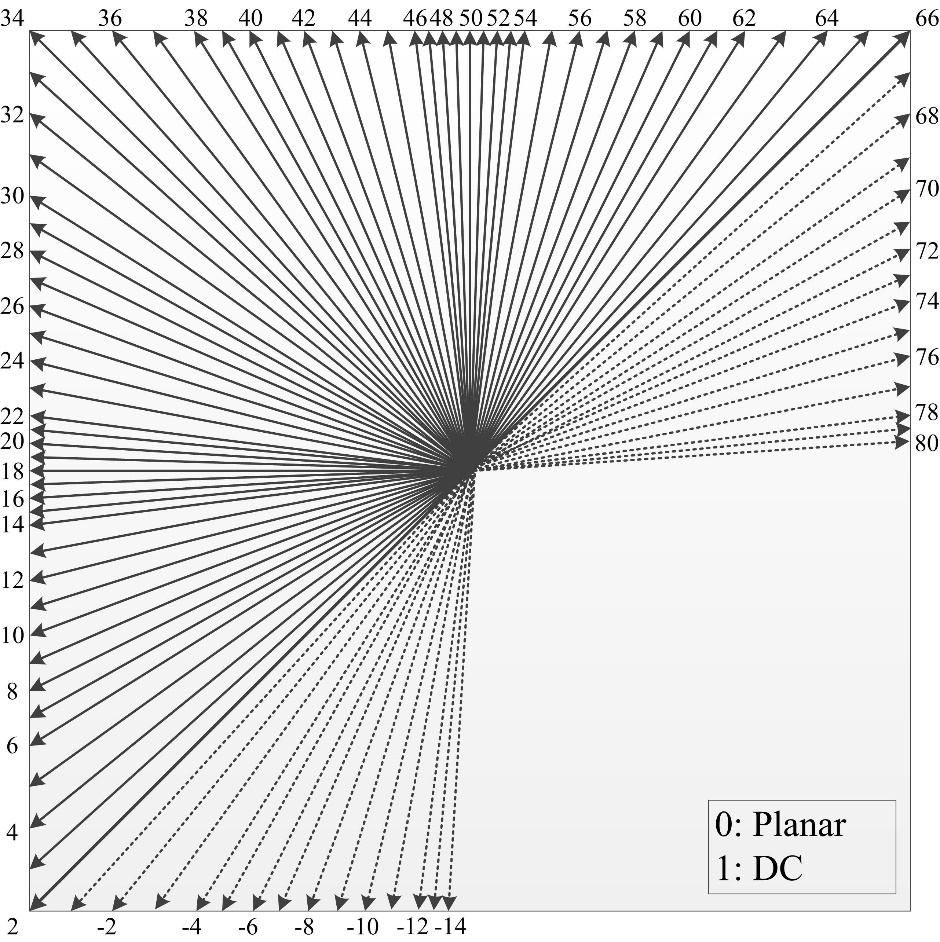


Figure 9 – Intra prediction directions (informative)

Figure 9 illustrates the 93 prediction directions, where the dashed directions are associated with the wide-angle modes that are only applied to non-square blocks.

Table 24 specifies the mapping table between predModeIntra and the angle parameter intraPredAngle.

Table 24 – Specification of intraPredAngle

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **predModeIntra** | **−14** | **−13** | **−12** | **−11** | **−10** | **−9** | **−8** | **−7** | **−6** | **−5** | **−4** | **−3** | **−2** | **−1** | **2** | **3** | **4** |
| **intraPredAngle** | 512 | 341 | 256 | 171 | 128 | 102 | 86 | 73 | 64 | 57 | 51 | 45 | 39 | 35 | 32 | 29 | 26 |
| **predModeIntra** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** | **21** |
| **intraPredAngle** | 23 | 20 | 18 | 16 | 14 | 12 | 10 | 8 | 6 | 4 | 3 | 2 | 1 | 0 | −1 | −2 | −3 |
| **predModeIntra** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** | **32** | **33** | **34** | **35** | **36** | **37** | **38** |
| **intraPredAngle** | −4 | −6 | −8 | −10 | −12 | −14 | −16 | −18 | −20 | −23 | −26 | −29 | −32 | −29 | −26 | −23 | −20 |
| **predModeIntra** | **39** | **40** | **41** | **42** | **43** | **44** | **45** | **46** | **47** | **48** | **49** | **50** | **51** | **52** | **53** | **54** | **55** |
| **intraPredAngle** | −18 | −16 | −14 | −12 | −10 | −8 | −6 | −4 | −3 | −2 | −1 | 0 | 1 | 2 | 3 | 4 | 6 |
| **predModeIntra** | **56** | **57** | **58** | **59** | **60** | **61** | **62** | **63** | **64** | **65** | **66** | **67** | **68** | **69** | **70** | **71** | **72** |
| **intraPredAngle** | 8 | 10 | 12 | 14 | 16 | 18 | 20 | 23 | 26 | 29 | 32 | 35 | 39 | 45 | 51 | 57 | 64 |
| **predModeIntra** | **73** | **74** | **75** | **76** | **77** | **78** | **79** | **80** |  |  |  |  |  |  |  |  |  |
| **intraPredAngle** | 73 | 86 | 102 | 128 | 171 | 256 | 341 | 512 |  |  |  |  |  |  |  |  |  |

The inverse angle parameter invAngle is derived based on intraPredAngle as follows:

invAngle = Round (332)

The interpolation filter coefficients fC[ phase ][ j ] and fG[ phase ][ j ] with phase = 0..31 and j = 0..3 are specified in Table 25.

Table 25 – Specification of interpolation filter coefficients fC and fG

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Fractional sample position p** | **fC interpolation filter coefficients** | | | | **fG interpolation filter coefficients** | | | |
| **fC[ p ][ 0 ]** | **fC[ p ][ 1 ]** | **fC[ p ][ 2 ]** | **fC[ p ][ 3 ]** | **fG[ p ][ 0 ]** | **fG[ p ][ 1 ]** | **fG[ p ][ 2 ]** | **fG[ p ][ 3 ]** |
| 0 | 0 | 64 | 0 | 0 | 16 | 32 | 16 | 0 |
| 1 | −1 | 63 | 2 | 0 | 16 | 32 | 16 | 0 |
| 2 | −2 | 62 | 4 | 0 | 15 | 31 | 17 | 1 |
| 3 | −2 | 60 | 7 | −1 | 15 | 31 | 17 | 1 |
| 4 | −2 | 58 | 10 | −2 | 14 | 30 | 18 | 2 |
| 5 | −3 | 57 | 12 | −2 | 14 | 30 | 18 | 2 |
| 6 | −4 | 56 | 14 | −2 | 13 | 29 | 19 | 3 |
| 7 | −4 | 55 | 15 | −2 | 13 | 29 | 19 | 3 |
| 8 | −4 | 54 | 16 | −2 | 12 | 28 | 20 | 4 |
| 9 | −5 | 53 | 18 | −2 | 12 | 28 | 20 | 4 |
| 10 | −6 | 52 | 20 | −2 | 11 | 27 | 21 | 5 |
| 11 | −6 | 49 | 24 | −3 | 11 | 27 | 21 | 5 |
| 12 | −6 | 46 | 28 | −4 | 10 | 26 | 22 | 6 |
| 13 | −5 | 44 | 29 | −4 | 10 | 26 | 22 | 6 |
| 14 | −4 | 42 | 30 | −4 | 9 | 25 | 23 | 7 |
| 15 | −4 | 39 | 33 | −4 | 9 | 25 | 23 | 7 |
| 16 | −4 | 36 | 36 | −4 | 8 | 24 | 24 | 8 |
| 17 | −4 | 33 | 39 | −4 | 8 | 24 | 24 | 8 |
| 18 | −4 | 30 | 42 | −4 | 7 | 23 | 25 | 9 |
| 19 | −4 | 29 | 44 | −5 | 7 | 23 | 25 | 9 |
| 20 | −4 | 28 | 46 | −6 | 6 | 22 | 26 | 10 |
| 21 | −3 | 24 | 49 | −6 | 6 | 22 | 26 | 10 |
| 22 | −2 | 20 | 52 | −6 | 5 | 21 | 27 | 11 |
| 23 | −2 | 18 | 53 | −5 | 5 | 21 | 27 | 11 |
| 24 | −2 | 16 | 54 | −4 | 4 | 20 | 28 | 12 |
| 25 | −2 | 15 | 55 | −4 | 4 | 20 | 28 | 12 |
| 26 | −2 | 14 | 56 | −4 | 3 | 19 | 29 | 13 |
| 27 | −2 | 12 | 57 | −3 | 3 | 19 | 29 | 13 |
| 28 | −2 | 10 | 58 | −2 | 2 | 18 | 30 | 14 |
| 29 | −1 | 7 | 60 | −2 | 2 | 18 | 30 | 14 |
| 30 | 0 | 4 | 62 | −2 | 1 | 17 | 31 | 15 |
| 31 | 0 | 2 | 63 | −1 | 1 | 17 | 31 | 15 |

The values of the prediction samples predSamples[ x ][ y ], with x = 0..nTbW − 1, y = 0..nTbH − 1 are derived as follows:

* If predModeIntra is greater than or equal to 34, the following ordered steps apply:

1. The reference sample array ref[ x ] is specified as follows:

* The following applies:

ref[ x ] = p[ −1 − refIdx + x ][ −1 − refIdx ], with x = 0..nTbW + refIdx + 1 (333)

* If intraPredAngle is less than 0, the main reference sample array is extended as follows:

ref[ x ] = p[ −1 − refIdx ][ −1 − refIdx + Min( ( x \* invAngle + 256 )  >>  9, nTbH ) ],  
 with x = −nTbH..−1 (334)

* Otherwise,

ref[ x ] = p[ −1 − refIdx + x ][ −1 − refIdx ], with x = nTbW + 2 + refIdx..refW + refIdx (335)

* The additional samples ref[ refW + refIdx +x ] with x = 1..( Max( 1, nTbW / nTbH ) \* refIdx + 1 ) are derived as follows:

ref[ refW + refIdx + x ] = p[ −1 + refW ][ −1 − refIdx ] (336)

1. The values of the prediction samples predSamples[ x ][ y ], with x = 0..nTbW − 1, y = 0..nTbH − 1 are derived as follows:

* The index variable iIdx and the multiplication factor iFact are derived as follows:

iIdx = ( ( ( y + 1 + refIdx ) \* intraPredAngle )  >>  5 ) + refIdx (337)

iFact = ( ( y + 1 + refIdx ) \* intraPredAngle ) & 31 (338)

* If cIdx is equal to 0, the following applies:
* The interpolation filter coefficients fT[ j ] with j = 0..3 are derived as follows:

fT[ j ] = filterFlag ? fG[ iFact ][ j ] : fC[ iFact ][ j ] (339)

* The value of the prediction samples predSamples[ x ][ y ] is derived as follows:

predSamples[ x ][ y ] = Clip1( ( (  ) + 32 )  >>  6 ) (340)

* Otherwise (cIdx is not equal to 0), depending on the value of iFact, the following applies:
* If iFact is not equal to 0, the value of the prediction samples predSamples[ x ][ y ] is derived as follows:

predSamples[ x ][ y ] =   
 ( ( 32 − iFact ) \* ref[ x + iIdx + 1 ] + iFact \* ref[ x + iIdx + 2 ] + 16 )  >>  5 (341)

* Otherwise, the value of the prediction samples predSamples[ x ][ y ] is derived as follows:

predSamples[ x ][ y ] = ref[ x + iIdx + 1 ] (342)

* Otherwise (predModeIntra is less than 34), the following ordered steps apply:

1. The reference sample array ref[ x ] is specified as follows:

* The following applies:

ref[ x ] = p[ −1 − refIdx ][ −1 − refIdx + x ], with x = 0..nTbH + refIdx + 1 (343)

* If intraPredAngle is less than 0, the main reference sample array is extended as follows:

ref[ x ] = p[ −1 − refIdx + Min( ( x \* invAngle + 256 )  >>  9, nTbW ) ][ −1 − refIdx ],  
 with x = −nTbW..−1 (344)

* Otherwise,

ref[ x ] = p[ −1 − refIdx ][ −1 − refIdx + x ], with x = nTbH + 2 + refIdx..refH + refIdx (345)

* The additional samples ref[ refH + refIdx +x ] with x = 1..( Max( 1, nTbH / nTbW ) \* refIdx + 1) are derived as follows:

ref[ refH + refIdx +x ] = p[ −1 + refH ][ −1 − refIdx ] (346)

1. The values of the prediction samples predSamples[ x ][ y ], with x = 0..nTbW − 1, y = 0..nTbH − 1 are derived as follows:

* The index variable iIdx and the multiplication factor iFact are derived as follows:

iIdx = ( ( ( x + 1 + refIdx ) \* intraPredAngle )  >>  5 ) + refIdx (347)

iFact = ( ( x + 1 + refIdx ) \* intraPredAngle ) & 31 (348)

* If cIdx is equal to 0, the following applies:
* The interpolation filter coefficients fT[ j ] with j = 0..3 are derived as follows:

fT[ j ] = filterFlag ? fG[ iFact ][ j ] : fC[ iFact ][ j ] (349)

* The value of the prediction samples predSamples[ x ][ y ] is derived as follows:

predSamples[ x ][ y ] = Clip1( ( (  ) + 32 )  >>  6 ) (350)

* Otherwise (cIdx is not equal to 0), depending on the value of iFact, the following applies:
* If iFact is not equal to 0, the value of the prediction samples predSamples[ x ][ y ] is derived as follows:

predSamples[ x ][ y ] =   
 ( ( 32 − iFact ) \* ref[ y + iIdx + 1 ] + iFact \* ref[ y + iIdx + 2 ] + 16 )  >>  5 (351)

* Otherwise, the value of the prediction samples predSamples[ x ][ y ] is derived as follows:

predSamples[ x ][ y ] = ref[ y + iIdx + 1 ] (352)

##### Specification of INTRA\_LT\_CCLM, INTRA\_L\_CCLM and INTRA\_T\_CCLM intra prediction mode

Inputs to this process are:

* the intra prediction mode predModeIntra,
* a sample location ( xTbC, yTbC ) of the top-left sample of the current transform block relative to the top-left sample of the current picture,
* a variable nTbW specifying the transform block width,
* a variable nTbH specifying the transform block height,
* a variable cIdx specifying the colour component of the current block,
* chroma neighbouring samples p[ x ][ y ], with x = −1, y = 0..2 \* nTbH − 1 and x = 0.. 2 \* nTbW − 1, y = − 1.

Output of this process are predicted samples predSamples[ x ][ y ], with x = 0..nTbW − 1, y = 0..nTbH − 1.

The current luma location ( xTbY, yTbY ) is derived as follows:

( xTbY, yTbY )  =  ( xTbC << ( SubWidthC − 1 ), yTbC << ( SubHeightC − 1 ) ) (353)

The variables availL, availT and availTL are derived as follows:

* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xTbY, yTbY ), the neighbouring luma location ( xTbY − 1, yTbY ), checkPredModeY set equal to FALSE, and cIdx as inputs, and the output is assigned to availL.
* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xTbY, yTbY ), the neighbouring luma location ( xTbY, yTbY − 1 ), checkPredModeY set equal to FALSE, and cIdx as inputs, and the output is assigned to availT.
* The variable availTL is derived as follows:

availTL  =  availL  &&  availT (354)

* 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 or x is equal to 2 \* nTbW − 1:
* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xTbY , yTbY ) the neighbouring luma location ( xTbY + x, 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 or y is equal to 2 \* nTbH − 1:
* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xTbY , yTbY ), the neighbouring luma location ( xTbY − 1, yTbY + y ), 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 (355)

numSampL = availL ? nTbH : 0 (356)

* Otherwise, the following applies:

numSampT = ( availT  &&  predModeIntra  = =  INTRA\_T\_CCLM ) ?  
 ( nTbW +Min( numTopRight, nTbH ) ) : 0 (357)

numSampL = ( availL  &&  predModeIntra  = =  INTRA\_L\_CCLM ) ? ( nTbH +  
 Min( numLeftBelow, nTbW ) ) : 0 (358)

The variable bCTUboundary is derived as follows:

bCTUboundary = ( yTbY & ( CtbSizeY − 1 )  = =  0 ) ? TRUE : FALSE. (359)

The variable cntN and array pickPosN with N being replaced by L and T, are derived as follows:

* The variable numIs4N is derived as follows:

numIs4N = ( ( availT  &&  availL  &&  predModeIntra  = =  INTRA\_LT\_CCLM ) ? 0 : 1) (360)

* The variable startPosN is set equal to numSampN  >>  ( 2 + numIs4N ).
* The variable pickStepN is set equal to Max( 1, numSampN  >>  ( 1 + numIs4N ) ).
* If availN is equal to TRUE and predModeIntra is equal to INTRA\_LT\_CCLM or INTRA\_N\_CCLM, the following assignments are made:
  + - cntN is set equal to Min( numSampN, ( 1 + numIs4N ) << 1 ).
    - pickPosN[ pos ] is set equal to (startPosN + pos \* pickStepN), with pos = 0.. cntN − 1.
* Otherwise, cntN is set equal to 0.

The prediction samples predSamples[ x ][ y ] with x = 0..nTbW − 1, y = 0..nTbH − 1 are derived as follows:

* If both numSampL and numSampT are equal to 0, the following applies:

predSamples[ x ][ y ] = 1  <<  ( BitDepth − 1 ) (361)

* Otherwise, the following ordered steps apply:
  1. The collocated luma samples pY[ x ][ y ] with x = 0..nTbW \* SubWidthC − 1, y= 0..nTbH \* SubHeightC − 1 are set equal to the reconstructed luma samples prior to the deblocking filter process at the locations ( xTbY + x, yTbY + y ).
  2. The neighbouring luma samples pY[ x ][ y ] are derived as follows:
     + When numSampL is greater than 0, the neighbouring left luma samples pY[ x ][ y ] with x = −1..− 3, y = 0..SubHeightC \* numSampL − 1, are set equal to the reconstructed luma samples prior to the deblocking filter process at the locations ( xTbY + x , yTbY +y ).
     + When availT is equal to FALSE, the neighbouring top luma samples pY[ x ][ y ] with x = −1..SubWidthC \* numSampT − 1, y = −1..−2, are set equal to the luma samples pY[ x ][ 0 ].
     + When availL is equal to FALSE, the neighbouring left luma samples pY[ x ][ y ] with x = −1..−3, y = −1..SubHeightC \* numSampL − 1, are set equal to the luma samples pY[ 0 ][ y ].
     + When numSampT is greater than 0, the neighbouring top luma samples pY[ x ][ y ] with x = 0..SubWidthC \* numSampT − 1, y = −1, −2, are set equal to the reconstructed luma samples prior to the deblocking filter process at the locations ( xTbY+ x, yTbY + y ).
     + When availTL is equal to TRUE, the neighbouring top-left luma samples pY[ x ][ y ] with x = −1, y = −1, −2, are set equal to the reconstructed luma samples prior to the deblocking filter process at the locations ( xTbY+ x, yTbY + 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 ] with x = 1..nTbW − 1, y = 1..nTbH − 1 is derived as follows:

pDstY[ x ][ y ] = pY[ x ][ y ] (362)

* + - Otherwise, the following applies:
      * The one-dimensional filter coefficients array F1 and F2, and the 2-dimensional filter coefficients arrays F3 and F4 are specified as follows.

F1[ 0 ] = 2, F1[ 1 ] = 0 (363)

F2[ 0 ] = 1, F2[ 1 ] = 2, F2[ 2 ] = 1 (364)

F3[ i ][ j ] = F4[ i ][ j ] = 0, with i = 0..2, j = 0..2 (365)

* + - * + If both SubWidthC and SubHeightC are equal to 2, the following applies:

F1[ 0 ] = 1, F1[ 1 ] = 1 (366)

F3[ 0 ][ 1 ] = 1, F3[ 1 ][ 1 ] = 4, F3[ 2 ][ 1 ] = 1, F3[ 1 ][ 0 ] = 1, F3[ 1 ][ 2 ] = 1 (367)

F4[ 0 ][ 1 ] = 1, F4[ 1 ][ 1 ] = 2, F4[ 2 ][ 1 ] = 1 (368)

F4[ 0 ][ 2 ] = 1, F4[ 1 ][ 2 ] = 2, F4[ 2 ][ 2 ] = 1 (369)

* + - * + Otherwise, the following applies:

F3[ 1 ][ 1 ] = 8 (370)

F4[ 0 ][ 1 ] = 2, F4[ 1 ][ 1 ] = 4, F4[ 2 ][ 1 ] = 2, (371)

* + - * If sps\_chroma\_vertical\_collocated\_flag is equal to 1, the following applies:
        + pDsY[ x ][ y ] with x = 0..nTbW − 1, y = 0..nTbH − 1 is derived as follows:

pDsY[ x ][ y ] = ( F3[ 1 ][ 0 ] \* pY[ SubWidthC \* x ][ SubHeightC \* y − 1 ] +  
 F3[ 0 ][ 1 ] \* pY[ SubWidthC \* x − 1 ][ SubHeightC \* y ] +  
 F3[ 1 ][ 1 ] \* pY[ SubWidthC \* x ][ SubHeightC \* y ] + (372)  
 F3[ 2 ][ 1 ] \* pY[ SubWidthC \* x + 1 ][ SubHeightC \* y ] +  
 F3[ 1 ][ 2 ] \* pY[ SubWidthC \* x ][ SubHeightC \* y + 1 ] + 4 ) >> 3

* + - * Otherwise (sps\_chroma\_vertical\_collocated\_flag is equal to 0), the following applies:
        + pDsY[ x ][ y ] with x = 0..nTbW − 1, y = 0..nTbH − 1 is derived as follows:

pDsY[ x ][ y ] = ( F4[ 0 ][ 1 ] \* pY[ SubWidthC \* x − 1 ][ SubHeightC \* y ] +  
 F4[ 0 ][ 2 ] \* pY[ SubWidthC \* x − 1 ][ SubHeightC \* y + 1 ] +  
 F4[ 1 ][ 1 ] \* pY[ SubWidthC \* x ][ SubHeightC \* y ] + (373)  
 F4[ 1 ][ 2 ] \* pY[ SubWidthC \* x ][ SubHeightC \* y + 1] +  
 F4[ 2 ][ 1 ] \* pY[ SubWidthC \* x + 1 ][ SubHeightC \* y ] +  
 F4[ 2 ][ 2 ] \* pY[ SubWidthC \* x + 1][ SubHeightC \* y + 1 ] + 4 ) >> 3

* 1. When numSampL is greater than 0, the selected neighbouring left chroma samples pSelC[ idx ] are set equal to p[ −1 ][ pickPosL[ idx ] ] with idx = 0..cntL − 1, and the selected down-sampled neighbouring left luma samples pSelDsY[ idx ] with idx = 0..cntL − 1 are derived as follows:
     + The variable y is set equal to pickPosL[ idx ].
     + If both SubWidthC and SubHeightC are equal to 1, the following applies:

pSelDsY[ idx ] = pY[ − 1][ y ] (374)

* + - Otherwise the following applies:
      * If sps\_chroma\_vertical\_collocated\_flag is equal to 1, the following applies:

pSelDsY[ idx ] = ( F3[ 1 ][ 0 ] \* pY[ − SubWidthC ][ SubHeightC \* y − 1 ] +  
 F3[ 0 ][ 1 ] \* pY[ −1 − SubWidthC ][ SubHeightC \* y ] +  
 F3[ 1 ][ 1 ] \* pY[ −SubWidthC ][ SubHeightC \* y ]  + (375)  
 F3[ 2 ][ 1 ] \* pY[ 1 − SubWidthC ][ SubHeightC \* y ] +  
 F3[ 1 ][ 2 ] \* pY[ −SubWidthC ][ SubHeightC \* y + 1 ] + 4 ) >> 3

* + - * Otherwise (sps\_chroma\_vertical\_collocated\_flag is equal to 0), the following applies:

pSelDsY[ idx ] = ( F4[ 0 ][ 1 ] \* pY[ −1 − SubWidthC ][ SubHeightC \* y ] +  
 F4[ 0 ][ 2 ] \* pY[ −1 − SubWidthC ][ SubHeightC \* y + 1 ] +  
 F4[ 1 ][ 1 ] \* pY[ −SubWidthC ][ SubHeightC \* y ] + (376)  
 F4[ 1 ][ 2 ] \* pY[ −SubWidthC ][ SubHeightC \* y + 1] +  
 F4[ 2 ][ 1 ] \* pY[ 1 − SubWidthC ][ SubHeightC \* y ] +  
 F4[ 2 ][ 2 ] \* pY[ 1 − SubWidthC][ SubHeightC \* y + 1 ] + 4 ) >> 3

* 1. When numSampT is greater than 0, the selected neighbouring top chroma samples pSelC[ idx ] are set equal to p[ pickPosT[ idx − cntL ] ][ -1 ] with idx = cntL..cntL + cntT − 1, and the down-sampled neighbouring top luma samples pSelDsY[ idx ] with idx = 0..cntL + cntT − 1 are specified as follows:
     + The variable x is set equal to pickPosT[ idx − cntL ].
     + If both SubWidthC and SubHeightC are equal to 1, the following applies:

pSelDsY[ idx ] = pY[ x ][ − 1] (377)

* + - Otherwise, the following applies:
      * If sps\_chroma\_vertical\_collocated\_flag is equal to 1, the following applies:
        + If bCTUboundary is equal to FALSE, the following applies:

pSelDsY[ idx ] = ( F3[ 1 ][ 0 ] \* pY[ SubWidthC \* x ][ − 1 − SubHeightC ] +  
 F3[ 0 ][ 1 ] \* pY[ SubWidthC \* x − 1 ][ −SubHeightC ] +  
 F3[ 1 ][ 1 ] \* pY[ SubWidthC \* x ][ −SubHeightC] + (378)  
 F3[ 2 ][ 1 ] \* pY[ SubWidthC \* x + 1 ][ −SubHeightC] +  
 F3[ 1 ][ 2 ] \* pY[ SubWidthC \* x ][ 1 − SubHeightC ] + 4 ) >> 3

* + - * + Otherwise (bCTUboundary is equal to TRUE), the following applies:

pSelDsY[ idx ] = ( F2[ 0 ] \* pY[ SubWidthC \* x − 1 ][ −1 ] +  
 F2[ 1 ] \* pY[ SubWidthC \* x ][ −1 ]  + (379)  
 F2[ 2 ] \* pY[ SubWidthC \* x + 1 ][ −1 ] + 2 ) >> 2

* + - * Otherwise (sps\_chroma\_vertical\_collocated\_flag is equal to 0), the following applies:
        + If bCTUboundary is equal to FALSE, the following applies:

pSelDsY[ idx ] = ( F4[ 0 ][ 1 ] \* pY[ SubWidthC x − 1 ][ −1 ] +  
 F4[ 0 ][ 2 ] \* pY[ SubWidthC \* x − 1 ][ −2 ] +  
 F4[ 1 ][ 1 ] \* pY[ SubWidthC \* x ][ −1 ] + (380)  
 F4[ 1 ][ 2 ] \* pY[ SubWidthC \* x ][ −2] +  
 F4[ 2 ][ 1 ] \* pY[ SubWidthC \* x + 1 ][ −1 ] +  
 F4[ 2 ][ 2 ] \* pY[ SubWidthC \* x + 1 ][ −2 ] + 4 ) >> 3

* + - * + Otherwise (bCTUboundary is equal to TRUE), the following applies:

pSelDsY[ idx ] = ( F2[ 0 ] \* pY[ SubWidthC \* x − 1 ][ −1 ] +  
 F2[ 1 ] \* pY[ SubWidthC \* x ][ −1 ] + (381)  
 F2[ 2 ] \* pY[ SubWidthC \* x + 1][ −1 ] + 2 ) >> 2

* 1. When cntT+ cntL is not equal to 0, the variables minY, maxY, minC and maxC are derived as follows:
     + When cntT + cntL is equal to 2, pSelComp[ 3 ] is set equal to pSelComp[ 0 ], pSelComp[ 2 ] is set equal to pSelComp[ 1 ], pSelComp[ 0 ] is set equal to pSelComp[ 1 ], and pSelComp[ 1 ] is set equal to pSelComp[ 3 ], with Comp being replaced by DsY and C.
     + The arrays minGrpIdx and maxGrpIdx are derived as follows:

minGrpIdx[ 0 ] = 0 (382)

minGrpIdx[ 1 ] = 2 (383)

maxGrpIdx[ 0 ] = 1 (384)

maxGrpIdx[ 1 ] = 3 (385)

* + - When pSelDsY[ minGrpIdx[ 0 ] ] is greater than pSelDsY[ minGrpIdx[ 1 ] ], minGrpIdx[ 0 ] and minGrpIdx[ 1 ] are swapped as follows:

( minGrpIdx[ 0 ], minGrpIdx[ 1 ] ) = Swap( minGrpIdx[ 0 ], minGrpIdx[ 1 ] ) (386)

* + - When pSelDsY[ maxGrpIdx[ 0 ] ] is greater than pSelDsY[ maxGrpIdx[ 1 ] ], maxGrpIdx[ 0 ] and maxGrpIdx[ 1 ] are swapped as follows:

( maxGrpIdx[ 0 ], maxGrpIdx[ 1 ] ) = Swap( maxGrpIdx[ 0 ], maxGrpIdx[ 1 ] ) (387)

* + - When pSelDsY[ minGrpIdx[ 0 ] ] is greater than pSelDsY[ maxGrpIdx[ 1 ] ], arrays minGrpIdx and maxGrpIdx are swapped as follows:

( minGrpIdx, maxGrpIdx ) = Swap( minGrpIdx, maxGrpIdx ) (388)

* + - When pSelDsY[ minGrpIdx[ 1 ] ] is greater than pSelDsY[ maxGrpIdx[ 0 ] ], minGrpIdx[ 1 ] and maxGrpIdx[ 0 ] are swapped as follows:

( minGrpIdx[ 1 ], maxGrpIdx[ 0 ] ) = Swap( minGrpIdx[ 1 ], maxGrpIdx[ 0 ] ) (389)

* + - The variables maxY, maxC, minY and minC are derived as follows:

maxY = ( pSelDsY[ maxGrpIdx[ 0 ] ] + pSelDsY[ maxGrpIdx[ 1 ] ] + 1 )  >>  1 (390)

maxC = ( pSelC[ maxGrpIdx[ 0 ] ] + pSelC[ maxGrpIdx[ 1 ] ] + 1 )  >>  1 (391)

minY = ( pSelDsY[ minGrpIdx[ 0 ] ] + pSelDsY[ minGrpIdx[ 1 ] ] + 1 )  >>  1 (392)

minC = ( pSelC[ minGrpIdx[ 0 ] ] + pSelC[ minGrpIdx[ 1 ] ] + 1 )  >>  1 (393)

* 1. The variables a, b, and k are derived as follows:
     + If numSampL is equal to 0, and numSampT is equal to 0, the following applies:

k = 0 (394)

a = 0 (395)

b = 1  <<  ( BitDepth − 1 ) (396)

* + - Otherwise, the following applies:

diff = maxY − minY (397)

* + - If diff is not equal to 0, the following applies:

diffC = maxC − minC (398)

x = Floor( Log2( diff ) ) (399)

normDiff = ( ( diff  <<  4 )  >>  x ) & 15 (400)

x  +=  ( normDiff  !=  0 ) ? 1 : 0 (401)

y = Abs( diffC ) > 0 ? Floor( Log2( Abs ( diffC ) ) ) + 1 : 0 (402)

a = ( diffC \* ( divSigTable[ normDiff ] | 8 ) + 2y − 1 )  >>  y (403)

k = ( ( 3 + x − y ) < 1 ) ? 1 : 3 + x − y (404)

a = ( ( 3 + x − y ) < 1 ) ? Sign( a ) \* 15 : a (405)

b = minC − ( ( a \* minY )  >>  k ) (406)

where divSigTable[ ] is specified as follows:

divSigTable[ ] = { 0, 7, 6, 5, 5, 4, 4, 3, 3, 2, 2, 1, 1, 1, 1, 0 } (407)

* + - Otherwise (diff is equal to 0), the following applies:

k = 0 (408)

a = 0 (409)

b = minC (410)

* 1. The prediction samples predSamples[ x ][ y ] with x = 0..nTbW − 1, y = 0.. nTbH − 1 are derived as follows:

predSamples[ x ][ y ] = Clip1( ( ( pDsY[ x ][ y ] \* a )  >>  k ) + b ) (411)

NOTE – This process uses sps\_chroma\_vertical\_collocated\_flag. However, in order to simplify implementation, it does not use sps\_chroma\_horizontal\_collocated\_flag.

##### Position-dependent intra prediction sample filtering process

Inputs to this process are:

* the intra prediction mode predModeIntra,
* a variable nTbW specifying the transform block width,
* a variable nTbH specifying the transform block height,
* a variable refW specifying the reference samples width,
* a variable refH specifying the reference samples height,
* the predicted samples predSamples[ x ][ y ], with x = 0..nTbW − 1, y = 0..nTbH − 1,
* the neighbouring samples p[ x ][ y ], with x = −1, y = −1..refH − 1 and x = 0..refW − 1, y = −1.

Outputs of this process are the modified predicted samples predSamples[ x ][ y ] with x = 0..nTbW − 1, y = 0..nTbH − 1.

The variable nScale is derived as follows:

* If predModeIntra is greater than INTRA\_ANGULAR50, nScale is set equal to Min( 2, Log2( nTbH ) − Floor( Log2( 3 \* invAngle − 2 ) ) + 8 ), using invAngle as specified in clause 8.4.5.2.12.
* Otherwise, if predModeIntra is less than INTRA\_ANGULAR18, not equal to INTRA\_PLANAR and not equal to INTRA\_DC, nScale is set equal to Min( 2, Log2( nTbW ) − Floor( Log2( 3 \* invAngle − 2 ) ) + 8 ), using invAngle as specified in clause 8.4.5.2.12.
* Otherwise, nScale is set equal to ( ( Log2( nTbW ) + Log2( nTbH ) − 2 )  >>  2 ).

The reference sample arrays mainRef[ x ] and sideRef[ y ], with x = 0..refW − 1 and y = 0..refH − 1 are derived as follows:

mainRef[ x ] = p[ x ][ −1 ] (412)  
sideRef[ y ] = p[ −1 ][ y ]

The variables refL[ x ][ y ], refT[ x ][ y ], wT[ y ], and wL[ x ] with x = 0..nTbW − 1, y = 0..nTbH − 1 are derived as follows:

* If predModeIntra is equal to INTRA\_PLANAR or INTRA\_DC, the following applies:

refL[ x ][ y ] = p[ −1 ][ y ] (413)

refT[ x ][ y ] = p[ x ][ −1 ] (414)

wT[ y ] = 32  >>  ( ( y  <<  1 )  >>  nScale ) (415)

wL[ x ] = 32  >>  ( ( x  <<  1 )  >>  nScale ) (416)

* Otherwise, if predModeIntra is equal to INTRA\_ANGULAR18 or INTRA\_ANGULAR50, the following applies:

refL[ x ][ y ] = p[ −1 ][ y ] − p[ −1 ][ −1 ] + predSamples[ x ][ y ] (417)

refT[ x ][ y ] = p[ x ][ −1 ] − p[ −1 ][ −1 ] + predSamples[ x ][ y ] (418)

wT[ y ] = ( predModeIntra  = = INTRA\_ANGULAR18 ) ? 32  >>  ( ( y  <<  1 )  >>  nScale ) : 0 (419)

wL[ x ] = ( predModeIntra  = = INTRA\_ANGULAR50 ) ? 32  >>  ( ( x  <<  1 )  >>  nScale ) : 0 (420)

* Otherwise, if predModeIntra is less than INTRA\_ANGULAR18 and nScale is equal to or greater than 0, the following ordered steps apply:

1. The variables dXInt[ y ] and dX[ x ][ y ] are derived as follows using invAngle as specified in clause 8.4.5.2.12 depending on intraPredMode:

dXInt[ y ] = ( ( y + 1 ) \* invAngle + 256 )  >>  9 (421)  
dX[ x ][ y ] = x + dXInt[ y ]

1. The variables refL[ x ][ y ], refT[ x ][ y ], wT[ y ], and wL[ x ] are derived as follows:

refL[ x ][ y ] = 0 (422)

refT[ x ][ y ] = ( y < ( 3 << nScale ) ) ? mainRef[ dX[ x ][ y ] ] : 0 (423)

wT[ y ] = 32  >>  ( ( y  <<  1 )  >>  nScale ) (424)

wL[ x ] = 0 (425)

* Otherwise, if predModeIntra is greater than INTRA\_ANGULAR50 and nScale is equal to or greater than 0, the following ordered steps apply:

1. The variables dYInt[ x ] and dY[ x ][ y ] are derived as follows using invAngle as specified in clause 8.4.5.2.12 depending on intraPredMode:

dYInt[ x ] = ( ( x + 1 ) \* invAngle + 256 )  >>  9 (8‑426)  
dY[ x ][ y ] = y + dYInt[ x ]

1. The variables refL[ x ][ y ], refT[ x ][ y ], wT[ y ], and wL[ x ] are derived as follows:

refL[ x ][ y ] = ( x < ( 3 << nScale ) ) ? sideRef[ dY[ x ][ y ] ] : 0 (427)

refT[ x ][ y ] = 0 (428)

wT[ y ] = 0 (429)

wL[ x ] = 32  >>  ( ( x  <<  1 )  >>  nScale ) (430)

* Otherwise, refL[ x ][ y ], refT[ x ][ y ], wT[ y ], and wL[ x ] are all set equal to 0.

The values of the modified predicted samples predSamples[ x ][ y ], with x = 0..nTbW − 1, y = 0..nTbH − 1 are derived as follows:

predSamples[ x ][ y ] = Clip1( (  refL[ x ][ y ] \* wL[ x ] + refT[ x ][ y ] \* wT[ y ] + (431) ( 64 − wL[ x ] − wT[ y ] ) \* predSamples[ x ][ y ] + 32  ) >> 6 )

#### Decoding process for palette mode

Inputs to this process are:

* a location ( xCbComp, yCbComp ) specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,
* a variable treeType specifying whether a single or a dual tree is used and if a dual tree is used, it specifies whether the current tree corresponds to the luma or chroma components,
* a variable cIdx specifying the colour component of the current block,
* two variables nCbW and nCbH specifying the width and height of the current coding block, respectively.

Output of this process is an array recSamples[ x ][ y ], with x = 0.. nCbW − 1, y = 0.. nCbH − 1 specifying reconstructed sample values for the block.

Depending on the value of treeType, the variables startComp, numComps and maxNumPalettePredictorSize are derived as follows:

* If treeType is equal to SINGLE\_TREE:

startComp = 0 (432)

numComps = ChromaArrayType = = 0 ? 1 : 3 (433)

maxNumPalettePredictorSize = 63 (434)

* Otherwise, treeType is equal to DUAL\_TREE\_LUMA:

startComp = 0 (435)

numComps = 1 (436)

maxNumPalettePredictorSize = 31 (437)

* Otherwise, treeType is equal to DUAL\_TREE\_CHROMA:

startComp = 1 (438)

numComps = 2 (439)

maxNumPalettePredictorSize = 31 (440)

Depending on the value of cIdx, the variables nSubWidth and nSubHeight are derived as follows:

* If cIdx is greater than 0 and startComp is equal to 0, nSubWidth is set equal to SubWidthC and nSubHeight is set equal to SubHeightC.
* Otherwise, nSubWidth is set equal to 1 and nSubHeight is set equal to 1.

The ( nCbW x nCbH ) block of the reconstructed sample array recSamples at location ( xCbComp, yCbComp ) is represented by recSamples[ x ][ y ] with x = 0..nCTbW − 1 and y = 0..nCbH − 1, and the value of recSamples[ x ][ y ] for each x in the range of 0 to nCbW − 1, inclusive, and each y in the range of 0 to nCbH − 1, inclusive, is derived as follows:

* The variables xL, yL, xCbL, and yCbL are derived as follows:

xL = x \* nSubWidth (441)

yL = y \* nSubHeight (442)

xCbL = xCbComp \* nSubWidth (443)

yCbL = yCbComp \* nSubHeight (444)

* The variable bIsEscapeSample is derived as follows:
* If PaletteIndexMap[ xCbL + xL ][ yCbL + yL ] is equal to MaxPaletteIndex and palette\_escape\_val\_present\_flag is equal to 1, bIsEscapeSample is set equal to 1.
* Otherwise, bIsEscapeSample is set equal to 0.
* If bIsEscapeSample is equal to 0, the following applies:

recSamples[ x ][ y ] = CurrentPaletteEntries[ cIdx ][ PaletteIndexMap[ xCbL + xL ][ yCbL + yL ] ] (445)

* Otherwise (bIsEscapeSample is equal to 1), the following ordered steps apply:

1. The quantization parameter qP is derived as follows:

* If cIdx is equal to 0,

qP = Max( QpPrimeTsMin, Qp′Y ) (446)

* Otherwise, if cIdx is equal to 1,

qP = Max( QpPrimeTsMin, Qp′Cb ) (447)

* Otherwise (cIdx is equal to 2),

qP = Max( QpPrimeTsMin, Qp′Cr ) (448)

1. The list levelScale[ ] is specified as levelScale[ k ] = { 40, 45, 51, 57, 64, 72 } with k = 0..5.
2. The following applies:

tmpVal = ( PaletteEscapeVal[ cIdx ][ xCbL + xL ][ yCbL + yL ] \*  
 levelScale[ qP%6 ] ) << ( qP / 6 ) + 32 ) >> 6 (449)

recSamples[ x ][ y ] = Clip3( 0, ( 1 << BitDepth ) − 1, tmpVal ) (450)

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[ startComp ] − 1:

CurrentPaletteEntries[ 1 ][ NumPredictedPaletteEntries + i ] = 1 << ( BitDepth − 1 ) (451)

CurrentPaletteEntries[ 2 ][ NumPredictedPaletteEntries + i ] = 1 << ( BitDepth − 1 ) (452)

* The variables CurrentPaletteSize[ 0 ], startComp, numComps and maxNumPalettePredictorSize are derived as follows:

CurrentPaletteSize[ 0 ] = CurrentPaletteSize[ startComp ] (453)

startComp = 0 (454)

numComps = 3 (455)

maxNumPalettePredictorSize = 63 (456)

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++ ) (457)  
 newPredictorPaletteEntries[ cIdx ][ newPredictorPaletteSize ] =  
 PredictorPaletteEntries[ cIdx ][ i ]  
 newPredictorPaletteSize++  
 }  
for( cIdx = startComp; cIdx < ( startComp + numComps ); cIdx++ )  
 for( i = 0; i < newPredictorPaletteSize; i++ )  
 PredictorPaletteEntries[ cIdx ][ i ] = newPredictorPaletteEntries[ cIdx ][ i ]  
PredictorPaletteSize[ startComp ] = newPredictorPaletteSize

When sps\_qtbtt\_dual\_tree\_intra\_flag is equal to 0 or sh\_slice\_type is not equal to I, the following applies:

PredictorPaletteSize[ 1 ] = newPredictorPaletteSize (458)

It is a requirement of bitstream conformance that the value of PredictorPaletteSize[ startComp ] shall be in the range of 0 to maxNumPalettePredictorSize, inclusive.

## Decoding process for coding units coded in inter prediction mode

### General decoding process for coding units coded in inter prediction mode

Inputs to this process are:

* a luma location ( xCb, yCb ) specifying the top-left sample of the current coding block relative to the top‑left luma sample of the current picture,
* a variable cbWidth specifying the width of the current coding block in luma samples,
* a variable cbHeight specifying the height of the current coding block in luma samples,
* a variable treeType specifying whether a single or a dual tree is used and if a dual tree is used, it specifies whether the current tree corresponds to the luma or chroma components.

Output of this process is a modified reconstructed picture before in-loop filtering.

The derivation process for quantization parameters as specified in clause 8.7.1 is invoked with the luma location ( xCb, yCb ), the width of the current coding block in luma samples cbWidth and the height of the current coding block in luma samples cbHeight, and the variable treeType as inputs.

The decoding process for coding units coded in inter prediction mode consists of the following ordered steps:

1. The variable dmvrFlag is set equal to 0, the variables cbProfFlagL0 and cbProfFlagL1 are both set equal to 0, and the variable hpelIfIdx is set equal to 0.
2. The motion vector components and reference indices of the current coding unit are derived as follows:

* If MergeGpmFlag[ xCb ][ yCb ], inter\_affine\_flag[ xCb ][ yCb ] and merge\_subblock\_flag[ xCb ][ yCb ] are all equal to 0, the following applies:
* The derivation process for motion vector components and reference indices as specified in clause 8.5.2.1 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width cbWidth and the luma coding block height cbHeight as inputs, and the luma motion vectors mvL0[ 0 ][ 0 ] and mvL1[ 0 ][ 0 ], the reference indices refIdxL0 and refIdxL1 and the prediction list utilization flags predFlagL0[ 0 ][ 0 ] and predFlagL1[ 0 ][ 0 ], the half sample interpolation filter index hpelIfIdx, and the bi-prediction weight index bcwIdx as outputs.
* When all of the following conditions are true, dmvrFlag is set equal to 1:
* ph\_dmvr\_disabled\_flag is equal to 0.
* general\_merge\_flag[ xCb ][ yCb ] is equal to 1.
* both predFlagL0[ 0 ][ 0 ] and predFlagL1[ 0 ][ 0 ] are equal to 1.
* mmvd\_merge\_flag[ xCb ][ yCb ] is equal to 0.
* ciip\_flag[ xCb ][ yCb ] is equal to 0.
* DiffPicOrderCnt( currPic, RefPicList[ 0 ][ refIdxL0 ]) is equal to DiffPicOrderCnt( RefPicList[ 1 ][ refIdxL1 ], currPic ).
* RefPicList[ 0 ][ refIdxL0 ] is a short-term reference picture and RefPicList[ 1 ][ refIdxL1 ] is a short-term reference picture.
* BcwIdx[ xCb ][ yCb ] 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.
* RprConstraintsActive[ 0 ][ refIdxL0 ] is equal to 0 and RprConstraintsActive[ 1 ][ refIdxL1 ] is equal to 0.
* If dmvrFlag is equal to 1, the following applies:
* For X being 0 and 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 clause 8.5.6.2 with X and refIdxLX as inputs.
* The number of luma coding subblocks in horizontal direction numSbX and in vertical direction numSbY, the subblock width sbWidth and the subblock height sbHeight are derived as follows:

numSbX = ( cbWidth > 16 ) ? ( cbWidth  >>  4 ) : 1 (459)

numSbY = ( cbHeight > 16 ) ? ( cbHeight  >>  4 ) : 1 (460)

sbWidth = ( cbWidth > 16 ) ? 16 : cbWidth (461)

sbHeight = ( cbHeight > 16 ) ? 16 : cbHeight (462)

* 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 equal to 0 and 1, and the luma location ( xSb[xSbIdx][ySbIdx], ySb[xSbIdx][ySbIdx] ) specifying the top-left sample of the coding subblock relative to the top‑left luma sample of the current picture are derived as follows:

mvLX[ xSbIdx ][ ySbIdx ] = mvLX[ 0 ][ 0 ] (463)

predFlagLX[ xSbIdx ][ ySbIdx ] = predFlagLX[ 0 ][ 0 ] (464)

xSb[ xSbIdx ][ ySbIdx ] =  xCb + xSbIdx \* sbWidth (465)

ySb[ xSbIdx ][ ySbIdx ] =  yCb + ySbIdx \* sbHeight (466)

* The decoder-side motion vector refimenent process specified in clause  8.5.3.1 is invoked with xSb[ xSbIdx ][ ySbIdx ], ySb[ xSbIdx ][ ySbIdx ], sbWidth, sbHeight, the motion vectors mvLX[ xSbIdx ][ ySbIdx ] and the reference picture array refPicLXL as inputs and delta motion vectors dMvLX[ xSbIdx ][ ySbIdx ] with X equal to 0 and 1, and the mimimum sum of absolute difference in decoder-side motion vector refimenent process dmvrSad[ xSbIdx ][ ySbIdx ] as outputs.
* When ChromaArrayType is not equal to 0, the derivation process for chroma motion vectors in clause 8.5.2.13 is invoked with mvLX[ xSbIdx ][ ySbIdx ] as input, and mvCLX[ xSbIdx ][ ySbIdx ] as output with X equal to 0 and 1.
* Otherwise (dmvrFlag is equal to 0), the following applies:
* When ChromaArrayType is not equal to 0, and treeType is equal to SINGLE\_TREE, and predFlagLX[ 0 ][0 ], with X being 0 or 1, is equal to 1, the derivation process for chroma motion vectors in clause 8.5.2.13 is invoked with mvLX[ 0 ][ 0 ] as input, and mvCLX[ 0 ][ 0 ] as output.
* The number of luma coding subblocks in horizontal direction numSbX and in vertical direction numSbY are both set equal to 1.
* Otherwise, if MergeGpmFlag[ xCb ][ yCb ] is equal to 1, inter\_affine\_flag[ xCb ][ yCb ] and merge\_subblock\_flag[ xCb ][ yCb ] are both equal to 0, the following applies:
* The derivation process for geometric partitioning mode motion vector components and reference indices as specified in clause 8.5.4.1 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width cbWidth and the luma coding block height cbHeight as inputs, and the luma motion vectors mvA and mvB, the chroma motion vectors mvCA and mvCB, the reference indices refIdxA and refIdxB and the prediction list flags predListFlagA and predListFlagB as outputs.
* The number of luma coding subblocks in horizontal direction numSbX and in vertical direction numSbY are both set equal to 1.
* Otherwise (inter\_affine\_flag[ xCb ][ yCb ] or merge\_subblock\_flag[ xCb ][ yCb ] is equal to 1), the derivation process for subblock motion vector components and reference indices as specified in clause 8.5.5.1 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width cbWidth, the luma coding block height cbHeight as inputs, and the reference indices refIdxL0 and refIdxL1, the number of luma coding subblocks in horizontal direction numSbX and in vertical direction numSbY, the prediction list utilization flags predFlagLX[ xSbIdx ][ ySbIdx ], the luma motion vector array mvLX[ xSbIdx ][ ySbIdx ], and the chroma motion vector array mvCLX[ xSbIdx ][ ySbIdx ] with xSbIdx = 0..numSbX − 1, and ySbIdx = 0..numSbY − 1, and with X being 0 or 1, the bi-prediction weight index bcwIdx, the prediction refinement utility flags cbProfFlagL0 and cbProfFlagL1, and motion vector difference arrays diffMvL0[ xIdx ][ yIdx ] and diffMvL1[ xIdx ][ yIdx ] with xIdx = 0.. cbWidth/numSbX − 1, yIdx = 0.. cbHeight/numSbY − 1 as outputs.

1. The arrays of luma and chroma motion vectors after decoder-side motion vector refinement, refMvLX[ xSbIdx ][ ySbIdx ] and refMvCLX[ xSbIdx ][ ySbIdx ], with X being 0 and 1, are derived as follows for xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1:

* If dmvrFlag is equal to 1, the derivation process for chroma motion vectors in clause 8.5.2.13 is invoked with refMvLX[ xSbIdx ][ ySbIdx ] as inputs, and refMvCLX[ xSbIdx ][ ySbIdx ] as output and the input refMvLX[ xSbIdx ][ ySbIdx ] is derived as follows;

refMvLX[ xSbIdx ][ ySbIdx ] = mvLX[ xSbIdx ][ ySbIdx ] + dMvLX[ xSbIdx ][ ySbIdx ] (467)

refMvLX[ xSbIdx ][ ySbIdx ][ 0 ] = Clip3( −217, 217 − 1, refMvLX[ xSbIdx ][ ySbIdx ][ 0 ] ) (468)

refMvLX[ xSbIdx ][ ySbIdx ][ 1 ] = Clip3( −217, 217− 1, refMvLX[ xSbIdx ][ ySbIdx ][ 1 ] ) (469)

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

refMvLX[ xSbIdx ][ ySbIdx ] = mvLX[ xSbIdx ][ ySbIdx ] (470)

refMvCLX [ xSbIdx ][ ySbIdx ] = mvCLX[ xSbIdx ][ ySbIdx ] (471)

NOTE – The array refMvLX is stored in MvDmvrLX and used in the derivation process for collocated motion vectors in clause 8.5.2.12. The array of non-refine luma motion vectors MvLX is used in the spatial motion vector prediction and deblocking boundary strength derivation processes.

1. The prediction samples of the current coding unit are derived as follows:

* If MergeGpmFlag[ xCb ][ yCb ] is equal to 0, the prediction samples of the current coding unit are derived as follows:
  + - * The decoding process for inter blocks as specified in clause 8.5.6.1 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width cbWidth and the luma coding block height cbHeight, the number of luma coding subblocks in horizontal direction numSbX and in vertical direction numSbY, the luma motion vectors mvL0[ xSbIdx ][ ySbIdx ] and mvL1[ xSbIdx ][ ySbIdx ], and the refined luma motion vectors refMvL0[ xSbIdx ][ ySbIdx ] and refMvL1[ xSbIdx ][ ySbIdx ] with xSbIdx = 0..numSbX − 1, and ySbIdx = 0..numSbY − 1, the reference indices refIdxL0 and refIdxL1, the prediction list utilization flags predFlagL0[ xSbIdx ][ ySbIdx ] and predFlagL1[ xSbIdx ][ ySbIdx ], the half sample interpolation filter index hpelIfIdx, the bi-prediction weight index bcwIdx, the mimimum sum of absolute difference values in decoder-side motion vector refimenent 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 predSamplesL of prediction luma samples as outputs.
      * When ChromaArrayType is not equal to 0. the decoding process for inter blocks as specified in clause 8.5.6.1 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width cbWidth and the luma coding block height cbHeight, the number of luma coding subblocks in horizontal direction numSbX and in vertical direction numSbY, the chroma motion vectors mvCL0[ xSbIdx ][ ySbIdx ] and mvCL1[ xSbIdx ][ ySbIdx ], and the refined chroma motion vectors refMvCL0[ xSbIdx ][ ySbIdx ] and refMvCL1[ xSbIdx ][ ySbIdx ] with xSbIdx = 0..numSbX − 1, and ySbIdx = 0..numSbY − 1, the reference indices refIdxL0 and refIdxL1, the prediction list utilization flags predFlagL0[ xSbIdx ][ ySbIdx ] and predFlagL1[ xSbIdx ][ ySbIdx ], the half sample interpolation filter index hpelIfIdx, the bi-prediction weight index bcwIdx, the mimimum sum of absolute difference values in decoder-side motion vector refimenent 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 predSamplesCb of prediction chroma samples for the chroma components Cb as outputs.
      * When ChromaArrayType is not equal to 0. the decoding process for inter blocks as specified in clause 8.5.6.1 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width cbWidth and the luma coding block height cbHeight, the number of luma coding subblocks in horizontal direction numSbX and in vertical direction numSbY, the chroma motion vectors mvCL0[ xSbIdx ][ ySbIdx ] and mvCL1[ xSbIdx ][ ySbIdx ], and the refined chroma motion vectors refMvCL0[ xSbIdx ][ ySbIdx ] and refMvCL1[ xSbIdx ][ ySbIdx ] with xSbIdx = 0..numSbX − 1, and ySbIdx = 0..numSbY − 1, the reference indices refIdxL0 and refIdxL1, the prediction list utilization flags predFlagL0[ xSbIdx ][ ySbIdx ] and predFlagL1[ xSbIdx ][ ySbIdx ], the half sample interpolation filter index hpelIfIdx, the bi-prediction weight index bcwIdx, the mimimum sum of absolute difference values in decoder-side motion vector refimenent process dmvrSad[ xSbIdx ][ ySbIdx ], the decoder-side motion vector refinement flag dmvrFlag, the variable cIdx set equal to 2, the prediction refinement utility flags cbProfFlagL0 and cbProfFlagL1, and motion vector difference arrays diffMvL0[ xIdx ][ yIdx ] and diffMvL1[ xIdx ][ yIdx ] with xIdx = 0..cbWidth / numSbX − 1, and yIdx = 0..cbHeight / numSbY − 1 as inputs, and the inter prediction samples (predSamples) that are an (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array predSamplesCr of prediction chroma samples for the chroma components Cr as outputs.
* Otherwise (MergeGpmFlag[ xCb ][ yCb ] is equal to 1), the decoding process for geometric partitioning mode inter blocks as specified in clause 8.5.7.1 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width cbWidth and the luma coding block height cbHeight, the luma motion vectors mvA and mvB, the chroma motion vectors mvCA and mvCB, the reference indices refIdxA and refIdxB, and the prediction list flags predListFlagA and predListFlagB as inputs, and the inter prediction samples (predSamples) that are an (cbWidth)x(cbHeight) array predSamplesL of prediction luma samples and, when ChromaArrayType is not equal to 0, two (cbWidth / SubWidthC)x(cbHeight / SubHeightC) arrays predSamplesCb and predSamplesCr of prediction chroma samples, one for each of the chroma components Cb and Cr, as outputs.

1. The variables NumSbX[ xCb ][ yCb ] and NumSbY[ xCb ][ yCb ] are set equal to numSbX and numSbY, respectively.
2. The residual samples of the current coding unit are derived as follows:

* The decoding process for the residual signal of coding blocks coded in inter prediction mode as specified in clause 8.5.8 is invoked with the location ( xTb0, yTb0 ) set equal to the luma location ( xCb, yCb ), the width nTbW set equal to the luma coding block width cbWidth, the height nTbH set equal to the luma coding block height cbHeight and the variable cIdx set equal to 0 as inputs, and the array resSamplesL as output.
* When ChromaArrayType is not equal to 0. the decoding process for the residual signal of coding blocks coded in inter prediction mode as specified in clause 8.5.8 is invoked with the location ( xTb0, yTb0 ) set equal to the chroma location ( xCb / SubWidthC, yCb / SubHeightC ), the width nTbW set equal to the chroma coding block width cbWidth / SubWidthC, the height nTbH set equal to the chroma coding block height cbHeight / SubHeightC and the variable cIdx set equal to 1 as inputs, and the array resSamplesCb as output.
* When ChromaArrayType is not equal to 0. the decoding process for the residual signal of coding blocks coded in inter prediction mode as specified in clause 8.5.8 is invoked with the location ( xTb0, yTb0 ) set equal to the chroma location ( xCb / SubWidthC, yCb / SubHeightC ), the width nTbW set equal to the chroma coding block width cbWidth / SubWidthC, the height nTbH set equal to the chroma coding block height cbHeight / SubHeightC and the variable cIdx set equal to 2 as inputs, and the array 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 clause 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.

1. The reconstructed samples of the current coding unit are derived as follows:

* The picture reconstruction process for a colour component as specified in clause 8.7.5 is invoked with the block location ( xB, yB ) set equal to ( xCb, yCb ), the block width bWidth set equal to cbWidth, the block height bHeight set equal to cbHeight, the variable cIdx set equal to 0, the (cbWidth)x(cbHeight) array predSamples set equal to predSamplesL and the (cbWidth)x(cbHeight) array resSamples set equal to resSamplesL as inputs, and the output is a modified reconstructed picture before in-loop filtering.
* When ChromaArrayType is not equal to 0. the picture reconstruction process for a colour component as specified in clause 8.7.5 is invoked with the block location ( xB, yB ) set equal to ( xCb / SubWidthC, yCb / SubHeightC ), the block width bWidth set equal to cbWidth / SubWidthC, the block height bHeight set equal to cbHeight / SubHeightC, the variable cIdx set equal to 1, the (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array predSamples set equal to predSamplesCb and the (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array resSamples set equal to resSamplesCb as inputs, and the output is a modified reconstructed picture before in-loop filtering.
* When ChromaArrayType is not equal to 0. the picture reconstruction process for a colour component as specified in clause 8.7.5 is invoked with the block location ( xB, yB ) set equal to ( xCb / SubWidthC, yCb / SubHeightC ), the block width bWidth set equal to cbWidth / SubWidthC, the block height bHeight set equal to cbHeight / SubHeightC, the variable cIdx set equal to 2, the (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array predSamples set equal to predSamplesCr and the (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array resSamples set equal to resSamplesCr as inputs, and the output is a modified reconstructed picture before in-loop filtering.

### Derivation process for motion vector components and reference indices

#### General

Inputs to this process are:

* a luma location ( xCb, yCb ) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
* a variable cbWidth specifying the width of the current coding block in luma samples,
* a variable cbHeight specifying the height of the current coding block in luma samples.

Outputs of this process are:

* the luma motion vectors in 1/16 fractional-sample accuracy mvL0[ 0 ][ 0 ] and mvL1[ 0 ][ 0 ],
* the reference indices refIdxL0 and refIdxL1,
* the prediction list utilization flags predFlagL0[ 0 ][ 0 ] and predFlagL1[ 0 ][ 0 ],
* the half sample interpolation filter index hpelIfIdx,
* the bi-prediction weight index bcwIdx.

Let the variable LX be RefPicList[ X ], with X being 0 or 1, of the current picture.

For the derivation of the variables mvL0[ 0 ][ 0 ] and mvL1[ 0 ][ 0 ], refIdxL0 and refIdxL1, as well as predFlagL0[ 0 ][ 0 ] and predFlagL1[ 0 ][ 0 ], the following applies:

* If general\_merge\_flag[ xCb ][ yCb ] is equal to 1, the derivation process for luma motion vectors for merge mode as specified in clause 8.5.2.2 is invoked with the luma location ( xCb, yCb ), the variables cbWidth and cbHeight inputs, and the output being the luma motion vectors mvL0[ 0 ][ 0 ], mvL1[ 0 ][ 0 ], the reference indices refIdxL0, refIdxL1, the prediction list utilization flags predFlagL0[ 0 ][ 0 ] and predFlagL1[ 0 ][ 0 ], the half sample interpolation filter index hpelIfIdx, the bi-prediction weight index bcwIdx and the merging candidate list mergeCandList.
* Otherwise, the following applies:
* For X being replaced by either 0 or 1 in the variables predFlagLX[ 0 ][0 ], mvLX[ 0 ][0 ] and refIdxLX, in PRED\_LX, and in the syntax elements ref\_idx\_lX and MvdLX, the following ordered steps apply:

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,

refIdxLX = ref\_idx\_lX[ xCb ][ yCb ] (472)

predFlagLX[ 0 ][0 ] = 1 (473)

* Otherwise, the variables refIdxLX and predFlagLX[ 0 ][0 ] are specified by:

refIdxLX = −1 (474)

predFlagLX[ 0 ][0 ] = 0 (475)

1. The variable mvdLX is derived as follows:

mvdLX[ 0 ] = MvdLX[ xCb ][ yCb ][ 0 ] (476)

mvdLX[ 1 ] = MvdLX[ xCb ][ yCb ][ 1 ] (477)

1. When predFlagLX[ 0 ][ 0 ] is equal to 1, the derivation process for luma motion vector prediction in clause 8.5.2.8 is invoked with the luma coding block location ( xCb, yCb ), the coding block width cbWidth, the coding block height cbHeight and the variable refIdxLX as inputs, and the output being mvpLX.
2. 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 ] + 218 ) % 218 (478)

mvLX[ 0 ][ 0 ][ 0 ] = ( uLX[ 0 ] >= 217 ) ? ( uLX[ 0 ] − 218 ) : uLX[ 0 ] (479)

uLX[ 1 ] = ( mvpLX[ 1 ] + mvdLX[ 1 ] + 218 ) % 218 (480)

mvLX[ 0 ][ 0 ][ 1 ] = ( uLX[ 1 ] >= 217 ) ? ( uLX[ 1 ] − 218 ) : uLX[ 1 ] (481)

NOTE 1– The resulting values of mvLX[ 0 ][ 0 ][ 0 ] and mvLX[ 0 ][ 0 ][ 1 ] as specified above will always be in the range of −217 to 217 − 1, inclusive.

* The half sample interpolation filter index hpelIfIdx is derived as follows:

hpelIfIdx = AmvrShift = =  3 ? 1: 0 (482)

* The bi-prediction weight index bcwIdx is set equal to bcw\_idx[ xCb ][ yCb ].

When all of the following conditions are true, refIdxL1 is set equal to −1, predFlagL1 is set equal to 0, and bcwIdx is set equal to 0:

* predFlagL0[ 0 ][ 0 ] is equal to 1.
* predFlagL1[ 0 ][ 0 ] is equal to 1.
* The value of ( cbWidth + cbHeight ) is equal to 12.

When ( ( xCb + cbWidth ) >> Log2ParMrgLevel is greater than ( xCb >> Log2ParMrgLevel ) and ( ( yCb + cbHeight ) >> Log2ParMrgLevel is greater than ( yCb >> Log2ParMrgLevel ), the updating process for the history-based motion vector predictor list as specified in clause 8.5.2.16 is invoked with luma motion vectors mvL0[ 0 ][ 0 ] and mvL1[ 0 ][ 0 ], reference indices refIdxL0 and refIdxL1, prediction list utilization flags predFlagL0[ 0 ][ 0 ] and predFlagL1[ 0 ][ 0 ], the bi-prediction weight index bcwIdx, and the half-sample interpolation filter index hpelIfIdx.

#### Derivation process for luma motion vectors for merge mode

This process is only invoked when general\_merge\_flag[ xCb ][ yCb ] is equal to 1, where ( xCb, yCb ) specify the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture.

Inputs to this process are:

* a luma location ( xCb, yCb ) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
* a variable cbWidth specifying the width of the current coding block in luma samples,
* a variable cbHeight specifying the height of the current coding block in luma samples.

Outputs of this process are:

* the luma motion vectors in 1/16 fractional-sample accuracy mvL0[ 0 ][ 0 ] and mvL1[ 0 ][ 0 ],
* the reference indices refIdxL0 and refIdxL1,
* the prediction list utilization flags predFlagL0[ 0 ][ 0 ] and predFlagL1[ 0 ][ 0 ],
* the half sample interpolation filter index hpelIfIdx,
* the bi-prediction weight index bcwIdx.
* the merging candidate list mergeCandList.

The bi-prediction weight index bcwIdx is set equal to 0.

The motion vectors mvL0[ 0 ][ 0 ] and mvL1[ 0 ][ 0 ], the reference indices refIdxL0 and refIdxL1 and the prediction utilization flags predFlagL0[ 0 ][ 0 ] and predFlagL1[ 0 ][ 0 ] are derived by the following ordered steps:

1. The derivation process for spatial merging candidates from neighbouring coding units as specified in clause 8.5.2.3 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width cbWidth, and the luma coding block height cbHeight as inputs, and the output being the availability flags availableFlagA0, availableFlagA1, availableFlagB0, availableFlagB1 and availableFlagB2, the reference indices refIdxLXA0, refIdxLXA1, refIdxLXB0, refIdxLXB1 and refIdxLXB2, the prediction list utilization flags predFlagLXA0, predFlagLXA1, predFlagLXB0, predFlagLXB1 and predFlagLXB2, and the motion vectors mvLXA0, mvLXA1, mvLXB0, mvLXB1 and mvLXB2, with X being 0 or 1, the half sample interpolation filter indices hpelIfIdxA0,hpelIfIdxA1, hpelIfIdxB0, hpelIfIdxB1, hpelIfIdxB2, and the bi-prediction weight indices bcwIdxA0,bcwIdxA1, bcwIdxB0,bcwIdxB1, bcwIdxB2.
2. The reference indices, refIdxLXCol, with X being 0 or 1, and the bi-prediction weight index bcwIdxCol for the temporal merging candidate Col are set equal to 0 and hpelIfIdxCol is set equal to 0.
3. The derivation process for temporal luma motion vector prediction as specified in in clause 8.5.2.11 is invoked with the luma location ( xCb, yCb ), the luma coding block width cbWidth, the luma coding block height cbHeight and the variable refIdxL0Col as inputs, and the output being the availability flag availableFlagL0Col and the temporal motion vector mvL0Col. The variables availableFlagCol, predFlagL0Col and predFlagL1Col are derived as follows:

availableFlagCol = availableFlagL0Col (483)

predFlagL0Col = availableFlagL0Col (484)

predFlagL1Col = 0 (485)

1. When sh\_slice\_type is equal to B, the derivation process for temporal luma motion vector prediction as specified in clause 8.5.2.11 is invoked with the luma location ( xCb, yCb ), the luma coding block width cbWidth, the luma coding block height cbHeight and the variable refIdxL1Col as inputs, and the output being the availability flag availableFlagL1Col and the temporal motion vector mvL1Col. The variables availableFlagCol and predFlagL1Col are derived as follows:

availableFlagCol = availableFlagL0Col  | |  availableFlagL1Col (486)

predFlagL1Col = availableFlagL1Col (487)

1. The merging candidate list, mergeCandList, is constructed as follows:

i = 0  
if( availableFlagB1 )  
 mergeCandList[ i++ ] = B1  
if( availableFlagA1 )  
 mergeCandList[ i++ ] = A1if( availableFlagB0 )  
 mergeCandList[ i++ ] = B0 (488)if( availableFlagA0 )  
 mergeCandList[ i++ ] = A0if( availableFlagB2 )  
 mergeCandList[ i++ ] = B2if( availableFlagCol )  
 mergeCandList[ i++ ] = Col

1. The variable numCurrMergeCand and numOrigMergeCand are set equal to the number of merging candidates in the mergeCandList.
2. 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.

1. When numCurrMergeCand is less than MaxNumMergeCand and greater than 1, the following applies:

* The derivation process for pairwise average merging candidate specified in clause 8.5.2.4 is invoked with mergeCandList, the reference indices refIdxL0N and refIdxL1N, the prediction list utilization flags predFlagL0N and predFlagL1N, the motion vectors mvL0N and mvL1N, and the half sample interpolation filter index hpelIfIdxN of every candidate N in mergeCandList, and numCurrMergeCand as inputs, and the output is assigned to mergeCandList, numCurrMergeCand, the reference indices refIdxL0avgCand and refIdxL1avgCand, the prediction list utilization flags predFlagL0avgCand and predFlagL1avgCand, the motion vectors mvL0avgCand and mvL1avgCand, and the half-sample interpolation filter index hpelIfIdxavgCand of candidate avgCand being added into mergeCandList. The bi-prediction weight index bcwIdx of candidate avgCand being added into mergeCandList is set equal to 0.
* numOrigMergeCand is set equal to numCurrMergeCand.

1. The derivation process for zero motion vector merging candidates specified in clause 8.5.2.5 is invoked with the mergeCandList, the reference indices refIdxL0N and refIdxL1N, the prediction list utilization flags predFlagL0N and predFlagL1N, the motion vectors mvL0N and mvL1N of every candidate N in mergeCandList and numCurrMergeCand as inputs, and the output is assigned to mergeCandList, numCurrMergeCand, the reference indices refIdxL0zeroCandm and refIdxL1zeroCandm, the prediction list utilization flags predFlagL0zeroCandm and predFlagL1zeroCandm and the motion vectors mvL0zeroCandm and mvL1zeroCandm of every new candidate zeroCandm being added into mergeCandList. The half sample interploation filter index hpelIfIdx of every new candidate zeroCandm being added into mergeCandList is set equal to 0. The bi-prediction weight index bcwIdx of every new candidate zeroCandm 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.
2. 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 ] ] ) and X being replaced by 0 or 1:

refIdxLX = refIdxLXN (489)

predFlagLX[ 0 ][ 0 ] = predFlagLXN (490)

mvLX[ 0 ][ 0 ][ 0 ] = mvLXN[ 0 ] (491)

mvLX[ 0 ][ 0 ][ 1 ] = mvLXN[ 1 ] (492)

hpelIfIdx = hpelIfIdxN (493)

bcwIdx = bcwIdxN (494)

1. 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.
* The motion vector difference mMvdLX is added to the merge motion vectors mvLX for X being 0 and 1 as follows:

mvLX[ 0 ][ 0 ][ 0 ] += mMvdLX[ 0 ] (495)

mvLX[ 0 ][ 0 ][ 1 ] += mMvdLX[ 1 ] (496)

mvLX[ 0 ][ 0 ][ 0 ] = Clip3( −217, 217 − 1, mvLX[ 0 ][ 0 ][ 0 ] ) (497)

mvLX[ 0 ][ 0 ] [ 1 ] = Clip3( −217, 217− 1, mvLX[ 0 ][ 0 ][ 1 ] ) (498)

#### Derivation process for spatial merging candidates

Inputs to this process are:

* a luma location ( xCb, yCb ) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
* a variable cbWidth specifying the width of the current coding block in luma samples,
* a variable cbHeight specifying the height of the current coding block in luma samples.

Outputs of this process are as follows, with X being 0 or 1:

* the availability flags availableFlagA0, availableFlagA1, availableFlagB0, availableFlagB1 and availableFlagB2 of the neighbouring coding units,
* the reference indices refIdxLXA0, refIdxLXA1, refIdxLXB0, refIdxLXB1 and refIdxLXB2 of the neighbouring coding units,
* the prediction list utilization flags predFlagLXA0, predFlagLXA1, predFlagLXB0, predFlagLXB1 and predFlagLXB2 of the neighbouring coding units,
* the motion vectors in 1/16 fractional-sample accuracy mvLXA0, mvLXA1, mvLXB0, mvLXB1 and mvLXB2 of the neighbouring coding units,
* the half sample interpolation filter indices hpelIfIdxA0,hpelIfIdxA1, hpelIfIdxB0,hpelIfIdxB1, and hpelIfIdxB2,
* the bi-prediction weight indices bcwIdxA0,bcwIdxA1, bcwIdxB0,bcwIdxB1, and bcwIdxB2.

For the derivation of availableFlagB1, refIdxLXB1, predFlagLXB1, mvLXB1, hpelIfIdxB1 and bcwIdxB1 the following applies:

* The luma location ( xNbB1, yNbB1 ) inside the neighbouring luma coding block is set equal to ( xCb + cbWidth − 1,  yCb − 1 ).
* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring luma location ( xNbB1, yNbB1 ), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag availableB1.
* When xCb >> Log2ParMrgLevel is equal to xNbB1 >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to yNbB1 >> Log2ParMrgLevel, availableB1 is set equal to FALSE.
* The variables availableFlagB1, refIdxLXB1, predFlagLXB1, mvLXB1, hpelIfIdxB1 and bcwIdxB1 are derived as follows:
* If availableB1 is equal to FALSE, availableFlagB1 is set equal to 0, both components of mvLXB1 are set equal to 0, refIdxLXB1 is set equal to −1 and predFlagLXB1 is set equal to 0, with X being 0 or 1, hpelIfIdxB1 is set equal to 0, and bcwIdxB1 is set equal to 0.
* Otherwise, availableFlagB1 is set equal to 1 and the following assignments are made:

mvLXB1 = MvLX[ xNbB1 ][ yNbB1 ] (499)

refIdxLXB1 = RefIdxLX[ xNbB1 ][ yNbB1 ] (500)

predFlagLXB1 = PredFlagLX[ xNbB1 ][ yNbB1 ] (501)

hpelIfIdxB1 = HpelIfIdx[ xNbB1 ][ yNbB1 ] (502)

bcwIdxB1 = BcwIdx[ xNbB1 ][ yNbB1 ] (503)

For the derivation of availableFlagA1, refIdxLXA1, predFlagLXA1, mvLXA1, hpelIfIdxA1 and bcwIdxA1 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 clause 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, mvLXA1, hpelIfIdxA1 and bcwIdxA1 are derived as follows:
* If one or more of the following conditions are true, 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 being 0 or 1, hpelIfIdxA1 is set equal to 0, and bcwIdxA1 is set equal to 0:
  + availableA1 is equal to FALSE.
  + availableB1 is equal to TRUE and the luma locations ( xNbA1, yNbA1 ) and ( xNbB1, yNbB1 ) have the same motion vectors and the same reference indices.
* Otherwise, availableFlagA1 is set equal to 1 and the following assignments are made:

mvLXA1 = MvLX[ xNbA1 ][ yNbA1 ] (504)

refIdxLXA1 = RefIdxLX[ xNbA1 ][ yNbA1 ] (505)

predFlagLXA1 = PredFlagLX[ xNbA1 ][ yNbA1 ] (506)

hpelIfIdxA1 = HpelIfIdx[ xNbA1 ][ yNbA1 ] (507)

bcwIdxA1 = BcwIdx[ xNbA1 ][ yNbA1 ] (508)

For the derivation of availableFlagB0, refIdxLXB0, predFlagLXB0, mvLXB0, hpelIfIdxB0 and bcwIdxB0 the following applies:

* The luma location ( xNbB0, yNbB0 ) inside the neighbouring luma coding block is set equal to ( xCb + cbWidth, yCb − 1 ).
* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring luma location ( xNbB0, yNbB0 ), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag availableB0.
* When xCb >> Log2ParMrgLevel is equal to xNbB0 >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to yNbB0 >> Log2ParMrgLevel, availableB0 is set equal to FALSE.
* The variables availableFlagB0, refIdxLXB0, predFlagLXB0, mvLXB0, hpelIfIdxB0 and bcwIdxB0 are derived as follows:
* If one or more of the following conditions are true, availableFlagB0 is set equal to 0, both components of mvLXB0 are set equal to 0, refIdxLXB0 is set equal to −1 and predFlagLXB0 is set equal to 0, with X being 0 or 1, hpelIfIdxB0 is set equal to 0, and bcwIdxB0 is set equal to 0:
  + availableB0 is equal to FALSE.
  + availableB1 is equal to TRUE and the luma locations ( xNbB1, yNbB1 ) and ( xNbB0, yNbB0 ) have the same motion vectors and the same reference indices.
* Otherwise, availableFlagB0 is set equal to 1 and the following assignments are made:

mvLXB0 = MvLX[ xNbB0 ][ yNbB0 ] (509)

refIdxLXB0 = RefIdxLX[ xNbB0 ][ yNbB0 ] (510)

predFlagLXB0 = PredFlagLX[ xNbB0 ][ yNbB0 ] (511)

hpelIfIdxB0 = HpelIfIdx[ xNbB0 ][ yNbB0 ] (512)

bcwIdxB0 = BcwIdx[ xNbB0 ][ yNbB0 ] (513)

For the derivation of availableFlagA0, refIdxLXA0, predFlagLXA0, mvLXA0, hpelIfIdxA0 and bcwIdxA0 the following applies:

* The luma location ( xNbA0, yNbA0 ) inside the neighbouring luma coding block is set equal to ( xCb − 1,  yCb + cbWidth ).
* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ). the neighbouring luma location ( xNbA0, yNbA0 ), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag availableA0.
* When xCb >> Log2ParMrgLevel is equal to xNbA0 >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to yNbA0 >> Log2ParMrgLevel, availableA0 is set equal to FALSE.
* The variables availableFlagA0, refIdxLXA0, predFlagLXA0, mvLXA0, hpelIfIdxA0 and bcwIdxA0 are derived as follows:
* If one or more of the following conditions are true, availableFlagA0 is set equal to 0, both components of mvLXA0 are set equal to 0, refIdxLXA0 is set equal to −1 and predFlagLXA0 is set equal to 0, with X being 0 or 1, hpelIfIdxA0 is set equal to 0, and bcwIdxA0 is set equal to 0:
  + availableA0 is equal to FALSE.
  + availableA1 is equal to TRUE and the luma locations ( xNbA1, yNbA1 ) and ( xNbA0, yNbA0 ) have the same motion vectors and the same reference indices.
* Otherwise, availableFlagA0 is set equal to 1 and the following assignments are made:

mvLXA0 = MvLX[ xNbA0 ][ yNbA0 ] (514)

refIdxLXA0 = RefIdxLX[ xNbA0 ][ yNbA0 ] (515)

predFlagLXA0 = PredFlagLX[ xNbA0 ][ yNbA0 ] (516)

hpelIfIdxA0 = HpelIfIdx[ xNbA0 ][ yNbA0 ] (517)

bcwIdxA0 = BcwIdx[ xNbA0 ][ yNbA0 ] (518)

For the derivation of availableFlagB2, refIdxLXB2, predFlagLXB2, mvLXB2, hpelIfIdxB2 and bcwIdxB2 the following applies:

* The luma location ( xNbB2, yNbB2 ) inside the neighbouring luma coding block is set equal to ( xCb − 1, yCb − 1 ).
* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring luma location ( xNbB2, yNbB2 ), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag availableB2.
* When xCb >> Log2ParMrgLevel is equal to xNbB2 >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to yNbB2 >> Log2ParMrgLevel, availableB2 is set equal to FALSE.
* The variables availableFlagB2, refIdxLXB2, predFlagLXB2, mvLXB2, hpelIfIdxB2 and bcwIdxB2  are derived as follows:
* If one or more of the following conditions are true, availableFlagB2 is set equal to 0, both components of mvLXB2 are set equal to 0, refIdxLXB2 is set equal to −1 and predFlagLXB2 is set equal to 0, with X being 0 or 1, hpelIfIdxB2 is set equal to 0, and bcwIdxB2 is set equal to 0:
  + availableB2 is equal to FALSE.
  + availableA1 is equal to TRUE and the luma locations ( xNbA1, yNbA1 ) and ( xNbB2, yNbB2 ) have the same motion vectors and the same reference indices.
  + availableB1 is equal to TRUE and the luma locations ( xNbB1, yNbB1 ) and ( xNbB2, yNbB2 ) have the same motion vectors and the same reference indices.
  + availableFlagA0 + availableFlagA1 + availableFlagB0 + availableFlagB1 is equal to 4.
* Otherwise, availableFlagB2 is set equal to 1 and the following assignments are made:

mvLXB2 = MvLX[ xNbB2 ][ yNbB2 ] (519)

refIdxLXB2 = RefIdxLX[ xNbB2 ][ yNbB2 ] (520)

predFlagLXB2 = PredFlagLX[ xNbB2 ][ yNbB2 ] (521)

hpelIfIdxB2 = HpelIfIdx[ xNbB2 ][ yNbB2 ] (522)

bcwIdxB2 = BcwIdx[ xNbB2 ][ yNbB2 ] (523)

#### Derivation process for pairwise average merging candidate

Inputs to this process are:

* a merging candidate list mergeCandList,
* the reference indices refIdxL0N and refIdxL1N of every candidate N in mergeCandList,
* the prediction list utilization flags predFlagL0N and predFlagL1N of every candidate N in mergeCandList,
* the motion vectors in 1/16 fractional-sample accuracy mvL0N and mvL1N of every candidate N in mergeCandList,
* the half sample interpolation filter index hpelIfIdxN of every candidate N in mergeCandList,
* the number of elements numCurrMergeCand within mergeCandList.

Outputs of this process are:

* the merging candidate list mergeCandList,
* the number of elements numCurrMergeCand within mergeCandList,
* the reference indices refIdxL0avgCand and refIdxL1avgCand of candidate avgCand added into mergeCandList during the invocation of this process,
* the prediction list utilization flags predFlagL0avgCand and predFlagL1avgCand of candidate avgCand added into mergeCandList during the invocation of this process,
* the motion vectors in 1/16 fractional-sample accuracy mvL0avgCand and mvL1avgCand of candidate avgCand added into mergeCandList during the invocation of this process,
* the half sample interpolation filter index hpelIfIdxavgCand of every candidate avgCand added into mergeCandList during the invocation of this process.

The variable numRefLists is derived as follows:

numRefLists = ( sh\_slice\_type = = B ) ? 2 : 1 (524)

The following assignments are made, with p0Cand being the candidate at position 0 and p1Cand being the candidate at position 1 in the merging candidate list mergeCandList:

p0Cand = mergeCandList[ 0 ] (525)

p1Cand = mergeCandList[ 1 ] (526)

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 reference picture list LX with X ranging from 0 to ( numRefLists − 1 ), the following applies:
  + If predFlagLXp0Cand is equal to 1 and predFlagLXp1Cand is equal to 1, the variables refIdxLXavgCand, predFlagLXavgCand, mvLXavgCand[ 0 ], and mvLXavgCand[ 1 ] are derived as follows:

refIdxLXavgCand = refIdxLXp0Cand (527)

predFlagLXavgCand = 1 (528)

* + - The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX[ 0 ] set equal to mvLXp0Cand[ 0 ] + mvLXp1Cand[ 0 ], mvX[ 1 ] set equal to mvLXp0Cand[ 1 ] + mvLXp1Cand[ 1 ], rightShift set equal to1, and leftShift set equal to 0 as inputs and the rounded mvLXavgCand[ 0 ] as output.
  + Otherwise, if predFlagLXp0Cand is equal to 1 and predFlagLXp1Cand is equal to 0, the variables refIdxLXavgCand, predFlagLXavgCand, mvLXavgCand[ 0 ], mvLXavgCand[ 1 ] are derived as follows:

refIdxLXavgCand = refIdxLXp0Cand (529)

predFlagLXavgCand = 1 (530)

mvLXavgCand[ 0 ] = mvLXp0Cand[ 0 ] (531)

mvLXavgCand[ 1 ] = mvLXp0Cand[ 1 ] (532)

* + Otherwise, if predFlagLXp0Cand is equal to 0 and predFlagLXp1Cand is equal to 1, the variables refIdxLXavgCand, predFlagLXavgCand, mvLXavgCand[ 0 ], mvLXavgCand[ 1 ] are derived as follows:

refIdxLXavgCand = refIdxLXp1Cand (533)

predFlagLXavgCand = 1 (534)

mvLXavgCand[ 0 ] = mvLXp1Cand[ 0 ] (535)

mvLXavgCand[ 1 ] = mvLXp1Cand[ 1 ] (536)

* + Otherwise, if predFlagLXp0Cand is equal to 0 and predFlagLXp1Cand is equal to 0, the variables refIdxLXavgCand, predFlagLXavgCand, mvLXavgCand[ 0 ], mvLXavgCand[ 1 ] are derived as follows:

refIdxLXavgCand = −1 (537)

predFlagLXavgCand = 0 (538)

mvLXavgCand[ 0 ] = 0 (539)

mvLXavgCand[ 1 ] = 0 (540)

* When numRefLists is equal to 1, the following applies:

refIdxL1avgCand = −1 (541)

predFlagL1avgCand = 0 (542)

* The half sample interpolation filter index hpelIfIdxavgCand is derived as follows:
* If hpelIfIdxp0Cand is equal to hpelIfIdxp1Cand, hpelIfIdxavgCand is set equal to hpelIfIdxp0Cand.
* Otherwise, hpelIfIdxavgCand is set equal to 0.

#### Derivation process for zero motion vector merging candidates

Inputs to this process are:

* a merging candidate list mergeCandList,
* the reference indices refIdxL0N and refIdxL1N of every candidate N in mergeCandList,
* the prediction list utilization flags predFlagL0N and predFlagL1N of every candidate N in mergeCandList,
* the motion vectors mvL0N and mvL1N of every candidate N in mergeCandList,
* the number of elements numCurrMergeCand within mergeCandList.

Outputs of this process are:

* the merging candidate list mergeCandList,
* the number of elements numCurrMergeCand within mergeCandList,
* the reference indices refIdxL0zeroCandm and refIdxL1zeroCandm of every new candidate zeroCandm added into mergeCandList during the invocation of this process,
* the prediction list utilization flags predFlagL0zeroCandm and predFlagL1zeroCandm of every new candidate zeroCandm added into mergeCandList during the invocation of this process,
* the motion vectors mvL0zeroCandm and mvL1zeroCandm of every new candidate zeroCandm added into mergeCandList during the invocation of this process.

The variable numRefIdx is derived as follows:

* If sh\_slice\_type is equal to P, numRefIdx is set equal to NumRefIdxActive[ 0 ].
* Otherwise (sh\_slice\_type is equal to B), numRefIdx is set equal to Min( NumRefIdxActive[ 0 ], NumRefIdxActive[ 1 ] ).

When numCurrMergeCand is less than MaxNumMergeCand, the variable numInputMergeCand is set equal to numCurrMergeCand, the variable zeroIdx is set equal to 0 and the following ordered steps are repeated until numCurrMergeCand is equal to MaxNumMergeCand:

1. For the derivation of the reference indices, the prediction list utilization flags and the motion vectors of the zero motion vector merging candidate, the following applies:
   * If sh\_slice\_type is equal to P, the candidate zeroCandm with m equal to ( numCurrMergeCand − numInputMergeCand ) is added at the end of mergeCandList, i.e., mergeCandList[ numCurrMergeCand ] is set equal to zeroCandm, and the reference indices, the prediction list utilization flags and the motion vectors of zeroCandm are derived as follows and numCurrMergeCand is incremented by 1:

refIdxL0zeroCandm = ( zeroIdx < numRefIdx ) ? zeroIdx : 0 (543)

refIdxL1zeroCandm = −1 (544)

predFlagL0zeroCandm = 1 (545)

predFlagL1zeroCandm = 0 (546)

mvL0zeroCandm[ 0 ] = 0 (547)

mvL0zeroCandm[ 1 ] = 0 (548)

mvL1zeroCandm[ 0 ] = 0 (549)

mvL1zeroCandm[ 1 ] = 0 (550)

numCurrMergeCand = numCurrMergeCand + 1 (551)

* + Otherwise (sh\_slice\_type is equal to B), the candidate zeroCandm with m equal to ( numCurrMergeCand − numInputMergeCand ) is added at the end of mergeCandList, i.e., mergeCandList[ numCurrMergeCand ] is set equal to zeroCandm, and the reference indices, the prediction list utilization flags and the motion vectors of zeroCandm are derived as follows and numCurrMergeCand is incremented by 1:

refIdxL0zeroCandm = ( zeroIdx < numRefIdx ) ? zeroIdx : 0 (552)

refIdxL1zeroCandm = ( zeroIdx < numRefIdx ) ? zeroIdx : 0 (553)

predFlagL0zeroCandm = 1 (554)

predFlagL1zeroCandm = 1 (555)

mvL0zeroCandm[ 0 ] = 0 (556)

mvL0zeroCandm[ 1 ] = 0 (557)

mvL1zeroCandm[ 0 ] = 0 (558)

mvL1zeroCandm[ 1 ] = 0 (559)

numCurrMergeCand = numCurrMergeCand + 1 (560)

1. The variable zeroIdx is incremented by 1.

#### 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 to this process are:

* the modified merging candidate list mergeCandList,
* the modified number of merging candidates in the list numCurrMergeCand.

The variables isPrunedA1 and isPrunedB1 are both set equal to FALSE.

For each candidate in HmvpCandList[ 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 A1 or B1, sameMotion and isPrunedN are both 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.
   * isPrunedN is equal to FALSE.
   * 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 ] (561)

#### Derivation process for merge motion vector difference

Inputs to this process are:

* a luma location ( xCb, yCb ) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
* reference indices refIdxL0 and refIdxL1,
* prediction list utilization flags predFlagL0 and predFlagL1.

Outputs of this process are the luma merge motion vector differences in 1/16 fractional-sample accuracy mMvdL0 and mMvdL1.

The variable currPic specifies the current picture.

The luma merge motion vector differences mMvdL0 and mMvdL1 are derived as follows:

* If both predFlagL0 and predFlagL1 are equal to 1, the following applies:

currPocDiffL0  =  DiffPicOrderCnt( currPic, RefPicList[ 0 ][ refIdxL0 ] ) (562)

currPocDiffL1  =  DiffPicOrderCnt( currPic, RefPicList[ 1 ][ refIdxL1 ] ) (563)

* If currPocDiffL0 is equal to currPocDiffL1, the following applies:

mMvdL0[ 0 ]  =  MmvdOffset[ xCb ][ yCb ][ 0 ] (564)

mMvdL0[ 1 ]  =  MmvdOffset[ xCb ][ yCb ][ 1 ] (565)

mMvdL1[ 0 ]  =  MmvdOffset[ xCb ][ yCb ][ 0 ] (566)

mMvdL1[ 1 ]  =  MmvdOffset[ xCb ][ yCb ][ 1 ] (567)

* Otherwise, if Abs( currPocDiffL0 ) is greater than or equal to Abs( currPocDiffL1 ), the following applies:

mMvdL0[ 0 ]  =  MmvdOffset[ xCb ][ yCb ][ 0 ] (568)

mMvdL0[ 1 ]  =  MmvdOffset[ xCb ][ yCb ][ 1 ] (569)

* + - If RefPicList[ 0 ][ refIdxL0 ] is not a long-term reference picture and RefPicList[ 1 ][ refIdxL1 ] is not a long-term reference picture, the following applies:

td = Clip3( −128, 127, currPocDiffL0 ) (570)

tb = Clip3( −128, 127, currPocDiffL1 ) (571)

tx = ( 16384 + ( Abs( td ) >> 1 ) ) / td (572)

distScaleFactor = Clip3( −4096, 4095, ( tb \* tx + 32 ) >> 6 ) (573)

mMvdL1[ 0 ] = Clip3( −217, 217 − 1, (distScaleFactor \* mMvdL0[ 0 ] + (574)  
 128 − ( distScaleFactor \* mMvdL0[ 0 ] >= 0 ) ) >> 8 )

mMvdL1[ 1 ] = Clip3( −217, 217 − 1, (distScaleFactor \* mMvdL0[ 1 ] + (575)  
 128 − ( distScaleFactor \* mMvdL0[ 1 ] >= 0 ) ) >> 8 )

* + - Otherwise, the following applies:

mMvdL1[ 0 ] = Sign( currPocDiffL0 )  = =  Sign( currPocDiffL1 ) ?   
 mMvdL0[ 0 ] : −mMvdL0[ 0 ] (576)

mMvdL1[ 1 ] = Sign( currPocDiffL0 )  = =  Sign( currPocDiffL1) ?   
 mMvdL0[ 1 ] : −mMvdL0[ 1 ] (577)

* Otherwise (Abs( currPocDiffL0 ) is less than Abs( currPocDiffL1 )), the following applies:

mMvdL1[ 0 ]  =  MmvdOffset[ xCb ][ yCb ][ 0 ] (578)

mMvdL1[ 1 ]  =  MmvdOffset[ xCb ][ yCb ][ 1 ] (579)

* + - If RefPicList[ 0 ][ refIdxL0 ] is not a long-term reference picture and RefPicList[ 1 ][ refIdxL1 ] is not a long-term reference picture, the following applies:

td = Clip3( −128, 127, currPocDiffL1 ) (580)

tb = Clip3( −128, 127, currPocDiffL0 ) (581)

tx = ( 16384 + ( Abs( td ) >> 1 ) ) / td (582)

distScaleFactor = Clip3( −4096, 4095, ( tb \* tx + 32 ) >> 6 ) (583)

mMvdL0[ 0 ] = Clip3( −217, 217 − 1, (distScaleFactor \* mMvdL1[ 0 ] + (584)  
  128 − (distScaleFactor \* mMvdL1[ 0 ] >= 0) ) >> 8 )

mMvdL0[ 1 ] = Clip3( −217, 217 − 1, , (distScaleFactor \* mMvdL1[ 1 ] + (585)  
  128 − (distScaleFactor \* mMvdL1[ 1 ] >= 0) ) >> 8 ) )

* + - Otherwise, the following applies:

mMvdL0[ 0 ] = Sign( currPocDiffL0 )  = =  Sign( currPocDiffL1 ) ?   
 mMvdL1[ 0 ] : −mMvdL1[ 0 ] (586)

mMvdL0[ 1 ] = Sign( currPocDiffL0 )  = =  Sign( currPocDiffL1) ?   
 mMvdL1[ 1 ] : −mMvdL1[ 1 ] (587)

* Otherwise ( predFlagL0 or predFlagL1 are equal to 1 ), the following applies for X being 0 and 1:

mMvdLX[ 0 ] = ( predFlagLX  = =  1 ) ? MmvdOffset[ xCb ][ yCb ][ 0 ] : 0 (588)

mMvdLX[ 1 ] = ( predFlagLX  = =  1 ) ? MmvdOffset[ xCb ][ yCb ][ 1 ] : 0 (589)

#### 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 with X being 0 or 1 is derived in the following ordered steps:

1. The derivation process for motion vector predictor candidate list as specified in clause 8.5.2.9 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width cbWidth, the luma coding block height cbHeight and refIdxLX, with X being 0 or 1 as inputs, and the motion vector predictor candidate list, mvpListLX with X being 0 or 1, as output.
2. The motion vector predictor mvpLX with X being 0 or 1 is derived as follows:

mvpLX = mvpListLX[ mvp\_lX\_flag[ xCb ][ yCb ] ] (590)

#### Derivation process for motion vector predictor candidate list

Inputs to this process are:

* a luma location ( xCb, yCb ) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
* a variable cbWidth specifying the width of the current coding block in luma samples,
* a variable cbHeight specifying the height of the current coding block in luma samples,
* the reference index of the current coding unit partition refIdxLX, with X being 0 or 1.

Output of this process is motion vector predictor candidate list mvpListLX in 1/16 fractional-sample accuracy with X being 0 or 1.

The motion vector predictor candidate list mvpListLX with X being 0 or 1 is derived in the following ordered steps:

1. The derivation process for spatial motion vector predictor candidates from neighbouring coding unit partitions as specified in clause 8.5.2.10 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width cbWidth, the luma coding block height cbHeight and refIdxLX, with X being 0 or 1 as inputs, and the availability flags availableFlagLXN and the motion vectors mvLXN, with N being replaced by A or B, as output.
2. The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to mvLXN, with N being replaced by A or B, rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded mvLXN, with N being replaced by A or B, as output.
3. If both availableFlagLXA and availableFlagLXB are equal to 1 and mvLXA is not equal to mvLXB, availableFlagLXCol is set equal to 0.
4. Otherwise, the following applies:

* The derivation process for temporal luma motion vector prediction as specified in clause 8.5.2.11 is with the luma coding block location ( xCb, yCb ), the luma coding block width cbWidth, the luma coding block height cbHeight and refIdxLX, with X being 0 or 1 as inputs, and with the output being the availability flag availableFlagLXCol and the temporal motion vector predictor mvLXCol.
* The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to mvLXCol, rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded mvLXCol as output.

1. The motion vector predictor candidate list, mvpListLX, is constructed as follows:

numCurrMvpCand = 0  
if( availableFlagLXA ) {  
 mvpListLX[ numCurrMvpCand++ ] = mvLXA  
 if( availableFlagLXB && ( mvLXA != mvLXB ) )  
 mvpListLX[ numCurrMvpCand++ ] = mvLXB (591)  
} else if( availableFlagLXB )  
 mvpListLX[ numCurrMvpCand++ ] = mvLXB  
if( numCurrMvpCand < 2 && availableFlagLXCol )  
 mvpListLX[ numCurrMvpCand++ ] = mvLXCol

1. 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 reference picture list LY with Y = X..( 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 reference picture list LY is the same as the reference picture corresponding to reference index refIdxLX in the reference picture list LX, the following applies:
* The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to the LY motion vector of the candidate HmvpCandList[ i − 1 ], rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded LY motion vector of the candidate HmvpCandList[ i − 1 ] as output is assigned to mvpListLX[ numCurrMvpCand++ ].

1. When numCurrMvpCand is less than 2, the following applies for until numCurrMvpCand is equal to 2:

mvpListLX[ numCurrMvpCand ][ 0 ] = 0 (592)

mvpListLX[ numCurrMvpCand ][ 1 ] = 0 (593)

numCurrMvpCand++ (594)

#### Derivation process for spatial motion vector predictor candidates

Inputs to this process are:

* a luma location ( xCb, yCb ) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
* a variable cbWidth specifying the width of the current coding block in luma samples,
* a variable cbHeight specifying the height of the current coding block in luma samples,
* the reference index of the current coding unit partition refIdxLX, with X being 0 or 1.

Outputs of this process are (with N being replaced by A or B):

* the motion vectors mvLXN in 1/16 fractional-sample accuracy of the neighbouring coding units,
* the availability flags availableFlagLXN of the neighbouring coding units.

Figure 10 provides an overview of spatial motion vector neighbours.



Figure 10 – Spatial motion vector neighbours (informative)

The variable currCb specifies the current luma coding block at luma location ( xCb, yCb ) and the variable currPic specifies the current picture.

The motion vector mvLXA and the availability flag availableFlagLXA are derived in the following ordered steps:

1. The sample location ( xNbA0, yNbA0 ) is set equal to ( xCb − 1, yCb + cbHeight ) and the sample location ( xNbA1, yNbA1 ) is set equal to ( xNbA0, yNbA0 − 1 ).
2. The availability flag availableFlagLXA is set equal to 0 and both components of mvLXA are set equal to 0.
3. The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring luma location ( xNbA0, yNbA0 ), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag availableA0.
4. The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring luma location ( 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.
5. The following applies for ( xNbAk, yNbAk ) from ( xNbA0, yNbA0 ) to ( xNbA1, yNbA1 ):

* When availableAk is equal to TRUE and availableFlagLXA is equal to 0, the following applies:
* If PredFlagLX[ xNbAk ][ yNbAk ] is equal to 1 and DiffPicOrderCnt( RefPicList[ X ][ RefIdxLX[ xNbAk ][ yNbAk ] ], RefPicList[ X ][ refIdxLX ] ) is equal to 0, availableFlagLXA is set equal to 1 and the following applies:

mvLXA = MvLX[ xNbAk ][ yNbAk ] (595)

* Otherwise, when PredFlagLY[ xNbAk ][ yNbAk ] (with Y = !X) is equal to 1 and DiffPicOrderCnt( RefPicList[ Y ][ RefIdxLY[ xNbAk ][ yNbAk ] ], RefPicList[ X ][ refIdxLX ] ) is equal to 0, availableFlagLXA is set equal to 1 and the following applies:

mvLXA = MvLY[ xNbAk ][ yNbAk ] (596)

The motion vector mvLXB and the availability flag availableFlagLXB are derived in the following ordered steps:

1. The sample locations ( xNbB0, yNbB0 ), ( xNbB1, yNbB1 ) and ( xNbB2, yNbB2 ) 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 the both components of mvLXB are set equal to 0.
3. The following applies for ( xNbBk, yNbBk ) from ( xNbB0, yNbB0 ) to ( xNbB2, yNbB2 ):

* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring luma location ( xNbBk, yNbBk ), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag availableBk.
* When availableBk is equal to TRUE and availableFlagLXB is equal to 0, the following applies:
* If PredFlagLX[ xNbBk ][ yNbBk ] is equal to 1, and DiffPicOrderCnt( RefPicList[ X ][ RefIdxLX[ xNbBk ][ yNbBk ] ], RefPicList[ X ][ refIdxLX ] ) is equal to 0, availableFlagLXB is set equal to 1 and the following assignments are made:

mvLXB = MvLX[ xNbBk ][ yNbBk ] (597)

refIdxB = RefIdxLX[ xNbBk ][ yNbBk ] (598)

* Otherwise, when PredFlagLY[ xNbBk ][ yNbBk ] (with Y = !X) is equal to 1 and DiffPicOrderCnt( RefPicList[ Y ][ RefIdxLY[ xNbBk ][ yNbBk ] ], RefPicList[ X ][ refIdxLX ] ) is equal to 0, availableFlagLXB is set equal to 1 and the following assignments are made:

mvLXB = MvLY[ xNbBk ][ yNbBk ] (599)

refIdxB = RefIdxLY[ xNbBk ][ yNbBk ] (600)

#### 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 (601)

yColBr = yCb + cbHeight (602)

rightBoundaryPos = sps\_subpic\_treated\_as\_pic\_flag[ CurrSubpicIdx ] ?  
 SubpicRightBoundaryPos : pps\_pic\_width\_in\_luma\_samples − 1 (603)

botBoundaryPos = sps\_subpic\_treated\_as\_pic\_flag[ CurrSubpicIdx ] ?  
 SubpicBotBoundaryPos : pps\_pic\_height\_in\_luma\_samples − 1 (604)

* 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 variable colCb specifies the luma coding block covering the modified location given by ( ( xColBr  >>  3 )  <<  3, ( yColBr  >>  3 )  <<  3 ) inside the collocated picture specified by ColPic.
* The luma location ( xColCb, yColCb ) is set equal to the top-left sample of the collocated luma coding block specified by colCb relative to the top-left luma sample of the collocated picture specified by ColPic.
* The derivation process for collocated motion vectors as specified in clause 8.5.2.12 is invoked with currCb, colCb, ( xColCb, yColCb ), refIdxLX and sbFlag set equal to 0 as inputs, and the output is assigned to mvLXCol and availableFlagLXCol.
* Otherwise, both components of mvLXCol are set equal to 0 and availableFlagLXCol is set equal to 0.

1. When availableFlagLXCol is equal to 0, the central collocated motion vector is derived as follows:

xColCtr = xCb + ( cbWidth  >>  1 ) (605)

yColCtr = yCb + ( cbHeight  >>  1 ) (606)

* The variable colCb specifies the luma coding block covering the modified location given by ( ( xColCtr  >>  3 )  <<  3, ( yColCtr  >>  3 )  <<  3 ) inside the collocated picture specified by ColPic.
* The luma location ( xColCb, yColCb ) is set equal to the top-left sample of the collocated luma coding block specified by colCb relative to the top-left luma sample of the collocated picture specified by ColPic.
* The derivation process for collocated motion vectors as specified in clause 8.5.2.12 is invoked with currCb, colCb, ( xColCb, yColCb ), refIdxLX and sbFlag set equal to 0 as inputs, and the output is assigned to mvLXCol and availableFlagLXCol.

#### 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 coding block inside the collocated picture specified by ColPic,
* a luma location ( xColCb, yColCb ) specifying the top-left sample of the collocated luma coding block specified by colCb 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 predFlagL0Col[ 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 predFlagL1Col[ 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.

[Ed. (BB): Define ColPic NoBackwardPredFlag.]

The function LongTermRefPic( aPic, aPb, refIdx, LX ), with X being 0 or 1, is defined as follows: [Ed. (YK): The definition of the function LongTermRefPic( ) here was copied from the HEVC spec. Double check to confirm that this works as is.]

* If the picture with index refIdx from reference picture list LX of the slice containing prediction block aPb in the picture aPic was marked as "used for long-term reference" at the time when aPic was the current picture, LongTermRefPic( aPic, aPb, refIdx, LX ) is equal to 1.
* Otherwise, LongTermRefPic( aPic, aPb, 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 predFlagL0Col[ 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 predFlagL0Col[ xColCb ][ yColCb ] is equal to 1 and predFlagL1Col[ xColCb ][ yColCb ] is equal to 0, mvCol, refIdxCol and listCol are set equal to mvL0Col[ xColCb ][ yColCb ], refIdxL0Col[ xColCb ][ yColCb ] and L0, respectively.
  + Otherwise (predFlagL0Col[ xColCb ][ yColCb ] is equal to 1 and predFlagL1Col[ 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 PredFlagLXCol[ 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 (PredFlagLXCol[ xColCb ][ yColCb ] is equal to 0), the following applies:
    - * If NoBackwardPredFlag is equal to 1 and PredFlagLYCol[ 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. 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 reference picture list listCol of the slice containing coding block colCb in the collocated picture specified by ColPic, and the following applies:

colPocDiff = DiffPicOrderCnt( ColPic, refPicList[ listCol ][ refIdxCol ] ) (607)

currPocDiff = DiffPicOrderCnt( currPic, RefPicList[ X ][ refIdxLX ] ) (608)

* + - * The temporal motion buffer compression process for collocated motion vectors as specified in clause 8.5.2.15 is invoked with mvCol as input, and the modified mvCol as output.
      * If RefPicList[ X ][ refIdxLX ] is a long-term reference picture, or colPocDiff is equal to currPocDiff, mvLXCol is derived as follows:

mvLXCol = Clip3( −131072, 131071, mvCol ) (609)

* + - * Otherwise, mvLXCol is derived as a scaled version of the motion vector mvCol as follows:

tx = ( 16384 + ( Abs( td )  >>  1 ) ) / td (610)

distScaleFactor = Clip3( −4096, 4095, ( tb \* tx + 32 )  >>  6 ) (611)

mvLXCol =  Clip3( −131072, 131071, (distScaleFactor \* mvCol +  
 128 − ( distScaleFactor \* mvCol  >= 0 ) )  >>  8 ) (612)

where td and tb are derived as follows:

td = Clip3( −128, 127, colPocDiff ) (613)

tb = Clip3( −128, 127, currPocDiff ) (614)

#### 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:

mvCLX[ 0 ] = mvLX[ 0 ] \* 2 / SubWidthC (615)

mvCLX[ 1 ] = mvLX[ 1 ] \* 2 / SubHeightC (616)

#### 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:

offset = ( rightShift  = =  0 ) ? 0 : ( 1  <<  ( rightShift − 1 ) ) (617)

mvX[ 0 ] = ( ( mvX[ 0 ] + offset − ( mvX[ 0 ] >= 0 ) ) >> rightShift ) << leftShift (618)

mvX[ 1 ] = ( ( mvX[ 1 ] + offset − ( mvX[ 1 ] >= 0 ) ) >> rightShift ) << leftShift (619)

#### Temporal motion buffer compression process for collocated motion vectors

Input to this process is a motion vector mv.

Outputs of this process is the rounded motion vector mv.

For each motion vector component compIdx being 0 and 1, mv[ compIdx ] is modified as follows:

s = mv[ compIdx ] >> 17 (620)

f = Floor( Log2( ( mv[ compIdx ] ^ s ) | 31 ) ) − 4 (621)

mask = ( −1 << f ) >> 1 (622)

round = ( 1 << f ) >> 2 (623)

mv[ compIdx ] = ( mv[ compIdx ] + round ) & mask (624)

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.

#### Updating process for the history-based motion vector predictor candidate list

Inputs to this process are:

* luma motion vectors in 1/16 fractional-sample accuracy mvL0 and mvL1,
* reference indices refIdxL0 and refIdxL1,
* prediction list utilization flags predFlagL0 and predFlagL1,
* the bi-prediction weight index bcwIdx,
* the half-sample interpolation filter index hpelIfIdx.

The MVP candidate hMvpCand consists of the luma motion vectors mvL0 and mvL1, the reference indices refIdxL0 and refIdxL1, the prediction list utilization flags predFlagL0 and predFlagL1, the bi-prediction weight index bcwIdx, and the half sample interpolation filter index hpelIfIdx.

The candidate list HmvpCandList is modified using the candidate hMvpCand by the following ordered steps:

1. The variable identicalCandExist is set equal to FALSE and the variable removeIdx is set equal to 0.
2. When NumHmvpCand is greater than 0, for each index hMvpIdx with hMvpIdx = 0..NumHmvpCand − 1, the following steps apply until identicalCandExist is equal to TRUE:
   * When hMvpCand and HmvpCandList[ hMvpIdx ] have the same motion vectors and the same reference indices, identicalCandExist is set equal to TRUE and removeIdx is set equal to hMvpIdx.
3. The candidate list HmvpCandList is updated as follows:
   * If identicalCandExist is equal to TRUE or NumHmvpCand is equal to 5, the following applies:
   * For each index i with i = ( removeIdx + 1 )..( NumHmvpCand − 1 ), HmvpCandList[ i − 1] is set equal to HmvpCandList[ i ].
   * HmvpCandList[ NumHmvpCand − 1 ] is set equal to hMvpCand.
   * Otherwise (identicalCandExist is equal to FALSE and NumHmvpCand is less than 5), the following applies:
   * HmvpCandList[ NumHmvpCand++ ] is set equal to hMvpCand.

### Decoder-side motion vector refinement process

#### General

Inputs to this process are:

* a luma location ( xSb, ySb ) specifying the top-left sample of the current coding subblock relative to the top‑left luma sample of the current picture,
* a variable sbWidth specifying the width of the current coding subblock in luma samples,
* a variable sbHeight specifying the height of the current coding subblock in luma samples,
* the luma motion vectors in 1/16 fractional-sample accuracy mvL0 and mvL1,
* the selected luma reference picture sample arrays refPicL0L and refPicL1L.

Outputs of this process are:

* delta luma motion vectors dMvL0 and dMvL1,
* a variable dmvrSad specifying the mimimum sum of absolute differences.

The variable subPelFlag is set equal to 0, the variable srRange is set equal to 2 and the integer sample offset ( intOffX, intOffY ) is set equal to ( 0, 0 ).

Both components of the delta luma motion vectors dMvL0 and dMvL1 are set equal to zero and modified as follows:

* For each X being 0 or 1, the ( sbWidth + 2 \* srRange ) x ( sbHeight + 2 \* srRange ) array predSamplesLXL 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 block width set equal to ( sbWidth + 2 \* srRange ), the prediction block height set equal to ( sbHeight + 2 \* srRange ), the reference picture sample array refPicLXL, the motion vector mvLX, and the refinement search range srRange as inputs.
* The variable minSad is derived by invoking the sum of absolute differences calculation process specified in clause 8.5.3.3 with the width sbW and height sbH of the current coding subblock set equal to sbWidth and sbHeight, the prediction sample arrays pL0 and pL1 set equal to predSamplesL0L and predSamplesL1L, and the offset ( dX, dY ) set equal to ( 0, 0 ) as inputs, and minSad as output.
* The variable dmvrSad is set equal to minSad.
* When minSad is greater than or equal to sbHeight \* sbWidth, the following applies:
* The 2-D array sadArray[ dX + 2 ][ dY + 2 ] with dX = −2..2 and dY = −2..2 is derived by invoking the sum of absolute differences calculation process specified in clause 8.5.3.3 with the width sbW and height sbH of the current coding subblock set equal to sbWidth and sbHeight, the prediction sample arrays pL0 and pL1 set equal to predSamplesL0L and predSamplesL1L, and the offset ( dX, dY ) as inputs, and sadArray[ dX + 2 ][ d Y + 2 ] as output.
* The integer sample offset ( intOffX, intOffY ) is modified by invoking the array entry selection process specified in clause 8.5.3.4 with the 2-D array sadArray[ dX + 2 ][ dY + 2 ] with dX = −2..2 and dY = −2..2, the best integer sample offset ( intOffX, intOffY ), and minSad as input, the modified best integer sample offset ( intOffX, intOffY ) and modified dmvrSad as outputs.
* When intOffX is not equal to −2 or 2, and intOffY is not equal to −2 or 2, subPelFlag is set equal to 1.
* The delta luma motion vector dMvL0 is modified as follows:

dMvL0[ 0 ] += 16 \* intOffX (625)

dMvL0[ 1 ] += 16 \* intOffY (626)

* When subPelFlag is equal to 1, the parametric motion vector refinement process specified in clause 8.5.3.5 is invoked with the 3x3 2-D array sadArray[ dX + 2 ][ dY + 2 ] with dX = intOffX − 1, intOffX, intOffX + 1 and dY = intOffY − 1, intOffY, intOffY + 1, and the delta motion vector dMvL0 as inputs and the modified dMvL0 as output.
* The delta motion vector dMvL1 is derived as follows:

dMvL1[ 0 ] = −dMvL0[ 0 ] (627)

dMvL1[ 1 ] = −dMvL0[ 1 ] (628)

#### Fractional sample bilinear interpolation process

##### 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 pbWidth specifying the width of the current prediction block in luma samples,
* a variable pbHeight specifying the height of the current prediction block in luma samples,
* a luma motion vector mvLX given in 1/16-luma-sample units,
* the selected reference picture sample array refPicLXL,
* the refinement search range srRange.

Output of this process is:

* a ( pbWidth ) x ( pbHeight ) array predSamplesLXL of luma prediction sample values.

Let ( xIntL, yIntL ) be a luma location given in full-sample units and ( xFracL, yFracL ) be an offset given in 1/16-sample units. These variables are used only in this clause for specifying fractional-sample locations inside the reference sample array refPicLXL.

For each luma sample location ( xL = 0..pbWidth − 1, yL = 0..pbHeight − 1 ) inside the luma prediction sample array predSamplesLXL, the corresponding luma prediction sample value predSamplesLXL[ xL ][ yL ] is derived as follows:

* The variables xIntL, yIntL, xFracL and yFracL are derived as follows:

xIntL = xSb + ( mvLX[ 0 ]  >>  4 ) + xL − srRange (629)

yIntL = ySb + ( mvLX[ 1 ]  >>  4 ) + yL − srRange (630)

xFracL = mvLX[ 0 ] & 15 (631)

yFracL = mvLX[ 1 ] & 15 (632)

* The luma prediction sample value predSamplesLXL[ xL ][ yL ] is derived by invoking the luma sample bilinear interpolation process specified in clause 8.5.3.2.2 with ( xIntL, yIntL ), ( xFracL, yFracL ), and refPicLXL as inputs.

##### Luma sample bilinear interpolation process

Inputs to this process are:

– a luma location in full-sample units ( xIntL, yIntL ),

– a luma location in fractional-sample units ( xFracL, yFracL ),

– the luma reference sample array refPicLXL.

Output of this process is a predicted luma sample value predSampleLXL.

The variables shift1, shift2, shift3, shift4, offset1, offset2 and offset3 are derived as follows:

shift1 = BitDepth − 6 (633)

offset1 = 1 << ( shift1 − 1 ) (634)

shift2 = 4 (635)

offset2 = 1 << ( shift2 − 1 ) (636)

shift3 = 10 − BitDepth (637)

shift4 = BitDepth − 10 (638)

offset4 = 1 << ( shift4 − 1 ) (639)

The variable picW is set equal to pps\_pic\_width\_in\_luma\_samples of the reference picture refPicLX and the variable picH is set equal to pps\_pic\_height\_in\_luma\_samples of the reference picture refPicLX.

The luma interpolation filter coefficients fbL[ p ] for each 1/16 fractional sample position p equal to xFracL or  yFracL are specified in Table 26.

The luma locations in full-sample units ( xInti, yInti ) are derived as follows for i = 0..1:

– If sps\_subpic\_treated\_as\_pic\_flag[ CurrSubpicIdx ] is equal to 1 and sps\_num\_subpics\_minus1 for the reference picture refPicLX is greater than 0, the following applies:

xInti = Clip3( SubpicLeftBoundaryPos, SubpicRightBoundaryPos, pps\_ref\_wraparound\_enabled\_flag ?  
 ClipH( ( PpsRefWraparoundOffset ) \* MinCbSizeY, picW, ( xIntL + i ) ) : xIntL + i ) (640)

yInti = Clip3( SubpicTopBoundaryPos, SubpicBotBoundaryPos, yIntL + i ) (641)

– 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:

xInti = Clip3( 0, picW − 1, pps\_ref\_wraparound\_enabled\_flag ?  
 ClipH( ( PpsRefWraparoundOffset ) \* MinCbSizeY, picW, ( xIntL + i ) ) : xIntL + i ) (642)

yInti = Clip3( 0, picH − 1, yIntL + i ) (643)

The predicted luma sample value predSampleLXL is derived as follows:

– If both xFracLand yFracL are equal to 0, the value of predSampleLXL is derived as follows:

predSampleLXL = BitDepth  <=  10 ? (refPicLXL[ xInt0 ][ yInt0 ]  <<  shift3 ) :  
 ( ( refPicLXL[ xInt0 ][ yInt0 ] + offset4 )  >>  shift4 ) (644)

– Otherwise, if xFracL is not equal to 0 and yFracL is equal to 0, the value of predSampleLXL is derived as follows:

predSampleLXL = (  + offset1 )  >>  shift1 (645)

– Otherwise, if xFracL is equal to 0 and yFracL is not equal to 0, the value of predSampleLXL is derived as follows:

predSampleLXL = (  + offset1 )  >>  shift1 (646)

– Otherwise, if xFracL is not equal to 0 and yFracL is not equal to 0, the value of predSampleLXL is derived as follows:

* The sample array temp[ n ] with n = 0..1, is derived as follows:

temp[ n ] = (  + offset1 )  >>  shift1 (647)

* The predicted luma sample value predSampleLXL is derived as follows:

predSampleLXL = (  + offset2 )  >>  shift2 (648)

Table 26 – Specification of the luma bilinear interpolation filter coefficients fL[ p ] for each 1/16 fractional sample position p

|  |  |  |
| --- | --- | --- |
| **Fractional sample position p** | **interpolation filter coefficients** | |
| **fbL[ p ][ 0 ]** | **fbL[ 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 |

#### Sum of absolute differences calculation process

Inputs to this process are:

* two variables nSbW and nSbH specifying the width and the height of the current subblock,
* two ( nSbW + 4 ) x ( nSbH + 4 ) arrays pL0 and pL1 containing the predicted samples for L0 and L1 respectively,
* an integer sample offset ( dX, dY ) in L0.

Output of this process is:

* the variable sad specifying the sum of absolute differences at the integer sample at the offset ( dX, dY ) in L0.

The variable sad is derived as follows:

(649)

When both dX and dY are equal to 0, the value of sad is modified as follows:

sad = sad − ( sad  >>  2 ) (650)

#### Array entry selection process

Inputs to this process are:

* a 2-D array of sum of absolute differences values sadArray[ dX + 2 ][ d Y + 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 ] (651)  
 intOffX = dX  
 intOffY = dY  
 }  
 }  
}

The variable dmvrSad is set equal to minSad.

#### Parametric motion vector refinement process

Inputs to this process are:

* a 3x3 2-D array sadArray[ dX + 1 ][ dY + 1 ] with dX = −1..1 and dY = −1..1,
* a delta luma motion vector dMvL0.

Output of this process is the modified delta luma motion vector dMvL0.

The variable dMvX is derived by invoking the derivation process for delta motion vector component offset specified in clause 8.5.3.5.1 with the SAD values sadMinus, sadCenter and sadPlus set equal to sadArray[ 0 ][ 1 ], sadArray[ 1 ][ 1 ], and sadArray[ 2 ][ 1 ] as inputs, and dMvX set equal to the output dMVc.

The variable dMvY is derived by invoking the derivation process for delta motion vector component offset specified in clause 8.5.3.5.1 with the SAD values sadMinus, sadCenter and sadPlus set equal to sadArray[ 1 ][ 0 ], sadArray[ 1 ][ 1 ], and sadArray[ 1 ][ 2 ] as inputs, and dMvY set equal to the output dMVc.

The delta luma motion vector dMvL0 is modified as follows:

dMvL0[ 0 ] += dMvX (652)

dMvL0[ 1 ] += dMvY (653)

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 can be implemented using compares, shifts, and subtractions.

##### 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:

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 (654)  
 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 1 – The above process is equivalent to an integer division of num = ( sadMinus − sadPlus ) << 3 by denom = ( sadMinus + sadPlus − ( sadCenter << 1 ). Given the fact that sadMinus, sadCenter, and sadPlus are all positive, and sadCenter is the smallest value among the three, the value is limited to be in the range of −8 to 8, inclusive.

### Derivation process for geometric partitioning mode motion vector components and reference indices

#### General

Inputs to this process are:

* a luma location ( xCb, yCb ) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
* a variable cbWidth specifying the width of the current coding block in luma samples,
* a variable cbHeight specifying the height of the current coding block in luma samples.

Outputs of this process are:

* the luma motion vectors in 1/16 fractional-sample accuracy mvA and mvB,
* the chroma motion vectors in 1/32 fractional-sample accuracy mvCA and mvCB,
* the reference indices refIdxA and refIdxB,
* the prediction list flags predListFlagA and predListFlagB.

The derivation process for luma motion vectors for geometric partitioning merge mode as specified in clause 8.5.4.2 is invoked with the luma location ( xCb, yCb ), the variables cbWidth and cbHeight as inputs, and the output being the luma motion vectors mvA, mvB, the reference indices refIdxA, refIdxB and the prediction list flags predListFlagA and predListFlagB.

The derivation process for chroma motion vectors in clause 8.5.2.13 is invoked with mvA as input, and the output being mvCA.

The derivation process for chroma motion vectors in clause 8.5.2.13 is invoked with mvB as input, and the output being mvCB.

#### Derivation process for luma motion vectors for geometric partitioning merge mode

This process is only invoked when MergeGpmFlag[ xCb ][ yCb ] is equal to 1, where ( xCb, yCb ) specify the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture.

Inputs to this process are:

* a luma location ( xCb, yCb ) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
* a variable cbWidth specifying the width of the current coding block in luma samples,
* a variable cbHeight specifying the height of the current coding block in luma samples.

Outputs of this process are:

* the luma motion vectors in 1/16 fractional-sample accuracy mvA and mvB,
* the reference indices refIdxA and refIdxB,
* the prediction list flags predListFlagA and predListFlagB.

The motion vectors mvA and mvB, the reference indices refIdxA and refIdxB and the prediction list flags predListFlagA and predListFlagB are derived by the following ordered steps:

1. The derivation process for luma motion vectors for merge mode as specified in clause 8.5.2.2 is invoked with the luma location ( xCb, yCb ), the variables cbWidth and cbHeight inputs, and the output being the luma motion vectors mvL0[ 0 ][ 0 ], mvL1[ 0 ][ 0 ], the reference indices refIdxL0, refIdxL1, the prediction list utilization flags predFlagL0[ 0 ][ 0 ] and predFlagL1[ 0 ][ 0 ], the bi-prediction weight index bcwIdx and the merging candidate list mergeCandList.
2. The variables m and n, being the merge index for the geometric partition 0 and 1 respectively, are derived using merge\_gpm\_idx0[ xCb ][ yCb ] and merge\_gpm\_idx1[ xCb ][ yCb ] as follows:

m = merge\_gpm\_idx0[ xCb ][ yCb ] (655)

n = merge\_gpm\_idx1[ xCb ][ yCb ] + ( merge\_gpm\_idx1[ xCb ][ yCb ] >= m ) ? 1 : 0 (656)

1. 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 ] ).
2. The variable X is set equal to ( m & 0x01 ).
3. When predFlagLXM is equal to 0, X is set equal to ( 1 − X ).
4. The following applies:

mvA[ 0 ] = mvLXM[ 0 ] (657)

mvA[ 1 ] = mvLXM[ 1 ] (658)

refIdxA = refIdxLXM (659)

predListFlagA = X (660)

1. Let refIdxL0N and refIdxL1N, predFlagL0N and predFlagL1N, and mvL0N and mvL1N be the reference indices, the prediction list utilization flags and the motion vectors of the merging candidate N at position m in the merging candidate list mergeCandList ( N = mergeCandList[ n ] ).
2. The variable X is set equal to ( n & 0x01 ).
3. When predFlagLXN is equal to 0, X is set equal to ( 1 − X ).
4. The following applies:

mvB[ 0 ] = mvLXN[ 0 ] (661)

mvB[ 1 ] = mvLXN[ 1 ] (662)

refIdxB = refIdxLXN (663)

predListFlagB = X (664)

### Derivation process for subblock motion vector components and reference indices

#### General

Inputs to this process are:

* a luma location ( xCb, yCb ) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
* a variable cbWidth specifying the width of the current coding block in luma samples,
* a variable cbHeight specifying the height of the current coding block in luma samples.

Outputs of this process are:

* the reference indices refIdxL0 and refIdxL1,
* the number of luma coding subblocks in horizontal direction numSbX and in vertical direction numSbY,
* the prediction list utilization flag arrays predFlagL0[ xSbIdx ][ ySbIdx ] and predFlagL1[ xSbIdx ][ ySbIdx ] with xSbIdx = 0..numSbX − 1, ySbIdx = 0 .. numSbX − 1,
* the luma subblock motion vector arrays in 1/16 fractional-sample accuracy mvL0[ xSbIdx ][ ySbIdx ] and mvL1[ xSbIdx ][ ySbIdx ] with xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1,
* the chroma subblock motion vector arrays in 1/32 fractional-sample accuracy mvCL0[ xSbIdx ][ ySbIdx ] and mvCL1[ xSbIdx ][ ySbIdx ] with xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1,
* the bi-prediction weight index bcwIdx,
* the prediction refinement utilization flags cbProfFlagL0, cbProfFlagL1,
* the motion vector difference arrays diffMvL0[ xIdx ][ yIdx ] and diffMvL1[ xIdx ][ yIdx ] with xIdx = 0..cbWidth/numSbX − 1, yIdx = 0..cbHeight/numSbY − 1.

The variable cbProfFlagL0 and cbProfFlagL1 are initialized to be equal to zero.

For the derivation of the variables mvL0[ xSbIdx ][ ySbIdx ], mvL1[ xSbIdx ][ ySbIdx ], mvCL0[ xSbIdx ][ ySbIdx ] and mvCL1[ xSbIdx ][ ySbIdx ], refIdxL0, refIdxL1, numSbX, numSbY, predFlagL0[ xSbIdx ][ ySbIdx ] and predFlagL1[ xSbIdx ][ ySbIdx ], the following applies:

* If merge\_subblock\_flag[ xCb ][ yCb ] is equal to 1, the derivation process for motion vectors and reference indices in subblock merge mode as specified in 8.5.5.2 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width cbWidth and the luma coding block height cbHeight as inputs, the number of luma coding 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 mvL0[ 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), for X being replaced by either 0 or 1 in the variables predFlagLX, cpMvLX, MvdCpLX, and refIdxLX, in PRED\_LX, and in the syntax element ref\_idx\_lX, the following ordered steps apply:
* For the derivation of the number of control point motion vectors numCpMv, the control point motion vectors cpMvLX[ cpIdx ] with cpIdx ranging from 0 to numCpMv − 1, refIdxLX, predFlagLX[ 0 ][ 0 ], the following applies:

1. The number of control point motion vectors numCpMv is set equal to MotionModelIdc[ xCb ][ yCb ] + 1.
2. The variables refIdxLX and predFlagLX are derived as follows:

* If inter\_pred\_idc[ xCb ][ yCb ] is equal to PRED\_LX or PRED\_BI,

refIdxLX = ref\_idx\_lX[ xCb ][ yCb ] (665)

predFlagLX[ 0 ][ 0 ] = 1 (666)

* Otherwise, the variables refIdxLX and predFlagLX are specified by:

refIdxLX = −1 (667)

predFlagLX[ 0 ][ 0 ] = 0 (668)

1. The variable mvdCpLX[ 0 ] is derived as follows:

mvdCpLX[ 0 ][ 0 ] = MvdCpLX[ xCb ][ yCb ][ 0 ][ 0 ] (669)

mvdCpLX[ 0 ][ 1 ] = MvdCpLX[ xCb ][ yCb ][ 0 ][ 1 ] (670)

1. The variable mvdCpLX[ cpIdx ] with cpIdx ranging from 1 to numCpMv − 1, is derived as follows:

mvdCpLX[ cpIdx ][ 0 ] = MvdCpLX[ xCb ][ yCb ][ cpIdx ][ 0 ] + mvdCpLX[ 0 ][ 0 ] (671)

mvdCpLX[ cpIdx ][ 1 ] = MvdCpLX[ xCb ][ yCb ][ cpIdx ][ 1 ] + mvdCpLX[ 0 ][ 1 ] (672)

1. When predFlagLX[ 0 ][ 0 ] is equal to 1, the derivation process for luma affine control point motion vector predictors as specified in clause 8.5.5.7 is invoked with the luma coding block location ( xCb, yCb ), and the variables cbWidth, cbHeight, refIdxLX, and the number of control point motion vectors numCpMv as inputs, and the output being mvpCpLX[ cpIdx ] with cpIdx ranging from 0 to numCpMv − 1.
2. 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:

uLX[ cpIdx ][ 0 ] = ( mvpCpLX[ cpIdx ][ 0 ] + mvdCpLX[ cpIdx ][ 0 ] + 218 ) % 218 (673)

cpMvLX[ cpIdx ][ 0 ] = (uLX[ cpIdx ][ 0 ] >= 217 ) ? (uLX[ cpIdx ][ 0 ] − 218 ) :   
 uLX[ cpIdx ][ 0 ] (674)

uLX[ cpIdx ][ 1 ] = ( mvpCpLX[ cpIdx ][ 1 ] + mvdCpLX[ cpIdx ][ 1 ] + 218 ) % 218 (675)

cpMvLX[ cpIdx ][ 1 ] = (uLX[ cpIdx ][ 1 ] >= 217 ) ? (uLX[ cpIdx ][ 1 ] − 218 ) :   
 uLX[ cpIdx ][ 1 ] (676)

* The variables numSbX and numSbY are derived as follows:

numSbX = ( cbWidth >> 2 ) (677)

numSbY = ( cbHeight >> 2 ) (678)

* For xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1, the following applies:

predFlagLX[ xSbIdx ][ ySbIdx ] = predFlagLX[ 0 ][ 0 ] (679)

* When 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 ( xCb, yCb ), the luma coding block width cbWidth, the luma prediction block height cbHeight, the number of control point motion vectors numCpMv, the control point motion vectors cpMvLX[ cpIdx ] with cpIdx being 0..2, the reference index refIdxLX and the number of luma coding subblocks in horizontal direction numSbX and in vertical direction numSbY as inputs, the luma motion vector array mvLX[ xSbIdx ][ ySbIdx ], the chroma motion vector array mvCLX[ xSbIdx ][ ySbIdx ] with xSbIdx = 0..numSbX − 1, ySbIdx = 0 .. numSbY − 1, the prediction refinement utility flag cbProfFlagLX, and motion vector difference array diffMvLX[ xIdx ][ yIdx ] with xIdx = 0..cbWidth / numSbX − 1, yIdx = 0 .. cbHeight / numSbY − 1 as outputs.
* The bi-prediction weight index bcwIdx is set equal to bcw\_idx[ xCb ][ yCb ].

#### Derivation process for motion vectors and reference indices in subblock merge mode

Inputs to this process are:

* a luma location ( xCb, yCb ) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
* two variables cbWidth and cbHeight specifying the width and the height of the luma coding block.

Outputs of this process are:

* the number of luma coding subblocks in horizontal direction numSbX and in vertical direction numSbY,
* the reference indices refIdxL0 and refIdxL1,
* the prediction list utilization flag arrays predFlagL0[ xSbIdx ][ ySbIdx ] and predFlagL1[ xSbIdx ][ ySbIdx ] with xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1,
* the luma subblock motion vector arrays in 1/16 fractional-sample accuracy mvL0[ xSbIdx ][ ySbIdx ] and mvL1[ xSbIdx ][ ySbIdx ] with xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1,
* the chroma subblock motion vector arrays in 1/32 fractional-sample accuracy mvCL0[ xSbIdx ][ ySbIdx ] and mvCL1[ xSbIdx ][ ySbIdx ] with xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1,
* the prediction refinement utilization flags cbProfFlagL0 and cbProfFlagL1,
* the motion vector difference arrays diffMvL0[ xIdx ][ yIdx ] and diffMvL1[ xIdx ][ yIdx ] with xIdx = 0..cbWidth / numSbX − 1, yIdx = 0..cbHeight / numSbY − 1 and X being 0 or 1.
* the bi-prediction weight index bcwIdx.

The variables numSbColX, numSbColY and the subblock merging candidate list, subblockMergeCandList are derived by the following ordered steps:

1. 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 clause 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 being 0 or 1, and bcwIdxA1 is set equal to 0.
    - Otherwise, availableFlagA1 is set equal to 1 and the following assignments are made:

mvLXA1 = MvLX[ xNbA1 ][ yNbA1 ] (680)

refIdxLXA1 = RefIdxLX[ xNbA1 ][ yNbA1 ] (681)

predFlagLXA1 = PredFlagLX[ xNbA1 ][ yNbA1 ] (682)

* The derivation process for subblock-based temporal merging candidates as specified in clause 8.5.5.3 is invoked with the luma location ( xCb, yCb ), the luma coding block width cbWidth, the luma coding block height cbHeight , the availability flag 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 coding 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 xSbIdx = 0..numSbColX − 1, ySbIdx = 0 .. numSbColY − 1 and X being 0 or 1.

1. 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:

( xNbA0, yNbA0 ) = ( xCb − 1, yCb + cbHeight ) (683)

( xNbA1, yNbA1 ) = ( xCb − 1, yCb + cbHeight − 1 ) (684)

( xNbA2, yNbA2 ) = ( xCb − 1, yCb ) (685)

( xNbB0, yNbB0 ) = ( xCb + cbWidth , yCb − 1 ) (686)

( xNbB1, yNbB1 ) = ( xCb + cbWidth − 1, yCb − 1 ) (687)

( xNbB2, yNbB2 ) = ( xCb − 1, yCb − 1 ) (688)

( xNbB3, yNbB3 ) = ( xCb, yCb − 1 ) (689)

1. When sps\_affine\_enabled\_flag is equal to 1, the variable availableFlagA is set equal to FALSE and the following applies for ( xNbAk, yNbAk ) from ( xNbA0, yNbA0 ) to ( xNbA1, yNbA1 ):

* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring luma location ( xNbAk, yNbAk ), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag availableAk.
* When xCb >> Log2ParMrgLevel is equal to xNbAk >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to yNbAk >> 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 ][ xNbAk ][ yNbAk ], CbPosY[ 0 ][ xNbAk ][ yNbAk ] ), nbW is set equal to CbWidth[ 0 ][ xNbAk ][ yNbAk ], nbH is set equal to CbHeight[ 0 ][ xNbAk ][ yNbAk ], numCpMv is set equal to MotionModelIdc[ xNbAk ][ yNbAk ] + 1, and bcwIdxA is set equal to BcwIdx[ xNbAk ][ yNbAk ].
    - For X being replaced by either 0 or 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 clause 8.5.5.5 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width and height (cbWidth, cbHeight), the neighbouring luma coding block location ( xNb, yNb ), the neighbouring luma coding block width and height (nbW, nbH), and the number of control point motion vectors numCpMv as input, the control point motion vector predictor candidates cpMvLXA[ cpIdx ] with cpIdx = 0 .. numCpMv − 1 as output.
      * The following assignments are made:

predFlagLXA = PredFlagLX[ xNbAk ][ yNbAk ] (690)

refIdxLXA = RefIdxLX[ xNbAk ][ yNbAk ] (691)

1. When sps\_affine\_enabled\_flag is equal to 1, the variable availableFlagB is set equal to FALSE and the following applies for ( xNbBk, yNbBk ) from ( xNbB0, yNbB0 ) to ( xNbB2, yNbB2 ):

* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring luma location ( xNbBk, yNbBk ), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag availableBk.
* When xCb >> Log2ParMrgLevel is equal to xNbBk >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to yNbBk >> Log2ParMrgLevel, availableBk is set equal to FALSE.
* When availableBk is equal to TRUE and MotionModelIdc[ xNbBk ][ yNbBk ] 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[ xNbBk ][ yNbBk ], ( xNb, yNb ) is set equal to ( CbPosX[ 0 ][ xNbAB ][ yNbBk ], CbPosY[ 0 ][ xNbBk ][ yNbBk ] ), nbW is set equal to CbWidth[ 0 ][ xNbBk ][ yNbBk ], nbH is set equal to CbHeight[ 0 ][ xNbBk ][ yNbBk ], numCpMv is set equal to MotionModelIdc[ xNbBk ][ yNbBk ] + 1, and bcwIdxB is set equal to BcwIdx[ xNbBk ][ yNbBk ].
    - For X being replaced by either 0 or 1, the following applies:
      * When PredFlagLX[ xNbBk ][ yNbBk ] is equal to TRUE, the derivation process for luma affine control point motion vectors from a neighbouring block as specified in clause 8.5.5.5 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width and height (cbWidth, cbHeight), the neighbouring luma coding block location ( xNb, yNb ), the neighbouring luma coding block width and height (nbW, nbH), and the number of control point motion vectors numCpMv as input, the control point motion vector predictor candidates cpMvLXB[ cpIdx ] with cpIdx = 0 .. numCpMv − 1 as output.
      * The following assignments are made:

predFlagLXB = PredFlagLX[ xNbBk ][ yNbBk ] (692)

refIdxLXB = RefIdxLX[ xNbBk ][ yNbBk ] (693)

1. When sps\_affine\_enabled\_flag is equal to 1, ther following applies:

* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring luma location ( xNbA2, yNbA2 ), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag availableA2.
* When xCb >> Log2ParMrgLevel is equal to xNbA2 >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to yNbA2 >> Log2ParMrgLevel, availableA2 is set equal to FALSE.
* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring luma location ( xNbB3, yNbB3 ), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag availableB3.
* When xCb >> Log2ParMrgLevel is equal to xNbB3 >> Log2ParMrgLevel and yCb >> Log2ParMrgLevel is equal to yNbB3 >> Log2ParMrgLevel, availableB3 is set equal to FALSE.
* The derivation process for constructed affine control point motion vector merging candidates as specified in clause 8.5.5.6 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width and height (cbWidth, cbHeight), the availability flags availableA0, availableA1, availableA2, availableB0, availableB1, availableB2, availableB3 as inputs, and the availability flags availableFlagConstK, the reference indices refIdxLXConstK, prediction list utilization flags predFlagLXConstK, motion model indices motionModelIdcConstK, bi-prediction weight indices bcwIdxConstK and cpMvpLXConstK[ cpIdx ] with X being 0 or 1, K = 1..6, cpIdx = 0..2 as outputs.

1. The initial subblock merging candidate list, subblockMergeCandList, is constructed as follows:

i = 0  
if( availableFlagSbCol )  
 subblockMergeCandList[ i++ ] = SbCol  
if( availableFlagA && i < MaxNumSubblockMergeCand )  
 subblockMergeCandList[ i++ ] = A  
if( availableFlagB && i < MaxNumSubblockMergeCand )  
 subblockMergeCandList[ i++ ] = Bif( availableFlagConst1 && i < MaxNumSubblockMergeCand )  
 subblockMergeCandList[ i++ ] = Const1 (694)if( availableFlagConst2 && i < MaxNumSubblockMergeCand )  
 subblockMergeCandList[ i++ ] = Const2if( availableFlagConst3 && i < MaxNumSubblockMergeCand )  
 subblockMergeCandList[ i++ ] = Const3  
if( availableFlagConst4 && i < MaxNumSubblockMergeCand )  
 subblockMergeCandList[ i++ ] = Const4if( availableFlagConst5 && i < MaxNumSubblockMergeCand )  
 subblockMergeCandList[ i++ ] = Const5  
if( availableFlagConst6 && i < MaxNumSubblockMergeCand )  
 subblockMergeCandList[ i++ ] = Const6

1. The variable numCurrMergeCand and numOrigMergeCand are set equal to the number of merging candidates in the subblockMergeCandList.
2. 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 zeroCandm with m equal to ( numCurrMergeCand − numOrigMergeCand ) are derived as follows:

refIdxL0ZeroCandm = 0 (695)

predFlagL0ZeroCandm = 1 (696)

cpMvL0ZeroCandm[ 0 ] = mvZero (697)

cpMvL0ZeroCandm[ 1 ] = mvZero (698)

cpMvL0ZeroCandm[ 2 ] = mvZero (699)

refIdxL1ZeroCandm = ( sh\_slice\_type = = B ) ? 0 : −1 (700)

predFlagL1ZeroCandm = ( sh\_slice\_type = = B ) ? 1 : 0 (701)

cpMvL1ZeroCandm[ 0 ] = mvZero (702)

cpMvL1ZeroCandm[ 1 ] = mvZero (703)

cpMvL1ZeroCandm[ 2 ] = mvZero (704)

motionModelIdcZeroCandm = 1 (705)

bcwIdxZeroCandm = 0 (706)

* The candidate zeroCandm with m equal to ( numCurrMergeCand − numOrigMergeCand ) is added at the end of subblockMergeCandList and numCurrMergeCand is incremented by 1 as follows:

subblockMergeCandList[ numCurrMergeCand++ ] = zeroCandm (707)

The variables numSbX and numSbY are derived as follows:

* If subblockMergeCandList[ merge\_subblock\_idx[ xCb ][ yCb ] ] is equal to SbCol, numSbX is set equal to numSbColX, and numSbY is set equal to numSbColY;
* Otherwise, the following applies:

numSbX = cbWidth >> 2 (708)

numSbY = cbHeight >> 2 (709)

The variables refIdxL0, refIdxL1, predFlagL0[ xSbIdx ][ ySbIdx ], predFlagL1[ xSbIdx ][ ySbIdx ], mvL0[ xSbIdx ][ ySbIdx ], mvL1[ xSbIdx ][ ySbIdx ], mvCL0[ xSbIdx ][ ySbIdx ], and mvCL1[ xSbIdx ][ ySbIdx ] with xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1 are derived as follows:

* If subblockMergeCandList[ merge\_subblock\_idx[ xCb ][ yCb ] ] is equal to SbCol, the bi-prediction weight index bcwIdx is set equal to 0 and the following applies with X being 0 or 1:

refIdxLX = refIdxLXSbCol (710)

* + - * For xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1, the following applies:

predFlagLX[ xSbIdx ][ ySbIdx ] = predFlagLXSbCol[ xSbIdx ][ ySbIdx ] (711)

mvLX[ xSbIdx ][ ySbIdx ][ 0 ] = mvLXSbCol[ xSbIdx ][ ySbIdx ][ 0 ] (712)

mvLX[ xSbIdx ][  ySbIdx ][ 1 ] = mvLXSbCol[ xSbIdx ][ ySbIdx ][ 1 ] (713)

* + - * When predFlagLX[ xSbIdx ][ ySbIdx ], is equal to 1, the derivation process for chroma motion vectors in clause 8.5.2.13 is invoked with mvLX[ xSbIdx ][ ySbIdx ] as input, and the output being mvCLX[ xSbIdx ][ ySbIdx ].
      * The following assignment is made for x = xCb ..xCb + cbWidth − 1 and y = yCb..yCb + cbHeight − 1:

MotionModelIdc[ x ][ y ] = 0  (714)

* Otherwise (subblockMergeCandList[ merge\_subblock\_idx[ xCb ][ yCb ] ] is not equal to SbCol), the following applies with X being 0 or 1:
  + - * The following assignments are made with N being the candidate at position merge\_subblock\_idx[ xCb ][ yCb ] in the subblock merging candidate list subblockMergeCandList ( N = subblockMergeCandList[ merge\_subblock\_idx[ xCb ][ yCb ] ] ):

refIdxLX = refIdxLXN (715)

predFlagLX[ 0][ 0 ] = predFlagLXN (716)

cpMvLX[ 0 ] = cpMvLXN[ 0 ] (717)

cpMvLX[ 1 ] = cpMvLXN[ 1 ] (718)

cpMvLX[ 2 ] = cpMvLXN[ 2 ] (719)

numCpMv = motionModelIdxN + 1 (720)

bcwIdx = bcwIdxN (721)

* + - * For xSbIdx = 0..numSbX − 1, ySbIdx = 0..numSbY − 1, the following applies:

predFlagLX[ xSbIdx ][ ySbIdx ] = predFlagLX[ 0 ][ 0 ] (722)

* + - * When 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 ( xCb, yCb ), the luma coding block width cbWidth, the luma prediction block height cbHeight, the number of control point motion vectors numCpMv, the control point motion vectors cpMvLX[ cpIdx ] with cpIdx being 0..2, the reference index refIdxLX, and the number of luma coding subblocks in horizontal direction numSbX and in vertical direction numSbY as inputs, the luma subblock motion vector array mvLX[ xSbIdx ][ ySbIdx ] and the chroma subblock motion vector array mvCLX[ xSbIdx ][ ySbIdx ] with xSbIdx = 0..numSbX − 1, ySbIdx = 0 .. numSbY − 1, the prediction refinement utility flag cbProfFlagLX, and motion vector difference array diffMvLX[ xIdx ][ yIdx ] with xIdx = 0..cbWidth / numSbX − 1, yIdx = 0 .. cbHeight / numSbY − 1 as outputs.
      * The following assignment is made for x = xCb ..xCb + cbWidth − 1 and y = yCb..yCb + cbHeight − 1:

MotionModelIdc[ x ][ y ] = numCpMv − 1 (723)

#### Derivation process for subblock-based temporal merging candidates

Inputs to this process are:

* a luma location ( xCb, yCb ) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
* a variable cbWidth specifying the width of the current coding block in luma samples,
* a variable cbHeight specifying the height of the current coding block in luma samples.
* the availability flag availableFlagA1 of the neighbouring coding unit,
* the reference index refIdxLXA1 of the neighbouring coding unit with X being 0 or 1,
* the prediction list utilization flag predFlagLXA1 of the neighbouring coding unit with X being 0 or 1,
* the motion vector in 1/16 fractional-sample accuracy mvLXA1 of the neighbouring coding unit with X being 0 or 1.

Outputs of this process are:

* the availability flag availableFlagSbCol,
* the number of luma coding subblocks in horizontal direction numSbX and in vertical direction numSbY,
* the reference indices refIdxL0SbCol and refIdxL1SbCol,
* the luma motion vectors in 1/16 fractional-sample accuracy mvL0SbCol[ xSbIdx ][ ySbIdx ] and mvL1SbCol[ xSbIdx ][ ySbIdx ] with xSbIdx = 0..numSbX − 1, ySbIdx = 0 .. numSbY − 1,
* the prediction list utilization flags predFlagL0SbCol[ xSbIdx ][ ySbIdx ] and predFlagL1SbCol[ xSbIdx ][ ySbIdx ] with xSbIdx = 0..numSbX − 1, ySbIdx = 0 .. numSbY − 1.

The availability flag availableFlagSbCol is derived as follows.

* If one or more of the following conditions are true, availableFlagSbCol is set equal to 0.
* ph\_temporal\_mvp\_enabled\_flag is equal to 0.
* sps\_sbtmvp\_enabled\_flag is equal to 0.
* cbWidth is less than 8.
* cbHeight is less than 8.
* Otherwise, the following ordered steps apply:

1. The location ( xCtb, yCtb ) of the top-left sample of the luma coding tree block that contains the current coding block and the location ( xCtr, yCtr ) of the below-right center sample of the current luma coding block are derived as follows:

xCtb = ( xCb  >>  CtuLog2Size )  <<  CtuLog2Size (724)

yCtb = ( yCb  >>  CtuLog2Size )  <<  CtuLog2Size (725)

xCtrCb = xCb + ( cbWidth / 2 ) (726)

yCtrCb = yCb + ( cbHeight / 2 ) (727)

1. The derivation process for subblock-based temporal merging base motion data as specified in clause 8.5.5.4 is invoked with the location ( xCtb, yCtb ), the location ( xCtrCb, yCtrCb ), the availability flag availableFlagA1, and the prediction list utilization flag predFlagLXA1, and the reference index refIdxLXA1, and the motion vector mvLXA1, with X being 0 and 1 as inputs and the motion vectors ctrMvLX, and the prediction list utilization flags ctrPredFlagLX of the collocated block, with X being 0 and 1, and the temporal motion vector tempMv as outputs.
2. The variable availableFlagSbCol is derived as follows:

* If both ctrPredFlagL0 and ctrPredFlagL1 are equal to 0, availableFlagSbCol is set equal to 0.
* Otherwise, availableFlagSbCol is set equal to 1.

When availableFlagSbCol is equal to 1, the following applies:

* The variables numSbX, numSbY, sbWidth, sbHeight and refIdxLXSbCol are derived as follows:

numSbX  =  cbWidth >> 3 (728)

numSbY  =  cbHeight >> 3 (729)

sbWidth  =  cbWidth / numSbX (730)

sbHeight  =  cbHeight / numSbY (731)

refIdxLXSbCol  =  0 (732)

* For xSbIdx = 0..numSbX − 1 and ySbIdx = 0 .. numSbY − 1, the motion vectors mvLXSbCol[ xSbIdx ][ ySbIdx ] and prediction list utilization flags predFlagLXSbCol[ xSbIdx ][ ySbIdx ] are derived as follows:
* The luma location ( xSb, ySb ) specifying the below-right center sample of the current coding subblock relative to the top‑left luma sample of the current picture is derived as follows:

xSb  =  xCb + xSbIdx \* sbWidth + sbWidth / 2 (733)

ySb  =  yCb + ySbIdx \* sbHeight + sbHeight / 2 (734)

* The location ( xColSb, yColSb ) of the collocated subblock inside ColPic is derived as follows.
  + - The following applies:

yColSb = Clip3( yCtb,  
 Min( pps\_pic\_height\_in\_luma\_samples − 1, yCtb + ( 1  <<  CtbLog2SizeY ) − 1 ), (735)  
 ySb + tempMv[1] )

* + If sps\_subpic\_treated\_as\_pic\_flag[ CurrSubpicIdx ] is equal to 1, the following applies:

xColSb = Clip3( xCtb,   
 Min( SubpicRightBoundaryPos, xCtb + ( 1  <<  CtbLog2SizeY ) + 3 ), (736)  
 xSb + tempMv[0] )

* + Otherwise ( sps\_subpic\_treated\_as\_pic\_flag[ CurrSubpicIdx ] is equal to 0), the following applies:

xColSb = Clip3( xCtb,  
 Min( pps\_pic\_width\_in\_luma\_samples − 1, xCtb + ( 1  <<  CtbLog2SizeY ) + 3 ), (737)  
 xSb + tempMv[0] )

* The variable currCb specifies the luma coding block covering the current coding subblock inside the current picture.
* The variable colCb specifies the luma coding block covering the modified location given by ( ( xColSb >> 3 ) << 3, ( yColSb >> 3 ) << 3 ) inside the ColPic.
* The luma location ( xColCb, yColCb ) is set equal to the top-left sample of the collocated luma coding block specified by colCb relative to the top-left luma sample of the collocated picture specified by ColPic.
* The derivation process for collocated motion vectors as specified in clause 8.5.2.12 is invoked with currCb, colCb, ( xColCb, yColCb ), 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 availableFlagL0SbCol.
* The derivation process for collocated motion vectors as specified in clause 8.5.2.12 is invoked with currCb, colCb, ( xColCb, yColCb ), refIdxL1 set equal to 0 and sbFlag set equal to 1 as inputs and the output being assigned to the motion vector of the subblock mvL1SbCol[ xSbIdx ][ ySbIdx ] and availableFlagL1SbCol.
* When availableFlagL0SbCol and availableFlagL1SbCol are both equal to 0, the following applies for X being 0 and 1:

mvLXSbCol[ xSbIdx ][ ySbIdx ] = ctrMvLX (738)

predFlagLXSbCol[ xSbIdx ][ ySbIdx ] = ctrPredFlagLX (739)

#### 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 index refIdxLXA1 of the neighbouring coding unit,
* the prediction list utilization flag predFlagLXA1 of the neighbouring coding unit,
* the motion vector in 1/16 fractional-sample accuracy mvLXA1 of the neighbouring coding unit.

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 (740)

tempMv[ 1 ] = 0 (741)

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,
  + DiffPicOrderCnt(ColPic, RefPicList[ 0 ][refIdxL0A1]) is equal to 0,
* Otherwise, if all of the following conditions are true, tempMv is set equal to mvL1A1:
  + sh\_slice\_type is equal to B,
  + predFlagL1A1 is equal to 1,
  + DiffPicOrderCnt(ColPic, RefPicList[ 1 ][refIdxL1A1]) is equal to 0.

The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to tempMv, rightShift set equal to 4, and leftShift set equal to 0 as inputs and the rounded tempMv as output.

The location ( xColCb, yColCb ) of the collocated block inside ColPic is derived as follows.

* + The following applies:

yColCb = Clip3( yCtb,  
 Min( pps\_pic\_height\_in\_luma\_samples − 1, yCtb + ( 1  <<  CtbLog2SizeY ) − 1 ), (742)  
 yCtrCb + tempMv[1] )

* + If sps\_subpic\_treated\_as\_pic\_flag[ CurrSubpicIdx ] is equal to 1, the following applies:

xColCb = Clip3( xCtb,  
 Min( SubpicRightBoundaryPos, xCtb + ( 1  <<  CtbLog2SizeY ) + 3 ), (743)  
 xCtrCb + tempMv[0] )

* + Otherwise ( sps\_subpic\_treated\_as\_pic\_flag[ CurrSubpicIdx ] is equal to 0), the following applies:

xColCb = Clip3( xCtb,  
 Min( pps\_pic\_width\_in\_luma\_samples − 1, xCtb + ( 1  <<  CtbLog2SizeY ) + 3 ), (744)  
 xCtrCb + tempMv[0] )

The array colPredMode is set equal to the prediction mode array CuPredMode[ 0 ] of the collocated picture specified by ColPic.

The motion vectors ctrMvL0 and ctrMvL1, and the prediction list utilization flags ctrPredFlagL0 and ctrPredFlagL1 are derived as follows:

* If colPredMode[ ( xColCb  >>  3 )  <<  3 ][ ( yColCb  >>  3 )  <<  3 ] is equal to MODE\_INTER, the following applies:
* The variable currCb specifies the luma coding block covering ( xCtrCb ,yCtrCb ) inside the current picture.
* The variable colCb specifies the luma coding block covering the modified location given by ( ( xColCb  >>  3 )  <<  3, ( yColCb  >>  3 )  <<  3 ) inside the ColPic.
* The luma location ( xColCb, yColCb ) is set equal to the top-left sample of the collocated luma coding block specified by colCb relative to the top-left luma sample of the collocated picture specified by ColPic.
  + The derivation process for collocated motion vectors specified in clause 8.5.2.12 is invoked with currCb, colCb, (xColCb, yColCb), refIdxL0 set equal to 0, and sbFlag set equal to 1 as inputs and the output being assigned to ctrMvL0 and ctrPredFlagL0.
  + The derivation process for collocated motion vectors specified in clause 8.5.2.12 is invoked with currCb, colCb, (xColCb, yColCb), refIdxL1 set equal to 0, and sbFlag set equal to 1 as inputs and the output being assigned to ctrMvL1 and ctrPredFlagL1.
* Otherwise, the following applies:

ctrPredFlagL0 = 0 (745)

ctrPredFlagL1 = 0 (746)

#### 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.

Output of this process are the luma affine control point vectors cpMvLX[ cpIdx ] with cpIdx = 0 .. numCpMv − 1 and X being 0 or 1.

The variable isCTUboundary is derived as follows:

* If all the following conditions are true, isCTUboundary is set equal to TRUE:
* ( ( yNb + nNbH ) % CtbSizeY ) is equal to 0
* yNb + nNbH is equal to yCb
* Otherwise, isCTUboundary is set equal to FALSE.

The variables log2NbW and log2NbH are derived as follows:

log2NbW = Log2( nNbW ) (747)

log2NbH = Log2( nNbH ) (748)

The variables mvScaleHor, mvScaleVer, dHorX and dVerX are derived as follows:

* If isCTUboundary is equal to TRUE, the following applies:

mvScaleHor = MvLX[ xNb ][ yNb + nNbH − 1 ][ 0 ] << 7 (749)

mvScaleVer = MvLX[ xNb ][ yNb + nNbH − 1 ][ 1 ] << 7 (750)

dHorX = ( MvLX[ xNb + nNbW − 1 ][ yNb + nNbH − 1 ][ 0 ] − MvLX[ xNb ][ yNb + nNbH − 1 ][ 0 ] )    
 << ( 7 − log2NbW ) (751)

dVerX = ( MvLX[ xNb + nNbW − 1 ][ yNb + nNbH − 1 ][ 1 ] − MvLX[ xNb ][ yNb + nNbH − 1 ][ 1 ] )    
 << ( 7 − log2NbW ) (752)

* Otherwise (isCTUboundary is equal to FALSE), the following applies:

mvScaleHor = CpMvLX[ xNb ][ yNb ][ 0 ][ 0 ] << 7 (753)

mvScaleVer = CpMvLX[ xNb ][ yNb ][ 0 ][ 1 ] << 7 (754)

dHorX = ( CpMvLX[ xNb + nNbW − 1 ][ yNb ][ 1 ][ 0 ] − CpMvLX[ xNb ][ yNb ][ 0 ][ 0 ] )   
 << ( 7 − log2NbW ) (755)

dVerX = ( CpMvLX[ xNb + nNbW − 1 ][ yNb ][ 1 ][ 1 ] − CpMvLX[ xNb ][ yNb ][ 0 ][ 1 ] )   
 << ( 7 − log2NbW ) (756)

The variables dHorY and dVerY are derived as follows:

* If isCTUboundary is equal to FALSE and MotionModelIdc[ xNb ][ yNb ] is equal to 2, the following applies:

dHorY = ( CpMvLX[ xNb ][ yNb + nNbH − 1 ][ 2 ][ 0 ] − CpMvLX[ xNb ][ yNb ][ 0 ][ 0 ] )   
 << ( 7 − log2NbH ) (757)

dVerY = ( CpMvLX[ xNb ][ yNb + nNbH − 1 ][ 2 ][ 1 ] − CpMvLX[ xNb ][ yNb ][ 0 ][ 1 ] )   
 << ( 7 − log2NbH ) (758)

* Otherwise (isCTUboundary is equal to TRUE or MotionModelIdc[ xNb ][ yNb ] is equal to 1), the following applies,

dHorY = − dVerX (759)

dVerY = dHorX (760)

The luma affine control point motion vectors cpMvLX[ cpIdx ] with cpIdx = 0 .. numCpMv − 1 and X being 0 or 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 ) ) (761)

cpMvLX[ 0 ][ 1 ] = ( mvScaleVer + dVerX \* ( xCb − xNb ) + dVerY \* ( yCb − yNb ) ) (762)

cpMvLX[ 1 ][ 0 ] = ( mvScaleHor + dHorX \* ( xCb + cbWidth − xNb ) + dHorY \* ( yCb − yNb ) ) (763)

cpMvLX[ 1 ][ 1 ] = ( mvScaleVer + dVerX \* ( xCb + cbWidth − xNb ) + dVerY \* ( yCb − yNb ) ) (764)

* 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 ) ) (765)

cpMvLX[ 2 ][ 1 ] = ( mvScaleVer + dVerX \* ( xCb − xNb ) + dVerY \* ( yCb + cbHeight − yNb ) ) (766)

* The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to cpMvLX[ cpIdx ], rightShift set equal to 7, and leftShift set equal to 0 as inputs and the rounded cpMvLX[ cpIdx ] as output, with X being 0 or 1 and cpIdx = 0 .. numCpMv − 1.
* The motion vectors cpMvLX[ cpIdx ] with cpIdx = 0 .. numCpMv − 1 are clipped as follows:

cpMvLX[ cpIdx ][ 0 ] = Clip3( −217, 217 − 1, cpMvLX[ cpIdx ][ 0 ] ) (767)

cpMvLX[ cpIdx ][ 1 ] = Clip3( −217, 217 − 1, cpMvLX[ cpIdx ][ 1 ] ) (768)

#### 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 availableA0, availableA1, availableA2, availableB0, availableB1, availableB2, availableB3,
* the sample locations ( xNbA0, yNbA0 ), ( xNbA1, yNbA1 ), ( xNbA2, yNbA2 ), ( xNbB0, yNbB0 ), ( xNbB1, yNbB1 ), ( xNbB2, yNbB2 ) and ( xNbB3, yNbB3 ).

Output of this process are:

* the availability flag of the constructed affine control point motion vector merging candidiates availableFlagConstK, with K = 1..6,
* the reference indices refIdxLXConstK, with K = 1..6, X being 0 or 1,
* the prediction list utilization flags predFlagLXConstK, with K = 1..6, X being 0 or 1,
* the affine motion model indices motionModelIdcConstK, with K = 1..6,
* the bi-prediction weight indices bcwIdxConstK, with K = 1..6,
* the constructed affine control point motion vectors cpMvLXConstK[ cpIdx ] with cpIdx = 0..2, K = 1..6 and X being 0 or 1.

The first (top-left) control point motion vector cpMvLXCorner[ 0 ], reference index refIdxLXCorner[ 0 ], prediction list utilization flag predFlagLXCorner[ 0 ], bi-prediction weight index bcwIdxCorner[ 0 ] and the availability flag availableFlagCorner[ 0 ] with X being 0 and 1 are derived as follows:

* The availability flag availableFlagCorner[ 0 ] is set equal to FALSE.
* The following applies for ( xNbTL, yNbTL ) with TL being replaced by B2, B3, and A2:
  + - * When availableTL is equal to TRUE and availableFlagCorner[ 0 ] is equal to FALSE, the following applies with X being 0 and 1:

refIdxLXCorner[ 0 ] = RefIdxLX[ xNbTL ][ yNbTL ] (769)

predFlagLXCorner[ 0 ] = PredFlagLX[ xNbTL ][ yNbTL ] (770)

cpMvLXCorner[ 0 ] = MvLX[ xNbTL ][ yNbTL ] (771)

bcwIdxCorner[ 0 ] = BcwIdx[ xNbTL ][ yNbTL ] (772)

availableFlagCorner[ 0 ] = TRUE (773)

The second (top-right) control point motion vector cpMvLXCorner[ 1 ], reference index refIdxLXCorner[ 1 ], prediction list utilization flag predFlagLXCorner[ 1 ], bi-prediction weight index bcwIdxCorner[ 1 ] and the availability flag availableFlagCorner[ 1 ] with X being 0 and 1 are derived as follows

* The availability flag availableFlagCorner[ 1 ] is set equal to FALSE.
* The following applies for ( xNbTR, yNbTR ) with TR being replaced by B1 and B0:
  + - * When availableTR is equal to TRUE and availableFlagCorner[ 1 ] is equal to FALSE, the following applies with X being 0 and 1:

refIdxLXCorner[ 1 ] = RefIdxLX[ xNbTR ][ yNbTR ] (774)

predFlagLXCorner[ 1 ] = PredFlagLX[ xNbTR ][ yNbTR ] (775)

cpMvLXCorner[ 1 ] = MvLX[ xNbTR ][ yNbTR ] (776)

bcwIdxCorner[ 1 ] = BcwIdx[ xNbTR ][ yNbTR ] (777)

availableFlagCorner[ 1 ] = TRUE (778)

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 being 0 and 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 A1 and A0:
  + - * When availableBL is equal to TRUE and availableFlagCorner[ 2 ] is equal to FALSE, the following applies with X being 0 and 1:

refIdxLXCorner[ 2 ] = RefIdxLX[ xNbBL ][ yNbBL ] (779)

predFlagLXCorner[ 2 ] = PredFlagLX[ xNbBL ][ yNbBL ] (780)

cpMvLXCorner[ 2 ] = MvLX[ xNbBL ][ yNbBL ] (781)

availableFlagCorner[ 2 ] = TRUE (782)

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 being 0 and 1 are derived as follows:

* The reference indices for the temporal merging candidate, refIdxLXCorner[ 3 ], with X being 0 or 1, are set equal to 0.
* The variables mvLXCol and availableFlagLXCol, with X being 0 or 1, are derived as follows:
* If ph\_temporal\_mvp\_enabled\_flag is equal to 0, both components of mvLXCol are set equal to 0 and availableFlagLXCol is set equal to 0.
* Otherwise (ph\_temporal\_mvp\_enabled\_flag is equal to 1), the following applies:

xColBr = xCb + cbWidth (783)

yColBr = yCb + cbHeight (784)

rightBoundaryPos = sps\_subpic\_treated\_as\_pic\_flag[ CurrSubpicIdx ] ? SubpicRightBoundaryPos :  
 pps\_pic\_width\_in\_luma\_samples − 1 (785)

botBoundaryPos = sps\_subpic\_treated\_as\_pic\_flag[ CurrSubpicIdx ] ? SubpicBotBoundaryPos :  
 pps\_pic\_height\_in\_luma\_samples − 1 (786)

* 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 variable colCb specifies the luma coding block covering the modified location given by ( ( xColBr  >>  3 )  <<  3, ( yColBr  >>  3 )  <<  3 ) inside the collocated picture specified by ColPic.
* The luma location ( xColCb, yColCb ) is set equal to the top-left sample of the collocated luma coding block specified by colCb relative to the top-left luma sample of the collocated picture specified by ColPic.
* The derivation process for collocated motion vectors as specified in clause 8.5.2.12 is invoked with currCb, colCb, ( xColCb, yColCb ), refIdxLXCorner[ 3 ] and sbFlag set equal to 0 as inputs, and the output is assigned to mvLXCol and availableFlagLXCol.
* Otherwise, both components of mvLXCol are set equal to 0 and availableFlagLXCol is set equal to 0.
* The variables availableFlagCorner[ 3 ], predFlagL0Corner[ 3 ], cpMvL0Corner[ 3 ] and predFlagL1Corner[ 3 ] are derived as follows:

availableFlagCorner[ 3 ] = availableFlagL0Col (787)

predFlagL0Corner[ 3 ] = availableFlagL0Col (788)

cpMvL0Corner[ 3 ] = mvL0Col (789)

predFlagL1Corner[ 3 ] = 0 (790)

* 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 (791)

predFlagL1Corner[ 3 ] = availableFlagL1Col (792)

cpMvL1Corner[ 3 ] = mvL1Col (793)

When sps\_affine\_type\_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 being 0 or 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 being replaced by 0 or 1, the following applies:
  + - * The variable availableFlagLX is derived as follows:
      * If all of 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 (794)

refIdxLXConst1 = refIdxLXCorner[ 0 ] (795)

cpMvLXConst1[ 0 ] = cpMvLXCorner[ 0 ] (796)

cpMvLXConst1[ 1 ] = cpMvLXCorner[ 1 ] (797)

cpMvLXConst1[ 2 ] = cpMvLXCorner[ 2 ] (798)

* 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.

1. 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 being replaced by 0 or 1, the following applies:
  + - * The variable availableFlagLX is derived as follows:
      * If all of 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 (799)

refIdxLXConst2 = refIdxLXCorner[ 0 ] (800)

cpMvLXConst2[ 0 ] = cpMvLXCorner[ 0 ] (801)

cpMvLXConst2[ 1 ] = cpMvLXCorner[ 1 ] (802)

cpMvLXConst2[ 2 ] = cpMvLXCorner[ 3 ] + cpMvLXCorner[ 0 ] − cpMvLXCorner[ 1 ] (803)

cpMvLXConst2[ 2 ][ 0 ] = Clip3( −217, 217 − 1, cpMvLXConst2[ 2 ][ 0 ] ) (804)

cpMvLXConst2[ 2 ][ 1 ] = Clip3( −217, 217− 1, cpMvLXConst2[ 2 ][ 1 ] ) (805)

* 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.

1. 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 being replaced by 0 or 1, the following applies:
  + - * The variable availableFlagLX is derived as follows:
      * If all of 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 (806)

refIdxLXConst3 = refIdxLXCorner[ 0 ] (807)

cpMvLXConst3[ 0 ] = cpMvLXCorner[ 0 ] (808)

cpMvLXConst3[ 1 ] = cpMvLXCorner[ 3 ] + cpMvLXCorner[ 0 ] − cpMvLXCorner[ 2 ] (809)

cpMvLXConst3[ 1 ][ 0 ] = Clip3( −217, 217 − 1, cpMvLXConst3[ 1 ][ 0 ] ) (810)

cpMvLXConst3[ 1 ][ 1 ] = Clip3( −217, 217 − 1, cpMvLXConst3[ 1 ][ 1 ] ) (811)

cpMvLXConst3[ 2 ] = cpMvLXCorner[ 2 ] (812)

* 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.

1. 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 being replaced by 0 or 1, the following applies:
  + - * The variable availableFlagLX is derived as follows:
      * If all of following conditions are TRUE, availableFlagLX is set equal to TRUE:
* predFlagLXCorner[ 1 ] is equal to 1
* predFlagLXCorner[ 2 ] is equal to 1
* predFlagLXCorner[ 3 ] is equal to 1
* refIdxLXCorner[ 1 ] is equal to refIdxLXCorner[ 2 ]
* refIdxLXCorner[ 1 ] is equal to refIdxLXCorner[ 3 ]
  + - * Otherwise, availableFlagLX is set equal to FALSE.
      * When availableFlagLX is equal to TRUE, the following assignments are made:

predFlagLXConst4 = 1 (813)

refIdxLXConst4 = refIdxLXCorner[ 1 ] (814)

cpMvLXConst4[ 0 ] = cpMvLXCorner[ 1 ] + cpMvLXCorner[ 2 ] − cpMvLXCorner[ 3 ] (815)

cpMvLXConst4[ 0 ][ 0 ] = Clip3( −217, 217 − 1, cpMvLXConst4[ 0 ][ 0 ] ) (816)

cpMvLXConst4[ 0 ][ 1 ] = Clip3( −217, 217 − 1, cpMvLXConst4[ 0 ][ 1 ] ) (817)

cpMvLXConst4[ 1 ] = cpMvLXCorner[ 1 ] (818)

cpMvLXConst4[ 2 ] = cpMvLXCorner[ 2 ] (819)

* The bi-prediction weight index bcwIdxConst4 is derived as follows:
  + - * If both availableFlagL0 and availableFlagL1 are equal to 1, bcwIdxConst4 is set equal to bcwIdxCorner[ 1 ].
      * Otherwise, bcwIdxConst4 is set equal to 0.
* The variables availableFlagConst4 and motionModelIdcConst4 are derived as follows:
  + - * If availableFlagL0 or availableFlagL1 is equal to 1, availableFlagConst4 is set equal to TRUE and motionModelIdcConst4 is set equal to 2.
      * Otherwise, availableFlagConst4 is set equal to FALSE and motionModelIdcConst4 is set equal to 0.

The last two constructed affine control point motion vector merging candidates ConstK with K = 5..6 including the availability flags availableFlagConstK, the reference indices refIdxLXConstK, the prediction list utilization flags predFlagLXConstK, the affine motion model indices motionModelIdcConstK, and the constructed affine control point motion vectors cpMvLXConstK[ cpIdx ] with cpIdx = 0..2 and X being 0 or 1 are derived as follows:

1. When availableFlagCorner[ 0 ] is equal to TRUE and availableFlagCorner[ 1 ] is equal to TRUE, the following applies:

* For X being replaced by 0 or 1, the following applies:
  + - * The variable availableFlagLX is derived as follows:
      * If all of following conditions are TRUE, availableFlagLX is set equal to TRUE:
* predFlagLXCorner[ 0 ] is equal to 1
* predFlagLXCorner[ 1 ] is equal to 1
* refIdxLXCorner[ 0 ] is equal to refIdxLXCorner[ 1 ]
  + - * Otherwise, availableFlagLX is set equal to FALSE.
      * When availableFlagLX is equal to TRUE, the following assignments are made:

predFlagLXConst5 = 1 (820)

refIdxLXConst5 = refIdxLXCorner[ 0 ] (821)

cpMvLXConst5[ 0 ] = cpMvLXCorner[ 0 ] (822)

cpMvLXConst5[ 1 ] = cpMvLXCorner[ 1 ] (823)

* 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.

1. When availableFlagCorner[ 0 ] is equal to TRUE and availableFlagCorner[ 2 ] is equal to TRUE, the following applies:

* For X being replaced by 0 or 1, the following applies:
  + - * The variable availableFlagLX is derived as follows:
      * If all of 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 ] << 7 ) +    
 ( ( cpMvLXCorner[ 2 ][ 1 ] − cpMvLXCorner[ 0 ][ 1 ] ) (824)  
  << ( 7 + Log2( cbWidth ) − Log2( cbHeight ) ) )

cpMvLXCorner[ 1 ][ 1 ] = ( cpMvLXCorner[ 0 ][ 1 ] << 7 ) −    
 ( ( cpMvLXCorner[ 2 ][ 0 ] − cpMvLXCorner[ 0 ][ 0 ] ) (825)  
  << ( 7 + Log2( cbWidth ) − Log2( cbHeight ) ) )

* + - * The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to cpMvLXCorner[ 1 ], rightShift set equal to 7, and leftShift set equal to 0 as inputs and the rounded cpMvLXCorner[ 1 ] as output.
      * The following assignments are made:

predFlagLXConst6 = 1 (826)

refIdxLXConst6 = refIdxLXCorner[ 0 ] (827)

cpMvLXConst6[ 0 ] = cpMvLXCorner[ 0 ] (828)

cpMvLXConst6[ 1 ] = cpMvLXCorner[ 1 ] (829)

cpMvLXConst6[ 1 ][ 0 ] = Clip3( −217, 217 − 1, cpMvLXConst6[ 1 ][ 0 ] ) (830)

cpMvLXConst6[ 1 ][ 1 ] = Clip3( −217, 217 − 1, cpMvLXConst6[ 1 ][ 1 ] ) (831)

* 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.

#### 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.

Output of this process are the luma affine control point motion vector predictors mvpCpLX[ 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 with X being 0 or 1, 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 ( xNbA0, yNbA0 ), ( xNbA1, yNbA1 ), ( xNbA2, yNbA2 ), ( xNbB0, yNbB0 ), ( xNbB1, yNbB1 ), and ( xNbB2, yNbB2 ) are derived as follows:

( xA0, yA0 ) = ( xCb − 1, yCb + cbHeight ) (832)

( xA1, yA1 ) = ( xCb − 1, yCb + cbHeight − 1 ) (833)

( xB0, yB0 ) = ( xCb + cbWidth , yCb − 1 ) (834)

( xB1, yB1 ) = ( xCb + cbWidth − 1, yCb − 1 ) (835)

( xB2, yB2 ) = ( xCb − 1, yCb − 1 ) (836)

1. The following applies for ( xNbAk, yNbAk ) from ( xNbA0, yNbA0 ) to ( xNbA1, yNbA1 ):

* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring luma location ( xNbAk, yNbAk ), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag availableAk.
* 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 ( xNb, yNb ) is set equal to ( CbPosX[ 0 ][ xNbAk ][ yNbAk ], CbPosY[ 0 ][ xNbAk ][ yNbAk ] ), nbW is set equal to CbWidth[ 0 ][ xNbAk ][ yNbAk ],and nbH is set equal to CbHeight[ 0 ][ xNbAk ][ yNbAk ].
    - If PredFlagLX[ xNbAk ][ yNbAk ] is equal to 1 and DiffPicOrderCnt( RefPicList[ X ][ RefIdxLX[ xNbAk ][ yNbAk ] ], RefPicList[ X ][ refIdxLX ] ) is equal to 0, the following applies:
      * The variable availableFlagA is set equal to TRUE
      * The derivation process for luma affine control point motion vectors from a neighbouring block as specified in clause 8.5.5.5 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width and height (cbWidth, cbHeight), the neighbouring luma coding block location ( xNb, yNb ), the neighbouring luma coding block width and height (nbW, nbH), and the number of control point motion vectors numCpMv as input, the control point motion vector predictor candidates cpMvpLX[ cpIdx ] with cpIdx = 0 .. numCpMv − 1 as output.
      * The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to cpMvpLX[ cpIdx ], rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded cpMvpLX[ cpIdx ] with cpIdx = 0 .. numCpMv − 1 as output.
      * The following assignments are made:

cpMvpListLX[ numCpMvpCandLX ][ 0 ] = cpMvpLX[ 0 ] (837)

cpMvpListLX[ numCpMvpCandLX ][ 1 ] = cpMvpLX[ 1 ] (838)

cpMvpListLX[ numCpMvpCandLX ][ 2 ] = cpMvpLX[ 2 ] (839)

numCpMvpCandLX = numCpMvpCandLX + 1 (840)

* + - Otherwise, if PredFlagLY[ xNbAk ][ yNbAk ] (with Y = !X) is equal to 1 and DiffPicOrderCnt( RefPicList[ Y ][ RefIdxLY[ xNbAk ][ yNbAk ] ], RefPicList[ X ][ refIdxLX ] ) is equal to 0, the following applies:
      * The variable availableFlagA is set equal to TRUE
      * The derivation process for luma affine control point motion vectors from a neighbouring block as specified in clause 8.5.5.5 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width and height (cbWidth, cbHeight), the neighbouring luma coding block location ( xNb, yNb ), the neighbouring luma coding block width and height (nbW, nbH), and the number of control point motion vectors numCpMv as input, the control point motion vector predictor candidates cpMvpLY[ cpIdx ] with cpIdx = 0 .. numCpMv − 1 as output.
      * The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to cpMvpLY[ cpIdx ], rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded cpMvpLY[ cpIdx ] with cpIdx = 0 .. numCpMv − 1 as output.
      * The following assignments are made:

cpMvpListLX[ numCpMvpCandLX ][ 0 ] = cpMvpLY[ 0 ] (841)

cpMvpListLX[ numCpMvpCandLX ][ 1 ] = cpMvpLY[ 1 ] (842)

cpMvpListLX[ numCpMvpCandLX ][ 2 ] = cpMvpLY[ 2 ] (843)

numCpMvpCandLX = numCpMvpCandLX + 1 (844)

1. The following applies for ( xNbBk, yNbBk ) from ( xNbB0, yNbB0 ) to ( xNbB2, yNbB2 ):

* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring luma location ( xNbBk, yNbBk ), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag availableBk.
* When availableBk is equal to TRUE and MotionModelIdc[ xNbBk ][ yNbBk ] is greater than 0 and availableFlagB is equal to FALSE, the following applies:
  + - The variable ( xNb, yNb ) is set equal to ( CbPosX[ 0 ][ xNbBk ][ yNbBk ], CbPosY[ 0 ][ xNbBk ][ yNbBk ] ), nbW is set equal to CbWidth[ 0 ][ xNbBk ][ yNbBk ],and nbH is set equal to CbHeight[ 0 ][ xNbBk ][ yNbBk ].
    - If PredFlagLX[ xNbBk ][ yNbBk ] is equal to 1 and DiffPicOrderCnt( RefPicList[ X ][ RefIdxLX[ xNbBk ][ yNbBk ] ], RefPicList[ X ][ refIdxLX ] ) is equal to 0, the following applies:
      * The variable availableFlagB is set equal to TRUE
      * The derivation process for luma affine control point motion vectors from a neighbouring block as specified in clause 8.5.5.5 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width and height (cbWidth, cbHeight), the neighbouring luma coding block location ( xNb, yNb ), the neighbouring luma coding block width and height (nbW, nbH), and the number of control point motion vectors numCpMv as input, the control point motion vector predictor candidates cpMvpLX[ cpIdx ] with cpIdx = 0 .. numCpMv − 1 as output.
      * The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to cpMvpLX[ cpIdx ], rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded cpMvpLX[ cpIdx ] with cpIdx = 0 .. numCpMv − 1 as output.
      * The following assignments are made:

cpMvpListLX[ numCpMvpCandLX ][ 0 ] = cpMvpLX[ 0 ] (845)

cpMvpListLX[ numCpMvpCandLX ][ 1 ] = cpMvpLX[ 1 ] (846)

cpMvpListLX[ numCpMvpCandLX ][ 2 ] = cpMvpLX[ 2 ] (847)

numCpMvpCandLX = numCpMvpCandLX + 1 (848)

* + - Otherwise, if PredFlagLY[ xNbBk ][ yNbBk ] (with Y = !X) is equal to 1 and DiffPicOrderCnt( RefPicList[ Y ][ RefIdxLY[ xNbBk ][ yNbBk ] ], RefPicList[ X ][ refIdxLX ] ) is equal to 0, the following applies:
      * The variable availableFlagB is set equal to TRUE
      * The derivation process for luma affine control point motion vectors from a neighbouring block as specified in clause 8.5.5.5 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width and height (cbWidth, cbHeight), the neighbouring luma coding block location ( xNb, yNb ), the neighbouring luma coding block width and height (nbW, nbH), and the number of control point motion vectors numCpMv as input, the control point motion vector predictor candidates cpMvpLY[ cpIdx ] with cpIdx = 0 .. numCpMv − 1 as output.
      * The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to cpMvpLY[ cpIdx ], rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded cpMvpLY[ cpIdx ] with cpIdx = 0 .. numCpMv − 1 as output.
      * The following assignments are made:

cpMvpListLX[ numCpMvpCandLX ][ 0 ] = cpMvpLY[ 0 ] (849)

cpMvpListLX[ numCpMvpCandLX ][ 1 ] = cpMvpLY[ 1 ] (850)

cpMvpListLX[ numCpMvpCandLX ][ 2 ] = cpMvpLY[ 2 ] (851)

numCpMvpCandLX = numCpMvpCandLX + 1 (852)

1. When numCpMvpCandLX is less than 2, the following applies

* The derivation process for constructed affine control point motion vector prediction candidate as specified in clause 8.5.5.8 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width cbWidth, the luma coding block height cbHeight, and the reference index of the current coding unit refIdxLX as inputs, and the availability flag availableConsFlagLX, the availability flags availableFlagLX[ cpIdx ] and cpMvpLX[ cpIdx ] with cpIdx = 0..numCpMv − 1 as outputs.
* When availableConsFlagLX is equal to 1, and numCpMvpCandLX is equal to 0, the following assignments are made:

cpMvpListLX[ numCpMvpCandLX ][ 0 ] = cpMvpLX[ 0 ] (853)

cpMvpListLX[ numCpMvpCandLX ][ 1 ] = cpMvpLX[ 1 ] (854)

cpMvpListLX[ numCpMvpCandLX ][ 2 ] = cpMvpLX[ 2 ] (855)

numCpMvpCandLX = numCpMvpCandLX + 1 (856)

1. The following applies for cpIdx = 2..0:

* When numCpMvpCandLX is less than 2 and availableFlagLX[ cpIdx ] is equal to 1, the following assignments are made:

cpMvpListLX[ numCpMvpCandLX ][ 0 ] = cpMvpLX[ cpIdx ] (857)

cpMvpListLX[ numCpMvpCandLX ][ 1 ] = cpMvpLX[ cpIdx ] (858)

cpMvpListLX[ numCpMvpCandLX ][ 2 ] = cpMvpLX[ cpIdx ] (859)

numCpMvpCandLX = numCpMvpCandLX + 1 (860)

1. When numCpMvpCandLX is less than 2, the following applies:

* The derivation process for temporal luma motion vector prediction as specified in clause 8.5.2.11 is with the luma coding block location ( xCb, yCb ), the luma coding block width cbWidth, the luma coding block height cbHeight and refIdxLX as inputs, and with the output being the availability flag availableFlagLXCol and the temporal motion vector predictor mvLXCol.
* When availableFlagLXCol is equal to 1, the following applies:
* The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to mvLXCol, rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded mvLXCol as output.
* The following assignments are made:

cpMvpListLX[ numCpMvpCandLX ][ 0 ] = mvLXCol (861)

cpMvpListLX[ numCpMvpCandLX ][ 1 ] = mvLXCol (862)

cpMvpListLX[ numCpMvpCandLX ][ 2 ] = mvLXCol (863)

numCpMvpCandLX = numCpMvpCandLX + 1 (864)

1. 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:

cpMvpListLX[ numCpMvpCandLX ][ 0 ] = mvZero (865)

cpMvpListLX[ numCpMvpCandLX ][ 1 ] = mvZero (866)

cpMvpListLX[ numCpMvpCandLX ][ 2 ] = mvZero (867)

numCpMvpCandLX = numCpMvpCandLX + 1 (868)

The affine control point motion vector predictor cpMvpLX with X being 0 or 1 is derived as follows:

cpMvpLX = cpMvpListLX[ mvp\_lX\_flag[ xCb ][ yCb ] ] (869)

#### 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 prediction unit partition refIdxLX, with X being 0 or 1,

Output of this process are:

* the availability flag of the constructed affine control point motion vector prediction candidiates availableConsFlagLX with X being 0 or 1,
* the availability flags availableFlagLX[ cpIdx ] with cpIdx = 0..2 and X being 0 or 1,
* the constructed affine control point motion vector prediction candidiates cpMvLX[ cpIdx ] with cpIdx = 0..2 and X being 0 or 1.

The first (top-left) control point motion vector cpMvLX[ 0 ] and the availability flag availableFlagLX[ 0 ] are derived in the following ordered steps:

1. The sample locations ( xNbB2, yNbB2 ), ( xNbB3, yNbB3 ) and ( xNbA2, yNbA2 ) 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 ) withTL being replaced by B2, B3, and A2:

* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the luma location ( xNbY, yNbY ) set equal to ( xNbTL, yNbTL ), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the coding block availability flag availableTL.
* When availableTL is equal to TRUE and availableFlagLX[ 0 ] is equal to 0, the following applies:
* If PredFlagLX[ xNbTL ][ yNbTL ] is equal to 1, and DiffPicOrderCnt( RefPicList[ X ][ RefIdxLX[ xNbTL ][ yNbTL ] ], RefPicList[ X ][ refIdxLX ] ) is equal to 0, availableFlagLX[ 0 ] is set equal to 1 and the following assignments are made:

cpMvLX[ 0 ] = MvLX[ xNbTL ][ yNbTL ] (870)

* 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 ] (871)

* When availableFlagLX[ 0 ] is equal to 1, the rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to cpMvLX[ 0 ], rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded cpMvLX[ 0 ] as output.

The second (top-right) control point motion vector cpMvLX[ 1 ] and the availability flag availableFlagLX[ 1 ] are derived in the following ordered steps:

1. The sample locations ( xNbB1, yNbB1 ) and ( xNbB0, yNbB0 ) 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 ) withTR being replaced by B1 and B0:

* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the luma location ( xNbY, yNbY ) set equal to ( xNbTR, yNbTR ), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the coding block availability flag availableTR.
* When availableTR is equal to TRUE and availableFlagLX[ 1 ] is equal to 0, the following applies:
* If PredFlagLX[ xNbTR ][ yNbTR ] is equal to 1, and DiffPicOrderCnt( RefPicList[ X ][ RefIdxLX[ xNbTR ][ yNbTR ] ], RefPicList[ X ][ refIdxLX ] ) is equal to 0, availableFlagLX[ 1 ] is set equal to 1 and the following assignments are made:

cpMvLX[ 1 ] = MvLX[ xNbTR ][ yNbTR ] (872)

* Otherwise, when PredFlagLY[ xNbTR ][ yNbTR ] (with Y = !X) is equal to 1 and DiffPicOrderCnt( RefPicList[ Y ][ RefIdxLY[ xNbTR ][ yNbTR ] ], RefPicList[ X ][ refIdxLX ] ) is equal to 0, availableFlagLX[ 1 ] is set equal to 1 and the following assignments are made:

cpMvLX[ 1 ] = MvLY[ xNbTR ][ yNbTR ] (873)

* When availableFlagLX[ 1 ] is equal to 1, the rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to cpMvLX[ 1 ], rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded cpMvLX[ 1 ] as output.

The third (bottom-left) control point motion vector cpMvLX[ 2 ] and the availability flag availableFlagLX[ 2 ] are derived in the following ordered steps:

1. The sample locations ( xNbA1, yNbA1 ) and ( xNbA0, yNbA0 ) 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 A1 and A0:

* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the luma location ( xNbY, yNbY ) set equal to ( xNbBL, yNbBL ), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the coding block availability flag availableBL.
* When availableBL is equal to TRUE and availableFlagLX[ 2 ] is equal to 0, the following applies:
* If PredFlagLX[ xNbBL ][ yNbBL ] is equal to 1, and DiffPicOrderCnt( RefPicList[ X ][ RefIdxLX[ xNbBL ][ yNbBL ] ], RefPicList[ X ][ refIdxLX ] ) is equal to 0, availableFlagLX[ 2 ] is set equal to 1 and the following assignments are made:

cpMvLX[ 2 ] = MvLX[ xNbBL ][ yNbBL ] (874)

* Otherwise, when PredFlagLY[ xNbBL ][ yNbBL ] (with Y = !X) is equal to 1 and DiffPicOrderCnt( RefPicList[ Y ][ RefIdxLY[ xNbBL ][ yNbBL ] ], RefPicList[ X ][ refIdxLX ] ) is equal to 0, availableFlagLX[ 2 ] is set equal to 1 and the following assignments are made:

cpMvLX[ 2 ] = MvLY[ xNbBL ][ yNbBL ] (875)

* When availableFlagLX[ 2 ] is equal to 1, the rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to cpMvLX[ 2 ], rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded cpMvLX[ 2 ] as output.

The variable availableConsFlagLX is derived as follows:

* + If availableFlagLX[ 0 ] is equal to 1 and availableFlagLX[ 1 ] is equal to 1 and availableFlagLX[ 2 ] is equal to 1, availableConsFlagLX is set equal to 1
  + Otherwise, if availableFlagLX[ 0 ] is equal to 1, and availableFlagLX[ 1 ] is equal to 1, and MotionModelIdc[ xCb ][ yCb ] is equal to 1, availableConsFlagLX is set equal to 1.
  + Otherwise, availableConsFlagLX is set equal to 0.

#### 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 reference index refIdxLX and X being 0 or 1,
* the number of luma coding 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 assignments are made for x = xCb..xCb + cbWidth − 1 and y = yCb..yCb + cbHeight − 1:

CpMvLX[ x ][ y ][ 0 ] = cpMvLX[ 0 ] (876)

CpMvLX[ x ][ y ][ 1 ] = cpMvLX[ 1 ] (877)

CpMvLX[ x ][ y ][ 2 ] = cpMvLX[ 2 ] (878)

The variables log2CbW and log2CbH are derived as follows:

log2CbW = Log2( cbWidth ) (879)

log2CbH = Log2( cbHeight ) (880)

The variables mvScaleHor, mvScaleVer, dHorX and dVerX are derived as follows:

mvScaleHor = cpMvLX[ 0 ][ 0 ]  <<  7 (881)

mvScaleVer = cpMvLX[ 0 ][ 1 ]  <<  7 (882)

dHorX = ( cpMvLX[ 1 ][ 0 ] − cpMvLX[ 0 ][ 0 ] )  <<  ( 7 − log2CbW ) (883)

dVerX = ( cpMvLX[ 1 ][ 1 ] − cpMvLX[ 0 ][ 1 ] )  <<  ( 7 − log2CbW ) (884)

The variables dHorY and dVerY are derived as follows:

* If numCpMv is equal to 3, the following applies:

dHorY = ( cpMvLX[ 2 ][ 0 ] − cpMvLX[ 0 ][ 0 ] )  <<  ( 7 − log2CbH ) (885)

dVerY = ( cpMvLX[ 2 ][ 1 ] − cpMvLX[ 0 ][ 1 ] )  <<  ( 7 − log2CbH ) (886)

* Otherwise ( numCpMv is equal to 2), the following applies:

dHorY = −dVerX (887)

dVerY = dHorX (888)

The variable fallbackModeTriggered is set equal to 1 and modified as follows:

* The variables bxWX4, bxHX4, bxWXh, bxHXh, bxWXvand bxHXv are derived as follows:

maxW4 = Max( 0, Max( 4 \* ( 2048 + dHorX ),   
 Max( 4\*dHorY, 4 \* ( 2048 + dHorX ) + 4 \* dHorY ) ) ) (889)

minW4 = Min( 0, Min( 4 \* ( 2048 + dHorX ),   
 Min( 4\*dHorY, 4 \* ( 2048 + dHorX ) + 4 \* dHorY ) ) ) (890)

maxH4 = Max( 0, Max( 4 \* dVerX,   
 Max( 4\* ( 2048 + dVerY ), 4 \* dVerX + 4 \* ( 2048 + dVerY ) ) ) ) (891)

minH4 = Min( 0, Min( 4 \* dVerX,   
 Min( 4\* ( 2048 + dVerY ), 4 \* dVerX + 4 \* ( 2048 + dVerY ) ) ) ) (892)

bxWX4 = ( ( maxW4 − minW4 )  >>  11 ) + 9 (893)

bxHX4 = ( ( maxH4 − minH4 ) >> 11 ) + 9 (894)

bxWXh = ( (Max( 0, 4 \* ( 2048 + dHorX ) ) − Min( 0, 4 \* ( 2048 + dHorX ) ) )  >>  11 ) + 9 (895)

bxHXh = ( ( Max( 0, 4 \* dVerX ) − Min( 0, 4 \* dVerX ) )  >>  11 ) + 9 (896)

bxWXv = ( ( Max( 0, 4 \* dHorY ) − Min( 0, 4 \* dHorY ) )  >>  11 ) + 9 (897)

bxHXv = ( ( Max( 0, 4 \* ( 2048 + dVerY ) ) − Min( 0, 4 \* ( 2048 + dVerY ) ) )  >>  11 ) + 9 (898)

* If inter\_pred\_idc[ xCb ][ yCb ] is equal to PRED\_BI and bxWX4 \* bxHX4 is less than or equal to 225, fallbackModeTriggered is set equal to 0.
* Otherwise, if both bxWXh \* bxHXh is less than or equal to 165 and bxWXv \* bxHXv is less than or equal to 165, fallbackModeTriggered is set equal to 0.

For xSbIdx = 0..numSbX − 1 and ySbIdx = 0..numSbY − 1, the following applies:

* The variables xPosCb and yPosCb are derived as follows
* If fallbackModeTriggered is equal to 1, the following applies:

xPosCb = ( cbWidth  >>  1 ) (899)

yPosCb = ( cbHeight  >>  1 ) (900)

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

xPosCb = 2 + ( xSbIdx  <<  2 ) (901)

yPosCb = 2 + ( ySbIdx  <<  2 ) (902)

* The luma motion vector mvLX[ xSbIdx ][ ySbIdx ] is derived as follows :

mvLX[ xSbIdx ][ ySbIdx ][ 0 ] = ( mvScaleHor + dHorX \* xPosCb + dHorY \* yPosCb ) (903)

mvLX[ xSbIdx ][ ySbIdx ][ 1 ] = ( mvScaleVer + dVerX \* xPosCb + dVerY \* yPosCb ) (904)

* The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to mvLX[ xSbIdx ][ ySbIdx ], rightShift set equal to 7, and leftShift set equal to 0 as inputs and the rounded mvLX[ xSbIdx ][ ySbIdx ] as output.
* The motion vectors mvLX[ xSbIdx ][ ySbIdx ] are clipped as follows:

mvLX[ xSbIdx ][ ySbIdx ][ 0 ] = Clip3( −217, 217 − 1, mvLX[ xSbIdx ][ ySbIdx ][ 0 ] ) (905)

mvLX[ xSbIdx ][ ySbIdx ][ 1 ] = Clip3( −217, 217 − 1, mvLX[ xSbIdx ][ ySbIdx ][ 1 ] ) (906)

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 ] (907)

* + Otherwise, the following applies:

mvAvgLX = mvLX[ ( xSbIdx  >>  ( SubWidthC − 1 )  <<  ( SubWidthC − 1 ) ) ]  
 [ (ySbIdx  >>  ( SubHeightC − 1 )  <<  ( SubHeightC − 1 ) ) ] +  
 mvLX[ ( xSbIdx  >>  (SubWidthC − 1 )  <<  ( SubWidthC − 1) ) + ( SubWidthC − 1 ) ] (908)  
 [ ( ySbIdx  >>  ( SubHeightC − 1 )  <<  ( SubHeightC − 1 ) ) + ( SubHeightC − 1 ) ]

mvAvgLX[ 0 ] = ( mvAvgLX[ 0 ] + 1 − ( mvAvgLX[ 0 ] >= 0 ) ) >> 1 (909)

mvAvgLX[ 1 ] = ( mvAvgLX[ 1 ] + 1 − ( mvAvgLX[ 1 ] >= 0 ) ) >> 1 (910)

* The derivation process for chroma motion vectors in clause 8.5.2.13 is invoked with mvAvgLX as input, and the chroma motion vector mvCLX[ xSbIdx ][ ySbIdx ] as output.

[Ed. (BB): This way four 2×2 chroma subblocks (4×4 chroma block) share the same motion vector which is derived from the average of two 4×4 luma subblock motion vectors. In the decoding process motion compensation is still performed on 2×2 chroma blocks which is however a motion compensation on a chroma 4×4 block because all chroma MVs inside a 4×4 chroma block are the same. I would prefer an editorial change that makes it more clear that affine chroma MC is performed on 4×4 chroma blocks.]

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 ].
  + RprConstraintsActive[ X ][ refIdxLX ] is equal to 1.
* Otherwise, cbProfFlagLX set equal to TRUE.

When cbProfFlagLX is 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 (911)

sbHeight  =  cbHeight / numSbY (912)

dmvLimit  =  1  <<  5 (913)

posOffsetX  =  6 \* dHorX + 6 \* dHorY (914)

posOffsetY  =  6 \* dVerX + 6 \* dVerY (915)

* For x = 0..sbWidth − 1 and y = 0..sbHeight − 1, the following applies:

diffMvLX[ x ][ y ][ 0 ] = x \* ( dHorX  <<  2 ) + y \* ( dHorY  <<  2 ) − posOffsetX (916)

diffMvLX[ x ][ y ][ 1 ] = x \* ( dVerX  <<  2 ) + y \* ( dVerY  <<  2 ) − posOffsetY (917)

* + For i = 0..1, the following applies:
    - The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to diffMvLX[ x ][ y ][ i ], rightShift set equal to 8, and leftShift set equal to 0 as inputs and the rounded diffMvLX[ x ][ y ][ i ] as output.
    - The value of diffMvLX[ x ][ y ][ i ] is clipped as follows:

diffMvLX[ x ][ y ][ i ] = Clip3( −dmvLimit + 1, dmvLimit − 1, diffMvLX[ x ][ y ][ i ]) (918)

### Decoding process for inter blocks

#### General

This process is invoked when decoding a coding unit coded in inter prediction mode.

Inputs to this process are:

* a luma location ( xCb, yCb ) specifying the top-left sample of the current coding block relative to the top‑left luma sample of the current picture,
* a variable cbWidth specifying the width of the current coding block in luma samples,
* a variable cbHeight specifying the height of the current coding block in luma samples,
* variables numSbX and numSbY specifying the number of luma coding subblocks in horizontal and vertical direction,
* the motion vectors mvL0[ xSbIdx ][ ySbIdx ] and mvL1[ xSbIdx ][ ySbIdx ] with xSbIdx = 0 .. numSbX − 1, and ySbIdx = 0 .. numSbY − 1,
* the refined motion vectors refMvL0[ xSbIdx ][ ySbIdx ] and refMvL1[ xSbIdx ][ ySbIdx ] with xSbIdx = 0 .. numSbX − 1, and ySbIdx = 0 .. numSbY − 1,
* the reference indices refIdxL0 and refIdxL1,
* the prediction list utilization flags predFlagL0[ xSbIdx ][ ySbIdx ] and predFlagL1[ xSbIdx ][ ySbIdx ] with xSbIdx = 0 .. numSbX − 1, and ySbIdx = 0 .. numSbY − 1,
* the half sample interpolation filter index hpelIfIdx,
* the bi-prediction weight index bcwIdx,
* the mimimum sum of absolute difference values in decoder-side motion vector refimenent process dmvrSad[ xSbIdx ][ ySbIdx ] with xSbIdx = 0 .. numSbX − 1, and ySbIdx = 0 .. numSbY − 1,
* the decoder-side motion vector refinement flag dmvrFlag,
* a variable cIdx specifying the colour component index of the current block,
* the prediction refinement utilization flag cbProfFlagL0 and cbProfFlagL1,
* a motion vector difference array diffMvL0[ xIdx ][ yIdx ] and diffMvL1[ xIdx ][ yIdx ] with xIdx = 0..cbWidth / numSbX − 1, and yIdx = 0..cbHeight / numSbY − 1.

Outputs of this process are:

* an array predSamples of prediction samples.

Let predSamplesL0L, predSamplesL1L and predSamplesIntraL be (cbWidth)x(cbHeight) arrays of predicted luma sample values and, predSamplesL0Cb, predSamplesL1Cb, predSamplesL0Cr and predSamplesL1Cr, predSamplesIntraCb, and predSamplesIntraCr be (cbWidth / SubWidthC)x(cbHeight / SubHeightC) arrays of predicted chroma sample values.

* + The variable currPic specifies the current picture and the variable bdofFlag is derived as follows:
    - If all of the following conditions are true, bdofFlag is set equal to TRUE.
      * ph\_bdof\_disabled\_flag is equal to 0.
      * predFlagL0[ xSbIdx ][ ySbIdx ] and predFlagL1[ xSbIdx ][ ySbIdx ] are both equal to 1.
      * DiffPicOrderCnt( currPic, RefPicList[ 0 ][ refIdxL0 ] ) is equal to DiffPicOrderCnt( RefPicList[ 1 ][ refIdxL1 ], currPic).
      * RefPicList[ 0 ][ refIdxL0 ] is a short-term reference picture and RefPicList[ 1 ][ refIdxL1 ] is a short-term reference picture.
      * 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[ xCb ][ yCb ] 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.
      * RprConstraintsActive[ 0 ][ refIdxL0 ] is equal to 0 and RprConstraintsActive[ 1 ][ refIdxL1 ] is equal to 0.
      * cIdx is equal to 0.
    - Otherwise, bdofFlag is set equal to FALSE.
* If numSbY is equal to 1 and numSbX is equal to 1 the following applies:
  + When bdofFlag is equal to TRUE, the variables numSbY, numSbX are modified as follows:

numSbX = ( cbWidth > 16 ) ? ( cbWidth  >>  4 ) : 1 (919)

numSbY = ( cbHeight > 16 ) ? ( cbHeight  >>  4 ) : 1 (920)

* + For X = 0..1, xSbIdx = 0..numSbX − 1 and ySbIdx = 0..numSbY − 1, the following applies:
    - predFlagLX[ xSbIdx ][ ySbIdx ] is set equal to predFlagLX[ 0 ][ 0 ].
    - refMvLX[ xSbIdx ][ ySbIdx ] is set equal to refMvLX[ 0 ][ 0 ].
    - mvLX[ xSbIdx ][ ySbIdx ] is set equal to mvLX[ 0 ][ 0 ].

The width and the height of the current coding sublock sbWidth, sbHeight in luma samples are derived as follows:

sbWidth  =  cbWidth / numSbX (921)

sbHeight  =  cbHeight / numSbY (922)

For each coding 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 coding subblock relative to the top‑left luma sample of the current picture is derived as follows:

( xSb, ySb )  =  ( xCb + xSbIdx \* sbWidth, yCb + ySbIdx \* sbHeight ) (923)

* For X being each of 0 and 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 clause 8.5.6.2 with X and refIdxLX as inputs.
  + The motion vector offset mvOffset is set equal to refMvLX[ xSbIdx ][ xSbIdx ] − mvLX[ xSbIdx ][ ySbIdx ].
  + If cIdx is equal to 0, the following applies:
    - The array predSamplesLXL is derived by invoking the fractional sample interpolation process specified in clause 8.5.6.3 with the luma location ( xSb, ySb ), the coding subblock width sbWidth, the coding subblock height sbHeight in luma samples, the luma motion vector offset mvOffset, the refined luma motion vector refMvLX[ xSbIdx ][ xSbIdx ], the reference array refPicLXL, bdofFlag, dmvrFlag, hpelIfIdx, cIdx, RprConstraintsActive[ X ][ refIdxLX ], and RefPicScale[ X ][ refIdxLX ] as inputs.
    - When cbProfFlagLX is equal to 1, the prediction refinement with optical flow process specified in clause 8.5.6.4 is invoked with sbWidth, sbHeight, the (sbWidth + 2)x(sbHeight + 2) array 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)x(sbHeight) array predSamplesLXL as output.
  + Otherwise, if cIdx is equal to 1, the following applies:
    - The array predSamplesLXCb is derived by invoking the fractional sample interpolation process specified in clause 8.5.6.3 with the luma location ( xSb, ySb ), the coding subblock width sbWidth / SubWidthC, the coding subblock height sbHeight / SubHeightC, the chroma motion vector offset mvOffset, the refined chroma motion vector refMvLX[ xSbIdx ][ ySbIdx ], the reference array refPicLXCb, bdofFlag, dmvrFlag, hpelIfIdx, cIdx, RprConstraintsActive[ 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 clause 8.5.6.3 with the luma location ( xSb, ySb ), the coding subblock width sbWidth / SubWidthC, the coding subblock height sbHeight / SubHeightC, the chroma motion vector offset mvOffset, the refined chroma motion vector refMvLX[ xSbIdx ][ xSbIdx ], the reference array refPicLXCr, bdofFlag, dmvrFlag, hpelIfIdx, cIdx, RprConstraintsActive[ X ][ refIdxLX ], and RefPicScale[ X ][ refIdxLX ] as inputs.
* The variable sbBdofFlag is set equal to FALSE.
* When bdofFlag is equal to TRUE, the variable sbBdofFlag is further modifed as follows:
* If dmvrFlag is equal to 1 and the variable dmvrSad[ xSbIdx ][ ySbIdx ] is less than ( 2 \* sbWidth \* sbHeight ), the variable sbBdofFlag is set equal to FALSE.
* Otherwise, the variable sbBdofFlag is set equal to TRUE.
* The array predSamples of prediction samples is derived as follows:
* If cIdx is equal to 0, the prediction samples inside the current luma coding subblock, predSamples[ xL + xSb ][ yL + ySb ] with xL = 0..sbWidth − 1 and yL = 0..sbHeight − 1, are derived as follows:
* If sbBdofFlag is equal to TRUE, the bi-directional optical flow sample prediction process as specified in clause 8.5.6.5 is invoked with nCbW set equal to the luma coding subblock width sbWidth, nCbH set equal to the luma coding subblock height sbHeight and the sample arrays predSamplesL0L and predSamplesL1L, and the variables predFlagL0[ xSbIdx ][ ySbIdx ], predFlagL1[ xSbIdx ][ ySbIdx ], refIdxL0, and refIdxL1 as inputs, and predSamples[ xL + xSb ][ yL + ySb ] as outputs.
* Otherwise (sbBdofFlag is equal to FALSE), the weighted sample prediction process as specified in clause 8.5.6.6 is invoked with the luma coding subblock width sbWidth, the luma coding subblock height sbHeight and the sample arrays predSamplesL0L and predSamplesL1L, and the variables predFlagL0[ xSbIdx ][ ySbIdx ], predFlagL1[ xSbIdx ][ ySbIdx ], refIdxL0, refIdxL1, bcwIdx, dmvrFlag, and cIdx as inputs, and predSamples[ xL + xSb ][ yL + ySb ] as outputs.
* Otherwise, if cIdx is equal to 1, the prediction samples inside the current chroma component Cb coding subblock, predSamples[ xC + xSb / SubWidthC ][ yC + ySb / SubHeightC ] with xC = 0..sbWidth / SubWidthC − 1 and yC = 0..sbHeight / SubHeightC − 1, are derived by invoking the weighted sample prediction process specified in clause 8.5.6.6 with nCbW set equal to sbWidth / SubWidthC, nCbH set equal to sbHeight / SubHeightC, the sample arrays predSamplesL0Cb and predSamplesL1Cb, 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 coding subblock, predSamples[ xC + xSb / SubWidthC ][ yC + ySb / SubHeightC ] with xC = 0..sbWidth / SubWidthC − 1 and yC = 0..sbHeight / SubHeightC − 1, are derived by invoking the weighted sample prediction process specified in clause 8.5.6.6 with nCbW set equal to sbWidth / SubWidthC, nCbH set equal to sbHeight / SubHeightC, the sample arrays predSamplesL0Cr and predSamplesL1Cr, 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 ] (924)

MvL1[ xSb + x ][ ySb + y ] = mvL1[ xSbIdx ][ ySbIdx ] (925)

MvDmvrL0[ xSb + x ][ ySb + y ] = refMvL0[ xSbIdx ][ ySbIdx ] (926)

MvDmvrL1[ xSb + x ][ ySb + y ] = refMvL1[ xSbIdx ][ ySbIdx ] (927)

RefIdxL0[ xSb + x ][ ySb + y ] = refIdxL0 (928)

RefIdxL1[ xSb + x ][ ySb + y ] = refIdxL1 (929)

PredFlagL0[ xSb + x ][ ySb + y ] = predFlagL0[ xSbIdx ][ ySbIdx ] (930)

PredFlagL1[ xSb + x ][ ySb + y ] = predFlagL1[ xSbIdx ][ ySbIdx ] (931)

HpelIfIdx[ xSb + x ][ ySb + y ] = hpelIfIdx (932)

BcwIdx[ xSb + x ][ ySb + y ] = bcwIdx (933)

When ciip\_flag[ xCb ][ yCb ] is equal to 1, the array predSamples of prediction samples is modified as follows:

* If cIdx is equal to 0, the following applies:
* The general intra sample prediction process as specified in clause 8.4.5.2.5 is invoked with the location ( xTbCmp, yTbCmp ) set equal to ( xCb, yCb ), the intra prediction mode predModeIntra set equal to INTRA\_PLANAR, the transform block width nTbW and height nTbH set equal to cbWidth and cbHeight, the coding block width nCbW and height nCbH set equal to cbWidth and cbHeight, and the variable cIdx as inputs, and the output is assigned to the (cbWidth)x(cbHeight) array predSamplesIntraL.
* The weighted sample prediction process for combined merge and intra prediction as specified in clause 8.5.6.7 is invoked with the location ( xTbCmp, yTbCmp ) set equal to ( xCb, yCb ), the coding block width cbWidth, the coding block height cbHeight, the sample arrays predSamplesInter and predSamplesIntra set equal to predSamples and predSamplesIntraL, respectively, and the colour component index cIdx as inputs, and the output is assigend to the (cbWidth)x(cbHeight) array predSamples.
* Otherwise, if cIdx is equal to 1 and cbWidth / SubWidthC is greater than or equal to 4, the following applies:
* The general intra sample prediction process as specified in clause 8.4.5.2.5 is invoked with the location ( xTbCmp, yTbCmp ) set equal to ( xCb / SubWidthC , yCb / SubHeightC ), the intra prediction mode predModeIntra set equal to INTRA\_PLANAR, the transform block width nTbW and height nTbH set equal to cbWidth / SubWidthC  and cbHeight / SubHeightC, the coding block width nCbW and height nCbH set equal to cbWidth / SubWidthC  and cbHeight / SubHeightC, and the variable cIdx as inputs, and the output is assigned to the (cbWidth / SubWidthC )x(cbHeight / SubHeightC) array predSamplesIntraCb.
* The weighted sample prediction process for combined merge and intra prediction as specified in clause 8.5.6.7 is invoked with the location ( xTbCmp, yTbCmp ) set equal to ( xCb, yCb ), the coding block width cbWidth / SubWidthC , the coding block height cbHeight / SubHeightC, the sample arrays predSamplesInter and predSamplesIntra set equal to predSamplesCb and predSamplesIntraCb, respectively, and the colour component index cIdx as inputs, and the output is assigend to the (cbWidth / SubWidthC )x(cbHeight / SubHeightC) array predSamples.
* Otherwise, if cIdx is equal to 2 and cbWidth / SubWidthC is greater than or equal to 4, the following applies:
* The general intra sample prediction process as specified in clause 8.4.5.2.5 is invoked with the location ( xTbCmp, yTbCmp ) set equal to ( xCb / SubWidthC , yCb / SubHeightC ), the intra prediction mode predModeIntra set equal to INTRA\_PLANAR, the transform block width nTbW and height nTbH set equal to cbWidth / SubWidthC  and cbHeight / SubHeightC, the coding block width nCbW and height nCbH set equal to cbWidth / SubWidthC  and cbHeight / SubHeightC, and the variable cIdx as inputs, and the output is assigned to the (cbWidth / SubWidthC )x(cbHeight / SubHeightC) array predSamplesIntraCr.
* The weighted sample prediction process for combined merge and intra prediction as specified in clause 8.5.6.7 is invoked with the location ( xTbCmp, yTbCmp ) set equal to ( xCb, yCb ), the coding block width cbWidth / SubWidthC , the coding block height cbHeight / SubHeightC, the sample arrays predSamplesInter and predSamplesIntra set equal to predSamplesCr and predSamplesIntraCr, respectively, and the colour component index cIdx as inputs, and the output is assigend to the (cbWidth / SubWidthC )x(cbHeight / SubHeightC) array predSamples.

#### Reference picture selection process

Inputs to this process are:

* a value X representing a reference list being equal to either 0 or 1,
* a reference index refIdxLX.

Output of this process is a reference picture consisting of a two-dimensional array of luma samples refPicLXL and two two-dimensional arrays of chroma samples refPicLXCb and refPicLXCr.

The output reference picture RefPicList[ X ][ refIdxLX ], where X is the value of X that this process is invoked for, consists of a pps\_pic\_width\_in\_luma\_samples by pps\_pic\_height\_in\_luma\_samples array of luma samples refPicLXL and two PicWidthInSamplesC by PicHeightInSamplesC arrays of chroma samples refPicLXCb and refPicLXCr.

The reference picture sample arrays refPicLXL, refPicLXCb and refPicLXCr correspond to decoded sample arrays SL, SCb and SCr derived in clause 8.8 for a previously-decoded picture.

#### Fractional sample interpolation process

##### General

Inputs to this process are:

* a luma location ( xSb, ySb ) specifying the top-left sample of the current coding subblock relative to the top‑left luma sample of the current picture,
* a variable sbWidth specifying the width of the current coding subblock,
* a variable sbHeight specifying the height of the current coding subblock,
* a motion vector offset mvOffset,
* a refined motion vector refMvLX,
* the selected reference picture sample array refPicLX,
* the bi-directional optical flow flag bdofFlag,
* the decoder-side motion vector refinement flag dmvrFlag,
* the half sample interpolation filter index hpelIfIdx,
* a variable cIdx specifying the colour component index of the current block,
* a variable refPicIsScaled indicating whether the selected reference picture requires scaling,
* a list of two scaling ratios, horizontal and vertical, scalingRatio.

Outputs of this process are:

* an (sbWidth + brdExtSize)x(sbHeight + brdExtSize) array predSamplesLX of prediction sample values.

The prediction block border extension size brdExtSize is derived as follows:

brdExtSize = ( bdofFlag | | ( inter\_affine\_flag[ xSb ][ ySb ] && !ph\_prof\_disabled\_flag ) ) ? 2 : 0 (934)

The variable refWraparoundEnabledFlag is set equal to ( pps\_ref\_wraparound\_enabled\_flag && !refPicIsScaled ).

The variable fRefLeftOffset is set equal to ( ( SubWidthC \* pps\_scaling\_win\_left\_offset )  <<  10 ), where pps\_scaling\_win\_left\_offset is the pps\_scaling\_win\_left\_offset for the reference picture.

The variable fRefTopOffset is set equal to ( ( SubHeightC \* pps\_scaling\_win\_top\_offset )  <<  10 ), where pps\_scaling\_win\_top\_offset is the pps\_scaling\_win\_top\_offset for the reference picture.

The (sbWidth + brdExtSize)x(sbHeight + brdExtSize) array predSamplesLX of prediction sample values is derived as follows:

* The motion vector mvLX is set equal to ( refMvLX − mvOffset ).
* If cIdx is equal to 0, the following applies:
  + Let ( xIntL, yIntL ) be a luma location given in full-sample units and ( xFracL, yFracL ) be an offset given in 1/16-sample units. These variables are used only in this clause for specifying fractional-sample locations inside the reference sample arrays refPicLX.
  + The top-left coordinate of the bounding block for reference sample padding ( xSbIntL, ySbIntL ) is set equal to ( xSb + ( mvLX[ 0 ]  >>  4 ), ySb + ( mvLX[ 1 ]  >>  4 ) ).
  + For each luma sample location ( xL = 0..sbWidth − 1 + brdExtSize, yL = 0..sbHeight − 1 + brdExtSize ) inside the prediction luma sample array predSamplesLX, the corresponding prediction luma sample value predSamplesLX[ xL ][ yL ] is derived as follows:
* Let ( refxSbL, refySbL ) and ( refxL, refyL ) be luma locations pointed to by a motion vector ( refMvLX[0], refMvLX[1] ) given in 1/16-sample units. The variables refxSbL, refxL, refySbL, and refyL are derived as follows:

refxSbL = ( ( ( xSb − ( SubWidthC \* pps\_scaling\_win\_left\_offset ) )  <<  4 ) +  
 refMvLX[ 0 ] ) \* scalingRatio[ 0 ] (935)

refxL = ( ( Sign( refxSbL ) \* ( ( Abs( refxSbL ) + 128 ) >> 8 ) +  
 xL \* ( ( scalingRatio[ 0 ] + 8 ) >> 4 ) ) + fRefLeftOffset + 32 ) >> 6 (936)

refySbL = ( ( ( ySb − ( SubHeightC \* pps\_scaling\_win\_top\_offset ) ) << 4 ) +  
 refMvLX[ 1 ] ) \* scalingRatio[ 1 ] (937)

refyL = ( ( Sign( refySbL ) \* ( ( Abs( refySbL ) + 128 ) >> 8 ) + yL \*  
 ( ( scalingRatio[ 1 ] + 8 ) >> 4 ) ) + fRefTopOffset + 32 ) >> 6 (938)

* The variables xIntL, yIntL, xFracL and yFracL are derived as follows:

xIntL = refxL  >>  4 (939)

yIntL = refyL  >>  4 (940)

xFracL = refxL & 15 (941)

yFracL = refyL & 15 (942)

* The prediction luma sample value predSamplesLX[ xL ][ yL ] is derived as follows:
* If bdofFlag is equal to TRUE or ( ph\_prof\_disabled\_flag is equal to FALSE and inter\_affine\_flag[ xSb ][ ySb ] is equal to TRUE ), and one or more of the following conditions are true, the prediction luma sample value predSamplesLX[ xL ][ yL ] is derived by invoking the luma integer sample fetching process as specified in clause 8.5.6.3.3 with ( xIntL + ( xFracL >> 3) − 1), yIntL + ( yFracL >> 3 ) − 1 ), refPicLX, and refWraparoundEnabledFlag as inputs.
  + - * xL is equal to 0.
      * xL is equal to sbWidth + 1.
      * yL is equal to 0.
      * yL is equal to sbHeight + 1.
* Otherwise, the prediction luma sample value predSamplesLX[ xL ][ yL ] is derived by invoking the luma sample 8-tap interpolation filtering process as specified in clause 8.5.6.3.2 with ( xIntL − ( brdExtSize > 0 ? 1 : 0 ), yIntL − ( brdExtSize > 0 ? 1 : 0 ) ), ( xFracL, yFracL ), ( xSbIntL, ySbIntL ), refPicLX, hpelIfIdx, sbWidth, sbHeight, dmvrFlag, refWraparoundEnabledFlag, scalingRatio[ 0 ], scalingRatio[ 1 ], and ( xSb, ySb ) as inputs.
* Otherwise (cIdx is not equal to 0), the following applies:
  + Let ( xIntC, yIntC ) be a chroma location given in full-sample units and ( xFracC, yFracC ) be an offset given in 1/32 sample units. These variables are used only in this clause for specifying general fractional-sample locations inside the reference sample arrays refPicLX.
  + The top-left coordinate of the bounding block for reference sample padding ( xSbIntC, ySbIntC ) 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 (refxSbC, refySbC) and ( refxC, refyC ) be chroma locations pointed to by a motion vector (refMvLX[ 0 ], refMvLX[ 1 ]) given in 1/32-sample units. The variables refxSbC, refySbC, refxC and refyC are derived as follows:

addX = sps\_chroma\_horizontal\_collocated\_flag ? 0 : 8 \* ( scalingRatio[ 0 ] − ( 1 << 14 ) ) (943)

addY = sps\_chroma\_vertical\_collocated\_flag ? 0 : 8 \* ( scalingRatio[ 1 ] − ( 1 << 14 ) ) (944)

refxSbC = ( ( ( xSb − ( SubWidthC \* pps\_scaling\_win\_left\_offset ) ) / SubWidthC << 5 ) +  
 refMvLX[ 0 ] ) \* scalingRatio[ 0 ] + addX (945)

refxC = ( ( Sign( refxSbC ) \* ( ( Abs( refxSbC ) + 256 ) >> 9 )  
 + xC \* ( ( scalingRatio[ 0 ] + 8 ) >> 4 ) )  + fRefLeftOffset / SubWidthC + 16 ) >> 5 (946)

refySbC =  ( ( ( ySb − ( SubHeightC \* pps\_scaling\_win\_top\_offset ) ) / SubHeightC  <<  5 ) +  
 refMvLX[ 1 ] ) \* scalingRatio[ 1 ] + addY (947)

refyC = ( ( Sign( refySbC ) \* ( ( Abs( refySbC ) + 256 ) >> 9 )  
 + yC\* ( ( scalingRatio[ 1 ] + 8 ) >> 4 ) ) + fRefTopOffset / SubHeightC + 16 ) >> 5 (948)

* The variables xIntC, yIntC, xFracC and yFracC are derived as follows:

xIntC = refxC  >>  5 (949)

yIntC = refyC  >>  5 (950)

xFracC = refxC & 31 (951)

yFracC = refyC & 31 (952)

* The prediction sample value predSamplesLX[ xC ][ yC ] is derived by invoking the process specified in clause 8.5.6.3.4 with ( xIntC, yIntC ), ( xFracC, yFracC ), ( xSbIntC, ySbIntC ), sbWidth, sbHeight, refPicLX, dmvrFlag, refWraparoundEnabledFlag, scalingRatio[ 0 ], and scalingRatio[ 1 ] as inputs.

NOTE – Unlike the process specified in clause 8.4.5.2.13, this process uses both sps\_chroma\_vertical\_collocated\_flag and sps\_chroma\_horizontal\_collocated\_flag.

##### Luma sample interpolation filtering process

Inputs to this process are:

* a luma location in full-sample units ( xIntL, yIntL ),
* a luma location in fractional-sample units ( xFracL, yFracL ),
* a luma location in full-sample units ( xSbIntL, ySbIntL ) 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 refPicLXL,
* the half sample interpolation filter index hpelIfIdx,
* a variable sbWidth specifying the width of the current subblock,
* a variable sbHeight specifying the height of the current subblock,
* the decoder-side motion vector refinement flag dmvrFlag,
* a variable refWraparoundEnabledFlag indicating whether horizontal wrap-around motion compensation is enabled,
* a fixedpoint representation of the horizontal scaling factor scalingRatio[ 0 ],
* a fixedpoint representation of the vertical scaling factor scalingRatio[ 1 ],
* a luma location ( xSb, ySb ) specifying the top-left sample of the current subblock relative to the top‑left luma sample of the current picture.

Output of this process is a predicted luma sample value predSampleLXL

The variables shift1, shift2 and shift3 are derived as follows:

* The variable shift1 is set equal to Min( 4, BitDepth − 8 ), the variable shift2 is set equal to 6 and the variable shift3 is set equal to Max( 2, 14 − BitDepth ).
* The variable picW is set equal to pps\_pic\_width\_in\_luma\_samples of the reference picture refPicLX and the variable picH is set equal to pps\_pic\_height\_in\_luma\_samples of the reference picture refPicLX.

The horizontal and vertical half sample interpolation filter indices hpelHorIfIdx and hpelVerIfIdx are derived as follows:

hpelHorIfIdx = ( scalingRatio[ 0 ] = = 16384 )  ?  hpelIfIdx  :  0 (953)

hpelVerIfIdx = ( scalingRatio[ 1 ] = = 16384 )  ?  hpelIfIdx  :  0 (954)

The horizontal luma interpolation filter coefficients fLH[ p ] for each 1/16 fractional sample position p equal to xFracL or  yFracL are derived as follows:

* If MotionModelIdc[ xSb ][ ySb ] is greater than 0, and sbWidth and sbHeight are both equal to 4, and scalingRatio[ 0 ] is greater than 28 672, luma interpolation filter coefficients fLH[ p ] are specified in Table 31.
* Otherwise, if MotionModelIdc[ xSb ][ ySb ] is greater than 0, and sbWidth and sbHeight are both equal to 4, and scalingRatio[ 0 ] is greater than 20 480, luma interpolation filter coefficients fLH[ p ] are specified in Table 32.
* Otherwise, if MotionModelIdc[ xSb ][ ySb ] is greater than 0, and sbWidth and sbHeight are both equal to 4, the luma interpolation filter coefficients fLH[ p ] are specified in Table 30.
* Otherwise, if scalingRatio[ 0 ] is greater than 28 672, luma interpolation filter coefficients fLH[ p ] are specified in Table 28.
* Otherwise, if scalingRatio[ 0 ] is greater than 20 480, luma interpolation filter coefficients fLH[ p ] are specified in Table 29.
* Otherwise, the luma interpolation filter coefficients fLH[ p ] are specified in Table 27 depending on hpelIfIdx set equal to hpelHorIfIdx.

The vertical luma interpolation filter coefficients fLV[ p ] for each 1/16 fractional sample position p equal to yFracL are derived as follows:

* If MotionModelIdc[ xSb ][ ySb ] is greater than 0, and sbWidth and sbHeight are both equal to 4, and scalingRatio[ 1 ] is greater than 28 672, the luma interpolation filter coefficients fLV[ p ] are specified in Table 31.
* Otherwise, if MotionModelIdc[ xSb ][ ySb ] is greater than 0, and sbWidth and sbHeight are both equal to 4, and scalingRatio[ 1 ] is greater than 20 480, the luma interpolation filter coefficients fLV[ p ] are specified in Table 32.
* Otherwise, if MotionModelIdc[ xSb ][ ySb ] is greater than 0, and sbWidth and sbHeight are both equal to 4, the luma interpolation filter coefficients fLV[ p ] are specified in Table 30.
* Otherwise, if scalingRatio[ 1 ] is greater than 28 672, luma interpolation filter coefficients fLV[ p ] are specified in Table 28.
* Otherwise, if scalingRatio[ 1 ] is greater than 20 480, luma interpolation filter coefficients fLV[ p ] are specified in Table 29.
* Otherwise, the luma interpolation filter coefficients fLV[ p ] are specified in Table 27 depending on hpelIfIdx set equal to hpelVerIfIdx.

The luma locations in full-sample units ( xInti, yInti ) are derived as follows for i = 0..7:

xInti = xIntL + i − 3  (955)

yInti = yIntL + i − 3  (956)

– When dmvrFlag is equal to 1, the following applies:

xInti = Clip3( xSbIntL − 3, xSbIntL + sbWidth + 4, xInti ) (957)

yInti = Clip3( ySbIntL − 3, ySbIntL + sbHeight + 4, yInti ) (958)

– 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:

xInti = Clip3( SubpicLeftBoundaryPos, SubpicRightBoundaryPos, refWraparoundEnabledFlag ?  
 ClipH( ( PpsRefWraparoundOffset ) \* MinCbSizeY, picW, xInti ) : xInti ) (959)

yInti = Clip3( SubpicTopBoundaryPos, SubpicBotBoundaryPos, yInti ) (960)

– 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:

xInti = Clip3( 0, picW − 1, refWraparoundEnabledFlag ?  
 ClipH( ( PpsRefWraparoundOffset ) \* MinCbSizeY, picW, xInti ) : xInti ) (961)

yInti = Clip3( 0, picH − 1, yInti ) (962)

The predicted luma sample value predSampleLXL is derived as follows:

* If both xFracLand yFracL are equal to 0, and both scalingRatio[ 0 ] and scalingRatio[ 1 ] are less than 20481, the value of predSampleLXL is derived as follows:

predSampleLXL = refPicLXL[ xInt3 ][ yInt3 ] << shift3 (963)

* Otherwise, if yFracL is equal to 0 and scalingRatio[ 1 ] is less than 20481, the value of predSampleLXL is derived as follows:

predSampleLXL =   >>  shift1 (964)

* Otherwise, if xFracL is equal to 0 and scalingRatio[ 0 ] is less than 20481, the value of predSampleLXL is derived as follows:

predSampleLXL =   >>  shift1 (965)

* Otherwise, the value of predSampleLXL is derived as follows:
* The sample array temp[ n ] with n = 0..7, is derived as follows:

temp[ n ] =   >>  shift1 (966)

* The predicted luma sample value predSampleLXL is derived as follows:

predSampleLXL =   >>  shift2 (967)

Table 27 – Specification of the luma interpolation filter coefficients fL[ p ] for each 1/16 fractional sample position p

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Fractional sample position p** | **interpolation filter coefficients** | | | | | | | |
| **fL[ p ][ 0 ]** | **fL[ p ][ 1 ]** | **fL[ p ][ 2 ]** | **fL[ p ][ 3 ]** | **fL[ p ][ 4 ]** | **fL[ p ][ 5 ]** | **fL[ p ][ 6 ]** | **fL[ p ][ 7 ]** |
| 1 | 0 | 1 | −3 | 63 | 4 | −2 | 1 | 0 |
| 2 | −1 | 2 | −5 | 62 | 8 | −3 | 1 | 0 |
| 3 | −1 | 3 | −8 | 60 | 13 | −4 | 1 | 0 |
| 4 | −1 | 4 | −10 | 58 | 17 | −5 | 1 | 0 |
| 5 | −1 | 4 | −11 | 52 | 26 | −8 | 3 | −1 |
| 6 | −1 | 3 | −9 | 47 | 31 | −10 | 4 | −1 |
| 7 | −1 | 4 | −11 | 45 | 34 | −10 | 4 | −1 |
| 8 (hpelIfIdx = = 0) | −1 | 4 | −11 | 40 | 40 | −11 | 4 | −1 |
| 8 (hpelIfIdx = = 1) | 0 | 3 | 9 | 20 | 20 | 9 | 3 | 0 |
| 9 | −1 | 4 | −10 | 34 | 45 | −11 | 4 | −1 |
| 10 | −1 | 4 | −10 | 31 | 47 | −9 | 3 | −1 |
| 11 | −1 | 3 | −8 | 26 | 52 | −11 | 4 | −1 |
| 12 | 0 | 1 | −5 | 17 | 58 | −10 | 4 | −1 |
| 13 | 0 | 1 | −4 | 13 | 60 | −8 | 3 | −1 |
| 14 | 0 | 1 | −3 | 8 | 62 | −5 | 2 | −1 |
| 15 | 0 | 1 | −2 | 4 | 63 | −3 | 1 | 0 |

Table 28 – Specification of the luma interpolation filter coefficients fL[ p ] for each 1/16 fractional sample position p

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Fractional sample position p** | **interpolation filter coefficients** | | | | | | | |
| **fL[ p ][ 0 ]** | **fL[ p ][ 1 ]** | **fL[ p ][ 2 ]** | **fL[ p ][ 3 ]** | **fL[ p ][ 4 ]** | **fL[ p ][ 5 ]** | **fL[ p ][ 6 ]** | **fL[ 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 fL[ p ] for each 1/16 fractional sample position

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Fractional sample position p** | **interpolation filter coefficients** | | | | | | | |
| **fL[ p ][ 0 ]** | **fL[ p ][ 1 ]** | **fL[ p ][ 2 ]** | **fL[ p ][ 3 ]** | **fL[ p ][ 4 ]** | **fL[ p ][ 5 ]** | **fL[ p ][ 6 ]** | **fL[ 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 fL[ p ] for each 1/16 fractional sample position p for affine motion mode

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Fractional sample position p** | **interpolation filter coefficients** | | | | | | | |
| **fL[ p ][ 0 ]** | **fL[ p ][ 1 ]** | **fL[ p ][ 2 ]** | **fL[ p ][ 3 ]** | **fL[ p ][ 4 ]** | **fL[ p ][ 5 ]** | **fL[ p ][ 6 ]** | **fL[ 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 fL[ p ] for each 1/16 fractional sample position p for affine motion mode

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Fractional sample position p** | **interpolation filter coefficients** | | | | | | | |
| **fL[ p ][ 0 ]** | **fL[ p ][ 1 ]** | **fL[ p ][ 2 ]** | **fL[ p ][ 3 ]** | **fL[ p ][ 4 ]** | **fL[ p ][ 5 ]** | **fL[ p ][ 6 ]** | **fL[ 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 fL[ p ] for each 1/16 fractional sample position  p for affine motion mode

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Fractional sample position p** | **interpolation filter coefficients** | | | | | | | |
| **fL[ p ][ 0 ]** | **fL[ p ][ 1 ]** | **fL[ p ][ 2 ]** | **fL[ p ][ 3 ]** | **fL[ p ][ 4 ]** | **fL[ p ][ 5 ]** | **fL[ p ][ 6 ]** | **fL[ 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 |

##### Luma integer sample fetching process

Inputs to this process are:

* a luma location in full-sample units ( xIntL, yIntL ),
* the luma reference sample array refPicLXL,
* a variable refWraparoundEnabledFlag indicating whether horizontal wrap-around motion compensation is enabled.

Output of this process is a predicted luma sample value predSampleLXL

The variable shift is set equal to Max( 2, 14 − BitDepth ).

The variable picW is set equal to pps\_pic\_width\_in\_luma\_samples of the reference picture refPicLX and the variable picH is set equal to pps\_pic\_height\_in\_luma\_samples of the reference picture refPicLX.

The luma locations in full-sample units ( xInt, yInt ) are derived as follows:

– If sps\_subpic\_treated\_as\_pic\_flag[ CurrSubpicIdx ] is equal to 1 and sps\_num\_subpics\_minus1 for the reference picture refPicLX is greater than 0, the following applies:

xInt = Clip3( SubpicLeftBoundaryPos, SubpicRightBoundaryPos, refWraparoundEnabledFlag ?  
 ClipH( ( PpsRefWraparoundOffset ) \* MinCbSizeY, picW, xIntL ) : xIntL ) (968)

yInt = Clip3( SubpicTopBoundaryPos, SubpicBotBoundaryPos, yIntL ) (969)

– 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 = Clip3( 0, picW − 1, refWraparoundEnabledFlag ? (970)  
 ClipH( ( PpsRefWraparoundOffset ) \* MinCbSizeY, picW, xIntL ) : xIntL )

yInt = Clip3( 0, picH − 1, yIntL ) (971)

The predicted luma sample value predSampleLXL is derived as follows:

predSampleLXL = refPicLXL[ xInt ][ yInt ] << shift3 (972)

##### Chroma sample interpolation process

Inputs to this process are:

– a chroma location in full-sample units ( xIntC, yIntC ),

– a chroma location in 1/32 fractional-sample units ( xFracC, yFracC ),

– a chroma location in full-sample units ( xSbIntC, ySbIntC ) 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 refPicLXC,

– 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 predSampleLXC

The variables shift1, shift2 and shift3 are derived as follows:

– The variable shift1 is set equal to Min( 4, BitDepth − 8 ), the variable shift2 is set equal to 6 and the variable shift3 is set equal to Max( 2, 14 − BitDepth ).

– The variable picWC is set equal to pps\_pic\_width\_in\_luma\_samples / SubWidthC of the reference picture refPicLX and the variable picHC is set equal to pps\_pic\_height\_in\_luma\_samples / SubHeightC of the reference picture refPicLX.

The horizontal chroma interpolation filter coefficients fCH[ p ] for each 1/32 fractional sample position p equal to xFracC are derived as follows:

– If scalingRatio[ 0 ] is greater than 28 672, chroma interpolation filter coefficients fCH[ p ] are specified in Table 35.

– Otherwise, if scalingRatio[ 0 ] is greater than 20 480, chroma interpolation filter coefficients fCH[ p ] are specified in Table 65.

– Otherwise, chroma interpolation filter coefficients fCH[ p ] are specified in Table 33.

The vertical chroma interpolation filter coefficients fCV[ p ] for each 1/32 fractional sample position p equal to yFracC are derived as follows:

* If scalingRatio[ 1 ] is greater than 28 672, chroma interpolation filter coefficients fCV[ p ] are specified in Table 35.
* Otherwise, if scalingRatio[ 1 ] is greater than 20 480, chroma interpolation filter coefficients fCV[ p ] are specified in Table 65.
* Otherwise, chroma interpolation filter coefficients fCV[ p ] are specified in Table 33.

The variable xOffset is set equal to ( PpsRefWraparoundOffset ) \* MinCbSizeY ) / SubWidthC.

The chroma locations in full-sample units ( xInti, yInti ) are derived as follows for i = 0..3:

xInti = xIntC + i − 1  (973)

yInti = yIntC + i − 1  (974)

– When dmvrFlag is equal to 1, the following applies:

xInti = Clip3( xSbIntC − 1, xSbIntC + sbWidth + 2, xInti ) (975)

yInti = Clip3( ySbIntC − 1, ySbIntC + sbHeight + 2, yInti ) (976)

– 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:

xInti = Clip3( SubpicLeftBoundaryPos / SubWidthC, SubpicRightBoundaryPos / SubWidthC,  
 refWraparoundEnabledFlag ? ClipH( xOffset, picWC, xInti ) : xInti ) (977)

yInti = Clip3( SubpicTopBoundaryPos / SubHeightC, SubpicBotBoundaryPos / SubHeightC, yInti ) (978)

– 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:

xInti = Clip3( 0, picWC − 1, refWraparoundEnabledFlag ? ClipH( xOffset, picWC, xInti ) : (979)  
 xIntC + i − 1 )

yInti = Clip3( 0, picHC − 1, yInti ) (980)

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 (981)

– Otherwise, if yFracC is equal to 0 and scalingRatio[ 1 ] is less than 20481, the value of predSampleLXC is derived as follows:

predSampleLXC =   >>  shift1 (982)

– Otherwise, if xFracC is equal to 0 and scalingRatio[ 0 ] is less than 20481, the value of predSampleLXC is derived as follows:

predSampleLXC =   >>  shift1 (983)

– Otherwise, the value of predSampleLXC is derived as follows:

* The sample array temp[ n ] with n = 0..3, is derived as follows:

temp[ n ] =   >>  shift1 (984)

* The predicted chroma sample value predSampleLXC is derived as follows:

predSampleLXC =( fCV[ ][ 0 ] \* temp[ 0 ] +  
  fCV[  ][ 1 ] \* temp[ 1 ] +  
  fCV[  ][ 2 ] \* temp[ 2 ] + (985)  
  fCV[  ][ 3 ] \* temp[ 3 ] ) >> shift2

Table 33 – Specification of the chroma interpolation filter coefficients fC[ p ] for each 1/32 fractional sample position p

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fractional sample position p** | **interpolation filter coefficients** | | | |
| **fC[ p ][ 0 ]** | **fC[ p ][ 1 ]** | **fC[ p ][ 2 ]** | **fC[ 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 fC[ p ] for each 1/32 fractional sample position p for scaling factors of around 1.5x

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fractional sample position p** | **interpolation filter coefficients** | | | |
| **fC[ p ][ 0 ]** | **fC[ p ][ 1 ]** | **fC[ p ][ 2 ]** | **fC[ p ][ 3 ]** |
| 0 | 12 | 40 | 12 | 0 |
| 1 | 11 | 40 | 13 | 0 |
| 2 | 10 | 40 | 15 | −1 |
| 3 | 9 | 40 | 16 | −1 |
| 4 | 8 | 40 | 17 | −1 |
| 5 | 8 | 39 | 18 | −1 |
| 6 | 7 | 39 | 19 | −1 |
| 7 | 6 | 38 | 21 | −1 |
| 8 | 5 | 38 | 22 | −1 |
| 9 | 4 | 38 | 23 | −1 |
| 10 | 4 | 37 | 24 | −1 |
| 11 | 3 | 36 | 25 | 0 |
| 12 | 3 | 35 | 26 | 0 |
| 13 | 2 | 34 | 28 | 0 |
| 14 | 2 | 33 | 29 | 0 |
| 15 | 1 | 33 | 30 | 0 |
| 16 | 1 | 31 | 31 | 1 |
| 17 | 0 | 30 | 33 | 1 |
| 18 | 0 | 29 | 33 | 2 |
| 19 | 0 | 28 | 34 | 2 |
| 20 | 0 | 26 | 35 | 3 |
| 21 | 0 | 25 | 36 | 3 |
| 22 | −1 | 24 | 37 | 4 |
| 23 | −1 | 23 | 38 | 4 |
| 24 | −1 | 22 | 38 | 5 |
| 25 | −1 | 21 | 38 | 6 |
| 26 | −1 | 19 | 39 | 7 |
| 27 | −1 | 18 | 39 | 8 |
| 28 | −1 | 17 | 40 | 8 |
| 29 | −1 | 16 | 40 | 9 |
| 30 | −1 | 15 | 40 | 10 |
| 31 | 0 | 13 | 40 | 11 |

Table 35 – Specification of the chroma interpolation filter coefficients fC[ p ] for each 1/32 fractional sample position p for scaling factors of around 2x

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fractional sample 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 |
| 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 |

#### Prediction refinement with optical flow process

Inputs to this process are:

* two variables sbWidth and sbHeight specifying the width and the height of the current subblock,
* one (sbWidth + 2)x(sbHeight + 2) prediction sample array predSamplesLXL,
* one (sbWidth)x( sbHeight) motion vector difference array diffMvLX.

Output of this process is the (sbWidth)x(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 ] >> shift1 ) −  (986)  
 ( predSamplesLXL[ x ][ y + 1 ] >> shift1 )

gradientV[ x ][ y ]  =  ( predSamplesLXL[ x + 1 ][ y + 2 ] >> shift1 ) −  (987)  
 ( predSamplesLXL[ x + 1 ][ y ] >> shift1 )

* The variable dI is derived as follows:

dI = gradientH[ x ][ y ] \* diffMvLX[ x ][ y ][ 0 ] + gradientV[ x ][ y ] \* diffMvLX[ x ][ y ][ 1 ] (988)

* Prediction sample value at location ( x, y ) in the subblock is derived as follows:

dILimit = ( 1 << max( 13, BitDepth + 1 ) ) (989)

sbSamplesLXL[ x ][ y ] = predSamplesLXL[ x + 1 ][ y + 1 ] + Clip3( − dILimit, dILimit − 1, dI) (990)

#### Bi-directional optical flow prediction process

Inputs to this process are:

* two variables nCbW and nCbH specifying the width and the height of the current coding block,
* two (nCbW + 2)x(nCbH + 2) luma prediction sample arrays predSamplesL0 and predSamplesL1,
* the prediction list utilization flags predFlagL0 and predFlagL1,
* the reference indices refIdxL0 and refIdxL1,

Output of this process is the (nCbW)x(nCbH) array pbSamples of luma prediction sample values.

The variables shift1, shift2, shift3, shift4, offset4, and mvRefineThres are derived as follows:

* The variable shift1 is set to be equal to 6.
* The variable shift2 is set to be equal to 4.
* The variable shift3 is set to be equal to 1.
* The variable shift4 is set equal to Max( 3, 15 − BitDepth ) and the variable offset4 is set equal to 1  <<  ( shift4 − 1 ).
* The variable mvRefineThres is set equal to 1  <<  4.

For xIdx = 0..( nCbW  >>  2 ) − 1 and yIdx = 0..( nCbH  >>  2 ) − 1, the following applies:

* The variable xSb is set equal to ( xIdx  <<  2) + 1 and ySb is set equal to ( yIdx  <<  2 ) + 1.
* The prediction sample values of the current subblock are derived as follows:
  + For x =xSb − 1..xSb + 4, y = ySb − 1..ySb + 4, the following ordered steps apply:

1. The locations ( hx, vy ) for each of the corresponding sample locations ( x, y ) inside the prediction sample arrays are derived as follows:

hx = Clip3( 1, nCbW, x ) (991)

vy = Clip3( 1, nCbH, y ) (992)

1. The variables gradientHL0[ x ][ y ], gradientVL0[ x ][ y ], gradientHL1[ x ][ y ] and gradientVL1[ x ][ y ] are derived as follows:

gradientHL0[ x ][ y ]  =  ( predSamplesL0[ hx + 1 ][vy]  >>  shift1 ) − (993)  
 ( predSampleL0[ hx − 1 ][ vy] )  >>  shift1 )

gradientVL0[ x ][ y ]  =  ( predSampleL0[ hx ][ vy + 1 ]  >>  shift1 ) − (994)  
 ( predSampleL0[ hx][vy − 1 ] )  >>  shift1 )

gradientHL1[ x ][ y ]  =  ( predSamplesL1[ hx + 1 ][vy]  >>  shift1 ) − (995)  
 ( predSampleL1[ hx − 1 ][ vy] )  >>  shift1 )

gradientVL1[ x ][ y ]  =   ( predSampleL1[ hx ][ vy + 1 ]  >>  shift1 ) − (996)  
 ( predSampleL1[ hx][vy − 1 ] )  >>  shift1 )

1. The variables diff[ x ][ y ], tempH[ x ][ y ] and tempV[ x ][ y ] are derived as follows:

diff[ x ][ y ] = (predSamplesL0[ hx ][ vy ]  >>  shift2 ) − ( predSamplesL1[ hx ][ vy ]  >>  shift2 ) (997)

tempH[ x ][ y ] = (gradientHL0[ x ][ y ] + gradientHL1[ x ][ y ] )  >>  shift3 (998)

tempV[ x ][ y ] = (gradientVL0[ x ][ y ] + gradientVL1[ x ][ y ] )  >>  shift3 (999)

* + The variables sGx2, sGy2, sGxGy, sGxdI and sGydI are derived as follows:

sGx2 = ΣiΣj Abs( tempH[ xSb + i ][ ySb + j ] ) with i, j = −1..4 (1000)

sGy2 = ΣiΣj Abs( tempV[ xSb + i ][ ySb + j ] ) with i, j = −1..4 (1001)

sGxGy = ΣiΣj( Sign( tempV[ xSb + i ][ ySb + j ] ) \* tempH[ xSb + i ][ ySb + j ] ) with i, j = −1..4 (1002)

sGxdI = ΣiΣj( −Sign( tempH[ xSb + i ][ ySb + j ] ) \* diff[ xSb + i ][ ySb + j ] ) with i, j = −1..4 (1003)

sGydI = ΣiΣj( −Sign( tempV[ xSb + i ][ ySb + j ] ) \* diff[ xSb + i ][ ySb + j ] ) with i, j = −1..4 (1004)

* + The horizontal and vertical motion offset of the current subblock are derived as:

vx = sGx2 > 0 ? Clip3( −mvRefineThres + 1, mvRefineThres − 1, (1005)  
 ( sGxdI  <<  2 )  >>  Floor( Log2( sGx2 ) ) ) : 0

vy = sGy2 > 0 ? Clip3( −mvRefineThres + 1, mvRefineThres − 1, ( ( sGydI  <<  2 ) − (1006)  
 ( ( vx \* sGxGy )  >>  1 ) )  >>  Floor( Log2( sGy2 ) ) ) : 0

* + For x =xSb − 1..xSb + 2, y = ySb − 1..ySb + 2, the prediction sample values of the current sub-block are derived as follows:

bdofOffset = vx \* ( gradientHL0[ x + 1 ][ y + 1 ] − gradientHL1[ x + 1 ][ y + 1 ] ) (1007)  
 + vy \* (gradientVL0[ x + 1 ][ y + 1 ] − gradientVL1[ x + 1 ][ y + 1 ] )

pbSamples[ x ][ y ] = Clip3( 0, ( 2BitDepth ) − 1, ( predSamplesL0[ x + 1 ][ y + 1 ] + offset4 + (1008)  
 predSamplesL1[ x + 1 ][ y + 1 ] + bdofOffset )  >>  shift4 )

#### Weighted sample prediction process

##### General

Inputs to this process are:

* two variables nCbW and nCbH specifying the width and the height of the current coding block,
* two (nCbW)x(nCbH) arrays predSamplesL0 and predSamplesL1,
* the prediction list utilization flags, predFlagL0 and predFlagL1,
* the reference indices refIdxL0 and refIdxL1,
* the bi-prediction weight index bcwIdx,
* the decoder-side motion vector refinement flag dmvrFlag,
* the variable cIdx specifying the colour component index.

Output of this process is the (nCbW)x(nCbH) array pbSamples of prediction sample values.

The variable weightedPredFlag is derived as follows:

* If sh\_slice\_type is equal to P, weightedPredFlag is set equal to pps\_weighted\_pred\_flag.
* Otherwise (sh\_slice\_type is equal to B), weightedPredFlag is set equal to ( pps\_weighted\_bipred\_flag  &&  !dmvrFlag ).

The following applies:

* If weightedPredFlag is equal to 0 or bcwIdx is not equal to 0, the array pbSamples of the prediction samples is derived by invoking the default weighted sample prediction process as specified in clause 8.5.6.6.2 with the coding block width nCbW, the coding block height nCbH, two (nCbW)x(nCbH) arrays predSamplesL0 and predSamplesL1, the prediction list utilization flags predFlagL0 and predFlagL1, the bi-prediction weight index bcwIdx and the bit depth BitDepth as inputs.
* Otherwise (weightedPredFlag is equal to 1 and bcwIdx is equal to 0), the array pbSamples of the prediction samples is derived by invoking the weighted sample prediction process as specified in clause 8.5.6.6.3 with the coding block width nCbW, the coding block height nCbH, two (nCbW)x(nCbH) arrays predSamplesL0 and predSamplesL1, the prediction list utilization flags predFlagL0 and predFlagL1, the reference indices refIdxL0 and refIdxL1, the colour component index cIdx and the bit depth BitDepth as inputs.

##### Default weighted sample prediction process

Inputs to this process are:

* two variables nCbW and nCbH specifying the width and the height of the current coding block,
* two (nCbW)x(nCbH) arrays predSamplesL0 and predSamplesL1,
* the prediction list utilization flags predFlagL0 and predFlagL1,
* the reference indices refIdxL0 and refIdxL1,
* the bi-prediction weight index bcwIdx.
* the sample bit depth, bitDepth.

Output of this process is the (nCbW)x(nCbH) array pbSamples of prediction sample values.

Variables shift1, shift2, offset1, offset2, and offset3 are derived as follows:

* The variable shift1 is set equal to Max( 2, 14 − bitDepth ) and the variable shift2 is set equal to Max( 3, 15 − bitDepth ).
* The variable offset1 is set equal to 1  <<  ( shift1 − 1 ).
* The variable offset2 is set equal to 1  <<  ( shift2 − 1 ).
* The variable offset3 is set equal to 1  <<  ( shift1 + 2 ).

Depending on the values of predFlagL0 and predFlagL1, the prediction samples pbSamples[ x ][ y ] with x = 0..nCbW − 1 and y = 0..nCbH − 1 are derived as follows:

* If predFlagL0 is equal to 1 and predFlagL1 is equal to 0, the prediction sample values are derived as follows:

pbSamples[ x ][ y ] = Clip3( 0, ( 1  <<  bitDepth ) − 1, ( predSamplesL0[ x ][ y ] + offset1 )  >>  shift1 ) (1009)

* Otherwise, if predFlagL0 is equal to 0 and predFlagL1 is equal to 1, the prediction sample values are derived as follows:

pbSamples[ x ][ y ] = Clip3( 0, ( 1  <<  bitDepth ) − 1, ( predSamplesL1[ x ][ y ] + offset1 )  >>  shift1 ) (1010)

* 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:

pbSamples[ x ][ y ] = Clip3( 0, ( 1  <<  bitDepth ) − 1, (1011)  
 ( predSamplesL0[ x ][ y ] + predSamplesL1[ x ][ y ] + offset2 )  >>  shift2 )

* Otherwise (bcwIdx is not equal to 0 and ciip\_flag[ xCb ][ yCb ] is equal to 0), the following applies:
* The variable w1 is set equal to bcwWLut[ bcwIdx ] with bcwWLut[ k ] = { 4, 5, 3, 10, −2 }.
* The variable w0 is set equal to ( 8 − w1 ).
* The prediction sample values are derived as follows.

pbSamples[ x ][ y ] = Clip3( 0, ( 1  <<  bitDepth ) − 1, (1012)  
 ( w0\*predSamplesL0[ x ][ y ] + w1\*predSamplesL1[ x ][ y ] + offset3 )  >>  (shift1+3) )

##### Explicit weighted sample prediction process

Inputs to this process are:

* two variables nCbW and nCbH specifying the width and the height of the current coding block,
* two (nCbW)x(nCbH) arrays predSamplesL0 and predSamplesL1,
* the prediction list utilization flags, predFlagL0 and predFlagL1,
* the reference indices, refIdxL0 and refIdxL1,
* the variable cIdx specifying the colour component index,
* the sample bit depth, bitDepth.

Output of this process is the (nCbW)x(nCbH) array pbSamples of prediction sample values.

The variable shift1 is set equal to Max( 2, 14 − bitDepth ).

The variables log2Wd, o0, o1, w0 and w1 are derived as follows:

* If cIdx is equal to 0 for luma samples, the following applies:

log2Wd = luma\_log2\_weight\_denom + shift1 (1013)

w0 = LumaWeightL0[ refIdxL0 ] (1014)

w1 = LumaWeightL1[ refIdxL1 ] (1015)

o0 = luma\_offset\_l0[ refIdxL0 ] << ( bitDepth − 8 ) (1016)

o1 = luma\_offset\_l1[ refIdxL1 ] << ( bitDepth − 8 ) (1017)

* Otherwise (cIdx is not equal to 0 for chroma samples), the following applies:

log2Wd = ChromaLog2WeightDenom + shift1 (1018)

w0 = ChromaWeightL0[ refIdxL0 ][ cIdx − 1 ] (1019)

w1 = ChromaWeightL1[ refIdxL1 ][ cIdx − 1 ] (1020)

o0 = ChromaOffsetL0[ refIdxL0 ][ cIdx − 1 ] << ( bitDepth − 8 ) (1021)

o1 = ChromaOffsetL1[ refIdxL1 ][ cIdx − 1 ] << ( bitDepth − 8 ) (1022)

The prediction sample pbSamples[ x ][ y ] with x = 0..nCbW − 1 and y = 0..nCbH − 1 are derived as follows:

* If predFlagL0 is equal to 1 and predFlagL1 is equal to 0, the prediction sample values are derived as follows:

if( log2Wd >= 1 )  
 pbSamples[ x ][ y ] = Clip3( 0, ( 1 << bitDepth ) − 1,  
 ( ( predSamplesL0[ x ][ y ] \* w0 + 2log2Wd − 1 ) >> log2Wd ) + o0 ) (1023)  
else  
 pbSamples[ x ][ y ] = Clip3( 0, ( 1 << bitDepth ) − 1, predSamplesL0[ x ][ y ] \* w0 + o0 )

* Otherwise, if predFlagL0 is equal to 0 and predFlagL1 is equal to 1, the prediction sample values are derived as follows:

if( log2Wd >= 1 )  
 pbSamples[ x ][ y ] = Clip3( 0, ( 1 << bitDepth ) − 1,  
 ( ( predSamplesL1[ x ][ y ] \* w1 + 2log2Wd − 1 ) >> log2Wd ) + o1 ) (1024)  
else  
 pbSamples[ x ][ y ] = Clip3( 0, ( 1 << bitDepth ) − 1, predSamplesL1[ x ][ y ] \* w1 + o1 )

* Otherwise (predFlagL0 is equal to 1 and predFlagL1 is equal to 1), the prediction sample values are derived as follows:

pbSamples[ x ][ y ] = Clip3( 0, ( 1  <<  bitDepth ) − 1,  
 ( predSamplesL0[ x ][ y ] \* w0 + predSamplesL1[ x ][ y ] \* w1 +  
 ( ( o0 + o1 + 1 )  <<  log2Wd ) )  >>  ( log2Wd + 1 ) ) (1025)

#### Weighted sample prediction process for combined merge and intra prediction

Inputs to this process are:

* a luma location ( xCb, yCb ) specifying the top-left sample of the current luma coding block relative to the top left luma sample of the current picture,
* the width of the current coding block cbWidth,
* the height of the current coding block cbHeight,
* two (cbWidth)x(cbHeight) arrays predSamplesInter and predSamplesIntra,
* a variable cIdx specifying the colour component index.

Output of this process is the (cbWidth)x(cbHeight) array predSamplesComb of prediction sample values.

The variable scallFact is derived as follows:

scallFactX = ( cIdx = = 0  | |  SubWidthC = = 1 ) ? 0 : 1 (1026)

scallFactY = ( cIdx = = 0  | |  SubHeightC = = 1 ) ? 0 : 1 (1027)

The neighbouring luma locations ( xNbA, yNbA ) and ( xNbB, yNbB ) are set equal to   
( xCb − 1, yCb − 1 + ( cbHeight  <<  scallFactY ) ) and ( xCb − 1 + (cbWidth  <<  scallFactX ), yCb − 1 ), respectively.

For X being replaced by either A or B, the variables availableX and isIntraCodedNeighbourX are derived as follows:

* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring location ( xNbY, yNbY ) set equal to ( xNbX, yNbX ), checkPredModeY set equal to FALSE, and cIdx set equal to 0 as inputs, and the output is assigned to availableX.
* The variable isIntraCodedNeighbourX is derived as follows:
* If availableX is equal to TRUE and CuPredMode[ 0 ][ xNbX ][ yNbX ] is equal to MODE\_INTRA, isIntraCodedNeighbourX is set equal to TRUE.
* Otherwise, isIntraCodedNeighbourX is set equal to FALSE.

The weight w is derived as follows:

* If isIntraCodedNeighbourA and isIntraCodedNeighbourB are both equal to TRUE, w is set equal to 3.
* Otherwise, if isIntraCodedNeighbourA and isIntraCodedNeighbourB are both equal to to FALSE, w is set equal to 1.
* Otherwise, w is set equal to 2.

When cIdx is equal to 0 and sh\_lmcs\_enabled\_flag is equal to 1, predSamplesInter[ x ][ y ] with x = 0..cbWidth − 1 and y = 0..cbHeight − 1 are modified as follows:

idxY = predSamplesInter[ x ][ y ] >> Log2( OrgCW )  
predSamplesInter [ x ][ y ] = Clip1( LmcsPivot[ idxY ] +  (1028)  
 ( ScaleCoeff[ idxY ] \* ( predSamplesInter[ x ][ y ] − InputPivot[ idxY ] ) +  
 ( 1  <<  10 ) )  >>  11 )

The prediction samples predSamplesComb[ x ][ y ] with x = 0..cbWidth − 1 and y = 0..cbHeight − 1 are derived as follows:

predSamplesComb[ x ][ y ] = ( w \* predSamplesIntra[ x ][ y ] +  (1029)  
 ( 4 − w ) \* predSamplesInter[ x ][ y ] + 2) >> 2

### Decoding process for geometric partitioning mode inter blocks

#### 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 ChromaArrayType is not equal to 0,
* an (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array predSamplesCr of chroma prediction samples for the component Cr, when ChromaArrayType is not equal to 0.

Let predSamplesLAL and predSamplesLBL be (cbWidth)x(cbHeight) arrays of predicted luma sample values and, when ChromaArrayType is not equal to 0, predSamplesLACb, predSamplesLBCb, predSamplesLACr and predSamplesLBCr be (cbWidth / SubWidthC)x(cbHeight / SubHeightC) arrays of predicted chroma sample values.

The predSamplesL, predSamplesCb and predSamplesCr 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 refPicLNL of luma samples and two ordered two-dimensional arrays refPicLNCb and refPicLNCr of chroma samples is derived by invoking the process specified in clause 8.5.6.2 with X set equal to predListFlagN and refIdxX set equal to refIdxN as input.
* The array predSamplesLNL is derived by invoking the fractional sample interpolation process specified in clause 8.5.6.3 with the luma location ( xCb, yCb ), the luma coding block width sbWidth set equal to cbWidth, the luma coding block height sbHeight set equal to cbHeight, the motion vector offset mvOffset set equal to ( 0, 0 ), the motion vector mvLX set equal to mvN and the reference array refPicLXL set equal to refPicLNL, the variable bdofFlag set euqal to FALSE, the variable cIdx is set equal to 0, RprConstraintsActive[ X ][ refIdxLX ], and RefPicScale[ predListFlagN ][ refIdxN ] as inputs.
* When ChromaArrayType is not equal to 0, the array predSamplesLNCb is derived by invoking the fractional sample interpolation process specified in clause 8.5.6.3 with the luma location ( xCb, yCb ), the coding block width sbWidth set equal to cbWidth / SubWidthC, the coding block height sbHeight set equal to cbHeight / SubHeightC, the motion vector offset mvOffset set equal to ( 0, 0 ), the motion vector mvLX set equal to mvCN, and the reference array refPicLXCb set equal to refPicLNCb, the variable bdofFlag set euqal to FALSE, the variable cIdx is set equal to 1, RprConstraintsActive[ X ][ refIdxLX ], and RefPicScale[ predListFlagN ][ refIdxN ] as inputs.
* When ChromaArrayType is not equal to 0, the array predSamplesLNCr is derived by invoking the fractional sample interpolation process specified in clause 8.5.6.3 with the luma location ( xCb, yCb ), the coding block width sbWidth set equal to cbWidth / SubWidthC, the coding block height sbHeight set equal to cbHeight / SubHeightC, the motion vector offset mvOffset set equal to ( 0, 0 ), the motion vector mvLX set equal to mvCN, and the reference array refPicLXCr set equal to refPicLNCr, the variable bdofFlag set euqal to FALSE, the variable cIdx is set equal to 2, RprConstraintsActive[ X ][ refIdxLX ], and RefPicScale[ predListFlagN ][ refIdxN ] as inputs.

1. 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.
2. The prediction samples inside the current luma coding block, predSamplesL[ xL ][ yL ] with xL = 0..cbWidth − 1 and yL = 0..cbHeight − 1, are derived by invoking the weighted sample prediction process for geometric partitioning mode specified in clause 8.5.7.2 with the coding block width nCbW set equal to cbWidth, the coding block height nCbH set equal to cbHeight, the sample arrays predSamplesLAL and predSamplesLBL, and the variables angleIdx, distanceIdx, and cIdx equal to 0 as inputs.
3. When ChromaArrayType is not equal to 0, the prediction samples inside the current chroma component Cb coding block, predSamplesCb[ xC ][ yC ] with xC = 0..cbWidth / SubWidthC − 1 and yC = 0..cbHeight / SubHeightC − 1, are derived by invoking the weighted sample prediction process for geometric partitioning mode specified in clause 8.5.7.2 with the coding block width nCbW set equal to cbWidth / SubWidthC, the coding block height nCbH set equal to cbHeight / SubHeightC, the sample arrays predSamplesLACb and predSamplesLBCb, and the variables angleIdx, distanceIdx, and cIdx equal to 1 as inputs.
4. When ChromaArrayType is not equal to 0, the prediction samples inside the current chroma component Cr coding block, predSamplesCr[ xC ][ yC ] with xC = 0..cbWidth / SubWidthC − 1 and yC = 0..cbHeight / SubHeightC − 1, are derived by invoking the weighted sample prediction process for geometric partitioning mode specified in clause 8.5.7.2 with the coding block width nCbW set equal to cbWidth / SubWidthC, the coding block height nCbH set equal to cbHeight / SubHeightC, the sample arrays predSamplesLACr and predSamplesLBCr, and the variables angleIdx, distanceIdx, and cIdx equal to 2 as inputs.
5. The motion vector storing process for merge geometric partitioning mode specified in clause 8.5.7.3 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width cbWidth, the luma coding block height cbHeight, the partition angle angleIdx and the distance distanceIdx, the luma motion vectors mvA and mvB, the reference indices refIdxA and refIdxB, and the prediction list flags predListFlagA and predListFlagB as inputs.

Table 36 – Specification of angleIdx and distanceIdx based on merge\_gpm\_partition\_idx.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **merge\_gpm\_partition\_idx** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| **angleIdx** | 0 | 0 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 5 | 5 |
| **distanceIdx** | 1 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 |
| **merge\_gpm\_partition\_idx** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| **angleIdx** | 5 | 5 | 8 | 8 | 11 | 11 | 11 | 11 | 12 | 12 | 12 | 12 | 13 | 13 | 13 | 13 |
| **distanceIdx** | 2 | 3 | 1 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 |
| **merge\_gpm\_partition\_idx** | **32** | **33** | **34** | **35** | **36** | **37** | **38** | **39** | **40** | **41** | **42** | **43** | **44** | **45** | **46** | **47** |
| **angleIdx** | 14 | 14 | 14 | 14 | 16 | 16 | 18 | 18 | 18 | 19 | 19 | 19 | 20 | 20 | 20 | 21 |
| **distanceIdx** | 0 | 1 | 2 | 3 | 1 | 3 | 1 | 2 | 3 | 1 | 2 | 3 | 1 | 2 | 3 | 1 |
| **merge\_gpm\_partition\_idx** | **48** | **49** | **50** | **51** | **52** | **53** | **54** | **55** | **56** | **57** | **58** | **59** | **60** | **61** | **62** | **63** |
| **angleIdx** | 21 | 21 | 24 | 24 | 27 | 27 | 27 | 28 | 28 | 28 | 29 | 29 | 29 | 30 | 30 | 30 |
| **distanceIdx** | 2 | 3 | 1 | 3 | 1 | 2 | 3 | 1 | 2 | 3 | 1 | 2 | 3 | 1 | 2 | 3 |

#### Weighted sample prediction process for geometric partitioning mode

Inputs to this process are:

* two variables nCbW and nCbH specifying the width and the height of the current coding block,
* two (nCbW)x(nCbH) arrays predSamplesLA and predSamplesLB,
* a variable angleIdx specifying the angle index of the geometric partition,
* a variable distanceIdx specifying the distance index of the geometric partition,
* a variable cIdx specifying colour component index.

Output of this process is the (nCbW)x(nCbH) array pbSamples of prediction sample values.

The variables nW, nH, shift1, offset1, hwRatio, displacementX, displacementY, partFlip and shiftHor are derived as follows:

nW = ( cIdx = = 0 ) ? nCbW : nCbW \* SubWidthC (1030)

nH = ( cIdx = = 0 ) ? nCbH : nCbH \* SubHeightC (1031)

shift1 = Max( 5, 17 − BitDepth ) (1032)

offset1 = 1  <<  ( shift1 − 1 ) (1033)

hwRatio = nH / nW (1034)

displacementX = angleIdx (1035)

displacementY = ( angleIdx + 8 ) % 32 (1036)

partFlip = ( angleIdx >= 13 && angleIdx <= 27 ) ? 0 : 1 (1037)

shiftHor = ( angleIdx % 16 = = 8 | | ( angleIdx % 16  !=  0 && hwRatio > 0 ) ) ? 0 : 1 (1038)

The variables offsetX and offsetY are derived as follows:

* If shiftHor is equal to 0, the following applies:

offsetX = ( −nW )  >>  1 (1039)

offsetY = ( ( −nH )  >>  1 ) +   
 ( angleIdx < 16 ? ( distanceIdx \* nH )  >>  3 : −( ( distanceIdx \* nH )  >>  3 ) ) (1040)

* Otherwise (shiftHor is equal to 1), the following applies:

offsetX = ( ( −nW )  >>  1 ) +   
 ( angleIdx < 16 ? ( distanceIdx \* nW )  >>  3 : −( ( distanceIdx \* nW )  >>  3 ) ) (1041)

offsetY = ( − nH )  >>  1 (1042)

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 (1043)

yL = ( cIdx = = 0 ) ? y : y \* SubHeightC (1044)

– The variable wValue specifying the weight of the prediction sample is derived based on the array disLut specified in Table 37 as follows:

weightIdx = ( ( ( xL + offsetX )  <<  1 ) + 1 ) \* disLut[ displacementX ] +   
 ( ( ( yL + offsetY )  <<  1 ) + 1 ) ) \* disLut[ displacementY ] (1045)

weightIdxL = partFlip  ?  32 + weightIdx  :  32 − weightIdx (1046)

wValue = Clip3( 0, 8, ( weightIdxL + 4 )  >>  3 ) (1047)

– The prediction sample values are derived as follows:

pbSamples[ x ][ y ] = Clip3( 0, ( 1  <<  BitDepth ) − 1, ( predSamplesLA[ x ][ y ] \* wValue + (1048)  
 predSamplesLB[ x ][ y ] \* ( 8 − wValue ) + offset1 )  >>  shift1 )

Table 37 ‑ Specification of the geometric partitioning distance array disLut.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **idx** | **0** | **2** | **3** | **4** | **5** | **6** | **8** | **10** | **11** | **12** | **13** | **14** |
| **disLut[idx]** | 8 | 8 | 8 | 4 | 4 | 2 | 0 | −2 | −4 | −4 | −8 | −8 |
| **idx** | **16** | **18** | **19** | **20** | **21** | **22** | **24** | **26** | **27** | **28** | **29** | **30** |
| **disLut[idx]** | −8 | −8 | −8 | −4 | −4 | −2 | 0 | 2 | 4 | 4 | 8 | 8 |

#### Motion vector storing process for geometric partitioning mode

This process is invoked when decoding a coding unit with MergeGpmFlag[ xCb ][ yCb ] equal to 1.

Inputs to this process are:

* a luma location ( xCb, yCb ) specifying the top-left sample of the current coding block relative to the top‑left luma sample of the current picture,
* a variable cbWidth specifying the width of the current coding block in luma samples,
* a variable cbHeight specifying the height of the current coding block in luma samples,
* a variable angleIdx specifying the angle index of the geometric partition,
* a variable distanceIdx specifying the distance index of the geometric partition,
* the luma motion vectors in 1/16 fractional-sample accuracy mvA and mvB,
* the reference indices refIdxA and refIdxB,
* the prediction list flags predListFlagA and predListFlagB.

The variables numSbX and numSbY, specifying the number of 4×4 blocks in the current coding block in the horizontal and vertical directions, respectivecly, are set equal to cbWidth >> 2 and cbHeight >> 2, respectively.

The variables hwRatio, displacementX, displacementY, partIdx and shiftHor are derived as follows:

hwRatio = cbHeight / cbWidth (1049)

displacementX = angleIdx (1050)

displacementY = ( angleIdx + 8 ) % 32 (1051)

partIdx = ( angleIdx >= 13 && angleIdx <= 27 ) ? 0 : 1 (1052)

shiftHor = ( angleIdx % 16 = = 8 | | ( angleIdx % 16  !=  0 && hwRatio > 0 ) ? 0 : 1 (1053)

The variables offsetX and offsetY are derived as follows:

* If shiftHor is equal to 0, the following applies:

offsetX = ( −cbWidth )  >>  1 (1054)

offsetY = ( ( −cbHeight )  >>  1 ) +  
 ( angleIdx < 16 ? ( distanceIdx \* cbHeight )  >>  3 : −( ( distanceIdx \* cbHeight )  >>  3 ) ) (1055)

* Otherwise (shiftHor is equal to 1), the following applies:

offsetX = ( ( −cbWidth )  >>  1 ) +   
 ( angleIdx < 16 ? ( distanceIdx \* cbWidth )  >>  3 : −( ( distanceIdx \* cbWidth )  >>  3 ) ) (1056)

offsetY = ( −cbHeight )  >>  1 (1057)

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:

motionIdx = ( ( ( 4 \* xSbIdx + offsetX )  <<  1 ) + 5 ) \* disLut[ displacementX ] +  
 ( ( ( 4 \* ySbIdx + offsetY  <<  1 ) + 5 ) ) \* disLut[ displacementY ] (1058)

* The variable sType is derived as follows:

sType = abs( motionIdx ) < 32 ? 2 : ( motionIdx  <=  0 ? ( 1 − partIdx ) : partIdx ) (1059)

* Depending on the value of sType, the following assignments are made:
* If sType is equal to 0, the following applies:

predFlagL0 = ( predListFlagA = = 0 ) ? 1 : 0 (1060)

predFlagL1 = ( predListFlagA = = 0 ) ? 0 : 1 (1061)

refIdxL0 = ( predListFlagA = = 0 ) ? refIdxA : −1 (1062)

refIdxL1 = ( predListFlagA = = 0 ) ? −1 : refIdxA (1063)

mvL0[ 0 ] = ( predListFlagA = = 0 ) ? mvA[ 0 ] : 0 (1064)

mvL0[ 1 ] = ( predListFlagA = = 0 ) ? mvA[ 1 ] : 0 (1065)

mvL1[ 0 ] = ( predListFlagA = = 0 ) ? 0 : mvA[ 0 ] (1066)

mvL1[ 1 ] = ( predListFlagA = = 0 ) ? 0 : mvA[ 1 ] (1067)

* Otherwise, if sType is equal to 1 or ( sType is equal to 2 and predListFlagA + predListFlagB is not equal to 1 ), the following applies:

predFlagL0 = ( predListFlagB = = 0 ) ? 1 : 0 (1068)

predFlagL1 = ( predListFlagB = = 0 ) ? 0 : 1 (1069)

refIdxL0 = ( predListFlagB = = 0 ) ? refIdxB : −1 (1070)

refIdxL1 = ( predListFlagB = = 0 ) ? −1 : refIdxB (1071)

mvL0[ 0 ] = ( predListFlagB = = 0 ) ? mvB[ 0 ] : 0 (1072)

mvL0[ 1 ] = ( predListFlagB = = 0 ) ? mvB[ 1 ] : 0 (1073)

mvL1[ 0 ] = ( predListFlagB = = 0 ) ? 0 : mvB[ 0 ] (1074)

mvL1[ 1 ] = ( predListFlagB = = 0 ) ? 0 : mvB[ 1 ] (1075)

* Otherwise (sType is equal to 2 and predListFlagA + predListFlagB is equal to 1), the following applies:

predFlagL0 = 1 (1076)

predFlagL1 = 1 (1077)

refIdxL0 = ( predListFlagA = = 0 ) ? refIdxA : refIdxB (1078)

refIdxL1 = ( predListFlagA = = 0 ) ? refIdxB : refIdxA (1079)

mvL0[ 0 ] = ( predListFlagA = = 0 ) ? mvA[ 0 ] : mvB[ 0 ] (1080)

mvL0[ 1 ] = ( predListFlagA = = 0 ) ? mvA[ 1 ] : mvB[ 1 ] (1081)

mvL1[ 0 ] = ( predListFlagA = = 0 ) ? mvB[ 0 ] : mvA[ 0 ] (1082)

mvL1[ 1 ] = ( predListFlagA = = 0 ) ? mvB[ 1 ] : mvA[ 1 ] (1083)

* The following assignments are made for x = 0..3 and y = 0..3:

MvL0[ ( xSbIdx  <<  2 ) + x ][ ( ySbIdx  <<  2 ) + y ] = mvL0 (1084)

MvL1[ ( xSbIdx  <<  2 ) + x ][ ( ySbIdx  <<  2 ) + y ] = mvL1 (1085)

MvDmvrL0[ ( xSbIdx  <<  2 ) + x ][ ( ySbIdx  <<  2 ) + y ] = mvL0 (1086)

MvDmvrL1[ ( xSbIdx  <<  2 ) + x ][ ( ySbIdx  <<  2 ) + y ] = mvL1 (1087)

RefIdxL0[ ( xSbIdx  <<  2 ) + x ][ ( ySbIdx  <<  2 ) + y ] = refIdxL0 (1088)

RedIdxL1[ ( xSbIdx  <<  2 ) + x ][ ( ySbIdx  <<  2 ) + y ] = refIdxL1 (1089)

PredFlagL0[ ( xSbIdx  <<  2 ) + x ][ ( ySbIdx  <<  2 ) + y ] = predFlagL0 (1090)

PredFlagL1[ ( xSbIdx  <<  2 ) + x ][ ( ySbIdx  <<  2 ) + y ] = predFlagL1 (1091)

BcwIdx[ ( xSbIdx  <<  2 ) + x ][ ( ySbIdx  <<  2 ) + y ] = 0 (1092)

### Decoding process for the residual signal of coding blocks coded in inter prediction mode

Inputs to this process are:

* a sample location ( xTb0, yTb0 ) specifying the top-left sample of the current transform block relative to the top‑left sample of the current picture,
* a variable nTbW specifying the width of the current transform block,
* a variable nTbH specifying the height of the current transform block,
* a variable cIdx specifying the colour component of the current block.

Output of this process is an (nTbW)x(nTbH) array resSamples.

The maximum transform block width maxTbWidth and height maxTbHeight are derived as follows:

maxTbWidth = ( cIdx  = =  0 ) ? MaxTbSizeY : MaxTbSizeY / SubWidthC (1093)

maxTbHeight = ( cIdx  = =  0 ) ? MaxTbSizeY : MaxTbSizeY / SubHeightC (1094)

The luma sample location is derived as follows:

( xTbY, yTbY ) = ( cIdx  = =  0 ) ? ( xTb0, yTb0 ) : ( xTb0 \* SubWidthC, yTb0 \* SubHeightC ) (1095)

Depending on maxTbSize, the following applies:

* If nTbW is greater than maxTbWidth or nTbH is greater than maxTbHeight, the following ordered steps apply.

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

verSplitFirst = ( nTbW \* ( cIdx = = 0 ? 1 : SubWidthC ) > nTbH \*( cIdx = = 0 ? 1 : SubHeightC ) ) (1096)  
 && ( nTbW > maxTbWidth )

newTbW = verSplitFirst ? ( nTbW / 2 ) : nTbW (1097)

newTbH = !verSplitFirst ? ( nTbH / 2 ) : nTbH (1098)

1. The decoding process process for the residual signal of coding units coded in inter prediction mode as specified in this clause is invoked with the location ( xTb0, yTb0 ), the transform block width nTbW set equal to newTbW and the height nTbH set equal to newTbH, and the variable cIdx as inputs, and the output is a modified reconstructed picture before in-loop filtering.
2. The following applies:

* If verSplitFirst is equal to TRUE, the decoding process process for the residual signal of coding units coded in inter prediction mode as specified in this clause is invoked with the location ( xTb0, yTb0 ) set equal to ( xTb0 + newTbW, yTb0 ), the transform block width nTbW set equal to newTbW and the height nTbH set equal to newTbH, and the variable cIdx as inputs, and the output is a modified reconstructed picture before in-loop filtering.
* Otherwise (verSplitFirst is equal to FALSE), the decoding process process for the residual signal of coding units coded in inter prediction mode as specified in this clause is invoked with the location ( xTb0, yTb0 ) set equal to ( xTb0, yTb0 + newTbH ), the transform block width nTbW set equal to newTbW and the height nTbH set equal to newTbH, and the variable cIdx as inputs, and the output is a modified reconstructed picture before in-loop filtering.
* Otherwise, if cu\_sbt\_flag is equal to 1, the following applies:
* The variables sbtMinNumFourths, wPartIdx and hPartIdx are derived as follows:

sbtMinNumFourths = cu\_sbt\_quad\_flag ? 1 : 2 (1099)

wPartIdx = cu\_sbt\_horizontal\_flag ? 4 : sbtMinNumFourths (1100)

hPartIdx = !cu\_sbt\_horizontal\_flag ? 4 : sbtMinNumFourths (1101)

* The variables xPartIdx and yPartIdx are derived as follows:
* If cu\_sbt\_pos\_flag is equal to 0, xPartIdx and yPartIdx are set equal to 0.
* Otherwise (cu\_sbt\_pos\_flag is equal to 1), the variables xPartIdx and yPartIdx are derived as follows:

xPartIdx = cu\_sbt\_horizontal\_flag ? 0 : ( 4 − sbtMinNumFourths ) (1102)

yPartIdx = !cu\_sbt\_horizontal\_flag ? 0 : ( 4 − sbtMinNumFourths ) (1103)

* The variables xTbYSub, yTbYSub, xTb0Sub, yTb0Sub, nTbWSub and nTbHSub are derived as follows:

xTbYSub = xTbY + ( ( nTbW \* ( ( cIdx  = =  0 ) ? 1: SubWidthC ) \* xPartIdx / 4 ) (1104)

yTbYSub = yTbY + ( ( nTbH \* ( ( cIdx  = =  0 ) ? 1: SubHeightC ) \* yPartIdx / 4 ) (1105)

xTb0Sub = xTb0 + ( nTbW \* xPartIdx / 4 ) (1106)

yTb0Sub = yTb0 + ( nTbH \* yPartIdx / 4 ) (1107)

nTbWSub = nTbW \* wPartIdx / 4 (1108)

nTbHSub = nTbH \* hPartIdx / 4 (1109)

* The scaling and transformation process as specified in clause 8.7.2 is invoked with the luma location ( xTbYSub , yTbYSub ), the variable cIdx, the variable predMode set equal to MODE\_INTER, nTbWSub and nTbHSub as inputs, and the output is an ( nTbWSub )x( nTbHSub ) array resSamplesTb.
* 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 ] = resSamplesTb[ x − xTb0Sub ][ y − yTb0Sub ] (1110)

* Otherwise, the scaling and transformation process as specified in clause 8.7.2 is invoked with the luma location ( xTbY, yTbY ), the variable cIdx, the variable predMode set equal to MODE\_INTER, the transform width nTbW and the transform height nTbH as inputs, and the output is an (nTbW)x(nTbH) array resSamples.

## Decoding process for coding units coded in IBC prediction mode

### General decoding process for coding units coded in IBC prediction mode

Inputs to this process are:

* a luma location ( xCb, yCb ) specifying the top-left sample of the current coding block relative to the top‑left luma sample of the current picture,
* a variable cbWidth specifying the width of the current coding block in luma samples,
* a variable cbHeight specifying the height of the current coding block in luma samples,
* a variable treeType specifying whether a single or a dual tree is used and if a dual tree is used, it specifies whether the current tree corresponds to the luma or chroma components.

Output of this process is a modified reconstructed picture before in-loop filtering.

The derivation process for quantization parameters as specified in clause 8.7.1 is invoked with the luma location ( xCb, yCb ), the width of the current coding block in luma samples cbWidth and the height of the current coding block in luma samples cbHeight, and the variable treeType as inputs.

The variable IsGt4by4 is derived as follows:

IsGt4by4 = ( cbWidth \* cbHeight ) > 16 (1111)

The decoding process for coding units coded in IBC prediction mode consists of the following ordered steps:

1. The block vector components of the current coding unit are derived as follows:

* The derivation process for block vector components as specified in clause 8.6.2.1 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width cbWidth and the luma coding block height cbHeight as inputs, and the luma block vector bvL as output.
* When treeType is equal to SINGLE\_TREE, the derivation process for chroma block vectors in clause 8.6.2.5 is invoked with luma block vector bvL as input, and chroma block vector bvC as output.

1. The prediction samples of the current coding unit are derived as follows:

* The decoding process for IBC blocks as specified in clause 8.6.3.1 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width cbWidth and the luma coding block height cbHeight, the luma block vector bvL, the variable cIdx set equal to 0 as inputs, and the IBC prediction samples (predSamples) that are an (cbWidth)x(cbHeight) array predSamplesL of prediction luma samples as outputs.
* When treeType is equal to SINGLE\_TREE, the prediction samples of the current coding unit are derived as follows:
  + - * The decoding process for IBC blocks as specified in clause 8.6.3.1 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width cbWidth and the luma coding block height cbHeight, the chroma block vector bvC and the variable cIdx set equal to 1 as inputs, and the IBC prediction samples (predSamples) that are an (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array predSamplesCb of prediction chroma samples for the chroma components Cb as outputs.
      * The decoding process for IBC blocks as specified in clause 8.6.3.1 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width cbWidth and the luma coding block height cbHeight, the chroma block vector bvC and the variable cIdx set equal to 2 as inputs, and the IBC prediction samples (predSamples) that are an (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array predSamplesCr of prediction chroma samples for the chroma components Cr as outputs.

1. The residual samples of the current coding unit are derived as follows:

* The decoding process for the residual signal of coding blocks coded in inter prediction mode as specified in clause 8.5.8 is invoked with the location ( xTb0, yTb0 ) set equal to the luma location ( xCb, yCb ), the width nTbW set equal to the luma coding block width cbWidth, the height nTbH set equal to the luma coding block height cbHeight and the variable cIdx set equal to 0 as inputs, and the array resSamplesL as output.
* When treeType is equal to SINGLE\_TREE, the decoding process for the residual signal of coding blocks coded in inter prediction mode as specified in clause 8.5.8 is invoked with the location ( xTb0, yTb0 ) set equal to the chroma location ( xCb / SubWidthC, yCb / SubHeightC ), the width nTbW set equal to the chroma coding block width cbWidth / SubWidthC, the height nTbH set equal to the chroma coding block height cbHeight / SubHeightC and the variable cIdx set equal to 1 as inputs, and the array resSamplesCb as output.
* When treeType is equal to SINGLE\_TREE, the decoding process for the residual signal of coding blocks coded in inter prediction mode as specified in clause 8.5.8 is invoked with the location ( xTb0, yTb0 ) set equal to the chroma location ( xCb / SubWidthC, yCb / SubHeightC ), the width 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 cIdxset equal to 2 as inputs, and the array resSamplesCr as output.

1. The reconstructed samples of the current coding unit are derived as follows:

* The picture reconstruction process for a colour component as specified in clause 8.7.5 is invoked with the block location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the block width nCurrSw set equal to cbWidth, the block height nCurrSh set equal to cbHeight, the variable cIdx set equal to 0, the (cbWidth)x(cbHeight) array predSamples set equal to predSamplesL and the (cbWidth)x(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, the picture reconstruction process for a colour component as specified in clause 8.7.5 is invoked with the block location ( xCurr, yCurr ) set equal to ( xCb / SubWidthC, yCb / SubHeightC ), the block width nCurrSw set equal to cbWidth / SubWidthC, the block height nCurrSh set equal to cbHeight / SubHeightC, the variable cIdx set equal to 1, the (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array predSamples set equal to predSamplesCb and the (cbWidth / SubWidthC)x(cbHeight / 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, the picture reconstruction process for a colour component as specified in clause 8.7.5 is invoked with the block location ( xCurr, yCurr ) set equal to ( xCb / SubWidthC, yCb / SubHeightC ), the block width nCurrSw set equal to cbWidth / SubWidthC, the block height nCurrSh set equal to cbHeight / SubHeightC, the variable cIdx set equal to 2, the (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array predSamples set equal to predSamplesCr and the (cbWidth / SubWidthC)x(cbHeight / SubHeightC) array resSamples set equal to resSamplesCr as inputs, and the output is a modified reconstructed picture before in-loop filtering.

### Derivation process for block vector components for IBC blocks

#### General

Inputs to this process are:

* a luma location ( xCb, yCb ) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
* a variable cbWidth specifying the width of the current coding block in luma samples,
* a variable cbHeight specifying the height of the current coding block in luma samples.

Outputs of this process are:

* the luma block vector in 1/16 fractional-sample accuracy bvL.

The luma block vector mvL is derived as follows:

* The derivation process for IBC luma block vector prediction as specified in clause 8.6.2.2 is invoked with the luma location ( xCb, yCb ), the variables cbWidth and cbHeight 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 ] (1112)

bvd[ 1 ] = MvdL0[ xCb ][ yCb ][ 1 ] (1113)

1. The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to bvL, rightShift set equal to AmvrShift, and leftShift set equal to AmvrShift as inputs and the rounded bvL as output.
2. The luma block vector bvL is modified as follows:

u[ 0 ] = ( bvL[ 0 ] + bvd[ 0 ] + 218 ) % 218 (1114)

bvL[ 0 ] = ( u[ 0 ] >= 217 ) ? ( u[ 0 ] − 218 ) : u[ 0 ] (1115)

u[ 1 ] = ( bvL[ 1 ] + bvd[ 1 ] + 218 ) % 218  (1116)

bvL[ 1 ] = ( u[ 1 ] >= 217 ) ? ( u[ 1 ] − 218 ) : u[ 1 ] (1117)

NOTE 1 – The resulting values of bvL[ 0 ] and bvL[ 1 ] as specified above will always be in the range of −217 to 217 − 1, inclusive.

When IsGt4by4 is equal to TRUE, the updating process for the history-based block vector predictor list as specified in clause 8.6.2.6 is invoked with luma block vector bvL.

It is a requirement of bitstream conformance that the luma block vector bvL shall obey the following constraints:

* CtbSizeY is greater than or equal to ( ( yCb + ( bvL[ 1 ] >> 4 ) ) & ( CtbSizeY − 1 ) ) + cbHeight.
* IbcVirBuf[ 0 ][ ( x + (bvL[ 0 ] >> 4 ) ) & ( IbcBufWidthY − 1 ) ][ ( y + (bvL[1] >> 4 ) ) & ( CtbSizeY − 1) ] shall not be equal to −1 for x = xCb..xCb + cbWidth − 1 and y = yCb..yCb + cbHeight − 1.

#### Derivation process for IBC luma block vector prediction

This process is only invoked when CuPredMode[ 0 ][ xCb ][ yCb ] is equal to MODE\_IBC, where ( xCb, yCb ) specify the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture.

Inputs to this process are:

* a luma location ( xCb, yCb ) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
* a variable cbWidth specifying the width of the current coding block in luma samples,
* a variable cbHeight specifying the height of the current coding block in luma samples.

Outputs of this process are:

* the luma block vector in 1/16 fractional-sample accuracy bvL.

The luma block vector bvL is derived by the following ordered steps:

1. When IsGt4by4 is equal to TRUE, the derivation process for spatial block vector candidates from neighbouring coding units as specified in clause 8.6.2.3 is invoked with the luma coding block location ( xCb, yCb ), the luma coding block width cbWidth and height cbHeight as inputs, and the outputs being the availability flags availableFlagA1, availableFlagB1 and the block vectors bvA1 and bvB1.
2. When IsGt4by4 is equal to TRUE, the block vector candidate list, bvCandList, is constructed as follows:

i = 0  
if( availableFlagA1 )  
 bvCandList [ i++ ] = bvA1 (1118)  
if( availableFlagB1 )  
 bvCandList [ i++ ] = bvB1

1. The variable numCurrCand is derived as follows:

* 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.

1. 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 8.6.2.4 is invoked with bvCandList, and numCurrCand as inputs, and modified bvCandList and numCurrCand as outputs.
2. 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.
3. The variable bvIdx is derived as follows:

bvIdx = general\_merge\_flag[ xCb ][ yCb ] ? merge\_idx[ xCb ][ yCb ] : mvp\_l0\_flag[ xCb ][ yCb ] (1119)

1. The following assignments are made:

bvL[ 0 ] = bvCandList[ bvIdx ][ 0 ] (1120)

bvL[ 1 ] = bvCandList[ bvIdx ][ 1 ] (1121)

#### 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 availableFlagA1 and availableFlagB1 of the neighbouring coding units,
* the block vectors in 1/16 fractional-sample accuracy bvA1, and bvB1 of the neighbouring coding units,

For the derivation of availableFlagA1 and mvA1 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 clause 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.
* The variables availableFlagA1 and bvA1 are derived as follows:
* If availableA1 is equal to FALSE, availableFlagA1 is set equal to 0 and both components of bvA1 are set equal to 0.
* Otherwise, availableFlagA1 is set equal to 1 and the following assignments are made:

bvA1 = MvL0[ xNbA1 ][ yNbA1 ] (1122)

For the derivation of availableFlagB1 and bvB1 the following applies:

* The luma location ( xNbB1, yNbB1 ) inside the neighbouring luma coding block is set equal to ( xCb + cbWidth − 1, yCb − 1 ).
* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring luma location ( xNbB1, yNbB1 ), checkPredModeY set equal to TRUE, and cIdx set equal to 0 as inputs, and the output is assigned to the block availability flag availableB1.
* The variables availableFlagB1 and bvB1 are derived as follows:
* If one or more of the following conditions are true, availableFlagB1 is set equal to 0 and both components of bvB1 are set equal to 0:
  + availableB1 is equal to FALSE.
  + availableA1 is equal to TRUE and the luma locations ( xNbA1, yNbA1 ) and ( xNbB1, yNbB1 ) have the same block vectors.
* Otherwise, availableFlagB1 is set equal to 1 and the following assignments are made:

bvB1 = MvL0[ xNbB1 ][ yNbB1 ] (1123)

#### Derivation process for IBC history-based block vector candidates

Inputs to this process are:

* a block vector candidate list bvCandList,
* the number of available block vector candidates in the list numCurrCand.

Outputs to this process are:

* the modified block vector candidate list bvCandList,
* the modified number of motion vector candidates in the list numCurrCand.

The variables isPrunedA1 and isPrunedB1 are set both equal to FALSE.

For each candidate in HmvpIbcCandList[ hMvpIdx ] with index hMvpIdx = 1..NumHmvpIbcCand, the following ordered steps are repeated until numCurrCand is equal to MaxNumIbcMergeCand:

1. The variable sameMotion is derived as follows:
   * If all of the following conditions are true for any block vector candidate N with N being A1 or B1, sameMotion and isPrunedN are both set equal to TRUE:
   * IsGt4by4 is equal to TRUE.
   * hMvpIdx is equal to 1.
   * The candidate HmvpIbcCandList[NumHmvpIbcCand − hMvpIdx] is equal to the block vector candidate N.
   * isPrunedN is equal to FALSE.
   * Otherwise, sameMotion is set equal to FALSE.
2. When sameMotion is equal to FALSE, the candidate HmvpIbcCandList[NumHmvpIbcCand − hMvpIdx] is added to the block vector candidate list as follows:

bvCandList[ numCurrCand++ ] = HmvpIbcCandList[ NumHmvpIbcCand − hMvpIdx ] (1124)

#### Derivation process for chroma block vectors

Input to this process is:

* a luma block vector in 1/16 fractional-sample accuracy bvL.

Output of this process is a chroma block vector in 1/32 fractional-sample accuracy bvC.

A chroma block vector is derived from the corresponding luma block vector.

The chroma block vector bvC is derived as follows:

bvC[ 0 ] = ( ( bvL[ 0 ] >> ( 3 + SubWidthC ) ) \* 32 (1125)

bvC[ 1 ] = ( ( bvL[ 1 ] >> ( 3 + SubHeightC ) ) \* 32 (1126)

#### Updating process for the history-based block vector predictor candidate list

Inputs to this process are:

* luma block vector bvL in 1/16 fractional-sample accuracy.

The candidate list HmvpIbcCandList is modified by the following ordered steps:

1. The variable identicalCandExist is set equal to FALSE and the variable removeIdx is set equal to 0.
2. When NumHmvpIbcCand is greater than 0, for each index hMvpIdx with hMvpIdx = 0..NumHmvpIbcCand − 1, the following steps apply until identicalCandExist is equal to TRUE:
   * When bvL is equal to HmvpIbcCandList[ hMvpIdx ], identicalCandExist is set equal to TRUE and removeIdx is set equal to hMvpIdx.
3. The candidate list HmvpIbcCandList is updated as follows:
   * If identicalCandExist is equal to TRUE or NumHmvpIbcCand  is equal to 5, the following applies:
   * For each index i with i = ( removeIdx + 1 )..( NumHmvpIbcCand − 1 ), HmvpIbcCandList[ i − 1] is set equal to HmvpIbcCandList [ i ].
   * HmvpIbcCandList[ NumHmvpIbcCand − 1 ] is set equal to bvL.
   * Otherwise (identicalCandExist is equal to FALSE and NumHmvpIbcCand  is less than 5), the following applies:
   * HmvpIbcCandList[ NumHmvpIbcCand ++ ] is set equal to bvL.

### Decoding process for IBC blocks

#### General

This process is invoked when decoding a coding unit coded in IBC prediction mode.

Inputs to this process are:

* a luma location ( xCb, yCb ) specifying the top-left sample of the current coding block relative to the top‑left luma sample of the current picture,
* a variable cbWidth specifying the width of the current coding block in luma samples,
* a variable cbHeight specifying the height of the current coding block in luma samples,
* the block vector bv,
* a variable cIdx specifying the colour component index of the current block.

Outputs of this process are:

* an array predSamples of prediction samples.

When cIdx is equal to 0, for x = xCb..xCb + cbWidth − 1 and y = yCb..yCb + cbHeight − 1, the following applies:

xVb = ( x + ( bv[ 0 ] >> 4 ) ) & ( IbcBufWidthY − 1 ) (1127)

yVb = ( y + ( bv[ 1 ] >> 4 ) ) & ( CtbSizeY − 1 ) (1128)

predSamples[ x ][ y ] = ibcVirBuf[ 0 ][ xVb ][ yVb ] (1129)

When cIdx is not equal to 0, for x = xCb / subWidthC..xCb / subWidthC + cbWidth / subWidthC − 1 and y = yCb / subHeightC..yCb / subHeightC + cbHeight / subHeightC − 1, the following applies:

xVb = ( x + ( bv[ 0 ] >> 5 ) ) & ( IbcBufWidthC − 1 ) (1130)

yVb = ( y + ( bv[ 1 ] >> 5 ) ) & ( ( CtbSizeY / subHeightC ) − 1 ) (1131)

predSamples[ x ][ y ] = ibcVirBuf[ cIdx ][ xVb ][ yVb ] (1132)

When cIdx is equal to 0, the following assignments are made for x = 0..cbWidth − 1 and y = 0..cbHeight − 1:

MvL0[ xCb + x ][ yCb + y ] = bv (1133)

MvL1[ xCb + x ][ yCb + y ] = 0 (1134)

RefIdxL0[ xCb + x ][ yCb + y ] = −1 (1135)

RefIdxL1[ xCb + x ][ yCb + y ] = −1 (1136)

PredFlagL0[ xCb + x ][ yCb + y ] = 0 (1137)

PredFlagL1[ xCb + x ][ yCb + y ] = 0 (1138)

BcwIdx[ xCb + x ][ yCb + y ] = 0 (1139)

## Scaling, transformation and array construction process

### Derivation process for quantization parameters

Inputs to this process are:

* a luma location ( xCb, yCb ) specifying the top-left luma sample of the current coding block relative to the top-left luma sample of the current picture,
* a variable cbWidth specifying the width of the current coding block in luma samples,
* a variable cbHeight specifying the height of the current coding block in luma samples,
* a variable treeType specifying whether a single tree (SINGLE\_TREE) or a dual tree is used to partition the coding tree node and, when a dual tree is used, whether the luma (DUAL\_TREE\_LUMA) or chroma components (DUAL\_TREE\_CHROMA) are currently processed.

In this process, the luma quantization parameter Qp′Y and the chroma quantization parameters Qp′Cb, Qp′Cr and Qp′CbCr are derived.

The luma location ( xQg, yQg ), specifies the top-left luma sample of the current quantization group relative to the top left luma sample of the current picture. The horizontal and vertical positions xQg and yQg are set equal to CuQgTopLeftX and CuQgTopLeftY, respectively.

NOTE – : The current quantization group is a rectangluar region inside a coding tree block that shares the same qPY\_PRED. Its width and height are equal to the width and height of the coding tree node of which the top-left luma sample position is assigned to the variables CuQgTopLeftX and CuQgTopLeftY.

When treeType is equal to SINGLE\_TREE or DUAL\_TREE\_LUMA, the predicted luma quantization parameter qPY\_PRED is derived by the following ordered steps:

1. The variable qPY\_PREV is derived as follows:

– If one or more of the following conditions are true, qPY\_PREV is set equal to 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.

– The current quantization group is the first quantization group in a CTB row of a tile and sps\_entropy\_coding\_sync\_enabled\_flag is equal to 1.

– Otherwise, qPY\_PREV is set equal to the luma quantization parameter QpY of the last luma coding unit in the previous quantization group in decoding order.

1. The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring location ( xNbY, yNbY ) set equal to ( xQg − 1, yQg ), checkPredModeY set equal to FALSE, and cIdx set equal to 0 as inputs, and the output is assigned to availableA. The variable qPY\_A is derived as follows:

– If one or more of the following conditions are true, qPY\_A is set equal to qPY\_PREV:

– availableA is equal to FALSE.

– The CTB containing the luma coding block covering the luma location ( xQg − 1, yQg ) is not equal to the CTB containing the current luma coding block at ( xCb, yCb ), i.e. one or more of the following conditions are true:

– ( xQg − 1 )  >>  CtbLog2SizeY is not equal to ( xCb )  >>  CtbLog2SizeY

– ( yQg )  >>  CtbLog2SizeY is not equal to ( yCb )  >>  CtbLog2SizeY

– Otherwise, qPY\_A is set equal to the luma quantization parameter QpY of the coding unit containing the luma coding block covering ( xQg − 1, yQg ).

1. The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location ( xCurr, yCurr ) set equal to ( xCb, yCb ), the neighbouring location ( xNbY, yNbY ) set equal to ( xQg, yQg − 1 ), checkPredModeY set equal to FALSE, and cIdx set equal to 0 as inputs, and the output is assigned to availableB. The variable qPY\_B is derived as follows:

– If one or more of the following conditions are true, qPY\_B is set equal to qPY\_PREV:

– availableB is equal to FALSE.

– The CTB containing the luma coding block covering the luma location ( xQg, yQg − 1 ) is not equal to the CTB containing the current luma coding block at ( xCb, yCb ), i.e. one or more of the following conditions are true:

– ( xQg )  >>  CtbLog2SizeY is not equal to ( xCb )  >>  CtbLog2SizeY

– ( yQg − 1 )  >>  CtbLog2SizeY is not equal to ( yCb )  >>  CtbLog2SizeY

– Otherwise, qPY\_B is set equal to the luma quantization parameter QpY of the coding unit containing the luma coding block covering ( xQg, yQg − 1 ).

1. The predicted luma quantization parameter qPY\_PRED is derived as follows:

* If all the following conditions are true, then qPY\_PRED is set equal to the luma quantization parameter QpY 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, qPY\_PRED is derived as follows:

qPY\_PRED = ( qPY\_A + qPY\_B + 1 ) >> 1 (1140)

The variable QpY is derived as follows:

QpY = ( ( qPY\_PRED + CuQpDeltaVal + 64 + 2 \* QpBdOffset ) % ( 64 + QpBdOffset ) ) − QpBdOffset (1141)

The luma quantization parameter Qp′Y is derived as follows:

Qp′Y = QpY + QpBdOffset (1142)

When ChromaArrayType 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 QpY is set equal to the luma quantization parameter QpY of the luma coding unit that covers the luma location ( xCb + cbWidth / 2, yCb + cbHeight / 2 ).

– The variables qPCb, qPCr and qPCbCr are derived as follows:

qPChroma = Clip3( −QpBdOffset, 63, QpY ) (1143)

qPCb = ChromaQpTable[ 0 ][ qPChroma ] (1144)

qPCr = ChromaQpTable[ 1 ][ qPChroma ] (1145)

qPCbCr = ChromaQpTable[ 2 ][ qPChroma ] (1146)

– The chroma quantization parameters for the Cb and Cr components, Qp′Cb and Qp′Cr, and joint Cb-Cr coding Qp′CbCr are derived as follows:

Qp′Cb =  Clip3( −QpBdOffset, 63, qPCb + pps\_cb\_qp\_offset + sh\_cb\_qp\_offset + CuQpOffsetCb )  
  + QpBdOffset (1147)

Qp′Cr = Clip3( −QpBdOffset, 63, qPCr + pps\_cr\_qp\_offset + sh\_cr\_qp\_offset + CuQpOffsetCr )  
  + QpBdOffset (1148)

Qp′CbCr = Clip3( −QpBdOffset, 63, qPCbCr + pps\_joint\_cbcr\_qp\_offset\_value +   
 sh\_joint\_cbcr\_qp\_offset +CuQpOffsetCbCr ) + QpBdOffset (1149)

### 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 nTbW specifying the transform block width,
* a variable nTbH specifying the transform block height.

Output of this process is the (nTbW)x(nTbH) array of residual samples resSamples[ x ][ y ] with x = 0..nTbW − 1, y = 0..nTbH − 1.

The variable codedCIdx is derived as follows:

* If cIdx is equal to 0 or TuCResMode[ xTbY ][ yTbY ] is equal to 0, codedCIdx is set equal to cIdx.
* Otherwise, if TuCResMode[ xTbY ][ yTbY ] is equal to 1 or 2, codedCIdx is set equal to 1.
* Otherwise, codedCIdx is set equal to 2.

The variable cSign is set equal to ( 1 − 2 \* ph\_joint\_cbcr\_sign\_flag ).

The (nTbW)x(nTbH) array of residual samples resSamples is derived as follows:

* 1. The scaling process for transform coefficients as specified in clause 8.7.3 is invoked with the transform block location ( xTbY, yTbY ), the transform block width nTbW and the transform block height nTbH, the prediction mode predMode, and the colour component variable cIdx being set equal to codedCIdx as inputs, and the output is an (nTbW)x(nTbH) array of scaled transform coefficients d.
  2. The (nTbW)x(nTbH) array of residual samples res is derived as follows:
* If transform\_skip\_flag[ xTbY ][ yTbY ][ codedCIdx ] is equal to 1, the residual sample array values res[ x ][ y ] with x = 0..nTbW − 1, y = 0..nTbH − 1 are derived as follows:

res[ x ][ y ] = d[ x ][ y ] (1150)

* Otherwise (transform\_skip\_flag[ xTbY ][ yTbY ][ codedCIdx ] is equal to 0), the transformation process for scaled transform coefficients as specified in clause 8.7.4.1 is invoked with the transform block location ( xTbY, yTbY ), the 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.
  1. The residual samples resSamples[ x ][ y ] with x = 0..nTbW − 1, y = 0..nTbH − 1 are derived as follows:
* If cIdx is equal to codedCIdx, the following applies:

resSamples[ x ][ y ] = res[ x ][ y ] (1151)

* Otherwise, if TuCResMode[ xTbY ][ yTbY ] is equal to 2, the following applies:

resSamples[ x ][ y ] = cSign \* res[ x ][ y ] (1152)

* Otherwise, the following applies:

resSamples[ x ][ y ] = ( cSign \* res[ x ][ y ] ) >> 1 (1153)

### Scaling process for transform coefficients

Inputs to this process are:

* a luma location ( xTbY, yTbY ) specifying the top-left sample of the current luma transform block relative to the top‑left luma sample of the current picture,
* a variable nTbW specifying the transform block width,
* a variable nTbH specifying the transform block height,
* a variable predMode specifying the prediction mode of the coding unit,
* a variable cIdx specifying the colour component of the current block.

Output of this process is the (nTbW)x(nTbH) array d of scaled transform coefficients with elements d[ x ][ y ].

The quantization parameter qP and the variable QpActOffset are derived as follows:

* If cIdx is equal to 0, the following applies:

qP = Qp′Y (1154)

QpActOffset = cu\_act\_enabled\_flag[ xTbY ][ yTbY ] ? −5 : 0 (1155)

* Otherwise, if TuCResMode[ xTbY ][ yTbY ] is equal to 2, the following applies:

qP = Qp′CbCr (1156)

QpActOffset = cu\_act\_enabled\_flag[ xTbY ][ yTbY ] ? 1 : 0 (1157)

* Otherwise, if cIdx is equal to 1, the following applies:

qP = Qp′Cb  (1158)

QpActOffset = cu\_act\_enabled\_flag[ xTbY ][ yTbY ] ? 1 : 0 (1159)

* Otherwise (cIdx is equal to 2), the following applies:

qP = Qp′Cr (1160)

QpActOffset = cu\_act\_enabled\_flag[ xTbY ][ yTbY ] ? 3 : 0 (1161)

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 ) (1162)

rectNonTsFlag = ( ( ( Log2( nTbW ) + Log2( nTbH ) ) & 1 )  = =  1 ) ? 1 : 0 (1163)

bdShift = BitDepth + rectNonTsFlag + (1164)  
 ( ( Log2( nTbW ) + Log2( nTbH ) ) / 2 ) − 5 + sh\_dep\_quant\_enabled\_flag

* Otherwise (transform\_skip\_flag[ xTbY ][ yTbY ][ cIdx ] is equal to 1), the following applies:

qP = Clip3( QpPrimeTsMin, 63 + QpBdOffset, qP + QpActOffset ) (1165)

rectNonTsFlag = 0 (1166)

bdShift = 10 (1167)

The variable bdOffset is derived as follows:

bdOffset = ( 1 << bdShift ) >> 1 (1168)

The list levelScale[ ][ ] is specified as levelScale[ j ][ k ] = { { 40, 45, 51, 57, 64, 72 }, { 57, 64, 72, 80, 90, 102 } } with j = 0..1, k = 0..5.

The (nTbW)x(nTbH) array dz is set equal to the (nTbW)x(nTbH) array TransCoeffLevel[ xTbY ][ yTbY ][ cIdx ].

For the derivation of the scaled transform coefficients d[ x ][ y ] with x = 0..nTbW − 1, y = 0..nTbH − 1, the following applies:

* The intermediate scaling factor m[ x ][ y ] is derived as follows:
* If one or more of the following conditions are true, m[ x ][ y ] is set equal to 16:
* sh\_explicit\_scaling\_list\_used\_flag is equal to 0.
* transform\_skip\_flag[ xTbY ][ yTbY ][ cIdx ] is equal to 1.
* sps\_scaling\_matrix\_for\_lfnst\_disabled\_flag is equal to 1 and ApplyLfnstFlag is equal to 1.
* sps\_scaling\_matrix\_for\_alternative\_colour\_space\_disabled\_flag is equal to 1 and sps\_scaling\_matrix\_designated\_colour\_space\_flag is equal to cu\_act\_enabled\_flag[ xTbY ][ yTbY ].
* Otherwise, the following applies:
* The variable id is derived based on predMode, cIdx, nTbW, and nTbH as specified in Table 38 and the variable log2MatrixSize is derived as follows:

log2MatrixSize = ( id < 2 ) ? 1 : ( id < 8 ) ? 2 : 3 (1169)

* The scaling factor m[ x ][ y ] is derived as follows:

m[ x ][ y ] = ScalingMatrixRec[ id ][ i ][ j ]  
 with i = ( x << log2MatrixSize ) >> Log2( nTbW ),  
 j = ( y << log2MatrixSize ) >> Log2( nTbH ) (1170)

* 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 ] = ScalingMatrixDCRec[ id − 14 ] (1171)

NOTE – A quantization matrix element m[ x ][ y ] can 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\_enabled\_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 ) (1172)

– Otherwise (sh\_dep\_quant\_enabled\_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 ) (1173)

* When BdpcmFlag[ xTbY ][ yYbY ][ cIdx ] is equal to 1, dz[ x ][ y ] is modified as follows:

– If BdpcmDir[ xTbY ][ yYbY ][ cIdx ] is equal to 0 and x is greater than 0, the following applies:

dz[ x ][ y ] = Clip3( CoeffMin, CoeffMax, dz[ x − 1 ][ y ] + dz[ x ][ y ]) (1174)

– Otherwise, if BdpcmDir[ xTbY ][ yTbY ][ cIdx ] is equal to 1 and y is greater than 0, the following applies:

dz[ x ][ y ] = Clip3( CoeffMin, CoeffMax, dz[ x ][ y − 1 ] + dz[ x ][ y ]) (1175)

* The value dnc[ x ][ y ] is derived as follows:

dnc[ x ][ y ] = ( dz[ x ][ y ] \* ls[ x ][ y ] +bdOffset )  >>  bdShift (1176)

* The scaled transform coefficient d[ x ][ y ] is derived as follows:

d[ x ][ y ] = Clip3( CoeffMin, CoeffMax, dnc[ x ][ y ] ) (1177)

Table 38 – Specification of the scaling matrix identifier variable id according to predMode, cIdx, nTbW, and nTbH

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **max( nTbW, nTbH )** | | **2** | **4** | **8** | **16** | **32** | **64** |
| **predMode = MODE\_INTRA** | **cIdx = 0 (Y)** |  | 2 | 8 | 14 | 20 | 26 |
| **cIdx = 1 (Cb)** |  | 3 | 9 | 15 | 21 | 21 |
| **cIdx = 2 (Cr)** |  | 4 | 10 | 16 | 22 | 22 |
| **predMode = MODE\_INTER or MODE\_IBC** | **cIdx = 0 (Y)** |  | 5 | 11 | 17 | 23 | 27 |
| **cIdx = 1 (Cb)** | 0 | 6 | 12 | 18 | 24 | 24 |
| **cIdx = 2 (Cr)** | 1 | 7 | 13 | 19 | 25 | 25 |

### Transformation process for scaled transform coefficients

#### General

Inputs to this process are:

* a luma location ( xTbY, yTbY ) specifying the top-left sample of the current luma transform block relative to the top‑left luma sample of the current picture,
* a variable nTbW specifying the width of the current transform block,
* a variable nTbH specifying the height of the current transform block,
* a variable cIdx specifying the colour component of the current block,
* an (nTbW)x(nTbH) array d[ x ][ y ] of scaled transform coefficients with x = 0..nTbW − 1, y = 0..nTbH − 1.

Output of this process is the (nTbW)x(nTbH) array res[ x ][ y ] of residual samples with x = 0..nTbW − 1, y = 0..nTbH − 1.

When ApplyLfnstFlag is equal to 1, transform\_skip\_flag[ xTbY ][ yTbY ][ cIdx ] is equal to 0 and both nTbW and nTbH are greater than or equal to 4, the following applies:

* The variables predModeIntra, nLfnstOutSize, log2LfnstSize, nLfnstSize, and nonZeroSize are derived as follows:

predModeIntra = ( cIdx  = =  0 ) ? IntraPredModeY[ xTbY ][ yTbY ] : IntraPredModeC[ xTbY ][ yTbY ] (1178)

nLfnstOutSize = ( nTbW  >=  8  &&  nTbH  >=  8 ) ? 48 : 16 (1179)

log2LfnstSize = ( nTbW  >=  8  &&  nTbH  >=  8 ) ? 3 : 2 (1180)

nLfnstSize = 1  <<  log2LfnstSize (1181)

nonZeroSize = ( ( nTbW  = =  4  &&  nTbH  = = 4 ) | | ( nTbW  = = 8  &&  nTbH  = =  8 ) ) ? 8 : 16 (1182)

* When intra\_mip\_flag[ xTbY ][ yTbY ] is equal to 1 and cIdx is equal to 0, predModeIntra is set equal to INTRA\_PLANAR.
* When predModeIntra is equal to either INTRA\_LT\_CCLM, INTRA\_L\_CCLM, or INTRA\_T\_CCLM, predModeIntra is derived as follows:
* If intra\_mip\_flag[ xTbY + nTbW \* SubWidthC / 2 ][ yTbY + nTbH \* SubHeightC / 2 ] is equal to 1, predModeIntra is set equal to INTRA\_PLANAR.
* Otherwise, if CuPredMode[ 0 ][ xTbY + nTbW \* SubWidthC / 2 ][ yTbY + nTbH \* SubHeightC / 2 ] is equal to MODE\_IBC or MODE\_PLT, predModeIntra is set equal to INTRA\_DC.
* Otherwise, predModeIntra is set equal to IntraPredModeY[ xTbY + nTbW \* SubWidthC / 2 ][ yTbY + nTbH \* SubHeightC / 2 ].
* The wide angle intra prediction mode mapping process as specified in clause 8.4.5.2.6 is invoked with predModeIntra, nTbW, nTbH and cIdx as inputs, and the modified predModeIntra as output.
* The values of the list u[ x ] with x = 0..nonZeroSize − 1 are derived as follows:

xC = DiagScanOrder[ 2 ][ 2 ][ x ][ 0 ] (1183)

yC = DiagScanOrder[ 2 ][ 2 ][ x ][ 1 ] (1184)

u[ x ] = d[ xC ][ yC ] (1185)

* The one-dimensional low frequency non-separable transformation process as specified in clause 8.7.4.2 is invoked with the input length of the scaled transform coefficients nonZeroSize, the transform output length nTrS set equal to nLfnstOutSize, the list of scaled non-zero transform coefficients u[ x ] with x = 0..nonZeroSize − 1, and the intra prediction mode for LFNST set selection predModeIntra as inputs, and the list v[ x ] with x = 0..nLfnstOutSize − 1 as output.
* The array d[ x ][ y ] with x = 0..nLfnstSize − 1, y = 0..nLfnstSize − 1 is derived as follows:
* If predModeIntra is less than or equal to 34, the following applies:

d[ x ][ y ] = ( y < 4 ) ? v[ x + ( y  <<  log2LfnstSize ) ] : (1186)  
 ( ( x < 4 ) ? v[ 32 + x + ( ( y − 4 )  <<  2 ) ] : d[ x ][ y ] )

* Otherwise, the following applies:

d[ x ][ y ] = ( x < 4 ) ? v[ y + ( x  <<  log2LfnstSize ) ] : (1187)  
 ( ( y < 4 ) ? v[ 32 + y + ( ( x − 4 )  <<  2 ) ] : d[ x ][ y ] )

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[ x0 ][ y0 ] is equal to 0 and intra\_mip\_flag[ x0 ][ y0 ] is equal to 0
* Otherwise, implicitMtsEnabled is set equal to 0.

The variable trTypeHor specifying the horizontal transform kernel and the variable trTypeVer specifying the vertical transform kernel are derived as follows:

* If one or more of the following conditions are true, trTypeHor and trTypeVer are set equal to 0.
* cIdx is greater than 0
* IntraSubPartitionsSplitType is not equal to ISP\_NO\_SPLIT and lfnst\_idx is not equal to 0
* Otherwise, if implicitMtsEnabled is equal to 1, the following applies:
* If cu\_sbt\_flag is equal to 1, trTypeHor and trTypeVer are specified in Table 40 depending on cu\_sbt\_horizontal\_flag and cu\_sbt\_pos\_flag.
* Otherwise (cu\_sbt\_flag is equal to 0), trTypeHor and trTypeVer are derived as follows:

trTypeHor = ( nTbW >= 4 && nTbW <= 16 ) ? 1 : 0 (1188)

trTypeVer = ( nTbH >= 4 && nTbH <= 16 ) ? 1 : 0 (1189)

* Otherwise, trTypeHor and trTypeVer are specified in Table 39 depending on mts\_idx.

The variables nonZeroW and nonZeroH are derived as follows:

* If ApplyLfnstFlag is equal to 1 and nTbW is greater than or equal to 4 and nTbH is greater than or equal to 4, the following applies:

nonZeroW = ( nTbW = = 4 | | nTbH = = 4 ) ? 4 : 8 (1190)

nonZeroH = ( nTbW = = 4 | | nTbH = = 4 ) ? 4 : 8 (1191)

* Otherwise, the following applies:

nonZeroW = Min( nTbW, ( trTypeHor > 0 ) ? 16 : 32 ) (1192)

nonZeroH = Min( nTbH, ( trTypeVer > 0 ) ? 16 : 32 ) (1193)

The (nTbW)x(nTbH) array r of residual samples is derived as follows:

1. When nTbH is greater than 1, each (vertical) column of scaled transform coefficients d[ x ][ y ] with x = 0..nonZeroW − 1, y = 0..nonZeroH − 1 is transformed to e[ x ][ y ] with x = 0..nonZeroW − 1, y = 0..nTbH − 1 by invoking the one-dimensional transformation process as specified in clause 8.7.4.4 for each column x = 0..nonZeroW − 1 with the height of the transform block nTbH, the non-zero height of the scaled transform coefficients nonZeroH, the list d[ x ][ y ] with y = 0..nonZeroH − 1 and the transform type variable trType set equal to trTypeVer as inputs, and the output is the list e[ x ][ y ] with y = 0..nTbH − 1.
2. When nTbH and nTbW are both greater than 1, the intermediate sample values g[ x ][ y ] with x = 0..nonZeroW − 1, y = 0..nTbH − 1 are derived as follows:

g[ x ][ y ] = Clip3( CoeffMin, CoeffMax, ( e[ x ][ y ] + 64 ) >> 7 ) (1194)

1. When nTbW is greater than 1, each (horizontal) row of the resulting array g[ x ][ y ] with x = 0..nonZeroW − 1, y = 0..nTbH − 1 is transformed to r[ x ][ y ] with x = 0..nTbW − 1, y = 0..nTbH − 1 by invoking the one-dimensional transformation process as specified in clause 8.7.4.4 for each row y = 0..nTbH − 1 with the width of the transform block nTbW, the non-zero width of the resulting array g[ x ][ y ] nonZeroW, the list g[ x ][ y ] with x = 0..nonZeroW − 1 and the transform type variable trType set equal to trTypeHor as inputs, and the output is the list r[ x ][ y ] with x = 0..nTbW − 1.
2. When nTbW is equal to 1, r[ x ][ y ] is set equal to e[ x ][ y ] for x = 0..nTbW − 1, y = 0..nTbH − 1.
3. The (nTbW)x(nTbH) array res[ x ][ y ] of residual samples with x = 0..nTbW − 1, y = 0..nTbH − 1 is derived as follows:

bdShift = Max( 20 − BitDepth, 0 ) (1195)

res[ x ][ y ] = ( r[ x ][ y ] + ( 1 << ( bdShift − 1 ) ) ) >> bdShift (1196)

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 |

#### Low frequency non-separable transformation process

Inputs to this process are:

* a variable nonZeroSize specifying the transform input length,
* a variable nTrS specifying the transform output length,
* a list of scaled non-zero transform coefficients x[ j ] with j = 0..nonZeroSize − 1,
* a variable predModeIntra specifying the intra prediction mode for LFNST set selection.

Output of this process is the list of transformed samples y[ i ] with i = 0..nTrS − 1.

The transformation matrix derivation process as specified in clause 8.7.4.3 is invoked with the transform output length nTrS, and the intra prediction mode for LFNST set selection predModeIntra as inputs, and the (nTrS)x(nonZeroSize) LFNST matrix lowFreqTransMatrix as output.

The list of transformed samples y[ i ] with i = 0..nTrS − 1 is derived as follows:

y[ i ] = Clip3( CoeffMin, CoeffMax, ( ( ) + 64 )  
 >> 7 ) (1197)

#### Low frequency non-separable transformation matrix derivation process

Inputs to this process are:

* a variable nTrS specifying the transform output length,
* a variable predModeIntra specifying the intra prediction mode for LFNST set selection.

Output of this process is the transformation matrix lowFreqTransMatrix.

The variable lfnstTrSetIdx is specified in Table 41 depending on predModeIntra.

Table 41 – Specification of lfnstTrSetIdx

|  |  |
| --- | --- |
| **predModeIntra** | **lfnstTrSetIdx** |
| predModeIntra < 0 | 1 |
| 0 <= predModeIntra <= 1 | 0 |
| 2 <= predModeIntra <= 12 | 1 |
| 13 <= predModeIntra <= 23 | 2 |
| 24 <= predModeIntra <= 44 | 3 |
| 45 <= predModeIntra <= 55 | 2 |
| 56 <= predModeIntra <= 80 | 1 |

The transformation matrix lowFreqTransMatrix is derived based on nTrS, lfnstTrSetIdx, and lfnst\_idx as follows:

* If nTrS is equal to 16, lfnstTrSetIdx is equal to 0, and lfnst\_idx is equal to 1, the following applies:

lowFreqTransMatrix[ m ][ n ] =

{

{ 108 -44 -15 1 -44 19 7 -1 -11 6 2 -1 0 -1 -1 0 }

{ -40 -97 56 12 -11 29 -12 -3 18 18 -15 -3 -1 -3 2 1 }

{ 25 -31 -1 7 100 -16 -29 1 -54 21 14 -4 -7 2 4 0 }

{ -32 -39 -92 51 -6 -16 36 -8 3 22 18 -15 4 1 -5 2 }

{ 8 -9 33 -8 -16 -102 36 23 -4 38 -27 -5 5 16 -8 -6 }

{ -25 5 16 -3 -38 14 11 -3 -97 7 26 1 55 -10 -19 3 }

{ 8 9 16 1 37 36 94 -38 -7 3 -47 11 -6 -13 -17 10 }

{ 2 34 -5 1 -7 24 -25 -3 8 99 -28 -29 6 -43 21 11 }

{ -16 -27 -39 -109 6 10 16 24 3 19 10 24 -4 -7 -2 -3 }

{ -9 -10 -34 4 -9 -5 -29 5 -33 -26 -96 33 14 4 39 -14 }

{ -13 1 4 -9 -30 -17 -3 -64 -35 11 17 19 -86 6 36 14 }

{ 8 -7 -5 -15 7 -30 -28 -87 31 4 4 33 61 -5 -17 22 }

{ -2 13 -6 -4 -2 28 -13 -14 -3 37 -15 -3 -2 107 -36 -24 }

{ 4 9 11 31 4 9 16 19 12 33 32 94 12 0 34 -45 }

{ 2 -2 8 -16 8 5 28 -17 6 -7 18 -45 40 36 97 -8 }

{ 0 -2 0 -10 -1 -7 -3 -35 -1 -7 -2 -32 -6 -33 -16 -112 }

},

* Otherwise, if nTrS is equal to 16, lfnstTrSetIdx is equal to 0, and lfnst\_idx is equal to 2, the following applies:

lowFreqTransMatrix[ m ][ n ] =

{

{ 119 -30 -22 -3 -23 -2 3 2 -16 3 6 0 -3 2 1 0 }

{ -27 -101 31 17 -47 2 22 3 19 30 -7 -9 5 3 -5 -1 }

{ 0 58 22 -15 -102 2 38 2 10 -13 -5 4 14 -1 -9 0 }

{ 23 4 66 -11 22 89 -2 -26 13 -8 -38 -1 -9 -20 -2 8 }

{ -19 -5 -89 2 -26 76 -11 -17 20 13 18 -4 1 -15 3 5 }

{ -10 -1 -1 6 23 25 87 -7 -74 4 39 -5 0 -1 -20 -1 }

{ -17 -28 12 -8 -32 14 -53 -6 -68 -67 17 29 2 6 25 4 }

{ 1 -24 -23 1 17 -7 52 9 50 -92 -15 27 -15 -10 -6 3 }

{ -6 -17 -2 -111 7 -17 8 -42 9 18 16 25 -4 2 -1 11 }

{ 9 5 35 0 6 21 -9 34 44 -3 102 11 -7 13 11 -20 }

{ 4 -5 -5 -10 15 19 -2 6 6 -12 -13 6 95 69 -29 -24 }

{ -6 -4 -9 -39 1 22 0 102 -19 19 -32 30 -16 -14 -8 -23 }

{ 4 -4 7 8 4 -13 -18 5 0 0 21 22 58 -88 -54 28 }

{ -4 -7 0 -24 -7 0 -25 3 -3 -30 8 -76 -34 4 -80 -26 }

{ 0 6 0 30 -6 1 -13 -23 1 20 -2 80 -44 37 -68 1 }

{ 0 0 -1 5 -1 -7 1 -34 -2 3 -6 19 5 -38 11 -115 }

},

* Otherwise, if nTrS is equal to 16, lfnstTrSetIdx is equal to 1, and lfnst\_idx is equal to 1, the following applies:

lowFreqTransMatrix[ m ][ n ] =

{

{ -111 39 4 3 44 11 -12 -1 7 -16 -5 2 3 -1 4 2 }

{ -47 -27 15 -1 -92 43 20 -2 20 39 -16 -5 10 -5 -13 2 }

{ -35 -23 4 4 -17 -72 32 6 -59 18 50 -6 0 40 0 -13 }

{ 13 93 -27 -4 -48 13 -34 4 -52 11 1 10 3 16 -3 1 }

{ -11 -27 1 2 -47 -4 -36 10 -2 -85 14 29 -20 -2 57 4 }

{ 0 -35 32 -2 26 60 -3 -17 -82 1 -30 0 -37 21 3 12 }

{ -17 -46 -92 14 7 -10 -39 29 -17 27 -28 17 1 -15 -13 17 }

{ 4 -10 -23 4 16 58 -17 26 30 21 67 2 -13 59 13 -40 }

{ 5 -20 32 -5 8 -3 -46 -7 -4 2 -15 24 100 44 0 5 }

{ -4 -1 38 -18 -7 -42 -63 -6 33 34 -23 15 -65 33 -20 2 }

{ -2 -10 35 -19 5 8 -44 14 -25 25 58 17 7 -84 -16 -18 }

{ 5 13 18 34 11 -4 18 18 5 58 -3 42 -2 -10 85 38 }

{ -5 -7 -34 -83 2 -1 -4 -73 4 20 15 -12 4 -3 44 12 }

{ 0 4 -2 -60 5 9 42 34 5 -14 9 80 -5 13 -38 37 }

{ -1 2 7 -57 3 -7 9 68 -9 6 -49 -20 6 -4 36 -64 }

{ -1 0 -12 23 1 -4 17 -53 -3 4 -21 72 -4 -8 -3 -83 }

},

* Otherwise, if nTrS is equal to 16, lfnstTrSetIdx is equal to 1, and lfnst\_idx is equal to 2, the following applies:

lowFreqTransMatrix[ m ][ n ] =

{

{ 88 -55 6 -3 -66 27 9 -2 11 11 -13 1 -2 -7 1 2 }

{ -58 -20 27 -2 -27 75 -29 0 47 -42 -11 11 -9 -3 19 -4 }

{ -51 23 -22 5 -63 3 37 -5 1 64 -35 -4 29 -31 -11 13 }

{ -27 -76 49 -2 40 14 9 -17 -56 36 -25 6 14 3 -6 8 }

{ 19 -4 -36 22 52 7 36 -23 28 -17 -64 15 -5 -44 48 9 }

{ 29 50 13 -10 1 34 -59 1 -51 4 -16 30 52 -33 24 -5 }

{ -12 -21 -74 43 -13 39 18 -5 -58 -35 27 -5 19 26 6 -5 }

{ 19 38 -10 -5 28 66 0 -5 -4 19 -30 -26 -40 28 -60 37 }

{ -6 27 18 -5 -37 -18 12 -25 -44 -10 -38 37 -66 45 40 -7 }

{ -13 -28 -45 -39 0 -5 -39 69 -23 16 -12 -18 -50 -31 24 13 }

{ -1 8 24 -51 -15 -9 44 10 -28 -70 -12 -39 24 -18 -4 51 }

{ -8 -22 -17 33 -18 -45 -57 -27 0 -31 -30 29 -2 -13 -53 49 }

{ 1 12 32 51 -8 8 -2 -31 -22 4 46 -39 -49 -67 14 17 }

{ 4 5 24 60 -5 -14 -23 38 9 8 -34 -59 24 47 42 28 }

{ -1 -5 -20 -34 4 4 -15 -46 18 31 42 10 10 27 49 78 }

{ -3 -7 -22 -34 -5 -11 -36 -69 -1 -3 -25 -73 5 4 4 -49 }

},

* Otherwise, if nTrS is equal to 16, lfnstTrSetIdx is equal to 2, and lfnst\_idx is equal to 1, the following applies:

lowFreqTransMatrix[ m ][ n ] =

{

{ -112 47 -2 2 -34 13 2 0 15 -7 1 0 8 -3 -1 0 }

{ 29 -7 1 -1 -108 40 2 0 -45 13 4 -1 8 -5 1 0 }

{ -36 -87 69 -10 -17 -33 26 -2 7 14 -11 2 6 8 -7 0 }

{ 28 -5 2 -2 -29 13 -2 0 103 -36 -4 1 48 -16 -4 1 }

{ -12 -24 15 -3 26 80 -61 9 15 54 -36 2 0 -4 6 -2 }

{ 18 53 69 -74 14 24 28 -30 -6 -7 -11 12 -5 -7 -6 8 }

{ 5 -1 2 0 -26 6 0 1 45 -9 -1 0 -113 28 8 -1 }

{ -13 -32 18 -2 15 34 -27 7 -25 -80 47 -1 -16 -50 28 2 }

{ -4 -13 -10 19 18 46 60 -48 16 33 60 -48 1 0 5 -2 }

{ 15 33 63 89 8 15 25 40 -4 -8 -15 -8 -2 -6 -9 -7 }

{ -8 -24 -27 15 12 41 26 -29 -17 -50 -39 27 0 35 -67 26 }

{ -2 -6 -24 13 -1 -8 37 -22 3 18 -51 22 -23 -95 17 17 }

{ -3 -7 -16 -21 10 24 46 75 8 20 38 72 1 2 1 7 }

{ 2 6 10 -3 -5 -16 -31 12 7 24 41 -16 -16 -41 -89 49 }

{ 4 8 21 40 -4 -11 -28 -57 5 14 31 70 7 18 32 52 }

{ 0 1 4 11 -2 -4 -13 -34 3 7 20 47 -6 -19 -42 -101 }

},

* Otherwise, if nTrS is equal to 16, lfnstTrSetIdx is equal to 2, and lfnst\_idx is equal to 2, the following applies:

lowFreqTransMatrix[ m ][ n ] =

{

{ -99 39 -1 2 65 -20 -5 0 -15 -2 5 -1 0 3 -1 0 }

{ 58 42 -33 3 33 -63 23 -1 -55 32 3 -5 21 -2 -8 3 }

{ -15 71 -44 5 -58 -29 25 3 62 -7 -4 -4 -19 4 0 1 }

{ 46 5 4 -6 71 -12 -15 5 52 -38 13 -2 -63 23 3 -3 }

{ -14 -54 -29 29 25 -9 61 -29 27 44 -48 5 -27 -21 12 7 }

{ -3 3 69 -42 -11 -50 -26 26 24 63 -19 -5 -18 -22 12 0 }

{ 17 16 -2 1 38 18 -12 0 62 1 -14 5 89 -42 8 -2 }

{ 15 54 -8 6 6 60 -26 -8 -30 17 -38 22 -43 -45 42 -7 }

{ -6 -17 -55 -28 9 30 -8 58 4 34 41 -52 -16 -36 -20 16 }

{ -2 -1 -9 -79 7 11 48 44 -13 -34 -55 6 12 23 20 -11 }

{ 7 29 14 -6 12 53 10 -11 14 59 -15 -3 5 71 -54 13 }

{ -5 -24 -53 15 -3 -15 -61 26 6 30 -16 23 13 56 44 -35 }

{ 4 8 21 52 -1 -1 -5 29 -7 -17 -44 -84 8 20 31 39 }

{ -2 -11 -25 -4 -4 -21 -53 2 -5 -26 -64 19 -8 -19 -73 39 }

{ -3 -5 -23 -57 -2 -4 -24 -75 1 3 9 -25 6 15 41 61 }

{ 1 1 7 18 1 2 16 47 2 5 24 67 3 9 25 88 }

},

* Otherwise, if nTrS is equal to 16, lfnstTrSetIdx is equal to 3, and lfnst\_idx is equal to 1, the following applies:

lowFreqTransMatrix[ m ][ n ] =

{

{ -114 37 3 2 -22 -23 14 0 21 -17 -5 2 5 2 -4 -1 }

{ -19 -41 19 -2 85 -60 -11 7 17 31 -34 2 -11 19 2 -8 }

{ 36 -25 18 -2 -42 -53 35 5 46 -60 -25 19 8 21 -33 -1 }

{ -27 -80 44 -3 -58 1 -29 19 -41 18 -12 -7 12 -17 7 -6 }

{ -11 -21 37 -10 44 -4 47 -12 -37 -41 58 18 10 -46 -16 31 }

{ 15 47 10 -6 -16 -44 42 10 -80 25 -40 21 -23 -2 3 -14 }

{ 13 25 79 -39 -13 10 31 -4 49 45 12 -8 3 -1 43 7 }

{ 16 11 -26 13 -13 -74 -20 -1 5 -6 29 -47 26 -49 54 2 }

{ -8 -34 -26 7 -26 -19 29 -37 1 22 46 -9 -81 37 14 20 }

{ -6 -30 -42 -12 -3 5 57 -52 -2 37 -12 6 74 10 6 -15 }

{ 5 9 -6 42 -15 -18 -9 26 15 58 14 43 23 -10 -37 75 }

{ -5 -23 -23 36 3 22 36 40 27 -4 -16 56 -25 -46 56 -24 }

{ 1 3 23 73 8 5 34 46 -12 2 35 -38 26 52 2 -31 }

{ -3 -2 -21 -52 1 -10 -17 44 -19 -20 30 45 27 61 49 21 }

{ -2 -7 -33 -56 -4 -6 21 63 15 31 32 -22 -10 -26 -52 -38 }

{ -5 -12 -18 -12 8 22 38 36 -5 -15 -51 -63 -5 0 15 73 }

},

* Otherwise, if nTrS is equal to 16, lfnstTrSetIdx is equal to 3, and lfnst\_idx is equal to 2, the following applies:

lowFreqTransMatrix[ m ][ n ] =

{

{ -102 22 7 2 66 -25 -6 -1 -15 14 1 -1 2 -2 1 0 }

{ 12 93 -27 -6 -27 -64 36 6 13 5 -23 0 -2 6 5 -3 }

{ -59 -24 17 1 -62 -2 -3 2 83 -12 -17 -2 -24 14 7 -2 }

{ -33 23 -36 11 -21 50 35 -16 -23 -78 16 19 22 15 -30 -5 }

{ 0 -38 -81 30 27 5 51 -32 24 36 -16 12 -24 -8 9 1 }

{ 28 38 8 -9 62 32 -13 2 51 -32 15 5 -66 28 0 -1 }

{ 11 -35 21 -17 30 -18 31 18 -11 -36 -80 12 16 49 13 -32 }

{ -13 23 22 -36 -12 64 39 25 -19 23 -36 9 -30 -58 33 -7 }

{ -9 -20 -55 -83 3 -2 1 62 8 2 27 -28 7 15 -11 5 }

{ -6 24 -38 23 -8 40 -49 0 -7 9 -25 -44 23 39 70 -3 }

{ 12 17 17 0 32 27 21 2 67 11 -6 -10 89 -22 -12 16 }

{ 2 -9 8 45 7 -8 27 35 -9 -31 -17 -87 -23 -22 -19 44 }

{ -1 -9 28 -24 -1 -10 49 -30 -8 -7 40 1 4 33 65 67 }

{ 5 -12 -24 -17 13 -34 -32 -16 14 -67 -7 9 7 -74 49 1 }

{ 2 -6 11 45 3 -10 33 55 8 -5 59 4 7 -4 44 -66 }

{ -1 1 -14 36 -1 2 -20 69 0 0 -15 72 3 4 5 65 }

},

* Otherwise, if nTrS is equal to 48, lfnstTrSetIdx is equal to 0, and lfnst\_idx is equal to 1 the following applies:

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol0to15[ m ][ n ] with m = 0..15, n = 0..15

lowFreqTransMatrixCol0to15 =

{

{ -117 28 18 2 4 1 2 1 32 -18 -2 0 -1 0 0 0 }

{ -29 -91 47 1 9 0 3 0 -54 26 -8 3 0 1 0 0 }

{ -10 62 -11 -8 -2 -2 -1 -1 -95 3 32 0 4 0 2 0 }

{ -15 15 -10 -2 1 0 1 0 10 112 -20 -17 -4 -4 -1 -2 }

{ 32 39 92 -44 4 -10 1 -4 26 12 -15 13 -5 2 -2 0 }

{ -10 1 50 -15 2 -3 1 -1 -28 -15 14 6 1 1 1 0 }

{ 1 -33 -11 -14 7 -2 2 0 29 -12 37 -7 -4 0 -1 0 }

{ 0 6 -6 21 -4 2 0 0 -20 -24 -104 30 5 5 1 2 }

{ -13 -13 -37 -101 29 -11 8 -3 -12 -15 -20 2 -11 5 -2 1 }

{ 6 1 -14 -36 9 -3 2 0 10 9 -18 -1 -3 1 0 0 }

{ -12 -2 -26 -12 -9 2 -1 1 -3 30 4 34 -4 0 -1 0 }

{ 0 -3 0 -4 -15 6 -3 1 -7 -15 -28 -86 19 -5 4 -1 }

{ -1 9 13 5 14 -2 2 -1 -8 3 -4 -62 4 1 1 0 }

{ 6 2 -3 2 10 -1 2 0 8 3 -1 -20 0 1 0 0 }

{ 6 9 -2 35 110 -22 11 -4 -2 0 -3 1 -18 12 -3 2 }

{ -1 7 -2 9 -11 5 -1 1 -7 2 -22 4 -13 0 -1 0 }

},

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol16to31[ m − 16 ][ n ] with m = 16..31, n = 0..15

lowFreqTransMatrixCol16to31 =

{

{ 14 -1 -3 0 -1 0 0 0 2 0 0 0 0 0 0 0 }

{ 33 5 -9 -1 -2 0 -1 0 -3 3 0 0 0 0 0 0 }

{ 32 -30 -4 4 -1 1 0 0 6 2 -5 0 0 0 0 0 }

{ -20 -26 31 1 0 0 0 0 2 -16 -1 6 0 1 0 0 }

{ 29 -16 -22 8 0 1 0 1 -20 6 4 -3 1 0 0 0 }

{ -99 -4 9 5 5 2 2 1 44 -10 -11 1 -2 0 -1 0 }

{ 6 -99 3 26 -1 5 0 2 14 30 -27 -2 1 -1 0 -1 }

{ -7 -46 10 -14 7 0 1 0 9 21 7 -6 -2 -1 0 -1 }

{ -12 10 26 12 -6 0 -1 0 -32 -2 11 3 3 -1 1 0 }

{ 38 26 -13 -1 -5 -1 -1 0 102 3 -14 -1 -5 -1 -2 0 }

{ -30 3 -92 14 19 0 3 0 -11 34 21 -33 1 -2 0 -1 }

{ -5 -17 -41 42 -6 2 -1 1 -1 -40 37 13 -4 2 -1 1 }

{ -12 23 16 -11 -17 0 -1 0 -11 97 -3 -3 0 -6 0 -2 }

{ -4 4 -16 0 -2 0 1 0 34 23 6 -7 -4 -2 -1 0 }

{ -5 -4 -22 8 -25 3 0 0 -3 -21 2 -3 9 -2 1 0 }

{ 0 28 0 76 4 -6 0 -2 -13 5 -76 -4 33 -1 3 0 }

},

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol32to47[ m − 32 ][ n ] with m = 32..47, n = 0..15

lowFreqTransMatrixCol32to47 =

{

{ 3 0 -1 0 1 0 0 0 1 0 0 0 1 0 0 0 }

{ 7 2 -2 0 -1 1 0 0 2 1 -1 0 0 0 0 0 }

{ 6 -3 0 0 2 0 -1 0 2 -1 0 0 1 0 0 0 }

{ 1 -4 0 0 0 -3 0 1 0 -1 0 0 0 -2 0 0 }

{ 1 -4 -3 2 -4 1 0 0 1 -1 -2 1 -2 0 0 0 }

{ -5 4 -3 0 8 -1 -2 0 -2 1 -1 0 4 0 -1 0 }

{ -6 6 6 -3 1 3 -3 0 -1 1 1 0 0 1 -1 0 }

{ 2 2 5 -2 0 3 4 -1 0 0 1 0 0 1 2 -1 }

{ 11 -5 -1 6 -4 2 1 0 3 -1 1 2 -1 0 0 0 }

{ -29 10 10 0 10 -4 -1 1 -7 1 2 1 2 -1 0 0 }

{ -9 -4 18 3 2 0 0 -2 -1 -1 3 0 0 0 0 -1 }

{ -10 13 -1 -4 4 -4 3 4 -2 2 -1 -1 1 -1 1 2 }

{ -21 -5 23 0 2 -2 -1 6 -3 -3 1 0 0 0 0 2 }

{ 108 -5 -30 6 -27 10 7 -2 11 -3 -1 1 -4 1 0 1 }

{ -7 1 3 -5 3 0 -1 0 0 1 0 -1 1 0 0 0 }

{ 9 18 -3 -35 -4 -1 6 1 1 2 0 -3 -1 0 2 0 }

},

* Otherwise, if nTrS is equal to 48, lfnstTrSetIdx is equal to 0, and lfnst\_idx is equal to 2 the following applies:

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol0to15[ m ][ n ] with m = 0..15, n = 0..15

lowFreqTransMatrixCol0to15 =

{

{ -108 48 9 1 1 1 0 0 44 -6 -9 -1 -1 0 -1 0 }

{ 55 66 -37 -5 -6 -1 -2 0 67 -30 -20 4 -2 0 -1 0 }

{ 2 86 -21 -13 -4 -2 -1 -1 -88 5 6 4 5 1 1 0 }

{ -24 -21 -38 19 0 4 -1 2 -23 -89 31 20 2 3 1 1 }

{ 9 20 98 -26 -3 -5 0 -2 -9 -26 15 -16 2 0 1 0 }

{ -21 -7 -37 10 2 2 -1 1 -10 69 -5 -7 -2 -2 0 -1 }

{ -10 -25 4 -17 8 -2 2 -1 -27 -17 -71 25 8 2 1 1 }

{ 2 5 10 64 -9 4 -3 1 -4 8 62 3 -17 1 -2 0 }

{ -11 -15 -28 -97 6 -1 4 -1 7 3 57 -15 10 -2 0 -1 }

{ 9 13 24 -6 7 -2 1 -1 16 39 20 47 -2 -2 -2 0 }

{ -7 11 12 7 2 -1 0 -1 -14 -1 -24 11 2 0 0 0 }

{ 0 0 7 -6 23 -3 3 -1 5 1 18 96 13 -9 -1 -1 }

{ -2 -6 -1 -10 0 1 1 0 -7 -2 -28 20 -15 4 -3 1 }

{ -1 6 -16 0 24 -3 1 -1 2 6 6 16 18 -7 1 -1 }

{ -5 -6 -3 -19 -104 18 -4 3 0 6 0 35 -41 20 -2 2 }

{ -1 -2 0 23 -9 0 -2 0 1 1 8 -1 29 1 1 0 }

},

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol16to31[ m − 16 ][ n ] with m = 16..31, n = 0..15

lowFreqTransMatrixCol16to31 =

{

{ 9 -9 -1 1 0 0 0 0 3 -1 1 0 0 0 0 0 }

{ -31 -19 14 4 1 1 1 0 -6 3 5 -2 0 0 0 0 }

{ 14 -5 0 3 0 0 0 0 10 -5 -2 0 -1 0 0 0 }

{ -30 26 36 -8 -2 -2 0 -1 14 18 -7 -9 -1 -1 0 0 }

{ -61 -3 -2 3 7 1 1 0 12 16 -6 -1 0 -1 0 0 }

{ -93 2 19 0 3 0 2 0 17 4 0 0 -1 0 0 0 }

{ -4 -66 28 36 -5 3 0 1 -10 20 33 -13 -8 0 0 -1 }

{ -3 -75 5 -14 1 4 0 1 -36 3 18 -4 4 0 1 0 }

{ -1 -27 13 6 1 -1 0 0 -34 -6 0 3 4 1 2 0 }

{ 28 23 76 -5 -25 -3 -3 -1 6 36 -7 -39 -4 -1 0 -1 }

{ -20 48 11 -13 -5 -2 0 -1 -105 -19 17 0 6 2 3 0 }

{ -21 -7 -42 14 -24 -3 0 0 11 -47 -7 3 -5 9 1 2 }

{ -2 -32 -2 -66 3 7 1 2 -11 13 -70 5 43 -2 3 0 }

{ -3 11 -63 9 4 -5 2 -1 -22 94 -4 -6 -4 -4 1 -2 }

{ -2 10 -18 16 21 3 -2 0 -2 11 6 -10 6 -3 -1 0 }

{ 3 -6 13 76 30 -11 -1 -2 -26 -8 -69 7 -9 -7 3 -1 }

},

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol32to47[ m − 32 ][ n ] with m = 32..47, n = 0..15

lowFreqTransMatrixCol32to47 =

{

{ 1 -1 0 0 1 0 0 0 0 -1 0 0 0 0 0 0 }

{ -7 -1 1 0 -1 1 1 0 -2 -1 1 0 0 0 0 0 }

{ 6 -5 0 1 2 -1 0 0 1 -1 0 0 1 0 0 0 }

{ 1 3 -2 -1 3 2 -2 -1 0 1 0 0 1 1 -1 0 }

{ 2 0 -8 1 3 1 -1 1 0 -1 -2 0 1 0 -1 0 }

{ 5 -4 -2 0 4 -2 0 1 0 0 0 0 2 -1 0 0 }

{ 3 6 -3 -7 -1 3 3 -1 1 0 -1 0 0 1 1 -1 }

{ 1 14 -2 -8 -2 1 -3 0 2 2 -1 -2 0 1 -1 0 }

{ -2 8 1 5 -2 0 -3 1 1 1 0 2 -1 0 -1 0 }

{ 2 -4 -18 -3 -1 -1 -2 -2 1 -2 -2 0 0 0 -1 -1 }

{ -14 8 8 2 1 2 -1 -2 3 0 -1 0 0 0 0 0 }

{ 0 -1 19 -1 1 0 -1 -6 -1 1 2 0 1 0 0 -2 }

{ 8 -14 -3 43 -1 2 7 -1 1 -2 1 3 -1 1 1 0 }

{ 10 23 -19 -5 0 -6 -4 6 3 -2 1 1 0 -1 0 0 }

{ -1 5 -1 -6 -1 -1 -1 -1 -1 0 0 0 0 0 0 -1 }

{ -10 -34 -25 13 -1 0 11 5 1 -1 1 -2 0 0 2 0 }

},

* Otherwise, if nTrS is equal to 48, lfnstTrSetIdx is equal to 1, and lfnst\_idx is equal to 1 the following applies:

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol0to15[ m ][ n ] with m = 0..15, n = 0..15

lowFreqTransMatrixCol0to15 =

{

{ 110 -49 -3 -4 -1 -1 0 -1 -38 -1 10 0 2 0 1 0 }

{ -43 -19 17 -1 3 0 1 0 -98 46 14 -1 2 0 1 0 }

{ -19 17 -7 3 -2 1 -1 0 -32 -59 29 3 4 0 2 0 }

{ -35 -103 39 1 7 0 2 0 38 -13 25 -6 1 -1 0 0 }

{ 9 5 -6 -1 -1 0 -1 0 42 4 21 -11 1 -3 1 -1 }

{ -5 -5 -28 9 -3 2 -1 1 -20 -78 22 16 1 3 0 1 }

{ 14 17 27 -12 1 -3 1 -1 8 19 -13 4 -2 1 -1 0 }

{ 7 35 17 -4 -1 0 0 0 3 8 54 -17 1 -2 1 -1 }

{ -13 -27 -101 24 -8 6 -3 2 11 43 6 28 -6 3 -1 1 }

{ -11 -13 -3 -10 3 -1 1 0 -19 -19 -37 8 4 2 0 1 }

{ -4 -10 -24 -11 3 -2 0 -1 -6 -37 -45 -17 8 -2 2 -1 }

{ -2 1 13 -17 3 -5 1 -2 3 0 -55 22 6 1 1 0 }

{ 3 1 5 -15 1 -2 1 -1 7 4 -7 29 -1 2 -1 1 }

{ -4 -8 -1 -50 6 -4 2 -2 -1 5 -22 20 6 1 0 0 }

{ 5 -1 26 102 -13 12 -4 4 -4 -2 -40 -7 -23 3 -5 1 }

{ -5 -6 -27 -22 -12 0 -3 0 -5 8 -20 -83 0 0 0 0 }

},

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol16to31[ m − 16 ][ n ] with m = 16..31, n = 0..15

lowFreqTransMatrixCol16to31 =

{

{ -9 13 1 -2 0 0 0 0 -4 2 -3 0 0 0 0 0 }

{ 26 26 -15 -3 -2 -1 -1 0 11 -7 -9 2 0 0 0 0 }

{ -72 43 34 -9 3 -2 1 -1 13 36 -18 -10 0 -2 0 -1 }

{ -1 7 6 -7 1 -1 0 0 -13 14 2 -4 2 -1 0 0 }

{ 21 70 -32 -21 0 -4 -1 -1 34 -26 -57 11 4 2 0 1 }

{ 80 -6 25 -5 -4 -1 -1 0 6 -24 7 -9 0 0 0 0 }

{ 48 -1 48 -15 -4 -2 -1 -1 1 60 -28 -42 5 -6 1 -2 }

{ 10 14 -11 -34 4 -4 1 -1 -80 -7 -6 2 15 0 3 0 }

{ -3 14 21 -12 -7 -2 -1 -1 -23 10 -4 -12 3 0 1 0 }

{ -12 -30 3 -9 5 0 1 0 -56 -9 -47 8 21 1 4 1 }

{ 17 14 -58 14 15 0 2 0 -10 34 -7 28 4 -1 1 0 }

{ 8 74 21 40 -14 0 -2 0 -36 -8 11 -13 -23 1 -3 0 }

{ 8 3 12 -14 -9 -1 -1 0 4 29 -15 31 10 4 1 1 }

{ -16 -15 18 -29 -11 2 -2 1 40 -45 -19 -22 31 2 4 1 }

{ -1 5 8 -23 7 2 1 1 10 -11 -13 -3 12 -3 2 0 }

{ 9 7 24 -20 41 3 6 1 15 20 12 11 17 -9 1 -2 }

},

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol32to47[ m − 32 ][ n ] with m = 32..47, n = 0..15

lowFreqTransMatrixCol32to47 =

{

{ -2 2 0 1 -1 1 0 0 -1 1 0 0 -1 0 0 0 }

{ 9 -3 -1 2 3 -3 0 0 4 -1 0 0 2 -1 0 0 }

{ 3 0 -12 3 6 1 -3 2 1 -1 -2 0 3 1 -1 1 }

{ -2 11 -6 -2 -2 4 -3 0 0 3 -2 0 -1 1 -1 0 }

{ -4 -32 5 24 1 -6 12 4 -3 -2 4 -2 0 -1 0 0 }

{ -7 3 13 -4 -3 5 1 -5 -2 3 1 -2 -1 2 -1 -2 }

{ 11 -11 -51 11 -2 -10 -2 13 2 -6 -4 4 -2 -3 2 2 }

{ -16 46 1 3 2 7 -24 0 2 -2 -5 8 1 -1 -2 2 }

{ 2 9 -10 0 1 -5 -4 4 2 -2 2 2 0 -2 1 0 }

{ -11 -30 10 59 -2 8 41 8 2 5 6 -7 -1 3 5 -2 }

{ 23 34 -31 4 10 -22 -30 22 4 -15 9 20 2 -5 9 4 }

{ -36 6 16 -14 2 19 -4 -12 -1 0 -7 -3 0 2 -2 -1 }

{ 61 22 55 14 13 3 -9 -65 1 -11 -21 -7 0 0 -1 3 }

{ -25 41 0 12 9 7 -42 12 -3 -14 2 28 5 1 6 2 }

{ -9 23 4 9 14 9 -14 -4 0 -12 -7 6 3 0 6 3 }

{ -26 -1 18 -1 -12 32 3 -18 -5 10 -25 -5 -2 1 -8 10 }

},

* Otherwise, if nTrS is equal to 48, lfnstTrSetIdx is equal to 1, and lfnst\_idx is equal to 2 the following applies:

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol0to15[ m ][ n ] with m = 0..15, n = 0..15

lowFreqTransMatrixCol0to15 =

{

{ 80 -49 6 -4 1 -1 1 -1 -72 36 4 0 1 0 0 0 }

{ -72 -6 17 0 3 0 1 0 -23 58 -21 2 -3 1 -1 0 }

{ -50 19 -15 4 -1 1 -1 1 -58 -2 30 -3 4 -1 2 0 }

{ -33 -43 28 -7 4 -2 2 -1 -38 11 -8 4 1 1 0 0 }

{ 10 66 -21 -3 -3 0 -1 0 -53 -41 -2 16 -1 4 -1 1 }

{ 18 14 13 -9 2 -2 1 -1 34 32 -31 12 -5 2 -2 1 }

{ 21 66 -1 9 -4 2 -1 1 -21 41 -30 -10 0 -2 0 -1 }

{ 1 -6 -24 17 -5 3 -2 1 24 10 39 -21 5 -4 2 -1 }

{ 9 33 -24 1 4 0 1 0 6 50 26 1 -10 0 -2 0 }

{ -7 -9 -32 14 -3 3 -1 1 -23 -28 0 -5 -1 0 0 0 }

{ 6 30 69 -18 5 -4 3 -1 -3 -11 -34 -16 9 -4 2 -1 }

{ 1 -8 24 -3 7 -2 2 -1 -6 -51 -6 -4 -5 0 -1 0 }

{ 4 10 4 17 -9 4 -2 1 5 14 32 -15 9 -3 2 -1 }

{ -3 -9 -23 10 -10 3 -3 1 -5 -14 -16 -27 13 -5 2 -1 }

{ 2 11 22 2 9 -2 2 0 -6 -7 20 -32 -3 -4 0 -1 }

{ 2 -3 8 14 -5 3 -1 1 -2 -11 5 -18 8 -3 2 -1 }

},

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol16to31[ m − 16 ][ n ] with m = 16..31, n = 0..15

lowFreqTransMatrixCol16to31 =

{

{ 26 0 -12 2 -2 1 -1 0 -7 -9 6 1 0 0 0 0 }

{ 55 -46 -1 6 -2 1 -1 0 -22 7 17 -7 2 -1 1 0 }

{ 6 57 -34 0 -2 0 -1 0 34 -48 -2 14 -4 3 -1 1 }

{ -55 24 26 -5 2 -1 1 0 15 46 -40 -1 -1 0 -1 0 }

{ 36 -5 41 -20 3 -3 1 -1 -30 26 -32 -3 7 -2 2 -1 }

{ 40 4 -4 -9 -3 -2 -1 -1 27 -31 -43 19 -2 3 -1 1 }

{ -35 -17 -3 26 -6 5 -2 2 56 3 18 -25 -1 -2 -1 -1 }

{ 33 32 -30 4 -3 -1 -1 0 -4 13 -16 -10 0 -1 0 0 }

{ -27 1 -28 -21 16 -5 3 -2 -23 36 -2 40 -17 4 -3 1 }

{ -36 -59 -24 14 4 2 1 1 -23 -26 23 26 -3 5 0 2 }

{ -16 35 -35 30 -9 3 -2 1 -57 -13 6 4 -5 5 -1 1 }

{ 38 -1 0 25 6 2 1 1 47 20 35 1 -27 1 -5 0 }

{ 7 13 19 15 -8 1 -1 0 3 25 30 -18 1 -2 0 -1 }

{ -1 -13 -30 11 -5 2 -1 0 -5 -8 -22 -16 10 0 1 0 }

{ 13 -5 -28 6 18 -4 3 -1 -26 27 -14 6 -20 0 -2 0 }

{ 12 -23 -19 22 2 0 1 0 23 41 -7 35 -10 4 -1 1 }

},

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol32to47[ m − 32 ][ n ] with m = 32..47, n = 0..15

lowFreqTransMatrixCol32to47 =

{

{ 3 5 -1 -2 -2 -2 -1 1 1 1 0 0 -1 -1 0 0 }

{ 9 5 -12 1 -3 -4 4 2 4 1 -2 -1 -1 -1 1 0 }

{ -10 7 21 -10 6 1 -11 0 -1 -1 4 2 3 0 -2 -1 }

{ 17 -38 1 17 -3 11 15 -11 3 -1 -10 1 0 1 3 2 }

{ 15 -8 1 17 -1 -2 4 -8 2 0 -1 3 0 0 0 -1 }

{ 7 -49 52 10 -11 22 7 -26 -1 -6 -9 6 -2 2 4 -2 }

{ -15 -13 -27 9 9 -6 20 5 -3 2 -6 -9 3 -3 1 5 }

{ 24 -26 -37 33 5 -32 55 -5 -7 22 -14 -22 1 -9 -3 13 }

{ 43 -13 4 -41 -19 -2 -24 17 11 -4 8 4 -3 -3 -3 -3 }

{ 10 -26 38 7 -12 11 42 -22 -5 20 -14 -15 -1 -2 1 6 }

{ 28 10 4 7 0 -15 7 -10 -1 7 -2 2 1 -3 0 0 }

{ 37 -37 -9 -47 -28 5 0 18 8 6 0 -8 -4 -3 -3 1 }

{ 11 24 22 -11 -3 37 -13 -58 -5 12 -63 26 9 -15 11 8 }

{ 0 -29 -27 6 -27 -10 -30 9 -3 -10 -7 77 9 -13 45 -8 }

{ -76 -26 -4 -7 12 51 5 24 7 -17 -16 -12 -5 4 2 13 }

{ 5 7 23 5 69 -38 -8 -32 -15 -31 24 11 2 18 11 -15 }

},

* Otherwise, if nTrS is equal to 48, lfnstTrSetIdx is equal to 2, and lfnst\_idx is equal to 1 the following applies:

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol0to15[ m ][ n ] with m = 0..15, n = 0..15

lowFreqTransMatrixCol0to15 =

{

{ -121 33 4 4 1 2 0 1 -1 -1 1 0 0 0 0 0 }

{ 0 -2 0 0 0 0 0 0 121 -23 -7 -3 -2 -1 -1 0 }

{ -20 19 -5 2 -1 1 0 0 16 3 -2 0 0 0 0 0 }

{ 32 108 -43 10 -9 3 -3 1 4 19 -7 1 -1 0 0 0 }

{ -3 0 -1 0 0 0 0 0 -29 11 -2 1 0 0 0 0 }

{ -4 -12 -3 1 -1 0 0 0 19 105 -31 7 -6 1 -2 0 }

{ 7 1 2 0 0 0 0 0 4 3 -2 0 0 0 0 0 }

{ -8 -31 14 -4 3 -1 1 0 9 43 0 1 -1 0 0 0 }

{ -15 -43 -100 23 -12 6 -4 2 -6 -17 -48 10 -5 2 -1 1 }

{ -3 1 2 0 0 0 0 0 -6 3 1 0 0 0 0 0 }

{ -1 -6 -3 2 -1 0 0 0 -6 -35 9 0 2 0 0 0 }

{ -5 -14 -48 2 -5 1 -2 0 10 24 99 -17 10 -4 3 -1 }

{ -2 0 2 0 0 0 0 0 -2 0 1 0 0 0 0 0 }

{ -2 -10 -4 0 0 0 0 0 3 11 -1 -1 0 0 0 0 }

{ -2 -3 -25 -2 -3 0 -1 0 -1 -3 -1 4 -2 2 0 1 }

{ 4 -4 28 103 -42 24 -9 7 1 2 4 0 3 -1 0 0 }

},

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol16to31[ m − 16 ][ n ] with m = 16..31, n = 0..15

lowFreqTransMatrixCol16to31 =

{

{ 24 -5 -1 -1 0 0 0 0 5 -1 0 0 0 0 0 0 }

{ 17 1 -2 0 0 0 0 0 -27 4 2 0 0 0 0 0 }

{ -120 14 8 1 3 1 1 0 -18 -2 3 0 1 0 0 0 }

{ 11 -30 9 -2 1 -1 0 0 0 -8 2 0 0 0 0 0 }

{ 12 7 -1 0 0 0 0 0 -117 12 9 1 3 0 1 0 }

{ 9 46 -6 0 0 0 0 0 8 -29 9 -3 1 0 0 0 }

{ 22 -8 1 -1 0 0 0 0 -28 -9 4 0 1 0 0 0 }

{ -13 -105 17 -2 2 0 0 0 -8 -25 -3 0 0 0 0 0 }

{ 1 -5 19 -6 3 -1 1 0 2 7 15 -3 1 -1 0 0 }

{ 0 3 -2 0 0 0 0 0 -20 8 -2 0 0 0 0 0 }

{ 1 -6 11 -2 2 0 1 0 -9 -100 17 -1 1 0 0 0 }

{ 4 14 32 0 2 0 1 0 -4 0 -39 6 -4 1 -1 0 }

{ -1 -1 1 -1 0 0 0 0 -1 -4 2 0 0 0 0 0 }

{ -6 -40 -15 6 -2 1 0 0 5 57 -6 2 0 0 0 0 }

{ -7 -8 -97 17 -9 3 -3 1 -8 -26 -61 -1 -3 -1 -1 -1 }

{ -1 0 -9 -42 17 -9 3 -2 -1 1 -14 6 -4 2 -1 0 }

},

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol32to47[ m − 32 ][ n ] with m = 32..47, n = 0..15

lowFreqTransMatrixCol32to47 =

{

{ 3 -1 0 0 2 -1 0 0 2 -1 0 0 1 0 0 0 }

{ -12 2 1 0 -5 1 0 0 -1 0 0 0 -2 0 0 0 }

{ 17 -3 -1 0 6 -1 -1 0 2 0 0 0 2 0 0 0 }

{ -7 -1 2 0 -3 -1 1 0 -2 -2 1 0 0 0 0 0 }

{ -32 -3 3 0 12 -2 -1 0 7 0 0 0 1 0 0 0 }

{ -3 -19 3 0 -4 -6 1 0 0 0 0 0 0 -1 0 0 }

{ 117 -10 -8 0 32 1 -4 0 3 1 -1 0 -3 1 0 0 }

{ -7 32 -5 1 -1 4 0 0 2 -1 0 0 1 0 -1 0 }

{ 4 10 5 -1 0 3 1 0 -2 1 2 0 -1 1 1 0 }

{ 30 13 -3 0 -116 6 10 0 -35 -5 4 0 -3 -1 0 0 }

{ -10 -63 1 2 -17 3 -4 0 -1 9 -1 0 3 4 -1 0 }

{ 2 -3 -4 0 2 -2 -2 0 0 0 -1 0 0 -1 -1 0 }

{ -8 -2 -1 1 30 4 -4 1 -102 4 8 -1 -69 -2 6 -1 }

{ 1 -95 18 -6 -10 -34 -2 0 -4 17 -2 0 0 2 1 0 }

{ 2 10 24 -7 5 9 19 -1 0 1 4 0 -2 0 1 0 }

{ -1 -2 -4 4 0 3 1 -1 0 2 0 -2 2 0 0 0 }

},

* Otherwise, if nTrS is equal to 48, lfnstTrSetIdx is equal to 2, and lfnst\_idx is equal to 2 the following applies:

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol0to15[ m ][ n ] with m = 0..15, n = 0..15

lowFreqTransMatrixCol0to15 =

{

{ 87 -41 3 -4 1 -1 0 -1 -73 28 2 1 1 1 0 0 }

{ -75 4 7 0 2 0 1 0 -41 36 -7 3 -1 1 0 0 }

{ 26 -44 22 -6 4 -2 1 -1 77 24 -22 2 -4 0 -1 0 }

{ -39 -68 37 -7 6 -2 2 0 -9 56 -21 1 -2 0 -1 0 }

{ 10 -20 2 0 1 0 0 0 50 -1 8 -5 1 -1 0 0 }

{ -21 -45 8 -2 3 -1 1 0 -7 -30 26 -8 3 -1 1 -1 }

{ -4 -2 -55 28 -8 5 -3 2 -2 37 43 -19 1 -2 1 -1 }

{ 2 19 47 -23 6 -4 2 -1 -23 -22 -44 17 -2 2 -1 0 }

{ -19 -62 -9 3 0 0 0 0 -12 -56 27 -7 3 -1 1 0 }

{ 1 9 -5 0 -1 0 0 0 0 22 -1 2 0 1 0 0 }

{ 5 17 -9 0 -2 1 0 0 13 54 -2 7 -1 1 0 0 }

{ 7 27 56 -2 10 -3 3 -1 -2 -6 8 -28 3 -4 1 -1 }

{ 0 0 19 -4 3 -2 2 -1 -3 -13 10 -4 1 0 0 0 }

{ -3 0 -27 -80 40 -16 6 -4 4 3 31 61 -22 7 -1 1 }

{ 1 2 -8 6 -1 1 0 0 2 8 -5 -1 0 0 0 0 }

{ -4 -18 -57 8 -8 1 -3 0 -5 -20 -69 7 -6 2 -2 1 }

},

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol16to31[ m − 16 ][ n ] with m = 16..31, n = 0..15

lowFreqTransMatrixCol16to31 =

{

{ 30 -5 -6 1 -1 0 0 0 -8 -3 3 0 0 0 0 0 }

{ 72 -29 -2 0 -1 0 -1 0 -37 6 7 -2 1 0 0 0 }

{ 7 -38 10 0 1 0 0 0 -51 27 4 -3 2 -1 1 0 }

{ -45 4 -3 6 -1 2 0 1 49 -13 3 -3 -1 0 0 0 }

{ 66 17 -24 4 -3 1 -1 0 13 -49 15 1 0 0 0 0 }

{ -9 69 -33 5 -2 0 -1 0 -44 -31 10 7 -2 2 0 1 }

{ -47 -34 -27 5 4 -1 1 0 -39 -2 27 4 -2 1 0 0 }

{ -33 3 22 -2 -4 1 -1 0 -58 -17 6 -6 7 -1 1 0 }

{ 7 -8 16 -6 4 -2 1 -1 -15 54 -23 2 -1 0 0 0 }

{ -13 17 0 -2 0 -1 0 0 -46 -10 -10 4 -1 1 0 0 }

{ 4 51 -3 -6 -1 -1 0 0 -20 6 -34 9 -2 2 -1 0 }

{ -1 -4 -68 35 -5 5 -2 1 0 35 43 -4 -6 1 -1 0 }

{ -6 -37 -18 -5 2 -2 1 -1 6 -6 -7 25 -6 4 -1 1 }

{ -4 -7 -26 -6 -10 6 -4 1 3 8 14 -18 15 -5 2 -1 }

{ 1 24 3 5 -1 1 0 0 -3 12 6 -10 1 -1 0 0 }

{ 1 4 0 33 -7 5 -2 1 0 -9 53 -22 3 -1 0 0 }

},

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol32to47[ m − 32 ][ n ] with m = 32..47, n = 0..15

lowFreqTransMatrixCol32to47 =

{

{ 3 2 -1 0 -2 -1 0 0 1 1 0 0 -1 0 0 0 }

{ 12 3 -4 0 -3 -2 1 0 4 0 0 0 -1 0 0 0 }

{ 31 -5 -8 3 -14 0 5 -1 6 1 -3 0 -4 -1 1 0 }

{ -19 2 0 0 5 1 1 0 -2 0 -1 0 1 0 0 0 }

{ -53 34 6 -5 30 -7 -11 3 -11 -2 5 1 4 2 -1 -1 }

{ 49 7 2 -6 -23 -3 -2 2 9 4 0 0 -2 -1 -1 0 }

{ -11 32 -8 -7 27 -12 -6 6 -13 0 4 -3 3 -1 -2 1 }

{ -23 40 -2 5 43 -11 -8 -1 -18 -4 5 2 4 3 0 -1 }

{ -42 -25 4 6 34 8 2 -2 -15 -1 0 -1 3 2 0 1 }

{ -80 -27 20 -4 -66 23 -2 -2 20 -3 -2 3 -14 2 3 -1 }

{ 16 -52 28 1 59 15 -8 -5 -28 -7 2 2 10 3 0 -1 }

{ -14 -38 -12 -10 9 5 7 6 -9 7 -4 -3 4 -4 0 3 }

{ 16 10 55 -24 15 46 -52 1 35 -43 10 12 -23 13 5 -8 }

{ -2 -4 -1 13 0 2 -4 -3 3 -1 2 1 -2 0 -2 -1 }

{ -9 -1 -25 10 45 -11 18 2 86 1 -13 -4 -65 -6 7 2 }

{ 4 -27 -2 -9 5 36 -13 5 -7 -17 1 2 4 6 4 -1 }

},

* Otherwise, if nTrS is equal to 48, lfnstTrSetIdx is equal to 3, and lfnst\_idx is equal to 1 the following applies:

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol0to15[ m ][ n ] with m = 0..15, n = 0..15

lowFreqTransMatrixCol0to15 =

{

{ -115 37 9 2 2 1 1 0 10 -29 8 0 1 0 1 0 }

{ 15 51 -18 0 -3 0 -1 0 -95 7 34 -3 5 -1 2 0 }

{ 29 -22 16 -6 3 -2 1 -1 -4 -80 12 15 0 3 0 1 }

{ -36 -98 25 5 4 1 2 1 -59 11 -17 1 1 1 0 0 }

{ -6 18 3 -3 -1 0 0 0 -50 -5 -38 12 0 2 0 1 }

{ 4 15 52 -13 5 -3 2 -1 -17 -45 16 24 -2 4 -1 2 }

{ -20 -7 -43 4 0 1 -1 1 -7 35 0 12 -4 1 -1 0 }

{ 4 29 1 26 -5 4 -2 1 -17 -7 -73 6 6 2 1 1 }

{ 12 13 10 2 -1 3 -1 1 17 -2 -46 12 7 0 2 0 }

{ 5 20 90 -17 4 -3 2 -1 6 66 8 28 -7 3 -1 1 }

{ -3 -4 -34 -12 2 -1 -1 0 5 25 11 43 -10 4 -2 1 }

{ -1 -3 2 19 -2 4 -1 2 9 3 -35 22 11 1 2 0 }

{ 10 -4 -6 12 5 1 1 0 11 -9 -12 -2 -7 0 -1 0 }

{ 4 6 14 53 -4 4 0 2 0 -1 -20 -13 3 2 -1 1 }

{ 2 9 13 37 19 6 2 2 -9 -3 -9 -28 -20 -4 -3 -1 }

{ 3 -3 12 84 -12 8 -2 3 6 13 50 -1 45 1 7 0 }

},

lowFreqTransMatrix [ m ][ n ] = lowFreqTransMatrixCol16to31[ m − 16 ][ n ] with m = 16..31, n = 0..15

lowFreqTransMatrixCol16to31 =

{

{ 23 -8 -8 1 -1 0 0 0 3 3 -2 -1 0 0 0 0 }

{ 23 -47 1 6 0 1 0 1 8 5 -12 0 -1 0 0 0 }

{ 45 7 -59 7 -2 1 -1 0 -15 41 -3 -16 2 -3 0 -1 }

{ 6 -13 7 -3 0 0 0 0 14 -4 -14 3 -1 0 0 0 }

{ 3 67 -7 -40 3 -6 1 -3 -12 -13 65 -3 -10 0 -1 0 }

{ -87 -8 -14 7 8 1 2 0 23 -35 -6 -3 1 1 0 0 }

{ -51 -2 -57 5 15 0 4 0 7 39 5 -55 1 -7 1 -3 }

{ -5 21 -3 5 -1 -3 0 -1 -11 2 -52 -3 27 -2 5 0 }

{ 16 -45 -9 -53 6 1 1 0 70 16 8 -4 -37 1 -7 0 }

{ 29 5 -19 12 9 -1 1 0 -10 14 -1 -13 7 0 1 0 }

{ 23 20 -40 12 21 -3 4 -1 25 -28 -10 5 8 6 0 2 }

{ -7 -65 -19 -22 11 4 2 1 -75 -18 3 -1 -10 2 0 1 }

{ 33 -10 -4 18 18 -4 4 -1 28 -72 1 -49 15 2 2 1 }

{ -3 1 -5 35 -16 -6 -1 -2 46 29 13 21 37 -5 4 -1 }

{ 1 18 9 28 24 6 2 2 -20 -5 -25 -33 -36 9 -2 2 }

{ -2 18 -22 -37 -13 14 0 3 1 -12 -3 2 -15 -8 1 -1 }

},

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol32to47[ m − 32 ][ n ] with m = 32..47, n = 0..15

lowFreqTransMatrixCol32to47 =

{

{ 4 0 0 -1 1 1 0 0 2 0 0 0 0 0 0 0 }

{ 3 -3 1 -1 2 1 -2 0 1 -1 0 0 1 1 -1 0 }

{ 1 0 7 -2 -3 6 1 -2 0 0 1 0 -1 2 0 -1 }

{ 2 8 -3 -5 2 0 0 0 0 3 0 -1 1 0 0 0 }

{ 9 -20 -5 22 -2 0 0 -1 2 -3 -2 3 -1 0 1 0 }

{ 2 5 -17 0 3 -1 -1 -5 0 1 -4 0 1 0 0 -2 }

{ 1 -10 41 2 4 -3 -2 3 -1 -2 7 1 1 -1 -1 0 }

{ 0 27 8 -58 2 -5 25 3 0 3 0 -5 0 -2 7 0 }

{ -12 29 3 21 4 0 5 -1 -3 4 1 4 2 0 1 0 }

{ 0 -6 13 -4 0 -4 1 5 0 -1 -1 1 0 -1 0 0 }

{ -4 21 -64 -8 -5 19 10 -48 3 -1 10 -3 0 4 3 -6 }

{ 2 -35 -27 4 1 8 -17 -19 3 0 3 -6 0 2 -1 -2 }

{ 56 -23 22 -1 4 -1 -15 26 6 4 -10 0 0 2 -3 2 }

{ -10 -53 -18 8 9 12 -41 -25 -2 2 13 -16 4 1 -5 1 }

{ -13 42 1 57 -22 -2 -25 -28 5 6 19 -12 -5 -3 -2 4 }

{ 19 14 -4 -12 -4 5 17 8 2 -4 -4 4 -2 2 1 0 }

},

* Otherwise, if nTrS is equal to 48, lfnstTrSetIdx is equal to 3, and lfnst\_idx is equal to 2 the following applies:

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol0to15[ m ][ n ] with m = 0..15, n = 0..15

lowFreqTransMatrixCol0to15 =

{

{ 109 -26 -8 -3 -2 -1 -1 0 -50 28 2 1 0 0 0 0 }

{ -39 31 -5 2 -1 1 0 0 -95 6 18 0 4 0 1 0 }

{ 29 -3 -2 -2 0 0 0 0 0 -41 9 0 2 0 1 0 }

{ 18 96 -23 2 -5 1 -2 0 -10 6 10 -2 1 -1 1 0 }

{ -29 -60 16 -2 3 -1 1 0 -52 9 -17 5 -2 1 -1 1 }

{ -23 -5 -15 5 -2 1 -1 1 2 79 -13 -4 -2 -1 -1 0 }

{ -7 -3 12 -3 3 -1 1 0 -31 -62 8 7 0 2 0 1 }

{ 1 -26 5 0 1 0 1 0 24 -3 43 -6 4 -2 1 -1 }

{ 11 14 6 -3 1 -1 1 0 10 -7 -9 3 -2 1 -1 0 }

{ -10 -11 -47 3 -4 1 -1 0 5 28 11 -2 -1 0 0 0 }

{ -8 -24 -99 11 -10 3 -4 1 -5 -36 19 -26 4 -5 1 -2 }

{ -5 1 -1 0 1 0 0 0 -10 -14 -6 8 0 1 0 0 }

{ 1 12 -20 21 -4 5 -2 2 -5 -2 -75 9 -1 2 -1 1 }

{ 2 -9 -18 8 -3 3 -1 1 3 -25 -62 -6 0 -2 0 -1 }

{ 4 9 39 18 0 2 0 1 -6 -16 -22 -37 5 -5 1 -2 }

{ -7 -2 15 -6 1 -1 1 -1 -11 -3 22 -14 0 -2 1 -1 }

},

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol16to31[ m − 16 ][ n ] with m = 16..31, n = 0..15

lowFreqTransMatrixCol16to31 =

{

{ -18 -8 6 0 1 0 1 0 6 -2 -3 0 0 0 0 0 }

{ 32 -49 5 1 1 0 0 0 27 -1 -14 2 -2 1 -1 0 }

{ 86 4 -33 2 -6 1 -2 0 -32 58 1 -7 0 -2 0 -1 }

{ -14 26 2 -4 1 -1 0 0 -43 -9 35 -2 4 -1 1 0 }

{ 13 56 -2 -9 0 -2 0 -1 -34 -18 41 0 3 0 1 0 }

{ -9 1 5 -1 1 0 0 0 -4 49 2 -14 1 -3 0 -1 }

{ -75 9 -45 5 -1 1 -1 0 14 35 0 -23 2 -5 1 -2 }

{ -7 -64 9 14 0 3 0 1 -12 -4 5 3 -1 1 0 0 }

{ 22 21 1 -21 2 -4 1 -2 92 1 53 0 -9 1 -2 0 }

{ -12 -2 -38 2 0 1 0 0 16 38 11 -16 -1 -3 0 -2 }

{ 0 25 41 5 -3 1 0 0 10 -5 -7 12 2 1 0 0 }

{ -17 -2 7 -5 3 -1 0 0 -16 13 3 31 -1 6 0 2 }

{ -1 -2 -16 -4 0 -1 0 0 -7 7 -31 0 3 0 0 0 }

{ -6 -61 14 -51 2 -6 0 -2 -19 0 40 -7 -17 0 -3 0 }

{ -5 15 63 9 -16 0 -3 0 18 42 -18 27 15 1 3 1 }

{ -18 -7 30 -9 -4 0 -1 0 -35 23 23 10 -17 1 -3 0 }

},

lowFreqTransMatrix[ m ][ n ] = lowFreqTransMatrixCol32to47[ m − 32 ][ n ] with m = 32..47, n = 0..15

lowFreqTransMatrixCol32to47 =

{

{ -3 2 1 -1 0 0 0 0 -2 0 0 0 0 0 0 0 }

{ 3 5 -3 -2 4 1 -1 -1 2 0 0 0 2 0 0 0 }

{ -14 -8 20 0 -2 -3 0 4 -1 -1 0 0 -1 1 0 0 }

{ 14 -40 1 10 2 1 -10 1 2 -4 -1 -1 0 0 -1 0 }

{ 19 -36 -10 13 3 6 -14 -1 3 1 -1 -3 1 1 -1 -1 }

{ -31 -14 56 -1 13 -37 -4 20 -2 2 -10 0 2 -4 0 -1 }

{ 1 -8 32 -1 7 -12 -4 10 0 2 -6 -1 2 0 0 -2 }

{ 8 -59 -3 26 14 6 -58 6 -5 17 -7 -18 3 3 -1 -5 }

{ -21 -11 1 40 -5 -4 -24 5 -4 5 -6 -5 0 0 0 -3 }

{ 12 -9 -22 7 -8 60 4 -36 -6 -15 54 7 3 -7 -8 14 }

{ -1 1 9 -3 -3 -14 -3 12 2 4 -13 -2 -1 3 2 -4 }

{ -93 -15 -46 -3 23 -19 0 -47 8 4 8 3 2 3 0 0 }

{ 4 11 -12 4 -12 14 -50 -1 -8 32 -4 -54 2 0 30 -15 }

{ 13 -4 11 9 17 0 24 5 1 -12 4 28 0 0 -15 8 }

{ 12 -34 9 -24 4 28 -2 4 -11 -4 30 2 5 -13 -4 18 }

{ -19 53 6 48 -65 12 -12 11 -8 -16 10 -21 -2 -12 6 2 }

},

#### Transformation process

Inputs to this process are:

* a variable nTbS specifying the horizontal sample size of transformed samples,
* a variable nonZeroS specifying the horizontal sample size of non-zero scaled transform coefficients,
* a list of scaled transform coefficients x[ j ] with j = 0..nonZeroS − 1,
* a transform kernel type variable trType.

Output of this process is the list of transformed samples y[ i ] with i = 0..nTbS − 1.

The transformation matrix derivation process as specified in clause 8.7.4.5 in invoked with the transform size nTbS and the transform kernel Type trType as inputs, and the transformation maxtrix transMatrix as output.

Depending on the value of trType, the following applies:, 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]= with i = 0..nTbS − 1 (1198)

* Otherwise (trType is equal to 1 or trType is equal to 2), the following transform matrix multiplication applies:

y[i]= with i = 0..nTbS − 1 (1199)

#### Transformation matrix derivation process

Inputs to this process are:

* a variable nTbS specifying the horizontal sample size of scaled transform coefficients,
* the transformation kernel type trType.

Output of this process is the transformation matrix transMatrix.

The transformation matrix transMatrix is derived based on trType and nTbs as follows:

* If trType is equal to 0, the following applies:

transMatrix[ m ][ n ] = transMatrixCol0to15[ m ][ n ] with m = 0..15, n = 0..63 (1200)

transMatrixCol0to15 = (1201)

{

{ 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 (1202)

transMatrixCol16to31 = (1203)

{

{ 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 }

{ 62 59 56 52 48 44 41 37 33 28 24 20 15 11 7 2 }

{ −4 −13 −22 −31 −38 −46 −54 −61 −67 −73 −78 −82 −85 −88 −90 −90 }

{ −69 −77 −83 −87 −90 −91 −90 −86 −81 −73 −65 −56 −44 −33 −20 −7 }

{ −90 −87 −80 −70 −57 −43 −25 −9 9 25 43 57 70 80 87 90 }

{ −56 −37 −15 7 28 48 65 79 87 91 88 81 69 52 33 11 }

{ 13 38 61 78 88 90 85 73 54 31 4 −22 −46 −67 −82 −90 }

{ 73 87 90 83 65 41 11 −20 −48 −71 −86 −91 −84 −69 −44 −15 }

{ 89 75 50 18 −18 −50 −75 −89 −89 −75 −50 −18 18 50 75 89 }

{ 48 11 −28 −62 −84 −90 −79 −52 −15 24 59 83 91 81 56 20 }

{ −22 −61 −85 −90 −73 −38 4 46 78 90 82 54 13 −31 −67 −88 }

{ −79 −91 −77 −41 7 52 83 90 71 33 −15 −59 −86 −88 −65 −24 }

{ −87 −57 −9 43 80 90 70 25 −25 −70 −90 −80 −43 9 57 87 }

{ −41 15 65 90 79 37 −20 −69 −90 −77 −33 24 71 91 73 28 }

{ 31 78 90 61 4 −54 −88 −82 −38 22 73 90 67 13 −46 −85 }

{ 83 86 44 −20 −73 −90 −59 2 62 90 71 15 −48 −87 −81 −33 }

{ 83 36 −36 −83 −83 −36 36 83 83 36 −36 −83 −83 −36 36 83 }

{ 33 −41 −87 −77 −15 56 90 65 −2 −69 −90 −52 20 79 86 37 }

{ −38 −88 −73 −4 67 90 46 −31 −85 −78 −13 61 90 54 −22 −82 }

{ −86 −73 −2 71 87 33 −48 −90 −59 20 83 79 11 −65 −90 −41 }

{ −80 −9 70 87 25 −57 −90 −43 43 90 57 −25 −87 −70 9 80 }

{ −24 62 88 28 −59 −90 −33 56 90 37 −52 −90 −41 48 91 44 }

{ 46 90 38 −54 −90 −31 61 88 22 −67 −85 −13 73 82 4 −78 }

{ 88 56 −41 −91 −37 59 87 15 −73 −79 7 84 65 −28 −90 −48 }

{ 75 −18 −89 −50 50 89 18 −75 −75 18 89 50 −50 −89 −18 75 }

{ 15 −79 −69 33 91 28 −71 −77 20 90 41 −62 −83 7 87 52 }

{ −54 −85 4 88 46 −61 −82 13 90 38 −67 −78 22 90 31 −73 }

{ −90 −33 73 69 −41 −88 −2 87 44 −65 −77 28 90 15 −83 −56 }

{ −70 43 87 −9 −90 −25 80 57 −57 −80 25 90 9 −87 −43 70 }

{ −7 88 33 −79 −56 62 73 −41 −86 15 91 11 −87 −37 77 59 }

{ 61 73 −46 −82 31 88 −13 −90 −4 90 22 −85 −38 78 54 −67 }

{ 91 7 −90 −15 88 24 −86 −33 83 41 −79 −48 73 56 −69 −62 }

{ 64 −64 −64 64 64 −64 −64 64 64 −64 −64 64 64 −64 −64 64 }

{ −2 −90 11 90 −20 −87 28 84 −37 −81 44 77 −52 −71 59 65 }

{ −67 −54 78 38 −85 −22 90 4 −90 13 88 −31 −82 46 73 −61 }

{ −90 20 84 −44 −71 65 52 −81 −28 90 2 −90 24 83 −48 −69 }

{ −57 80 25 −90 9 87 −43 −70 70 43 −87 −9 90 −25 −80 57 }

{ 11 84 −52 −59 81 20 −91 24 79 −62 −48 86 7 −90 37 71 }

{ 73 31 −90 22 78 −67 −38 90 −13 −82 61 46 −88 4 85 −54 }

{ 90 −44 −59 84 2 −86 56 48 −88 11 81 −65 −37 90 −24 −73 }

{ 50 −89 18 75 −75 −18 89 −50 −50 89 −18 −75 75 18 −89 50 }

{ −20 −71 81 2 −83 69 24 −90 52 44 −90 33 62 −86 11 77 }

{ −78 −4 82 −73 −13 85 −67 −22 88 −61 −31 90 −54 −38 90 −46 }

{ −87 65 20 −86 69 15 −84 71 11 −83 73 7 −81 77 2 −79 }

{ −43 90 −57 −25 87 −70 −9 80 −80 9 70 −87 25 57 −90 43 }

{ 28 52 −91 56 24 −84 77 −7 −69 88 −37 −44 90 −62 −15 81 }

{ 82 −22 −54 90 −61 −13 78 −85 31 46 −90 67 4 −73 88 −38 }

{ 84 −81 24 48 −90 71 −7 −62 91 −59 −11 73 −88 44 28 −83 }

{ 36 −83 83 −36 −36 83 −83 36 36 −83 83 −36 −36 83 −83 36 }

{ −37 −28 79 −88 52 11 −69 91 −65 7 56 −90 77 −24 −41 84 }

{ −85 46 13 −67 90 −73 22 38 −82 88 −54 −4 61 −90 78 −31 }

{ −81 90 −62 11 44 −83 88 −59 7 48 −84 87 −56 2 52 −86 }

{ −25 70 −90 80 −43 −9 57 −87 87 −57 9 43 −80 90 −70 25 }

{ 44 2 −48 81 −90 73 −37 −11 56 −84 90 −69 28 20 −62 87 }

{ 88 −67 31 13 −54 82 −90 78 −46 4 38 −73 90 −85 61 −22 }

{ 77 −90 86 −65 33 7 −44 73 −90 87 −69 37 2 −41 71 −88 }

{ 18 −50 75 −89 89 −75 50 −18 −18 50 −75 89 −89 75 −50 18 }

{ −52 24 7 −37 62 −81 90 −88 77 −56 28 2 −33 59 −79 90 }

{ −90 82 −67 46 −22 −4 31 −54 73 −85 90 −88 78 −61 38 −13 }

{ −71 83 −90 90 −86 77 −62 44 −24 2 20 −41 59 −73 84 −90 }

{ −9 25 −43 57 −70 80 −87 90 −90 87 −80 70 −57 43 −25 9 }

{ 59 −48 37 −24 11 2 −15 28 −41 52 −62 71 −79 84 −88 90 }

{ 90 −90 88 −85 82 −78 73 −67 61 −54 46 −38 31 −22 13 −4 }

{ 65 −69 71 −73 77 −79 81 −83 84 −86 87 −88 90 −90 90 −91 }

},

transMatrix[ m ][ n ] = ( n & 1 ? −1 : 1 ) \* transMatrixCol16to31[47 −m ][ n ] (1204)  
 with m =32..47, n = 0..63

transMatrix[ m ][ n ] = ( n & 1 ? −1 : 1 ) \* transMatrixCol0to15[63 −m ][ n ] (1205)  
 with m =48..63, n = 0..63

* Otherwise, if trType is equal to 1 and nTbs is equal to 4, the following applies:

transMatrix[ m ][ n ] = (1206)

{

{ 29 55 74 84 }

{ 74 74 0 −74 }

{ 84 −29 −74 55 }

{ 55 −84 74 −29 }

},

* Otherwise, if trType is equal to 1 and nTbs is equal to 8, the following applies:

transMatrix[ m ][ n ] = (1207)

{

{ 17 32 46 60 71 78 85 86 }

{ 46 78 86 71 32 −17 −60 −85 }

{ 71 85 32 −46 −86 −60 17 78 }

{ 85 46 −60 −78 17 86 32 −71 }

{ 86 −17 −85 32 78 −46 −71 60 }

{ 78 −71 −17 85 −60 −32 86 −46 }

{ 60 −86 71 −17 −46 85 −78 32 }

{ 32 −60 78 −86 85 −71 46 −17 }

},

* Otherwise, if trType is equal to 1 and nTbs is equal to 16, the following applies:

transMatrix[ m ][ n ] = (1208)

{

{ 8 17 25 33 40 48 55 62 68 73 77 81 85 87 88 88 }

{ 25 48 68 81 88 88 81 68 48 25 0 −25 −48 −68 −81 −88 }

{ 40 73 88 85 62 25 −17 −55 −81 −88 −77 −48 −8 33 68 87 }

{ 55 87 81 40 −17 −68 −88 −73 −25 33 77 88 62 8 −48 −85 }

{ 68 88 48 −25 −81 −81 −25 48 88 68 0 −68 −88 −48 25 81 }

{ 77 77 0 −77 −77 0 77 77 0 −77 −77 0 77 77 0 −77 }

{ 85 55 −48 −87 −8 81 62 −40 −88 −17 77 68 −33 −88 −25 73 }

{ 88 25 −81 −48 68 68 −48 −81 25 88 0 −88 −25 81 48 −68 }

{ 88 −8 −88 17 87 −25 −85 33 81 −40 −77 48 73 −55 −68 62 }

{ 87 −40 −68 73 33 −88 8 85 −48 −62 77 25 −88 17 81 −55 }

{ 81 −68 −25 88 −48 −48 88 −25 −68 81 0 −81 68 25 −88 48 }

{ 73 −85 25 55 −88 48 33 −87 68 8 −77 81 −17 −62 88 −40 }

{ 62 −88 68 −8 −55 88 −73 17 48 −87 77 −25 −40 85 −81 33 }

{ 48 −81 88 −68 25 25 −68 88 −81 48 0 −48 81 −88 68 −25 }

{ 33 −62 81 −88 85 −68 40 −8 −25 55 −77 88 −87 73 −48 17 }

{ 17 −33 48 −62 73 −81 87 −88 88 −85 77 −68 55 −40 25 −8 }

},

* Otherwise, if trType is equal to 1 and nTbs is equal to 32, the following applies:

transMatrix[ m ][ n ] = transMatrixCol0to15[ m ][ n ] with m = 0..15, n = 0..15 (1209)

transMatrixCol0to15 = (1210)

{

{ 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 (1211)

transMatrixCol16to31 = (1212)

{

{ 66 68 72 74 77 78 80 82 84 85 86 87 88 89 90 90 }

{ 56 46 34 21 9 −4 −17 −30 −42 −53 −63 −72 −78 −84 −87 −90 }

{ −74 −84 −89 −89 −84 −74 −60 −42 −21 0 21 42 60 74 84 89 }

{ −46 −17 13 42 66 82 90 86 74 53 26 −4 −34 −60 −78 −88 }

{ 80 90 82 60 26 −13 −50 −77 −89 −85 −66 −34 4 42 72 87 }

{ 34 −13 −56 −84 −88 −68 −30 17 60 85 87 66 26 −21 −63 −86 }

{ −85 −85 −53 0 53 85 85 53 0 −53 −85 −85 −53 0 53 85 }

{ −21 42 84 84 42 −21 −74 −89 −60 0 60 89 74 21 −42 −84 }

{ 88 72 9 −60 −90 −63 4 68 89 53 −17 −77 −86 −42 30 82 }

{ 9 −66 −88 −42 38 87 68 −4 −74 −85 −30 50 90 60 −17 −80 }

{ −90 −50 38 89 56 −30 −87 −63 21 85 68 −13 −82 −74 4 78 }

{ 4 82 68 −21 −87 −56 38 90 42 −53 −88 −26 66 84 9 −77 }

{ 89 21 −74 −74 21 89 42 −60 −84 0 84 60 −42 −89 −21 74 }

{ −17 −90 −30 74 68 −38 −88 −9 84 53 −56 −82 13 89 34 −72 }

{ −86 9 90 21 −82 −50 66 72 −42 −85 13 90 17 −84 −46 68 }

{ 30 86 −17 −89 4 90 9 −88 −21 85 34 −80 −46 74 56 −66 }

},

* Otherwise, if trType is equal to 2 and nTbs is equal to 4, the following applies:

transMatrix[ m ][ n ] = (1213)

{

{ 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 ] = (1214)

{

{ 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 ] = (1215)

{

{ 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 (1216)

transMatrixCol0to15 = (1217)

{

{ 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 (1218)

transMatrixCol16to31 = (1219)

{

{ 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 }

},

#### Residual modification process for blocks using colour space conversion

Inputs to this process are:

– a variable nTbW specifying the block width,

– a variable nTbH specifying the block height,

– an (nTbW)x(nTbH) array of luma residual samples rY with elements rY[ x ][ y ],

– an (nTbW)x(nTbH) array of chroma residual samples rCb with elements rCb[ x ][ y ],

– an (nTbW)x(nTbH) array of chroma residual samples rCr with elements rCr[ x ][ y ].

Outputs of this process are:

– a modified (nTbW)x(nTbH) array rY of luma residual samples,

– a modified (nTbW)x(nTbH) array rCb of chroma residual samples,

– a modified (nTbW)x(nTbH) array rCr of chroma residual samples.

The (nTbW)x(nTbH) arrays of residual samples rY, rCb and rCr are modified as follows:

rY[ x ][ y ] = Clip3( − ( 1  <<  ( BitDepth + 1 ) ), ( 1  <<  ( BitDepth + 1 ) ) − 1, rY[ x ][ y ] ) (1220)

rCb[ x ][ y ] = Clip3( − ( 1  <<  ( BitDepth + 1 ) ), ( 1  <<  ( BitDepth + 1 ) ) − 1, rCb[ x ][ y ] ) (1221)

rCr[ x ][ y ] = Clip3( − ( 1  <<  ( BitDepth + 1 ) ), ( 1  <<  ( BitDepth + 1 ) ) − 1, rCr[ x ][ y ] ) (1222)

tmp = rY[ x ][ y ] − ( rCb[ x ][ y ]  >>  1 ) (1223)

rY[ x ][ y ] = tmp + rCb[ x ][ y ] (1224)

rCb[ x ][ y ] = tmp − ( rCr[ x ][ y ]  >>  1 ) (1225)

rCr[ x ][ y ]  +=  rCb[ x ][ y ] (1226)

### Picture reconstruction process

#### General

Inputs to this process are:

* a location ( xCurr, yCurr ) specifying the top-left sample of the current block relative to the top‑left sample of the current picture component,
* the variables nCurrSw and nCurrSh specifying the width and height, respectively, of the current block,
* a variable cIdx specifying the colour component of the current block,
* an (nCurrSw) x (nCurrSh) array predSamples specifying the predicted samples of the current block,
* an (nCurrSw) x (nCurrSh) array resSamples specifying the residual samples of the current block.

Output of this process is a reconstructed picture sample array recSamples.

Depending on the value of the colour component cIdx, the following assignments are made:

* If cIdx is equal to 0, recSamples corresponds to the reconstructed picture sample array SL.
* Otherwise, if cIdx is equal to 1, tuCbfChroma is set equal to tu\_cb\_coded\_flag[ xCurr ][ yCurr ], recSamples corresponds to the reconstructed chroma sample array SCb.
* Otherwise (cIdx is equal to 2), tuCbfChroma is set equal to tu\_cr\_coded\_flag[ xCurr ][ yCurr ], recSamples corresponds to the reconstructed chroma sample array SCr.

Depending on the value of sh\_lmcs\_enabled\_flag, the following applies:

* If sh\_lmcs\_enabled\_flag is equal to 0, the (nCurrSw)x(nCurrSh) block of the reconstructed samples recSamples at location ( xCurr, yCurr ) is derived as follows for i = 0..nCurrSw − 1, j = 0..nCurrSh − 1:

recSamples[ xCurr + i ][ yCurr + j ] = Clip1( predSamples[ i ][ j ] + resSamples[ i ][ j ] ) (1227)

* Otherwise (sh\_lmcs\_enabled\_flag is equal to 1), the following applies:
* If cIdx is equal to 0, the following applies:
* The picture reconstruction with mapping process for luma samples as specified in clause 8.7.5.2 is invoked with the luma location ( xCurr, yCurr ), the block width nCurrSw and height nCurrSh, the predicted luma sample array predSamples, and the residual luma sample array resSamples as inputs, and the output is the reconstructed luma sample array recSamples.
* Otherwise (cIdx is greater than 0), the picture reconstruction with luma dependent chroma residual scaling process for chroma samples as specified in clause 8.7.5.3 is invoked with the chroma location ( xCurr, yCurr ), the transform block width nCurrSw and height nCurrSh, the coded block flag of the current chroma transform block tuCbfChroma, the predicted chroma sample array predSamples, and the residual chroma sample array resSamples as inputs, and the output is the reconstructed chroma sample array recSamples.

The following assignments are made for i = 0..nCurrSw − 1, j = 0..nCurrSh − 1:

xVb = ( xCurr + i ) % ( ( cIdx = = 0 ) ? IbcBufWidthY : IbcBufWidthC ) (1228)

yVb = ( yCurr + j ) % ( ( cIdx = = 0 ) ? CtbSizeY : ( CtbSizeY / subHeightC ) ) (1229)

IbcVirBuf[ cIdx ][ xVb ][ yVb ] = recSamples[ xCurr + i ][ yCurr + j ] (1230)

IsAvailable[ cIdx ][ xCurr + i ][ yCurr + j ] = TRUE (1231)

#### 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.

Outputs of this process is a reconstructed luma picture sample array recSamples.

The (nCurrSw)x(nCurrSh) array of mapped predicted luma samples predMapSamples is derived as follows:

* If one of the following conditions is true, predMapSamples[ i ][ j ] is set equal to predSamples[ i ][ j ] for i = 0..nCurrSw − 1, j = 0..nCurrSh − 1:
* CuPredMode[ 0 ][ xCurr ][ yCurr ] is equal to MODE\_INTRA.
* CuPredMode[ 0 ][ xCurr ][ yCurr ] is equal to MODE\_IBC.
* CuPredMode[ 0 ][ xCurr ][ yCurr ] is equal to MODE\_PLT.
* CuPredMode[ 0 ][ xCurr ][ yCurr ] is equal to MODE\_INTER and ciip\_flag[ xCurr ][ yCurr ] is equal to 1.
* Otherwise (CuPredMode[ 0 ][ xCurr ][ yCurr ] is equal to MODE\_INTER and ciip\_flag[ xCurr ][ yCurr ] is equal to 0), the following applies:

idxY = predSamples[ i ][ j ] >> Log2( OrgCW )  
predMapSamples[ i ][ j ] =  LmcsPivot[ idxY ]   
  + ( ScaleCoeff[ idxY ] \* ( predSamples[ i ][ j ] − InputPivot[ idxY ] ) + ( 1 << 10 ) ) >> 11  (1232)  
 with i = 0..nCurrSw − 1, j = 0..nCurrSh − 1

The reconstructed luma picture sample recSamples is derived as follows:

recSamples[ xCurr + i ][ yCurr + j ] = Clip1( predMapSamples[ i ][ j ]+ resSamples[ i ][ j ] ] ) (1233)  
 with i = 0..nCurrSw − 1, j = 0..nCurrSh − 1

#### Picture reconstruction with luma dependent chroma residual scaling process for chroma samples

Inputs to this process are:

* a chroma location ( xCurr, yCurr ) of the top-left chroma sample of the current chroma transform block relative to the top-left chroma sample of the current picture,
* a variable nCurrSw specifying the chroma transform block width,
* a variable nCurrSh specifying the chroma transform block height,
* a variable tuCbfChroma specifying the coded block flag of the current chroma transform block,
* an (nCurrSw)x(nCurrSh) array predSamples specifying the chroma prediction samples of the current block,
* an (nCurrSw)x(nCurrSh) array resSamples specifying the chroma residual samples of the current block,

Output of this process is a reconstructed chroma picture sample array recSamples.

The variable sizeY is set equal to Min( CtbSizeY, 64 ).

The reconstructed chroma picture sample recSamples is derived as follows for i = 0..nCurrSw − 1, j = 0..nCurrSh − 1:

* If one or more of the following conditions are true, recSamples[ xCurr + i ][ yCurr + j ] is set equal to Clip1( predSamples[ i ][ j ] + resSamples[ i ][ j ] ):
* ph\_chroma\_residual\_scale\_flag is equal to 0.
* sh\_lmcs\_enabled\_flag is equal to 0.
* nCurrSw \* nCurrSh is less than or euqal to 4.
* tu\_cb\_coded\_flag [ xCurr ][ yCurr ] is equal to 0 and tu\_cr\_coded\_flag [ xCurr ][ yCurr ] is equal to 0.
* Otherwise, the following applies:
* The current luma location ( xCurrY, yCurrY ) is derived as follows:

( xCurrY, yCurrY ) = ( xCurr \* SubWidthC, yCurr \* SubHeightC ) (1234)

* The luma location ( xCuCb, yCuCb ) is specified as the top-left luma sample location of the coding unit that contains the luma sample at ( xCurrY / sizeY \* sizeY, yCurrY / sizeY \* sizeY ).
* The variables availL and availT are derived as follows:
* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location ( xCurr, yCurr ) set equal to ( xCuCb, yCuCb ), the neighbouring luma location ( xNbY, yNbY ) set equal to ( xCuCb − 1, yCuCb ), checkPredModeY set equal to FALSE, and cIdx set equal to 0 as inputs, and the output is assigned to availL.
* The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location ( xCurr, yCurr ) set equal to ( xCuCb, yCuCb ), the neighbouring luma location ( xNbY, yNbY ) set equal to ( xCuCb, yCuCb − 1 ), checkPredModeY set equal to FALSE, and cIdx set equal to 0 as inputs, and the output is assigned to availT.
* The variable currPic specifies the array of reconstructed luma samples in the current picture.
* For the derivation of the variable varScale the following ordered steps apply:

1. The variable invAvgLuma is derived as follows:

* The array recLuma[ i ] with i=0..( 2 \* sizeY − 1 ) and the variable cnt are derived as follows:
* The variable cnt is set equal to 0.
* When availL is equal to TRUE, the array recLuma[ i ] with i = 0..sizeY − 1 is set equal to currPic[ xCuCb − 1 ][ Min( yCuCb + i, pps\_pic\_height\_in\_luma\_samples − 1 ) ] with i = 0..sizeY − 1, and cnt is set equal to sizeY.
* When availT is equal to TRUE, the array recLuma[ cnt + i ] with i = 0..sizeY − 1 is set equal to currPic[ Min( xCuCb + i, pps\_pic\_width\_in\_luma\_samples − 1 ) ][ yCuCb − 1 ] with i = 0..sizeY − 1, and cnt is set equal to ( cnt + sizeY ).
* The variable invAvgLuma is derived as follows:
* If cnt is greater than 0, the following applies:

invAvgLuma = ( + ( cnt  >>  1 ) ) >> Log2( cnt ) (1235)

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

invAvgLuma = 1  <<  ( BitDepth − 1 ) (1236)

1. The variable idxYInv is derived by invoking the identification of piece-wise function index process for a luma sample as specified in clause 8.8.2.3 with the variable lumaSample set equal to invAvgLuma as the input and idxYInv as the output.
2. The variable varScale is derived as follows:

varScale = ChromaScaleCoeff[ idxYInv ] (1237)

* The reconstructed chroma picture sample array recSamples is derived as follows:
* If tuCbfChroma is equal to 1, the following applies:

resSamples[ i ][ j ] = Clip3( −( 1 << BitDepth ), ( 1 << BitDepth ) − 1, resSamples[ i ][ j ] ) (1238)

recSamples[ xCurr + i ][ yCurr + j ] = Clip1( predSamples[ i ][ j ] +  (1239)  
 Sign( resSamples[ i ][ j ] ) \* ( ( Abs( resSamples[ i ][ j ] ) \* varScale + ( 1 << 10 ) ) >> 11 ) )

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

recSamples[ xCurr + i ][ yCurr + j ] = predSamples[ i ][ j ] (1240)

## In-loop filter process

### 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 sps\_lmcs\_enabled\_flag is equal to 1, the following applies:

* The picture inverse mapping process for luma samples as specified in clause 8.8.2.1 is invoked with the reconstructed luma sample array SL 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 SL (which represent the decoded picture).

1. For the deblocking filter, the following applies:

* The deblocking filter process as specified in clause 8.8.3.1 is invoked with the reconstructed picture sample array SL and, when ChromaArrayType is not equal to 0, the arrays SCb and SCr as inputs, and the modified reconstructed picture sample array S′L and, when ChromaArrayType is not equal to 0, the arrays S′Cb and S′Cr after deblocking as outputs.
* The array S′L and, when ChromaArrayType is not equal to 0, the arrays S′Cb and S′Cr are assigned to the array SL and, when ChromaArrayType is not equal to 0, the arrays SCb and SCr (which represent the decoded picture), respectively.

1. When sps\_sao\_enabled\_flag is equal to 1, the following applies:

* The sample adaptive offset process as specified in clause 8.8.4.1 is invoked with the reconstructed picture sample array SL and, when ChromaArrayType is not equal to 0, the arrays SCb and SCr as inputs, and the modified reconstructed picture sample array S′L and, when ChromaArrayType is not equal to 0, the arrays S′Cb and S′Cr after sample adaptive offset as outputs.
* The array S′L and, when ChromaArrayType is not equal to 0, the arrays S′Cb and S′Cr are assigned to the array SL and, when ChromaArrayType is not equal to 0, the arrays SCb and SCr (which represent the decoded picture), respectively.

1. When sps\_alf\_enabled\_flag is equal to 1, the following applies:

* The adaptive loop filter process as specified in clause 8.8.5.1 is invoked with the reconstructed picture sample array SL and, when ChromaArrayType is not equal to 0, the arrays SCb and SCr as inputs, and the modified reconstructed picture sample array S′L and, when ChromaArrayType is not equal to 0, the arrays S′Cb and S′Cr after adaptive loop filter as outputs.
* The array S′L and, when ChromaArrayType is not equal to 0, the arrays S′Cb and S′Cr are assigned to the array SL and, when ChromaArrayType is not equal to 0, the arrays SCb and SCr (which represent the decoded picture), respectively.

### Picture inverse mapping process for luma samples

#### General

Input to this process is a reconstructed picture luma sample array SL.

The output to this process is a modified reconstructed picture luma sample array SL.

The inverse mapping process for a luma sample SL[ x ][ y ] with x = 0..pps\_pic\_width\_in\_luma\_samples − 1, y = 0..pps\_pic\_height\_in\_luma\_samples − 1 is invoked as specified in clause 8.8.2.2 with the variable lumaSample set equal to SL[ x ][ y ] as the input and the output is assigned to the luma sample SL[ x ][ y ].

#### 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\_enabled\_flag of the slice that contains the luma sample lumaSample is equal to 1, the following ordered steps apply:

1. The variable idxYInv is derived by invoking the identification of piece-wise function index process for a luma sample as specified in clause 8.8.2.3 with lumaSample as the input and idxYInv as the output.
2. The variable invSample is derived as follows:

invSample = InputPivot[ idxYInv ] + ( InvScaleCoeff[ idxYInv ] \*  (1241)  
 ( lumaSample − LmcsPivot[ idxYInv ] ) + ( 1 << 10 ) ) >> 11

1. The inverse mapped luma sample invLumaSample is derived as follows:

invLumaSample = Clip1( invSample ) (1242)

* Otherwise, invLumaSample is set equal to lumaSample.

#### 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 identifing 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 ] ) (1243)  
 break  
}  
idxYInv = Min( idxYInv, 15 )

### Deblocking filter process

#### General

Inputs to this process are the reconstructed picture prior to deblocking, i.e., the array recPictureL and, when ChromaArrayType is not equal to 0, the arrays recPictureCb and recPictureCr.

Outputs of this process are the modified reconstructed picture after deblocking, i.e., the array recPictureL and, when ChromaArrayType is not equal to 0, the arrays recPictureCb and recPictureCr.

The vertical edges in a picture are filtered first. Then the horizontal edges in a picture are filtered with samples modified by the vertical edge filtering process as input. The vertical and horizontal edges in the CTBs of each CTU are processed separately on a coding unit basis. The vertical edges of the coding blocks in a coding unit are filtered starting with the edge on the left-hand side of the coding blocks proceeding through the edges towards the right-hand side of the coding blocks in their geometrical order. The horizontal edges of the coding blocks in a coding unit are filtered starting with the edge on the top of the coding blocks proceeding through the edges towards the bottom of the coding blocks in their geometrical order.

NOTE – Although the filtering process is specified on a picture basis in this Specification, the filtering process can 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 coding subblock edges and transform block edges of a picture, except the following types of edges:

* Edges that are at the boundary of the picture,
* Edges that coincide with the boundaries of a subpicture with subpicture index subpicIdx and sps\_loop\_filter\_across\_subpic\_enabled\_flag[ subpicIdx ] is equal to 0,
* Edges that coincide with the virtual boundaries of the picture when VirtualBoundariesPresentFlag is equal to 1,
* Edges that coincide with tile boundaries when pps\_loop\_filter\_across\_tiles\_enabled\_flag is equal to 0,
* Edges that coincide with slice boundaries when pps\_loop\_filter\_across\_slices\_enabled\_flag is equal to 0,
* Edges that coincide with upper or left boundaries of slices with sh\_deblocking\_filter\_disabled\_flag equal to 1,
* Edges within slices with sh\_deblocking\_filter\_disabled\_flag equal to 1,
* Edges that do not correspond to 4×4 sample grid boundaries of the luma component,
* Edges that do not correspond to 8×8 sample grid boundaries of the chroma component,
* Edges within the luma component for which both sides of the edge have intra\_bdpcm\_luma\_flag equal to 1,
* Edges within the chroma components for which both sides of the edge have intra\_bdpcm\_chroma\_flag equal to 1,
* Edges of chroma subblocks that are not edges of the associated transform unit.

The edge type, vertical or horizontal, is represented by the variable edgeType as specified in Table 42.

Table 42 – Name of association to edgeType

|  |  |
| --- | --- |
| edgeType | Name of edgeType |
| 0 (vertical edge) | EDGE\_VER |
| 1 (horizontal edge) | EDGE\_HOR |

When sh\_deblocking\_filter\_disabled\_flag of the current slice is equal to 0, the following applies:

* The variable treeType is set equal to DUAL\_TREE\_LUMA.
* The vertical edges are filtered by invoking the deblocking filter process for one direction as specified in clause 8.8.3.2 with the variable treeType, the reconstructed picture prior to deblocking, i.e., the array 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 clause 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 ChromaArrayType is not equal to 0, the following applies:
* The variable treeType is set equal to DUAL\_TREE\_CHROMA
* The vertical edges are filtered by invoking the deblocking filter process for one direction as specified in clause 8.8.3.2 with the variable treeType, the reconstructed picture prior to deblocking, i.e., the arrays 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 clause 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.

#### 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 ChromaArrayType 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 ChromaArrayType 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:

firstCompIdx = ( treeType = = DUAL\_TREE\_CHROMA ) ? 1 : 0 (1244)

lastCompIdx = ( treeType = = DUAL\_TREE\_LUMA | | ChromaArrayType = = 0 ) ? 0 : 2 (1245)

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 luma 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.

1. All elements of the two-dimensional (nCbW)x(nCbH) array edgeFlags, maxFilterLengthQs and maxFilterlengthPs are initialized to be equal to zero.
2. The derivation process of transform block boundary specified in clause 8.8.3.3 is invoked with the location ( xCb, yCb ), the coding block width nCbW, the coding block height nCbH, the variable cIdx, the variable filterEdgeFlag, the array edgeFlags, the maximum filter length arrays maxFilterLengthPs and maxFilterLengthQs, and the variable edgeType as inputs, and the modified array edgeFlags, the modified maximum filter length arrays maxFilterLengthPs and maxFilterLengthQs as outputs.
3. When cIdx is equal to 0, the derivation process of coding subblock boundary specified in clause 8.8.3.4 is invoked with the location ( xCb, yCb ), the coding block width nCbW, the coding block height nCbH, the array edgeFlags, the maximum filter length arrays maxFilterLengthPs and maxFilterLengthQs, and the variable edgeType as inputs, and the modified array edgeFlags, the modified maximum filter length arrays maxFilterLengthPs and maxFilterLengthQs as outputs.
4. 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 recPictureL.
* Otherwise, if cIdx is equal to 1, recPicture is set equal to the reconstructed chroma picture sample array prior to deblocking recPictureCb.
* Otherwise (cIdx is equal to 2), recPicture is set equal to the reconstructed chroma picture sample array prior to deblocking recPictureCr.

1. The derivation process of the boundary filtering strength specified in clause 8.8.3.5 is invoked with the picture sample array recPicture, the luma location ( xCb, yCb ), the coding block width nCbW, the coding block height nCbH, the variable edgeType, the variable cIdx, and the array edgeFlags as inputs, and an (nCbW)x(nCbH) array bS as output.
2. The edge filtering process for one direction is invoked for a coding block as specified in clause 8.8.3.6 with the variable edgeType, the variable cIdx, the reconstructed picture prior to deblocking recPicture, the location ( xCb, yCb ), the coding block width nCbW, the coding block height nCbH, and the arrays bS, maxFilterLengthPs, and maxFilterLengthQs, as inputs, and the modified reconstructed picture recPicture as output.

#### Derivation process of transform block boundary

Inputs to this process are:

* a location ( xCb, yCb ) specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,
* a variable nCbW specifying the width of the current coding block,
* a variable nCbH specifying the height of the current coding block,
* a variable cIdx specifying the colour component of the current coding block,
* a variable filterEdgeFlag,
* a two-dimensional (nCbW)x(nCbH) array edgeFlags,
* two-dimensional (nCbW)x(nCbH) arrays maxFilterLengthQs and maxFilterLengthPs,
* a variable edgeType specifying whether a vertical (EDGE\_VER) or a horizontal (EDGE\_HOR) edge is filtered.

Outputs of this process are:

* the modified two-dimensional (nCbW)x(nCbH) array edgeFlags,
* the modified two-dimensional (nCbW)x(nCbH) arrays maxFilterLengthQs, maxFilterLengthPs.

Depending on edgeType, the arrays edgeFlags, maxFilterLengthPs and maxFilterLengthQs are derived as follows:

* The variable gridSize is set as follows:

gridSize = cIdx  = =  0 ? 4 : 8 (1246)

* If edgeType is equal to EDGE\_VER, the following applies:
* The variable numEdges is set equal to Max( 1, nCbW / gridSize ).
* For xEdge = 0..numEdges − 1 and y = 0..nCbH − 1, the following applies:
* The horizontal position x inside the current coding block is set equal to xEdge \* gridSize.
* The value of edgeFlags[ x ][ y ] is derived as follows:
  + - If x is equal to 0, edgeFlags[ x ][ y ] is set equal to filterEdgeFlag.
    - Otherwise, if the location ( xCb + x , yCb + y ) is at a transform block edge, edgeFlags[ x ][ y ] is set equal to 1.
* When edgeFlags[ x ][ y ] is equal to 1,the following applies:
  + - If cIdx is equal to 0, the following applies:
    - The value of maxFilterLengthQs[ x ][ y ] is derived as follows:
    - If the width in luma samples of the transform block at luma location ( xCb + x, yCb + y ) is equal to or less than 4 or the width in luma samples of the transform block at luma location ( xCb + x − 1, yCb + y ) is equal to or less than 4, maxFilterLengthQs[ x ][ y ] is set equal to 1.
    - Otherwise, if the width in luma samples of the transform block at luma location ( xCb + x, yCb + y ) is equal to or greater than 32, maxFilterLengthQs[ x ][ y ] is set equal to 7.
    - Otherwise, maxFilterLengthQs[ x ][ y ] is set equal to 3.
    - The value of maxFilterLengthPs[ x ][ y ] is derived as follows:
    - If the width in luma samples of the transform block at luma location ( xCb + x, yCb + y ) is equal to or less than 4 or the width in luma samples of the transform block at luma location ( xCb + x − 1, yCb + y ) is equal to or less than 4 , maxFilterLengthPs[ x ][ y ] is set equal to 1.
    - Otherwise, if the width in luma samples of the transform block at luma location ( xCb + x − 1, yCb + y ) is equal to or greater than 32, maxFilterLengthPs[ x ][ y ] is set equal to 7.
    - Otherwise, maxFilterLengthPs[ x ][ y ] is set equal to 3.
    - Otherwise (cIdx is not equal to 0), the values of maxFilterLengthPs[ x][ y ] and maxFilterLengthQs[ x ][ y ] are derived as follows:
    - If the width in chroma samples of the transform block at chroma location ( xCb + x, yCb + y ) and the width at chroma location ( xCb + x − 1,  yCb + y ) are both equal to or greater than 8, maxFilterLengthPs[ x ][ y ] and maxFilterLengthQs[ x ][ y ] are set equal to 3.
    - Otherwise, maxFilterLengthPs[ x ][ y ] and maxFilterLengthQs[ x ][ y ] are set equal to 1.
* Otherwise (edgeType is equal to EDGE\_HOR), the following applies:
* The variable numEdges is set equal to Max( 1, nCbH / gridSize ).
* For yEdge = 0..numEdges − 1 and x = 0..nCbW − 1, the following applies:
* The vertical position y inside the current coding block is set equal to yEdge \* gridSize.
* The value of edgeFlags[ x ][ y ] is derived as follows:
  + - If y is equal to 0, edgeFlags[ x ][ y ] is set equal to filterEdgeFlag.
    - Otherwise, if the location ( xCb + x , yCb + y ) is at a transform block edge, edgeFlags[ x ][ y ] is set equal to 1.
* When edgeFlags[ x ][ y ] is equal to 1,the following applies:
  + - If cIdx is equal to 0, the following applies:
    - The value of maxFilterLengthQs[ x ][ y ] is derived as follows:
    - If the height in luma samples of the transform block at luma location ( xCb + x, yCb + y ) is equal to or less than 4 or the height in luma samples of the transform block at luma location ( xCb + x, yCb + y − 1 ) is equal to or less than 4, maxFilterLengthQs[ x ][ y ] is set equal to 1.
    - Otherwise, if the height in luma samples of the transform block at luma location ( xCb + x, yCb + y ) is equal to or greater than 32, maxFilterLengthQs[ x ][ y ] is set equal to 7.
    - Otherwise, maxFilterLengthQs[ x ][ y ] is set equal to 3.
    - The value of maxFilterLengthPs[ x ][ y ] is derived as follows:
    - If the height in luma samples of the transform block at luma location ( xCb + x, yCb + y ) is equal to or less than 4 or the height in luma samples of the transform block at luma location ( xCb + x, yCb + y − 1 ) is equal to or less than 4, maxFilterLengthPs[ x ][ y ] is set equal to 1.
    - Otherwise, if the height in luma samples of the transform block at luma location ( xCb + x, yCb + y − 1 ) is equal to or greater than 32, maxFilterLengthPs[ x ][ y ] is set equal to 7.
    - Otherwise, maxFilterLengthPs[ x ][ y ] is set equal to 3.
    - Otherwise (cIdx is not equal to 0), the values of maxFilterLengthPs[ x][ y ] and maxFilterLengthQs[ x ][ y ] are derived as follows:
    - If the height in chroma samples of the transform block at chroma location ( xCb + x, yCb + y ) and the height in chroma samples of the transform block at chroma location ( xCb + x,  yCb + y − 1 ) are both equal to or greater than 8, the following applies:
    - If ( yCb + y ) % CtbHeightC is greater than 0, i.e. the horizontal edge do not overlap with the upper chroma CTB boundary, both maxFilterLengthPs[ x ][ y ] and maxFilterLengthQs[ x ][ y ] are set equal to 3
    - Otherwise ( ( yCb + y ) % CtbHeightC is equal to 0, i.e. the horizontal edge overlaps with the upper chroma CTB boundary), maxFilterLengthPs[ x ][ y ] is set equal to 1 and maxFilterLengthQs[ x ][ y ] is set equal to 3.
    - Otherwise, maxFilterLengthPs[ x ][ y ] and maxFilterLengthQs[ x ][ y ] are set equal to 1.

#### Derivation process of coding subblock boundary

Inputs to this process are:

* a location ( xCb, yCb ) specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,
* a variable nCbW specifying the width of the current coding block,
* a variable nCbH specifying the height of the current coding block,
* a two-dimensional (nCbW)x(nCbH) array edgeFlags,
* two-dimensional (nCbW)x(nCbH) arrays maxFilterLengthQs and maxFilterLengthPs,
* a variable edgeType specifying whether a vertical (EDGE\_VER) or a horizontal (EDGE\_HOR) edge is filtered.

Outputs of this process are:

* the modified two-dimensional (nCbW)x(nCbH) array edgeFlags,
* the modified two-dimensional (nCbW)x(nCbH) arrays maxFilterLengthQs and maxFilterLengthPs.

The number of coding 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 edgeFlags.
* For xEdge = 0..min( ( nCbW / 8 ) − 1, numSbX − 1), y = 0..nCbH − 1:
* The horizontal position x inside the current coding block is set equal to xEdge \*sbW.
* The value of edgeFlags[ x ][ y ] is derived as follows:

edgeFlags[ x ][ y ] = 2 (1247)

* When edgeFlags[ 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 ] ) (1248)

* When inter\_affine\_flag[ xCb − 1 ][ yCb + y ] is equal to 1 or merge\_subblock\_flag[ xCb − 1 ][ yCb + y ] is equal to 1, the following applies:

maxFilterLengthPs[ x ][ y ] = Min( 5, maxFilterLengthPs[ x ][ y ] ) (1249)

* Otherwise, if edgeTbFlags[ x ][ y ] is equal to 1, the following applies:

maxFilterLengthPs[ x ][ y ] = Min( 5, maxFilterLengthPs[ x ][ y ] ) (1250)

maxFilterLengthQs[ x ][ y ] = Min( 5, maxFilterLengthQs[ x ][ y ] ) (1251)

* Otherwise, if one or more of the following conditions are true:
* ( x + 4 ) is greater than or equal to nCbW,
* edgeTbFlags[ x − 4 ][ y ] is equal to 1,
* edgeTbFlags[ x + 4 ][ y ] is equal to 1,

the following applies:

maxFilterLengthPs[ x ][ y ] = 1 (1252)

maxFilterLengthQs[ x ][ y ] = 1 (1253)

* Otherwise, if one or more of the following conditions are true:
* xEdge is equal to 1,
* xEdge is equal to ( nCbW / 8 ) − 1,
* edgeTbFlags[ x − sbW ][ y ] is equal to 1,
* edgeTbFlags[ x + sbW ][ y ] is equal to 1,

the following applies:

maxFilterLengthPs[ x ][ y ] = 2 (1254)

maxFilterLengthQs[ x ][ y ] = 2 (1255)

* Otherwise, the following applies:

maxFilterLengthPs[ x ][ y ] = 3 (1256)

maxFilterLengthQs[ x ][ y ] = 3 (1257)

* Otherwise, if edgeType is equal to EDGE\_HOR, the following applies:
* The variable sbH is set equal to Max( 8, nCbH / numSbY ).
* The array edgeTbFlags is set equal to edgeFlags.
* For yEdge = 0..min( ( nCbH / 8 ) − 1, numSbY − 1 ), x = 0..nCbW − 1:
* The vertical position y inside the current coding block is set equal to yEdge \*sbH.
* The value of edgeFlags[ x ][ y ] is derived as follows:

edgeFlags[ x ][ y ] = 2 (1258)

* When edgeFlags[ x ][ y ] is equal to 1 or 2, the values of maxFilterLengthPs[ x ][ y ] and maxFilterLengthQs[ x ][ y ] are modified as follows:
* If y is equal to 0, the following applies:
* When numSbY is greater than 1, the following applies:

maxFilterLengthQs[ x ][ y ] = Min( 5, maxFilterLengthQs[ x ][ y ] ) (1259)

* When inter\_affine\_flag[ xCb + x ][ yCb − 1 ] is equal to 1 or merge\_subblock\_flag[ xCb + x ][ yCb − 1 ] is equal to 1, the following applies:

maxFilterLengthPs[ x ][ y ] = Min( 5, maxFilterLengthPs[ x ][ y ] ) (1260)

* Otherwise, if edgeTbFlags[ x ][ y ] is equal to 1, the following applies:

maxFilterLengthPs[ x ][ y ] = Min( 5, maxFilterLengthPs[ x ][ y ] ) (1261)

maxFilterLengthQs[ x ][ y ] = Min( 5, maxFilterLengthQs[ x ][ y ] ) (1262)

* Otherwise, if one or more of the following conditions are true:
* ( y + 4 ) is greater than or equal to nCbH,
* edgeTbFlags[ x ][ y − 4 ] is equal to 1,
* edgeTbFlags[ x ][ y + 4 ] is equal to 1,

the following applies:

maxFilterLengthPs[ x ][ y ] = 1 (1263)

maxFilterLengthQs[ x ][ y ] = 1 (1264)

* Otherwise, if one or more of the following conditions are true:
* yEdge is equal to 1,
* yEdge is equal to ( nCbH / 8 ) − 1,
* edgeTbFlags[ x ][ y − sbH ] is equal to 1,
* edgeTbFlags[ x ][ y + sbH ] is equal to 1,

the following applies:

maxFilterLengthPs[ x ][ y ] = 2 (1265)

maxFilterLengthQs[ x ][ y ] = 2 (1266)

* Otherwise, the following applies:

maxFilterLengthPs[ x ][ y ] = 3 (1267)

maxFilterLengthQs[ x ][ y ] = 3 (1268)

#### Derivation process of boundary filtering strength

Inputs to this process are:

* a picture sample array recPicture,
* a location ( xCb, yCb ) specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,
* a variable nCbW specifying the width of the current coding block,
* a variable nCbH specifying the height of the current coding block,
* a variable edgeType specifying whether a vertical (EDGE\_VER) or a horizontal (EDGE\_HOR) edge is filtered,
* a variable cIdx specifying the colour component of the current coding block,
* a two-dimensional (nCbW)x(nCbH) array edgeFlags.

Output of this process is a two-dimensional (nCbW)x(nCbH) array bS specifying the boundary filtering strength.

The variables xDi, yDj, xN and yN are derived as follows:

* The variable gridSize is set as follows:

gridSize = cIdx  = =  0 ? 4 : 8 (1269)

* If edgeType is equal to EDGE\_VER,

xDi = ( i \* gridSize ) (1270)

yDj = cIdx = = 0 ? ( j  <<  2 ) : ( j  <<  1 ) (1271)

xN is set equal to Max( 0, ( nCbW / gridSize ) − 1 ) (1272)

yN = cIdx = = 0 ? ( nCbH / 4 ) − 1 : ( nCbH / 2 ) − 1 (1273)

* Otherwise (edgeType is equal to EDGE\_HOR),

xDi = cIdx = = 0 ? ( i  <<  2 ) : ( i  <<  1 ) (1274)

yDj =  j \* gridSize (1275)

xN = cIdx = = 0 ? ( nCbW / 4 ) − 1 : ( nCbW / 2 ) − 1 (1276)

yN = Max( 0, ( nCbH / gridSize ) − 1 ) (1277)

For xDi with i = 0..xN and yDj with j = 0..yN, the following applies:

* If edgeFlags[ xDi ][ yDj ] is equal to 0, the variable bS[ xDi ][ yDj ] is set equal to 0.
* Otherwise, if edgeType is equal to EDGE\_VER, VirtualBoundariesPresentFlag equal to 1, and ( xCb + xDi ) is equal to VirtualBoundariesPosX[ n ] for any n = 0..NumVerVirtualBoundaries − 1, the variable bS[ xDi ][ yDj ] is set equal to 0.
* Otherwise, if edgeType is equal to EDGE\_HOR, VirtualBoundariesPresentFlag equal to 1, and ( yCb + yDj ) is equal to VirtualBoundariesPosY[ n ] for any n = 0..NumHorVirtualBoundaries − 1, the variable bS[ xDi ][ yDj ] is set equal to 0.
* Otherwise, the following applies:
* The sample values p0 and q0 are derived as follows:
  + - If edgeType is equal to EDGE\_VER, p0 is set equal to recPicture[ xCb + xDi − 1 ][ yCb + yDj ] and q0 is set equal to recPicture[ xCb + xDi ][ yCb + yDj ].
    - Otherwise (edgeType is equal to EDGE\_HOR), p0 is set equal to recPicture[ xCb + xDi ][ yCb + yDj − 1 ] and q0 is set equal to recPicture[ xCb + xDi ][ yCb + yDj ].
* The variable bS[ xDi ][ yDj ] is derived as follows:
  + - If cIdx is equal to 0 and both samples p0 and q0 are in a coding block with intra\_bdpcm\_luma\_flag equal to 1, bS[ xDi ][ yDj ] is set equal to 0.
    - Otherwise, if cIdx is greater than 0 and both samples p0 and q0 are in a coding block with intra\_bdpcm\_chroma\_flag equal to 1, bS[ xDi ][ yDj ] is set equal to 0.
    - Otherwise, if the sample p0 or q0 is in the coding block of a coding unit coded with CuPredMode equal to MODE\_INTRA, bS[ xDi ][ yDj ] is set equal to 2.
    - Otherwise, if the block edge is also a coding block edge and the sample p0 or q0 is in a coding block with ciip\_flag equal to 1, bS[ xDi ][ yDj ] is set equal to 2.
    - Otherwise, if the block edge is also a transform block edge and the sample p0 or q0 is in a transform block which contains one or more non-zero transform coefficient levels, bS[ xDi ][ yDj ] is set equal to 1.
    - Otherwise, if cIdx is equal to 0, edgeFlags[ xDi ][ yDj ] is equal to 2, and one or more of the following conditions are true, bS[ xDi ][ yDj ] is set equal to 1:
      * The CuPredMode of the coding subblock containing the sample p0 is different from the CuPredMode of the coding subblock containing the sample q0.
      * The coding subblock containing the sample p0 and the coding subblock containing the sample q0 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 coding subblocks is greater than or equal to 8 in units of 1/16 luma samples.
      * For the prediction of the coding subblock containing the sample p0 different reference pictures or a different number of motion vectors are used than for the prediction of the coding subblock containing the sample q0.

NOTE 1 – The determination of whether the reference pictures used for the two coding sublocks are the same or different is based only on which pictures are referenced, without regard to whether a prediction is formed using an index into reference picture list 0 or an index into reference picture list 1, and also without regard to whether the index position within a reference picture list is different.

NOTE 2 – The number of motion vectors that are used for the prediction of a coding subblock with top-left sample covering ( xSb, ySb ), is equal to PredFlagL0[ xSb ][ ySb ] + PredFlagL1[ xSb ][ ySb ].

* + - * One motion vector is used to predict the coding subblock containing the sample p0 and one motion vector is used to predict the coding subblock containing the sample q0, 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 coding subblock containing the sample p0, two motion vectors for the same two reference pictures are used to predict the coding subblock containing the sample q0 and the absolute difference between the horizontal or vertical component of the two motion vectors used in the prediction of the two coding 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 coding subblock containing the sample p0, two motion vectors for the same reference picture are used to predict the coding subblock containing the sample q0 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 coding 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 coding 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 coding subblock containing the sample p0 and the list 1 motion vector used in the prediction of the coding subblock containing the sample q0 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 coding subblock containing the sample p0 and list 0 motion vector used in the prediction of the coding subblock containing the sample q0 is greater than or equal to 8 in units of 1/16 luma samples.
    - Otherwise, the variable bS[ xDi ][ yDj ] is set equal to 0.

#### Edge filtering process for one direction

Inputs to this process are:

* a variable edgeType specifying whether vertical edges (EDGE\_VER) or horizontal edges (EDGE\_HOR) are currently processed,
* a variable cIdx specifying the current colour component,
* the reconstructed picture prior to deblocking recPicture,
* a location ( xCb, yCb ) specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,
* a variable nCbW specifying the width of the current coding block,
* a variable nCbH specifying the height of the current coding block,
* the array bS specifying the boundary 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:

gridSize = cIdx  = =  0 ? 4 : 8 (1278)

* The variables subW, subH, xN, yN are derived as follows:

subW = cIdx = = 0 ? 1 : SubWidthC (1279)

subH = cIdx = = 0 ? 1 : SubHeightC (1280)

xN = edgeType = = EDGE\_VER ? Max( 0, ( nCbW / gridSize ) − 1 ) : ( nCbW / 4 / subW ) − 1 (1281)

yN = edgeType = = EDGE\_VER ? ( nCbH / 4 / subH ) − 1 : Max( 0, ( nCbH / gridSize ) − 1 ) (1282)

* The variables xDk with k = 0..xN and yDm with m = 0..yN are derived as follows:

xDk = edgeType = = EDGE\_VER ? ( k \*gridSize ) : ( k  <<  ( 2 / subW ) ) (1283)

yDm = edgeType = = EDGE\_VER ? ( m  <<  ( 2 / subH ) ) : ( m \* gridSize ) (1284)

* 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 clause 8.8.3.6.1 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 clause 8.8.3.6.2 is invoked with the luma picture sample array recPicture, the location of the luma coding block ( xCb, yCb ), the luma location of the block ( xBl, yBl ) set equal to ( 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 clause 8.8.3.6.3 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 clause 8.8.3.6.4 is invoked with the chroma picture sample array recPicture, the location of the chroma coding block ( xCb, yCb ), the 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.

##### 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:

qj,k = recPicture[ xCb + xBl + j ][ yCb + yBl + k ] (1285)

pi,k = recPicture[ xCb + xBl − i − 1 ][ yCb + yBl + k ] (1286)

* Otherwise (edgeType is equal to EDGE\_HOR), the following applies:

qj,k = recPicture[ xCb + xBl + k ][ yCb + yBl + j ] (1287)

pi,k = recPicture[ xCb + xBl + k ][ yCb + yBl − i − 1 ] (1288)

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 follow:

lumaLevel = ( ( p0,0 + p0,3 + q0,0 + q0,3 ) >> 2 ), (1289)

* 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 ] (1290)  
 else  
 break  
}

* Otherwise, qpOffset is set equal to 0.

The variables QpQ and QpP are set equal to the QpY values of the coding units which include the coding blocks containing the sample q0,0 and p0,0, respectively.

The variable qP is derived as follows:

qP = ( ( QpQ + QpP + 1 )  >>  1 ) + qpOffset (1291)

The value of the variable β′ is determined as specified in Table 43 based on the quantization parameter Q derived as follows:

Q = Clip3( 0, 63, qP + ( sh\_luma\_beta\_offset\_div2  <<  1 ) ) (1292)

where sh\_luma\_beta\_offset\_div2 is the value of the syntax element sh\_luma\_beta\_offset\_div2 for the slice that contains sample q0,0.

The variable β is derived as follows:

β = β′ \* ( 1  <<  ( BitDepth − 8 ) ) (1293)

The value of the variable tC′ is determined as specified in Table 43 based on the quantization parameter Q derived as follows:

Q = Clip3( 0, 65, qP + 2 \* ( bS − 1 ) + ( sh\_luma\_tc\_offset\_div2  <<  1 ) ) (1294)

where sh\_luma\_tc\_offset\_div2 is the value of the syntax element sh\_luma\_tc\_offset\_div2 for the slice that contains sample q0,0.

The variable tC is derived as follows:

roundOffset = 1  <<  ( 9 − BitDepth ) (1295)

tC = BitDepth < 10 ? ( tC′ + roundOffset )  >>  ( 10 − BitDepth ) : tC′ \* ( 1  <<  ( BitDepth − 10 ) ) (1296)

The following ordered steps apply:

1. The variables dp0, dp3, dq0 and dq3 are derived as follows:

dp0 = Abs( p2,0 − 2 \* p1,0 + p0,0 ) (1297)

dp3 = Abs( p2,3 − 2 \* p1,3 + p0,3 ) (1298)

dq0 = Abs( q2,0 − 2 \* q1,0 + q0,0 ) (1299)

dq3 = Abs( q2,3 − 2 \* q1,3 + q0,3 ) (1300)

1. When maxFilterLengthP and maxFilterLengthQ both are equal to or greater than 3 the variables sp0, sq0, spq0, sp3, sq3 and spq3 are derived as follows:

sp0 = Abs( p3,0 − p0,0 ) (1301)

sq0 = Abs( q0,0 − q3,0 ) (1302)

spq0 = Abs( p0,0 − q0,0 ) (1303)

sp3 = Abs( p3,3 − p0,3 ) (1304)

sq3 = Abs( q0,3 − q3,3 ) (1305)

spq3 = Abs( p0,3 − q0,3 ) (1306)

1. The variables sidePisLargeBlk and sideQisLargeBlk are set equal to 0.
2. When maxFilterLengthP is greater than 3, sidePisLargeBlk is set equal to 1.
3. When maxFilterLengthQ is greater than 3, sideQisLargeBlk is set equal to 1.
4. When edgeType is equal to EDGE\_HOR and (yCb + yBl ) % CtbSizeY is equal to 0, sidePisLargeBlk is set equal to 0.
5. The variables dSam0 and dSam3 are initialized to 0.
6. When sidePisLargeBlk or sideQisLargeBlk is greater than 0, the following applies:
7. The variables dp0L, dp3L are derived and maxFilterLengthP is modified as follows:

* If sidePisLargeBlk is equal to 1, the following applies:

dp0L = ( dp0 + Abs( p5,0 − 2 \* p4,0 + p3,0 ) + 1 ) >> 1 (1307)

dp3L = ( dp3 + Abs( p5,3 − 2 \* p4,3 + p3,3 ) + 1 ) >> 1 (1308)

* Otherwise, the following applies:

dp0L = dp0 (1309)

dp3L = dp3 (1310)

maxFilterLengthP = 3 (1311)

1. The variables dq0L and dq3L are derived as follows:

* If sideQisLargeBlk is equal to 1, the following applies:

dq0L = ( dq0 + Abs( q5,0 − 2 \* q4,0 + q3,0 ) + 1 ) >> 1 (1312)

dq3L = ( dq3 + Abs( q5,3 − 2 \* q4,3 + q3,3 ) + 1 ) >> 1 (1313)

* Otherwise, the following applies:

dq0L = dq0 (1314)

dq3L = dq3 (1315)

1. The variables sp0L and sp3L are derived as follows:

* If maxFilterLengthP is equal to 7, the following applies:

sp0L = sp0 + Abs( p7,0 − p6,0 − p5,0 + p4,0) (1316)

sp3L = sp3 + Abs( p7,3 − p6,3 − p5,3 + p4,3) (1317)

* Otherwise, the following applies:

sp0L = sp0 (1318)

sp3L = sp3 (1319)

1. The variables sq0L and sq3L are derived as follows:

* If maxFilterLengthQ is equal to 7, the following applies:

sq0L = sq0 + Abs( q4,0 − q5,0 − q6,0 + q7,0 ) (1320)

sq3L = sq3 + Abs( q4,3 − q5,3 − q6,3 + q7,3 ) (1321)

* Otherwise, the following applies:

sq0L = sq0 (1322)

sq3L = sq3 (1323)

1. The variables dpq0L, dpq3L, and dL are derived as follows:

dpq0L = dp0L + dq0L (1324)

dpq3L = dp3L + dq3L (1325)

dL = dpq0L + dpq3L (1326)

1. When dL is less than β, the following ordered steps apply:
   * 1. The variable dpq is set equal to 2 \* dpq0L.
     2. 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.
     3. The variables p0 p3 qo and q3 are first initialized to 0 and then modified according to sidePisLargeBlk and sideQisLargeBlk as follows:

* When sidePisLargeBlk is equal to 1, the following applies:

p3 = p3,0 (1327)

p0 = pmaxFilterLengthP,0 (1328)

* When sideQisLargeBlk is equal to 1, the following applies:

q3 = q3,0 (1329)

q0 = qmaxFilterLengthQ,0 (1330)

* + 1. For the sample location ( xCb + xBl, yCb + yBl ), the decision process for a luma sample as specified in clause 8.8.3.6.5 is invoked with the sample values p0, p3, q0, q3, the variables dpq, sp, sq, spq, sidePisLargeBlk, sideQisLargeBlk, β and tC as inputs, and the output is assigned to the decision dSam0.
    2. The variable dpq is set equal to 2 \* dpq3L.
    3. 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.
    4. The variables p0 p3 q0 and q3 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:

p3 = p3,3 (1331)

p0 = pmaxFilterLengthP,3 (1332)

* When sideQisLargeBlk is equal to 1, the following applies:

q3 = q3,3 (1333)

q0 = qmaxFilterLengthQ,3 (1334)

* + 1. When edgeType is equal to EDGE\_VER for the sample location ( xCb + xBl, yCb + yBl + 3 ) or when edgeType is equal to EDGE\_HOR for the sample location ( xCb + xBl + 3, yCb + yBl ), the decision process for a luma sample as specified in clause 8.8.3.6.5 is invoked with the sample values p0, p3, q0, q3, the variables dpq, sp, sq, spq, sidePisLargeBlk, sideQisLargeBlk, β and tC as inputs, and the output is assigned to the decision dSam3.

1. 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:

1. The variables dpq0, dpq3, dp, dq and d are derived as follows:

dpq0 = dp0 + dq0 (1335)

dpq3 = dp3 + dq3 (1336)

dp = dp0 + dp3 (1337)

dq = dq0 + dq3 (1338)

d = dpq0 + dpq3 (1339)

1. The variables dE, dEp, dEq, sidePisLargeBlk and sideQisLargeBlk are set equal to 0.
2. When d is less than β and both maxFilterLengthP and maxFilterLengthQ are greater than 2, the following ordered steps apply:
3. The variable dpq is set equal to 2 \* dpq0.
4. The variable sp is set equal to sp0, the variable sq is set equal to sq0 and the variable spq is set equal to spq0.
5. For the sample location ( xCb + xBl, yCb + yBl ), the decision process for a luma sample as specified in clause 8.8.3.6.5 is invoked with the variables p0, p3, q0, q3 all set equal to 0, the variables dpq, sp, sq, spq, sidePisLargeBlk, sideQisLargeBlk, β and tC as inputs, and the output is assigned to the decision dSam0.
6. The variable dpq is set equal to 2 \* dpq3.
7. The variable sp is set equal to sp3, the variable sq is set equal to sq3 and the variable spq is set equal to spq3.
8. When edgeType is equal to EDGE\_VER for the sample location ( xCb + xBl, yCb + yBl + 3 ) or when edgeType is equal to EDGE\_HOR for the sample location ( xCb + xBl + 3, yCb + yBl ), the decision process for a sample as specified in clause 8.8.3.6.5 is invoked with the variables p0, p3, q0, q3 all set equal to 0, the variables dpq, sp, sq, spq, sidePisLargeBlk, sideQisLargeBlk, β and tC as inputs, and the output is assigned to the decision dSam3.
9. When d is less than β, the following ordered steps apply:
10. The variable dE is set equal to 1.
11. When dSam0 is equal to 1 and dSam3 is equal to 1, the variable dE is set equal to 2 and both maxFilterLengthP and maxFilterLengthQ are set equal to 3.
12. When maxFilterLengthP is greater than 1, and maxFilterLengthQ is greater than 1, and dp is less than ( β + ( β  >>  1 ) )  >>  3, the variable dEp is set equal to 1.
13. When maxFilterLengthP is greater than 1, and maxFilterLengthQ is greater than 1, and dq is less than ( β + ( β  >>  1 ) )  >>  3, the variable dEq is set equal to 1.
14. When dE is equal to 1, maxFilterLengthP is set equal to 1 + dEp and maxFilterLengthQ is set equal to 1 + dEq.

Table 43 – Derivation of threshold variables β′ and tC′ from input Q

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Q** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| **β**′ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 |
| **tC**′ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Q** | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 |
| **β**′ | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 20 | 22 | 24 | 26 | 28 |
| **tC**′ | 0 | 3 | 4 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 7 | 7 | 8 | 9 | 10 | 10 | 11 |
| **Q** | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| **β**′ | 30 | 32 | 34 | 36 | 38 | 40 | 42 | 44 | 46 | 48 | 50 | 52 | 54 | 56 | 58 | 60 | 62 |
| **tC**′ | 13 | 14 | 15 | 17 | 19 | 21 | 24 | 25 | 29 | 33 | 36 | 41 | 45 | 51 | 57 | 64 | 71 |
| **Q** | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 |  |  |
| **β**′ | 64 | 66 | 68 | 70 | 72 | 74 | 76 | 78 | 80 | 82 | 84 | 86 | 88 | - | - |  |  |
| **tC**′ | 80 | 89 | 100 | 112 | 125 | 141 | 157 | 177 | 198 | 222 | 250 | 280 | 314 | 352 | 395 |  |  |

##### Filtering process for luma block edges

Inputs to this process are:

* a picture sample array recPicture,
* a location ( xCb, yCb ) specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,
* a location ( xBl, yBl ) specifying the top-left sample of the current block relative to the top-left sample of the current coding block,
* a variable edgeType specifying whether a vertical (EDGE\_VER) or a horizontal (EDGE\_HOR) edge is filtered,
* the variables dE, dEp and dEq containing decisions,
* the variables maxFilterLengthP and maxFilterLengthQ containing maximum filter lengths,
* the variable tC.

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 pi,k and qj,k with i = 0..maxFilterLengthP, j = 0..maxFilterLengthQ and k = 0..3 are derived as follows:

qj,k = recPicture[ xCb + xBl + j ][ yCb + yBl + k ] (1340)

pi,k = recPicture[ xCb + xBl − i − 1 ][ yCb + yBl + k ] (1341)

1. 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:
2. The filtering process for a luma sample using short filters as specified in clause 8.8.3.6.6 is invoked with the variables maxFilterLengthP, maxFilterLengthQ, the sample values pi,k, qj,k with i = 0..maxFilterLengthP and j = 0..maxFilterLengthQ, the decision dE, the variables dEp and dEq and the variable tC as inputs, and the number of filtered samples nDp and nDq from each side of the block boundary and the filtered sample values pi′ and qj′ as outputs.
3. When nDp is greater than 0, the filtered sample values pi′ with i = 0..nDp − 1 replace the corresponding samples inside the sample array recPicture as follows:

recPicture[ xCb + xBl − i − 1 ][ yCb + yBl + k ] = pi′ (1342)

1. When nDq is greater than 0, the filtered sample values qj′ with j = 0..nDq − 1 replace the corresponding samples inside the sample array recPicture as follows:

recPicture[ xCb + xBl + j ][ yCb + yBl + k ] = qj′ (1343)

1. When dE is equal to 3, for each sample location ( xCb + xBl, yCb + yBl + k ), k = 0..3, the following ordered steps apply:
2. The filtering process for a luma sample using long filters as specified in clause 8.8.3.6.7 is invoked with the variables maxFilterLengthP, maxFilterLengthQ, the sample values pi,k, qj,k with i = 0..maxFilterLengthP and j = 0..maxFilterLengthQ, and tC as inputs and the filtered samples values pi′ and qj′ as outputs.
3. The filtered sample values pi′ with i = 0..maxFilterLengthP − 1 replace the corresponding samples inside the sample array recPicture as follows:

recPicture[ xCb + xBl − i − 1 ][ yCb + yBl + k ] = pi′ (1344)

1. The filtered sample values qj′ with j = 0..maxFilterLengthQ − 1 replace the corresponding samples inside the sample array recPicture as follows:

recPicture[ xCb + xBl + j ][ yCb + yBl + k ] = qj′ (1345)

* Otherwise (edgeType is equal to EDGE\_HOR), the following ordered steps apply:

1. The sample values pi,k and qj,k with i = 0..maxFilterLengthP, j = 0..maxFilterLengthQ and k = 0..3 are derived as follows:

qj,k = recPicture[ xCb + xBl + k ][ yCb + yBl + j ] (1346)

pi,k = recPicture[ xCb + xBl + k ][ yCb + yBl − i − 1 ] (1347)

1. When dE is not equal to 0 and dE is not equal to 3, for each sample location ( xCb + xBl + k, yCb + yBl ), k = 0..3, the following ordered steps apply:
2. The filtering process for a luma sample using short filters as specified in clause 8.8.3.6.6 is invoked with the variables maxFilterLengthP, maxFilterLengthQ, the sample values pi,k, qi,k with i = 0..maxFilterLengthP and j = 0..maxFilterLengthQ, the decision dE, the variables dEp and dEq, and the variable tC as inputs, and the number of filtered samples nDp and nDq from each side of the block boundary and the filtered sample values pi′ and qj′ as outputs.
3. When nDp is greater than 0, the filtered sample values pi′ with i = 0..nDp − 1 replace the corresponding samples inside the sample array recPicture as follows:

recPicture[ xCb + xBl + k ][ yCb + yBl − i − 1 ] = pi′ (1348)

1. When nDq is greater than 0, the filtered sample values qj′ with j = 0..nDq − 1 replace the corresponding samples inside the sample array recPicture as follows:

recPicture[ xCb + xBl + k ][ yCb + yBl + j ] = qj′ (1349)

1. When dE is equal to 3, for each sample location ( xCb + xBl + k, yCb + yBl ), k = 0..3, the following ordered steps apply:
2. The filtering process for a luma sample using long filters as specified in clause 8.8.3.6.7 is invoked with the variables maxFilterLengthP, maxFilterLengthQ, the sample values pi,k, qj,k with i = 0..maxFilterLengthP and j = 0..maxFilterLengthQ, and the variable tC as inputs, and the filtered sample values pi′ and qj′ as outputs.
3. The filtered sample values pi′ with i = 0..maxFilterLengthP − 1 replace the corresponding samples inside the sample array recPicture as follows:

recPicture[ xCb + xBl + k ][ yCb + yBl − i − 1 ] = pi′ (1350)

1. The filtered sample values qj′ with j = 0..maxFilterLengthQ − 1 replace the corresponding samples inside the sample array recPicture as follows:

recPicture[ xCb + xBl + k ][ yCb + yBl + j ] = qj′ (1351)

##### Decision process for chroma block edges

This process is only invoked when ChromaArrayType is not equal to 0.

Inputs to this process are:

* a chroma picture sample array recPicture,
* a chroma location ( xCb, yCb ) specifying the top-left sample of the current chroma coding block relative to the top-left chroma sample of the current picture,
* a chroma location ( xBl, yBl ) specifying the top-left sample of the current chroma block relative to the top-left sample of the current chroma coding block,
* a variable edgeType specifying whether a vertical (EDGE\_VER) or a horizontal (EDGE\_HOR) edge is filtered,
* a variable cIdx specifying the colour component index,
* a variable bS specifying the boundary filtering strength,
* a variable maxFilterLengthP specifying the maximum filter length,
* a variable maxFilterLengthQ specifying the maximum filter length.

Outputs of this process are

* the modified filter length variables maxFilterLengthP and maxFilterLengthQ,
* the variable tC.

The variable maxK is derived as follows:

* If edgeType is equal to EDGE\_VER, the following applies:

maxK = ( SubHeightC = = 1 ) ? 3 : 1 (1352)

* Otherwise (edgeType is equal to EDGE\_HOR), the following applies:

maxK = ( SubWidthC = = 1 ) ? 3 : 1 (1353)

The values pi,k and qj,k with i = 0..maxFilterLengthP, j = 0..maxFilterLengthQ and k = 0..maxK are derived as follows:

* If edgeType is equal to EDGE\_VER, the following applies:

qj,k = recPicture[ xCb + xBl + j ][ yCb + yBl + k ] (1354)

pi,k = recPicture[ xCb + xBl − i − 1 ][ yCb + yBl + k ] (1355)

subSampleC = SubHeightC (1356)

* Otherwise (edgeType is equal to EDGE\_HOR), the following applies:

qj,k = recPicture[ xCb + xBl + k ][ yCb + yBl + j ] (1357)

pi,k = recPicture[ xCb + xBl + k ][ yCb + yBl − i − 1 ] (1358)

subSampleC = SubWidthC (1359)

The variable QpP is derived as follows:

* The luma location ( xTbP, yTbP ) is set as the top-left luma sample position of the transform block containing the sample p0,0, relative to the top-left luma sample of the picture.
* If TuCResMode[ xTbP ][ yTbP ] is equal to 2, QpP is set equal to Qp′CbCr of the transform block containing the sample p0,0.
* Otherwise, if cIdx is equal to 1, QpP is set equal to Qp′Cb of the transform block containing the sample p0,0.
* Otherwise, QpP is set equal to Qp′Cr of the transform block containing the sample p0,0.

The variable QpQ is derived as follows:

* The luma location ( xTbQ,  yTbQ ) is set as the top-left luma sample position of the transform block containing the sample q0,0, relative to the top-left luma sample of the picture.
* If TuCResMode[ xTbQ ][ yTbQ ] is equal to 2, QpQ is set equal to Qp′CbCr of the transform block containing the sample q0,0.
* Otherwise, if cIdx is equal to 1, QpQ is set equal to Qp′Cb of the transform block containing the sample q0,0.
* Otherwise, QpQ is set equal to Qp′Cr of the transform block containing the sample q0,0.

The variable QpC is derived as follows:

QpC = ( QpQ − QpBdOffset + QpP − QpBdOffset + 1 )  >>  1 (1360)

The value of the variable β′ is determined as specified in Table 43 based on the quantization parameter Q derived as follows:

sliceBetaOffsetDiv2 = ( cIdx = = 1 ? sh\_cb\_beta\_offset\_div2 : sh\_cr\_beta\_offset\_div2 )  
Q = Clip3( 0, 63, QpC + ( sliceBetaOffsetDiv2  <<  1 ) ) (1361)

where sh\_cb\_beta\_offset\_div2 and sh\_cr\_beta\_offset\_div2 are the values of the syntax elements sh\_cb\_beta\_offset\_div2 and sh\_cr\_beta\_offset\_div2, respectively, for the slice that contains sample q0,0.

The variable β is derived as follows:

β = β′ \* ( 1  <<  ( BitDepth − 8 ) ) (1362)

The value of the variable tC′ is determined as specified in Table 43 based on the chroma quantization parameter Q derived as follows:

sliceTcOffsetDiv2 = ( cIdx = = 1 ? sh\_cb\_tc\_offset\_div2 : sh\_cr\_tc\_offset\_div2 )  
Q = Clip3( 0, 65, QpC + 2 \* ( bS − 1 ) + ( sliceTcOffsetDiv2  <<  1 ) ) (1363)

where sh\_cb\_tc\_offset\_div2 and sh\_cr\_tc\_offset\_div2 are the values of the syntax elements sh\_cb\_tc\_offset\_div2 and sh\_cr\_tc\_offset\_div2, respectivelyl, for the slice that contains sample q0,0.

The variable tC is derived as follows:

tC = ( BitDepth < 10 ) ? ( tC′ + 2 ) >> ( 10 − BitDepth ) : tC′ \* ( 1  <<  ( BitDepth − 10 ) ) (1364)

When both maxFilterLengthP and maxFilterLengthQ are equal to 1 and bS is not equal to 2, maxFilterLengthP and maxFilterLengthQ are both set equal to 0.

When maxFilterLengthQ is equal to 3, the following ordered steps apply:

1. The variables n1 is derived as follows:

n1 = subSampleC = = 2 ? 1 : 3 (1365)

1. When maxFilterLengthP is equal to 1, the samples p3,0 and p2,0 are both set equal to p1,0 and the samples p3,n1, p2,n1 are both set equal to p1,n1.
2. The variables dpq0, dpq1, dp, dq and d are derived as follows:

dp0 = Abs( p2,0 − 2 \* p1,0 + p0,0 ) (1366)

dp1 = Abs( p2,n1 − 2 \* p1,n1 + p0,n1 ) (1367)

dq0 = Abs( q2,0 − 2 \* q1,0 + q0,0 ) (1368)

dq1 = Abs( q2,n1 − 2 \* q1,n1 + q0,n1 ) (1369)

dpq0 = dp0 + dq0 (1370)

dpq1 = dp1 + dq1 (1371)

dp = dp0 + dp1 (1372)

dq = dq0 + dq1 (1373)

d = dpq0 + dpq1 (1374)

1. The variables dSam0 and dSam1 are both set equal to 0.
2. When d is less than β, the following ordered steps apply:
3. The variable dpq is set equal to 2 \* dpq0.
4. The variable dSam0 is derived by invoking the decision process for a chroma sample as specified in clause 8.8.3.6.8 for the sample location ( xCb + xBl, yCb + yBl ) with sample values p0,0, p3,0, q0,0, and q3,0, the variables dpq, β and tC as inputs, and the output is assigned to the decision dSam0.
5. The variable dpq is set equal to 2 \* dpq1.
6. The variable dSam1 is modified as follows:

* If edgeType is equal to EDGE\_VER, for the sample location ( xCb + xBl, yCb + yBl + n1 ), the decision process for a chroma sample as specified in clause 8.8.3.6.8 is invoked with sample values p0,n1, p3,n1, q0,n1, and q3,n1, the variables dpq, β and tC as inputs, and the output is assigned to the decision dSam1.
* Otherwise (edgeType is equal to EDGE\_HOR), for the sample location ( xCb + xBl + n1, yCb + yBl ), the decision process for a chroma sample as specified in clause 8.8.3.6.8 is invoked with sample values p0,n1, p3,n1, q0,n1 and q3,n1, the variables dpq, β and tC as inputs, and the output is assigned to the decision dSam1.

1. When dSam0 is equal to 0 or dSam1 is equal to 0, maxFilterLengthP and maxFilterLengthQ are both set equal to 1.

##### Filtering process for chroma block edges

This process is only invoked when ChromaArrayType is not equal to 0.

Inputs to this process are:

* a chroma picture sample array recPicture,
* a chroma location ( xCb, yCb ) specifying the top-left sample of the current chroma coding block relative to the top-left chroma sample of the current picture,
* a chroma location ( xBl, yBl ) specifying the top-left sample of the current chroma block relative to the top-left sample of the current chroma coding block,
* a variable edgeType specifying whether a vertical (EDGE\_VER) or a horizontal (EDGE\_HOR) edge is filtered,
* a variable maxFilterLengthP specifying the maximum filter length,
* a variable maxFilterLengthQ specifying the maximum filter length,
* the variable tC.

Output of this process is the modified chroma picture sample array recPicture.

The variable maxK is derived as follows:

* If edgeType is equal to EDGE\_VER, the following applies:

maxK = ( SubHeightC = = 1 ) ? 3 : 1 (1375)

* Otherwise (edgeType is equal to EDGE\_HOR), the following applies:

maxK = ( SubWidthC = = 1 ) ? 3 : 1 (1376)

The values pi,k with i = 0..maxFilterLengthP, qj,k with j = 0..maxFilterLengthQ, and k = 0..maxK are derived as follows:

* If edgeType is equal to EDGE\_VER, the following applies:

qj,k = recPicture[ xCb + xBl + j ][ yCb + yBl + k ] (1377)

pi,k = recPicture[ xCb + xBl − i − 1 ][ yCb + yBl + k ] (1378)

* Otherwise (edgeType is equal to EDGE\_HOR), the following applies:

qj,k = recPicture[ xCb + xBl + k ][ yCb + yBl + j ] (1379)

pi,k = recPicture[ xCb + xBl + k ][ yCb + yBl − i − 1 ] (1380)

Depending on the value of edgeType, the following applies:

* If edgeType is equal to EDGE\_VER, for each sample location ( xCb + xBl, yCb + yBl + k ), k = 0..maxK, the following ordered steps apply:

1. The filtering process for a chroma sample as specified in clause 8.8.3.6.9 is invoked with the variables maxFilterLengthP and maxFilterLengthQ, the sample values pi,k, qj,k with i = 0..maxFilterLengthP and j = 0..maxFilterLengthQ, and the variable tC as inputs, and the filtered sample values pi′ and qj′ with i = 0..maxFilterLengthP − 1 and j = 0..maxFilterLengthQ − 1as outputs.
2. The filtered sample values pi′ and qj′ with i = 0..maxFilterLengthP − 1 and j = 0..maxFilterLengthQ − 1 replace the corresponding samples inside the sample array recPicture as follows:

recPicture[ xCb + xBl + j ][ yCb + yBl + k ] = qj′ (1381)

recPicture[ xCb + xBl − i − 1 ][ yCb + yBl + k ] = pi′ (1382)

* Otherwise (edgeType is equal to EDGE\_HOR), for each sample location ( xCb + xBl + k, yCb + yBl ), k = 0..maxK, the following ordered steps apply:

1. The filtering process for a chroma sample as specified in clause 8.8.3.6.9 is invoked with the variable maxFilterLengthP and maxFilterLengthQ, the sample values pi,k, qj,k, with i = 0..maxFilterLengthP and j = 0..maxFilterLengthQ, and the variable tC as inputs, and the filtered sample values pi′ and qj′ with i = 0..maxFilterLengthP − 1 and j = 0..maxFilterLengthQ − 1as outputs.
2. The filtered sample values pi′ and qj′ with i = 0..maxFilterLengthP − 1 and j = 0..maxFilterLengthQ − 1 replace the corresponding samples inside the sample array recPicture as follows:

recPicture[ xCb + xBl + k ][ yCb + yBl + j ] = qj′ (1383)

recPicture[ xCb + xBl + k ][ yCb + yBl − i − 1 ] = pi′ (1384)

##### Decision process for a luma sample

Inputs to this process are:

* the sample values p0, p3, q0 and q3,
* the variables dpq, sp, sq, spq, sidePisLargeBlk, sideQisLargeBlk, β and tC.

Output of this process is the variable dSam containing a decision.

The variables sp and sq are modified as follows:

* When sidePisLargeBlk is equal to 1, the following applies:

sp = ( sp + Abs( p3 − p0 ) + 1 )  >>  1 (1385)

* When sideQisLargeBlk is equal to 1, the following applies:

sq = ( sq + Abs( q3 − q0 ) + 1 )  >>  1 (1386)

The variables sThr1 and sThr2 are is derived as follows:

* If sidePisLargeBlk is equal to 1 or sideQisLargeBlk is equal to 1, the following applies:

sThr1 = 3 \* β  >>  5 (1387)

sThr2 = β  >>  4 (1388)

* Otherwise, the following applies:

sThr1 = β  >>  3 (1389)

sThr2 = β  >>  2 (1390)

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 \* tC + 1 )  >>  1.
* Otherwise, dSam is set equal to 0.

##### Filtering process for a luma sample using short filters

Inputs to this process are:

* the variables maxFilterLengthP and maxFilterLengthQ,
* the sample values pi and qj 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 tC.

Outputs of this process are:

* the number of filtered samples nDp and nDq,
* the filtered sample values pi′ and qj′ 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:

p0′ = Clip3( p0 − 3 \* tC, p0 + 3 \* tC, ( p2 + 2 \* p1 + 2 \* p0 + 2 \* q0 + q1 + 4 )  >>  3 ) (1391)

p1′ = Clip3( p1 − 2 \* tC, p1 + 2 \* tC, ( p2 + p1 + p0 + q0 + 2 )  >>  2 ) (1392)

p2′ = Clip3( p2 − 1 \* tC, p2 + 1\*tC, ( 2 \* p3 + 3 \* p2 + p1 + p0 + q0 + 4 )  >>  3 ) (1393)

q0′ = Clip3( q0 − 3 \* tC, q0 + 3 \* tC, ( p1 + 2 \* p0 + 2 \* q0 + 2 \* q1 + q2 + 4 )  >>  3 ) (1394)

q1′ = Clip3( q1 − 2 \* tC, q1 + 2 \* tC, ( p0 + q0 + q1 + q2 + 2 )  >>  2 ) (1395)

q2′= Clip3( q2 − 1 \* tC, q2 + 1 \* tC, ( p0 + q0 + q1 + 3 \* q2 + 2 \* q3 + 4 )  >>  3 ) (1396)

* Otherwise, nDp and nDq are set both equal to 0 and the following weak filtering applies:
  + The following applies:

Δ = ( 9 \* ( q0 −  p0 ) − 3 \* ( q1 − p1 ) + 8 )  >>  4 (1397)

* + When Abs(Δ) is less than tC \* 10, the following ordered steps apply:
    - The filtered sample values p0′ and q0′ are specified as follows:

Δ = Clip3( −tC, tC, Δ ) (1398)

p0′ = Clip1( p0 + Δ ) (1399)

q0′ = Clip1( q0 − Δ ) (1400)

* + - When dEp is equal to 1, the filtered sample value p1′ is specified as follows:

Δp = Clip3( −( tC  >>  1 ), tC  >>  1, ( ( ( p2 + p0 + 1 )  >>  1 ) − p1 + Δ )  >>  1 ) (1401)

p1′ = Clip1( p1 + Δp ) (1402)

* + - When dEq is equal to 1, the filtered sample value q1′ is specified as follows:

Δq = Clip3( −( tC  >>  1 ), tC  >>  1, ( ( ( q2 + q0 + 1 )  >>  1 ) − q1 − Δ )  >>  1 ) (1403)

q1′ = Clip1( q1 + Δq ) (1404)

* + - 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 p0 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 q0 is equal to 1, nDq is set equal to 0:

##### Filtering process for a luma sample using long filters

Inputs to this process are:

* the variables maxFilterLengthP and maxFilterLengthQ,
* the sample values pi and qj with i = 0..maxFilterLengthP and j = 0..maxFilterLengthQ,
* a variable tC.

Outputs of this process are:

* the filtered sample values pi′ and qj′ with i = 0..maxFilterLengthP − 1, j = 0..maxFilterLengthQ − 1.

The variable refMiddle is derived as follows:

* If maxFilterLengthP is equal to maxFilterLengthQ and maxFilterLengthP is equal to 5, the following applies:

refMiddle = ( p4 + p3 + 2\* ( p2 + p1 + p0 + q0 + q1 + q2 ) + q3 + q4 + 8)  >>  4 (1405)

* Otherwise, if maxFilterLengthP is equal to maxFilterLengthQ and maxFilterLengthP is not equal to 5, the following applies:

refMiddle = ( p6 + p5 + p4 + p3 + p2 + p1 + 2\* ( p0 + q0 ) + q1 + q2 + q3 + q4 + q5 + q6 + 8 )  >>  4 (1406)

* Otherwise, if one of the following conditions are true,
* maxFilterLengthQ is equal to 7 and maxFilterLengthP is equal to 5,
* maxFilterLengthQ is equal to 5 and maxFilterLengthP is equal to 7,

the following applies:

refMiddle = ( p5 + p4 + p3 + p2 + 2\* ( p1 + p0 + q0 + q1 )  + q2 + q3 + q4 + q5 + 8 )  >>  4 (1407)

* Otherwise, if one of the following conditions are true,
* maxFilterLengthQ is equal to 5 and maxFilterLengthP is equal to 3,
* maxFilterLengthQ is equal to 3 and maxFilterLengthP is equal to 5,

the following applies:

refMiddle = ( p3 + p2 + p1 + p0 + q0 + q1 + q2 + q3 + 4)  >>  3 (1408)

* Otherwise, if maxFilterLengthQ is equal to 7 and maxFilterLengthP is equal to 3, the following applies:

refMiddle = ( 2 \* ( p2 + p1 + p0 + q0 ) + p0 + p1 + q1 + q2 + q3 + q4 + q5 + q6 + 8 )  >>  4 (1409)

* Otherwise, the following applies:

refMiddle = ( p6 + p5 + p4 + p3 + p2 + p1 + 2\*( q2 + q1 + q0 + p0)  + q0 + q1 + 8 )  >>  4 (1410)

The variables refP and refQ are derived as follows:

refP = ( pmaxFilterLengtP + pmaxFilterLengthP-1 + 1 )  >>  1 (1411)

refQ = ( qmaxFilterLengtQ + qmaxFilterLengthQ-1 + 1 )  >>  1 (1412)

The variables fi and tCPDi are defined as follows:

* If maxFilterLengthP is equal to 7, the following applies:

f0..6 = { 59, 50, 41, 32, 23, 14, 5 } (1413)

tCPD0..6 = { 6,  5,  4,  3,  2,  1,  1 } (1414)

* Otherwise, if maxFilterLengthP is equal to 5, the following applies:

f0..4 = { 58, 45, 32, 19, 6 } (1415)

tCPD0..4 = { 6, 5, 4, 3, 2 } (1416)

* Otherwise, the following applies:

f0..2 = { 53, 32, 11 } (1417)

tCPD0..2 = { 6, 4, 2} (1418)

The variables gj and tCQDj are defined as follows:

* If maxFilterLengthQ is equal to 7, the following applies:

g0..6 = { 59, 50, 41, 32, 23, 14, 5 } (1419)

tCQD0..6 = { 6, 5, 4, 3, 2, 1, 1 } (1420)

* Otherwise, if maxFilterLengthQ is equal to 5, the following applies:

g0..4 = { 58, 45, 32, 19, 6 } (1421)

tCQD0..4 = { 6, 5, 4, 3, 2 } (1422)

* Otherwise, the following applies:

g0..2 = { 53, 32, 11 } (1423)

tCQD0..2 = { 6, 4, 2 } (1424)

The filtered sample values pi′ and qj′ with i = 0..maxFilterLengthP − 1 and j = 0..maxFilterLengthQ − 1 are derived as follows:

pi′ = Clip3( pi − ( tC \* tCPDi  >>  1 ), pi + ( tC \* tCPDi  >>  1 ), ( refMiddle \* fi +  
 refP \* ( 64 − fi ) + 32)  >>  6 ) (1425)

qj′ = Clip3( qj − ( tC \* tCQDj  >>  1 ), qj + ( tC \* tCQDj  >>  1 ), ( refMiddle \* gj +  
 refQ \* ( 64 − gj ) + 32)  >>  6 ) (1426)

When pred\_mode\_plt\_flag of the coding unit that includes the coding block containing the sample pi is equal to 1, the filtered sample value, pi′ is substituted by the corresponding input sample value pi with i = 0..maxFilterLengthP − 1.

When pred\_mode\_plt\_flag of the coding unit that includes the coding block containing the sample qi is equal to 1, the filtered sample value, qi′ is substituted by the corresponding input sample value qj with j = 0..maxFilterLengthQ − 1.

##### Decision process for a chroma sample

Inputs to this process are:

* the sample values p0, p3, q0 and q3,
* the variables dpq, β and tC.

Output of this process is the variable dSam containing a decision.

The variable dSam is specified as follows:

* If all of the following conditions are true, dSam is set equal to 1:
* dpq is less than ( β  >>  2 ),
* Abs( p3 − p0 ) + Abs( q0 − q3 ) is less than ( β  >>  3 ),
* Abs( p0 − q0 ) is less than ( 5 \* tC + 1 )  >>  1.
* Otherwise, dSam is set equal to 0.

##### Filtering process for a chroma sample

This process is only invoked when ChromaArrayType is not equal to 0.

Inputs to this process are:

* the variables maxFilterLengthP and maxFilterLengthQ,
* the chroma sample values pi and qj with i = 0..maxFilterLengthP and j = 0..maxFilterLengthQ,
* a variable tC.

Outputs of this process are the filtered sample values pi′ and qj′ with i = 0..maxFilterLengthP − 1 and j = 0..maxFilterLengthQ − 1.

The filtered sample values pi′ and qj′ with i = 0..maxFilterLengthP − 1 and j = 0..maxFilterLengthQ − 1 are derived as follows:

* If both of maxFilterLengthP and maxFilterLengthQ is equal to 3, the following strong filtering applies:

p0′ = Clip3( p0 − tC, p0 + tC, ( p3 + p2 + p1 + 2 \* p0 + q0 + q1 + q2  + 4 )  >>  3 ) (1427)

p1′ = Clip3( p1 − tC, p1 + tC, ( 2 \* p3 + p2 + 2 \* p1 + p0 + q0 + q1  + 4 )  >>  3 ) (1428)

p2′ = Clip3( p2 − tC, p2 + tC, ( 3 \* p3 + 2 \* p2 + p1 + p0 + q0 + 4 )  >>  3 ) (1429)

q0′ = Clip3( q0 − tC, q0 + tC, ( p2 + p1 + p0 + 2 \* q0 + q1 + q2 + q3  + 4 )  >>  3 ) (1430)

q1′ = Clip3( q1 − tC, q1 + tC, ( p1 + p0 + q0 + 2 \* q1 + q2 + 2 \* q3  + 4 )  >>  3 ) (1431)

q2′= Clip3( q2 − tC, q2 + tC, ( p0 + q0 + q1 + 2 \* q2 + 3 \* q3 + 4 )  >>  3 ) (1432)

* Otherwise, if the variable maxFilterLengthP is equal to 1 and maxFilterLengthQ is equal to 3, the following filtering applies:

p0′ = Clip3( p0 − tC, p0 + tC, ( 3 \* p1 + 2 \* p0 + q0 + q1 + q2  + 4 )  >>  3 ) (1433)

q0′ = Clip3( q0 − tC, q0 + tC, ( 2 \*  p1 + p0 + 2 \* q0 + q1 + q2 + q3  + 4 )  >>  3 ) (1434)

q1′ = Clip3( q1 − tC, q1 + tC, ( p1 + p0 + q0 + 2 \* q1 + q2 + 2 \* q3  + 4 )  >>  3 ) (1435)

q2′= Clip3( q2 − tC, q2 + tC, ( p0 + q0 + q1 + 2 \* q2 + 3 \* q3 + 4 )  >>  3 ) (1436)

* Otherwise, the following weak filtering applies:

Δ = Clip3( −tC, tC, ( ( ( ( q0 − p0 )  <<  2 ) + p1 − q1 + 4 )  >>  3 ) ) (1437)

p0′ = Clip1( p0 + Δ ) (1438)

q0′ = Clip1( q0 − Δ ) (1439)

When pred\_mode\_plt\_flag of the coding unit that includes the coding block containing the sample pi is equal to 1, the filtered sample value, pi′ is substituted by the corresponding input sample value pi with i = 0..maxFilterLengthP − 1.

When pred\_mode\_plt\_flag of the coding unit that includes the coding block containing the sample qi is equal to 1, the filtered sample value, qi′ is substituted by the corresponding input sample value qi with i = 0..maxFilterLengthQ − 1:

### Sample adaptive offset process

#### General

Inputs to this process are the reconstructed picture sample array prior to sample adaptive offset recPictureL and, when ChromaArrayType 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 ChromaArrayType 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 ChromaArrayType 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 ChromaArrayType 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\_flag of the current slice is equal to 1, the CTB modification process as specified in clause 8.8.4.2 is invoked with recPicture set equal to recPictureL, cIdx set equal to 0, ( rx, ry ), and both nCtbSw and nCtbSh set equal to CtbSizeY as inputs, and the modified luma picture sample array saoPictureL as output.

– When ChromaArrayType is not equal to 0 and sh\_sao\_chroma\_flag of the current slice is equal to 1, the CTB modification process as specified in clause 8.8.4.2 is invoked with recPicture set equal to recPictureCb, cIdx set equal to 1, ( rx, ry ), nCtbSw set equal to ( 1  <<  CtbLog2SizeY ) / SubWidthC and nCtbSh set equal to ( 1  <<  CtbLog2SizeY ) / SubHeightC as inputs, and the modified chroma picture sample array saoPictureCb as output.

– When ChromaArrayType is not equal to 0 and sh\_sao\_chroma\_flag of the current slice is equal to 1, the CTB modification process as specified in clause 8.8.4.2 is invoked with recPicture set equal to recPictureCr, cIdx set equal to 2, ( rx, ry ), nCtbSw set equal to ( 1  <<  CtbLog2SizeY ) / SubWidthC and nCtbSh set equal to ( 1  <<  CtbLog2SizeY ) / SubHeightC as inputs, and the modified chroma picture sample array saoPictureCr as output.

#### CTB modification process

Inputs to this process are:

– the picture sample array recPicture for the colour component cIdx,

– a variable cIdx specifying the colour component index,

– a pair of variables ( rx, ry ) specifying the CTB location,

– the CTB width nCtbSw and height nCtbSh.

Output of this process is a modified picture sample array saoPicture for the colour component cIdx.

The variables scaleWidth and scaleHeight are derived as follows:

scaleWidth = ( cIdx = = 0 ) ? 1 : SubWidthC (1440)

scaleHeight = ( cIdx = = 0 ) ? 1 : SubHeightC (1441)

The location ( xCtb, yCtb ), specifying the top-left sample of the current CTB for the colour component cIdx relative to the top-left sample of the current picture component cIdx, is derived as follows:

( xCtb, yCtb ) = ( rx \* nCtbSw, ry \* nCtbSh ) (1442)

The sample locations inside the current CTB are derived as follows:

( xSi, ySj ) = ( xCtb + i, yCtb + j ) (1443)

( xYi, yYj ) = ( cIdx  = =  0 ) ? ( xSi, ySj ) : ( xSi \* SubWidthC, ySj \* SubHeightC ) (1444)

For all sample locations ( xSi, ySj ) and ( xYi, yYj ) with i = 0..nCtbSw − 1 and j = 0..nCtbSh − 1, the following applies:

– If one or more of the following conditions are true, saoPicture[ xSi ][ ySj ] is not modified:

* SaoTypeIdx[ cIdx ][ rx ][ ry ] is equal to 0.
* VirtualBoundariesPresentFlag is equal to 1 and xSj is equal to ( ( VirtualBoundaryPosX[ n ] / scaleWidth ) − 1 ) for any n = 0..NumVerVirtualBoundaries − 1 and SaoTypeIdx[ cIdx ][ rx ][ ry ] is equal to 2 and SaoEoClass[ cIdx ][ rx ][ ry ] is not equal to 1.
* VirtualBoundariesPresentFlag is equal to 1 and xSj is equal to ( VirtualBoundaryPosX[ n ] / scaleWidth ) for any n = 0..NumVerVirtualBoundaries − 1 and SaoTypeIdx[ cIdx ][ rx ][ ry ] is equal to 2 and SaoEoClass[ cIdx ][ rx ][ ry ] is not equal to 1.
* VirtualBoundariesPresentFlag is equal to 1 and ySj is equal to ( ( VirtualBoundaryPosY[ n ] / scaleHeight ) − 1 ) for any n = 0..NumHorVirtualBoundaries − 1 and SaoTypeIdx[ cIdx ][ rx ][ ry ] is equal to 2 and SaoEoClass[ cIdx ][ rx ][ ry ] is not equal to 0.
* VirtualBoundariesPresentFlag is equal to 1 and ySj is equal to ( VirtualBoundaryPosY[ n ] / scaleHeight ) for any n = 0..NumHorVirtualBoundaries − 1 and SaoTypeIdx[ cIdx ][ rx ][ ry ] is equal to 2 and SaoEoClass[ cIdx ][ rx ][ ry ] is not equal to 0.

– 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 ( xSik′, ySjk′ ) and ( xYik′, yYjk′ ) are derived as follows:

( xSik′, ySjk′ ) = ( xSi + hPos[ k ], ySj + vPos[ k ] ) (1445)

( xYik′, yYjk′ ) = ( cIdx  = =  0 ) ? ( xSik′, ySjk′ ) : ( xSik′ \* SubWidthC, ySjk′ \* SubHeightC ) (1446)

* If one or more of the following conditions for all sample locations ( xSik′, ySjk′ ) and ( xYik′, yYjk′ ) with k = 0..1 are true, edgeIdx is set equal to 0:
* The sample at location ( xSik′, ySjk′ ) is outside the picture boundaries.
* The sample at location ( xSik′, ySjk′ ) belongs to a different subpicture and sps\_loop\_filter\_across\_subpic\_enabled\_flag[ CurrSubpicIdx ] for the subpicture to which the sample recPicture[ xSi ][ ySj ] belongs to is equal to 0.
* pps\_loop\_filter\_across\_slices\_enabled\_flag is equal to 0 and the sample at location ( xSik′, ySjk′ ) belongs to a different slice.
* pps\_loop\_filter\_across\_tiles\_enabled\_flag is equal to 0 and the sample at location ( xSik′, ySjk′ ) belongs to a different tile.
* Otherwise, edgeIdx is derived as follows:
* The following applies:

edgeIdx = 2 + Sign( recPicture[ xSi ][ ySj ] − recPicture[ xSi + hPos[ 0 ] ][ ySj + vPos[ 0 ] ] ) +  
 Sign( recPicture[ xSi ][ ySj ] − recPicture[ xSi + hPos[ 1 ] ][ ySj + vPos[ 1 ] ] ) (1447)

* When edgeIdx is equal to 0, 1, or 2, edgeIdx is modified as follows:

edgeIdx = ( edgeIdx = = 2 ) ? 0 : ( edgeIdx + 1 ) (1448)

1. The modified picture sample array saoPicture[ xSi ][ ySj ] is derived as follows:

saoPicture[ xSi ][ ySj ] = Clip3( 0, ( 1  <<  BitDepth ) − 1, recPicture[ xSi ][ ySj ] +  
 SaoOffsetVal[ cIdx ][ rx ][ ry ][ edgeIdx ] ) (1449)

* Otherwise (SaoTypeIdx[ cIdx ][ rx ][ ry ] is equal to 1), the following ordered steps apply:

1. The variable bandShift is set equal to BitDepth − 5.
2. The variable saoLeftClass is set equal to sao\_band\_position[ cIdx ][ rx ][ ry ].
3. The list bandTable is defined with 32 elements and all elements are initially set equal to 0. Then, four of its elements (indicating the starting position of bands for explicit offsets) are modified as follows:

for( k = 0; k < 4; k++ )  
 bandTable[ ( k + saoLeftClass ) & 31 ] = k + 1 (1450)

1. The variable bandIdx is set equal to bandTable[ recPicture[ xSi ][ ySj ]  >>  bandShift ].
2. The modified picture sample array saoPicture[ xSi ][ ySj ] is derived as follows:

saoPicture[ xSi ][ ySj ] = Clip3( 0, ( 1  <<  BitDepth ) − 1, recPicture[ xSi ][ ySj ] +  
 SaoOffsetVal[ cIdx ][ rx ][ ry ][ bandIdx ] ) (1451)

Table 44 – Specification of hPos and vPos according to the sample adaptive offset class

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SaoEoClass[ cIdx ][ rx ][ ry ] | 0 | 1 | 2 | 3 |
| hPos[ 0 ] | −1 | 0 | −1 | 1 |
| hPos[ 1 ] | 1 | 0 | 1 | −1 |
| vPos[ 0 ] | 0 | −1 | −1 | −1 |
| vPos[ 1 ] | 0 | 1 | 1 | 1 |

### Adaptive loop filter process

#### General

Inputs of this process are the reconstructed picture sample array prior to adaptive loop filter recPictureL and, when ChromaArrayType is not equal to 0, the arrays recPictureCb and recPictureCr.

Outputs of this process are the modified reconstructed picture sample array after adaptive loop filter alfPictureL and, when ChromaArrayType is not equal to 0, the arrays ccAlfPictureCb and ccAlfPictureCr.

The sample values in the modified reconstructed picture sample array after adaptive loop filter alfPictureL and, when ChromaArrayType is not equal to 0, the arrays alfPictureCb and alfPictureCr are initially set equal to the sample values in the reconstructed picture sample array prior to adaptive loop filter recPictureL and, when ChromaArrayType is not equal to 0, the arrays recPictureCb and recPictureCr, respectively.

The following ordered steps apply:

* + For every coding tree unit with luma coding tree block location ( rx, ry ), where rx = 0..PicWidthInCtbsY − 1 and ry = 0..PicHeightInCtbsY − 1, the following applies:
  + When alf\_ctb\_flag[ 0 ][ rx ][ ry ] is equal to 1, the coding tree block filtering process for luma samples as specified in clause 8.8.5.2 is invoked with recPictureL, alfPictureL, and the luma coding tree block location ( xCtb, yCtb ) set equal to ( rx  <<  CtbLog2SizeY, ry  <<  CtbLog2SizeY ) as inputs, and the output is the modified filtered picture alfPictureL.
  + When ChromaArrayType is not equal to 0 and alf\_ctb\_flag[ 1 ][ rx ][ ry ] is equal to 1, the coding tree block filtering process for chroma samples as specified in clause 8.8.5.4 is invoked with recPicture set equal to recPictureCb, alfPicture set equal to alfPictureCb, the chroma coding tree block location ( xCtbC, yCtbC ) set equal to ( ( rx  <<  CtbLog2SizeY ) / SubWidthC, ( ry  <<  CtbLog2SizeY ) / SubHeightC ), and the alternative chroma filter index altIdx set equal to alf\_ctb\_filter\_alt\_idx[ 0 ][ rx ][ ry ] as inputs, and the output is the modified filtered picture alfPictureCb.
  + When ChromaArrayType is not equal to 0 and alf\_ctb\_flag[ 2 ][ rx ][ ry ] is equal to 1, the coding tree block filtering process for chroma samples as specified in clause 8.8.5.4 is invoked with recPicture set equal to recPictureCr, alfPicture set equal to alfPictureCr, the chroma coding tree block location ( xCtbC, yCtbC ) set equal to ( ( rx  <<  CtbLog2SizeY ) / SubWidthC, ( ry  <<  CtbLog2SizeY ) / SubHeightC ), and the alternative chroma filter index altIdx set equal to alf\_ctb\_filter\_alt\_idx[ 1 ][ rx ][ ry ] as inputs, and the output is the modified filtered picture alfPictureCr.
  + When ChromaArrayType is not equal to 0, the sample values in the arrays ccAlfPictureCb and ccAlfPictureCr are set equal to the sample values in the arrays alfPictureCb and alfPictureCr, 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 ChromaArrayType is not equal to 0 and alf\_ctb\_cc\_cb\_idc[ rx ][ ry ] is not equal to 0, the cross-component filtering process as specified in clause 8.8.5.7 is invoked with recPictureL set equal to recPictureL, alfPictureC set equal to alfPictureCb, the chroma coding tree block location ( xCtbC, yCtbC ) set equal to ( ( rx << CtbLog2SizeY ) / SubWidthC, ( ry << CtbLog2SizeY  )/ SubHeightC ) ), the luma coding tree block location ( xCtb, yCtb ) set equal to ( rx  <<  CtbLog2SizeY, ry  <<  CtbLog2SizeY ), ccAlfWidth set equal to ( 1 << CtbLog2SizeY ) / SubWidthC, ccAlfHeight set equal to ( 1<< CtbLog2SizeY ) / SubHeightC, and the cross-component filter coefficients CcAlfCoeff[ j ] set equal to CcAlfApsCoeffCb[ sh\_cc\_alf\_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 ccAlfPictureCb.
  + When ChromaArrayType is not equal to 0 and alf\_ctb\_cc\_cr\_idc[ rx ][ ry ] is not equal to 0, the cross-component filtering process as specified in clause 8.8.5.7 is invoked with recPictureL set equal to recPictureL, alfPictureC set equal to alfPictureCr, the chroma coding tree block location ( xCtbC, yCtbC ) set equal to ( ( rx << CtbLog2SizeY ) / SubWidthC, ( ry << CtbLog2SizeY  )/ SubHeightC ) ), ccAlfWidth set equal to ( 1 << CtbLog2SizeY ) / SubWidthC, ccAlfHeight set equal to ( 1<< CtbLog2SizeY ) / SubHeightC, and the cross-component filter coefficients CcAlfCoeff[ j ] set equal to CcAlfApsCoeffCr[ sh\_cc\_alf\_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 ccAlfPictureCr.

#### 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 alfPictureL,
* a luma location ( xCtb, yCtb ) specifying the top-left sample of the current luma coding tree block relative to the top left sample of the current picture.

Output of this process is the modified filtered reconstructed luma picture sample array alfPictureL.

The derivation process for filter index clause 8.8.5.3 is invoked with the location ( xCtb, yCtb ) and the reconstructed luma picture sample array recPicture as inputs, and filtIdx[ x ][ y ] and transposeIdx[ x ][ y ] with x, y = 0..CtbSizeY − 1 as outputs.

For the derivation of the filtered reconstructed luma samples alfPictureL[ x ][ y ], each reconstructed luma sample inside the current luma coding tree block recPicture[ x ][ y ] is filtered as follows with x, y = 0..CtbSizeY − 1:

* + The array of luma filter coefficients f[ j ] and the array of luma clipping values c[ j ] corresponding to the filter specified by filtIdx[ x ][ y ] is derived as follows with j = 0..11:
  + If AlfCtbFiltSetIdxY[ xCtb  >>  CtbLog2SizeY ][ yCtb  >>  CtbLog2SizeY ] is less than 16, the following applies:

i = AlfCtbFiltSetIdxY[ xCtb  >>  CtbLog2SizeY ][ yCtb  >>  CtbLog2SizeY ] (1452)

f[ j ] = AlfFixFiltCoeff[ AlfClassToFiltMap[ i ][ filtIdx[ x ][ y ] ] ][ j ] (1453)

c[ j ] = 2BitDepth (1454)

* + Otherwise (AlfCtbFiltSetIdxY[ xCtb  >>  CtbLog2SizeY ][ yCtb  >>  CtbLog2SizeY ] is greater than or equal to 16, the following applies:

i = sh\_alf\_aps\_id\_luma[ AlfCtbFiltSetIdxY[ xCtb  >>  CtbLog2SizeY ][ yCtb  >>  CtbLog2SizeY ] − 16 ]  
 (1455)

f[ j ] = AlfCoeffL[ i ][ filtIdx[ x ][ y ] ][ j ] (1456)

c[ j ] = AlfClipL[ i ][ filtIdx[ x ][ y ] ][ j ] (1457)

* + The luma filter coefficients and clipping values index idx are derived depending on transposeIdx[ x ][ y ] as follows:
  + If transposeIndex[ x ][ y ] is equal to 1, the following applies:

idx[ ] = { 9, 4, 10, 8, 1, 5, 11, 7, 3, 0, 2, 6 } (1458)

* + Otherwise, if transposeIndex[ x ][ y ] is equal to 2, the following applies:

idx[ ] = { 0, 3, 2, 1, 8, 7, 6, 5, 4, 9, 10, 11 } (1459)

* + Otherwise, if transposeIndex[ x ][ y ] is equal to 3, the following applies:

idx[ ] = { 9, 8, 10, 4, 3, 7, 11, 5, 1, 0, 2, 6 } (1460)

* + Otherwise, the following applies:

idx[ ] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 } (1461)

* + The locations ( hx + i, vy + 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:

hx + i = Clip3( 0, pps\_pic\_width\_in\_luma\_samples − 1, xCtb + x + i ) (1462)

vy + j = Clip3( 0, pps\_pic\_height\_in\_luma\_samples − 1, yCtb + y + j ) (1463)

* + The variables clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag are derived by invoking the ALF boundary position derivation process as specified in clause 8.8.5.5 with ( xCtb, yCtb ) and ( x, y ) as inputs.
  + The variables hx + i and vy + j are modified by invoking the ALF sample padding process as specified in clause 8.8.5.6 with ( xCtb, yCtb ), ( hx + i, vy + j ), 0, clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag as input.
  + The variable applyAlfLineBufBoundaryis derived as follows:
  + If the bottom boundary of the current coding tree block is the bottom boundary of current picture and pps\_pic\_height\_in\_luma\_samples − yCtb  <=  CtbSizeY − 4, applyAlfLineBufBoundary is set equal to 0:
  + Otherwise, applyAlfLineBufBoundary is set equal to 1.
  + The vertical sample position offsets y1, y2, y3 and the variable alfShiftY are specified in Table 45 according to the vertical luma sample position y and applyAlfLineBufBoundary.
  + The variable curr is derived as follows:

curr = recPicture[ hx ][ vy ] (1464)

* + The variable sum is derived as follows:

sum = f[ idx[ 0 ] ]   \* (  Clip3( −c[ idx[ 0 ] ], c[ idx[ 0 ] ],     recPicture[ hx ][ vy + y3 ] − curr ) +  
 Clip3( −c[ idx[ 0 ] ], c[ idx[ 0 ] ],     recPicture[ hx ][ vy − y3 ] − curr ) ) +  
 f[ idx[ 1 ] ]   \* (  Clip3( −c[ idx[ 1 ] ], c[ idx[ 1 ] ],     recPicture[ hx + 1 ][ vy + y2 ] − curr ) +  
 Clip3( −c[ idx[ 1 ] ], c[ idx[ 1 ] ],     recPicture[ hx − 1 ][ vy − y2 ] − curr ) ) +  
 f[ idx[ 2 ] ]   \* (  Clip3( −c[ idx[ 2 ] ], c[ idx[ 2 ] ],     recPicture[ hx ][ vy + y2 ] − curr ) +  
 Clip3( −c[ idx[ 2 ] ], c[ idx[ 2 ] ],     recPicture[ hx ][ vy − y2 ] − curr ) ) +  
 f[ idx[ 3 ] ]   \* (  Clip3( −c[ idx[ 3 ] ], c[ idx[ 3 ] ],     recPicture[ hx − 1 ][ vy + y2 ] − curr ) +  
 Clip3( −c[ idx[ 3 ] ], c[ idx[ 3 ] ],     recPicture[ hx + 1 ][ vy − y2 ] − curr ) ) +  
 f[ idx[ 4 ] ]   \* (  Clip3( −c[ idx[ 4 ] ], c[ idx[ 4 ] ],     recPicture[ hx + 2 ][ vy + y1 ] − curr ) +  
 Clip3( −c[ idx[ 4 ] ], c[ idx[ 4 ] ],     recPicture[ hx − 2 ][ vy − y1 ] − curr ) ) +  
 f[ idx[ 5 ] ]   \* (  Clip3( −c[ idx[ 5 ] ], c[ idx[ 5 ] ],     recPicture[ hx + 1 ][ vy + y1 ] − curr ) +  
 Clip3( −c[ idx[ 5 ] ], c[ idx[ 5 ] ],     recPicture[ hx − 1 ][ vy − y1 ] − curr ) ) +  
 f[ idx[ 6 ] ]   \* (  Clip3( −c[ idx[ 6 ] ], c[ idx[ 6 ] ],     recPicture[ hx ][ vy + y1 ] − curr ) +  
 Clip3( −c[ idx[ 6 ] ], c[ idx[ 6 ] ],     recPicture[ hx ][ vy − y1 ] − curr ) ) + (1465)  
 f[ idx[ 7 ] ]   \* (  Clip3( −c[ idx[ 7 ] ], c[ idx[ 7 ] ],     recPicture[ hx − 1 ][ vy + y1 ] − curr ) +  
 Clip3( −c[ idx[ 7 ] ], c[ idx[ 7 ] ],     recPicture[ hx + 1 ][ vy − y1 ] − curr ) ) +  
 f[ idx[ 8 ] ]   \* (  Clip3( −c[ idx[ 8 ] ], c[ idx[ 8 ] ],     recPicture[ hx − 2 ][ vy + y1 ] − curr ) +  
 Clip3( −c[ idx[ 8 ] ], c[ idx[ 8 ] ],     recPicture[ hx + 2 ][ vy − y1 ] − curr ) ) +  
 f[ idx[ 9 ] ]   \* (  Clip3( −c[ idx[ 9 ] ], c[ idx[ 9 ] ],     recPicture[ hx + 3 ][ vy ] − curr ) +  
 Clip3( −c[ idx[ 9 ] ], c[ idx[ 9 ] ],     recPicture[ hx − 3 ][ vy ] − curr ) ) +  
 f[ idx[ 10 ] ] \* (  Clip3( −c[ idx[ 10 ] ], c[ idx[ 10 ] ], recPicture[ hx + 2 ][ vy ] − curr ) +  
 Clip3( −c[ idx[ 10 ] ], c[ idx[ 10 ] ], recPicture[ hx − 2 ][ vy ] − curr ) ) +  
 f[ idx[ 11 ] ] \* (  Clip3( −c[ idx[ 11 ] ], c[ idx[ 11 ] ], recPicture[ hx + 1 ][ vy ] − curr ) +  
 Clip3( −c[ idx[ 11 ] ], c[ idx[ 11 ] ], recPicture[ hx − 1 ][ vy ] − curr ) )

sum = curr + ( ( sum + ( 1  <<  ( alfShiftY − 1) ) )  >>  alfShiftY ) (1466)

* + The modified filtered reconstructed luma picture sample alfPictureL[ xCtb + x ][ yCtb + y ] is derived as follows:

alfPictureL[ xCtb + x ][ yCtb + y ] = Clip3( 0, ( 1  <<  BitDepth ) − 1, sum ) (1467)

Table 45 – Specification of y1, y2, y3 and alfShiftY according to the vertical luma sample position y and applyAlfLineBufBoundary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Condition | alfShiftY | y1 | y2 | y3 |
| ( y  = =  CtbSizeY − 5 | | y  = =  CtbSizeY − 4 ) &&  ( applyAlfLineBufBoundary  = =  1 ) | 10 | 0 | 0 | 0 |
| ( y  = =  CtbSizeY − 6 | | y  = =  CtbSizeY − 3 ) &&  ( applyAlfLineBufBoundary  = =  1 ) | 7 | 1 | 1 | 1 |
| ( y  = =  CtbSizeY − 7 | | y  = =  CtbSizeY − 2 ) &&  ( applyAlfLineBufBoundary  = =  1 ) | 7 | 1 | 2 | 2 |
| otherwise | 7 | 1 | 2 | 3 |

#### Derivation process for ALF transpose and filter index for luma samples

Inputs of this process are:

* a luma location ( xCtb, yCtb ) specifying the top-left sample of the current luma coding tree block relative to the top left sample of the current picture,
* a reconstructed luma picture sample array recPicture prior to the adaptive loop filtering process.

Outputs of this process are

* the classification filter index array filtIdx[ x ][ y ] with x, y = 0..CtbSizeY − 1,
* the transpose index array transposeIdx[ x ][ y ] with x, y = 0..CtbSizeY − 1.

The variables ac[ x ][ y ], sumH[ x ][ y ], sumV[ x ][ y ], sumD0[ x ][ y ], sumD1[ x ][ y ] and sumOfHV[ x ][ y ] with x, y = 0..( CtbSizeY − 1 )  >>  2 are derived as follows:

* + The variables x4 and y4 are set as ( x  <<  2 ) and ( y  <<  2 ), respectively.
* The variables minY, maxY, and ac[ x ][ y ] are derived as follows:
* If y4 is equal to ( CtbSizeY − 8 ) and one of the following condition is true, minY is set equal to −2, maxY is set equal to 3, and ac[ x ][ y ] is set equal to 3.
  + The bottom boundary of the current coding tree block is the bottom boundary of the picture and pps\_pic\_height\_in\_luma\_samples − yCtb > CtbSizeY − 4.
  + The bottom boundary of the current coding tree block is not the bottom boudary of the picture.
* Otherwise, if y4 is equal to ( CtbSizeY − 4 ) and one of the following condition is true, minY is set equal to 0, maxY is set equal to 5, and ac[ x ][ y ] is set equal to 3.
  + The bottom boundary of the current coding tree block is the bottom boundary of the picture and pps\_pic\_height\_in\_luma\_samples − yCtb > CtbSizeY − 4.
  + The bottom boundary of the current coding tree block is not the bottom boudary of the picture
* Otherwise, minY is set equal to −2 and maxY is set equal to 5, and ac[ x ][ y ] is set equal to 2.
  + The variables clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag are derived by invoking the ALF boundary position derivation process as specified in clause 8.8.5.5 with ( xCtb, yCtb ) and ( x4, y4 ) as inputs.
  + The locations ( hx4 + i, vy4 + 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:

hx4 + i = Clip3( 0, pps\_pic\_width\_in\_luma\_samples − 1, xCtb + x4 + i ) (1468)

vy4 + j = Clip3( 0, pps\_pic\_height\_in\_luma\_samples − 1, yCtb + y4 + j ) (1469)

* The variables hx4 + i and vy4 + j are modified by invoking the ALF sample padding process as specified in clause 8.8.5.6 with ( xCtb, yCtb ), ( hx4 + i, vy4 + j ), the variable isChroma set equal to 0, clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag as input.
  + 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:

filtH[ i ][ j ] = Abs( ( recPicture[ hx4 + i ][ vy4 + j ]  <<  1 ) − recPicture[ hx4 +  i − 1 ][ vy4 +j ] − (1470)  
  recPicture[ hx4 + i + 1 ][ vy4 + j ] )

filtV[ i ][ j ] = Abs( ( recPicture[ hx4 + i ][ vy4 + j ]  <<  1 ) − recPicture[ hx4 + i ][ vy4 + j − 1 ] − (1471)  
  recPicture[ hx4 + i ][ vy4 + j + 1 ] )

filtD0[ i ][ j ] = Abs( ( recPicture[ hx4 + i ][ vy4 + j ]  <<  1 ) − recPicture[ hx4 + i − 1 ][ vy4 + j − 1 ] − (1472)  
 recPicture[ hx4 + i + 1 ][ vy4 + j + 1 ] )

filtD1[ i ][ j ] = Abs( ( recPicture[ hx4 + i ][ vy4 + j ]  <<  1 ) − recPicture[ hx4 + i + 1 ][ vy4 + j − 1 ] − (1473)  
 recPicture[ hx4 + i − 1 ][ vy4 + j + 1 ] )

* + 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:

sumH[ x ][ y ] = ΣiΣj filtH[ i ][ j ], with i = −2..5, j = minY..maxY (1474)

sumV[ x ][ y ] = ΣiΣj filtV[ i ][ j ], with i = −2..5, j = minY..maxY (1475)

sumD0[ x ][ y ] = ΣiΣj filtD0[ i ][ j ], with i = −2..5, j = minY..maxY (1476)

sumD1[ x ][ y ] = ΣiΣj filtD1[ i ][ j ], with i = −2..5, j = minY..maxY (1477)

sumOfHV[ x ][ y ] = sumH[ x ][ y ] + sumV[ x ][ y ] (1478)

The classification filter index array filtIdx and transpose index array transposeIdx are derived by the following steps:

1. The variables dir1[ x ][ y ], dir2[ x ][ y ] and dirS[ x ][ y ] with x, y = 0..CtbSizeY − 1 are derived as follows:

* The variables hv1, hv0 and dirHV are derived as follows:
* If sumV[ x  >>  2 ][ y  >>  2 ] is greater than sumH[ x  >>  2 ][ y  >>  2 ], the following applies:

hv1 = sumV[ x  >>  2 ][ y  >>  2 ] (1479)

hv0 = sumH[ x  >>  2 ][ y  >>  2 ]  (1480)

dirHV = 1 (1481)

* Otherwise, the following applies:

hv1 = sumH[ x  >>  2 ][ y  >>  2 ] (1482)

hv0 = sumV[ x  >>  2 ][ y  >>  2 ] (1483)

dirHV = 3 (1484)

* The variables d1, d0 and dirD are derived as follows:
* If sumD0[ x  >>  2 ][ y  >>  2 ] is greater than sumD1[ x  >>  2 ][ y  >>  2 ], the following applies:

d1 = sumD0[ x  >>  2 ][ y  >>  2 ] (1485)

d0 = sumD1[ x  >>  2 ][ y  >>  2 ] (1486)

dirD = 0 (1487)

* Otherwise, the following applies:

d1 = sumD1[ x  >>  2 ][ y  >>  2 ] (1488)

d0 = sumD0[ x  >>  2 ][ y  >>  2 ] (1489)

dirD = 2 (1490)

* The variables hvd1, hvd0, are derived as follows:

hvd1 = ( d1 \* hv0 > hv1 \* d0 ) ? d1 : hv1 (1491)

hvd0 = ( d1 \* hv0 > hv1 \* d0 ) ? d0 : hv0 (1492)

* The variables dirS[ x ][ y ], dir1[ x ][ y ] and dir2[ x ][ y ] derived as follows:

dir1[ x ][ y ] = ( d1 \* hv0 > hv1 \* d0 ) ? dirD : dirHV (1493)

dir2[ x ][ y ] = ( d1 \* hv0 > hv1 \* d0 ) ? dirHV : dirD (1494)

dirS[ x ][ y ] = ( hvd1 \*2 > 9 \* hvd0 ) ? 2 : ( ( hvd1 > 2 \* hvd0 ) ? 1 : 0 ) (1495)

1. The variable avgVar[ x ][ y ] with x, y = 0..CtbSizeY − 1 is derived as follows:

varTab[ ] = { 0, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 4 } (1496)

avgVar[ x ][ y ] = varTab[ Clip3( 0, 15, ( sumOfHV[ x  >>  2 ][ y  >>  2 ] \* (1497)  
 ac[ x  >>  2 ][ y  >>  2 ] )  >>  ( BitDepth − 1 ) ) ]

1. The classification filter index array filtIdx[ x ][ y ] and the transpose index array transposeIdx[ x ][ y ] with x = y = 0..CtbSizeY − 1 are derived as follows:

transposeTable[ ] = { 0, 1, 0, 2, 2, 3, 1, 3 }

transposeIdx[ x ][ y ] = transposeTable[ dir1[ x ][ y ] \* 2 + ( dir2[ x ][ y ]  >>  1 ) ]

filtIdx[ x ][ y ] = avgVar[ x ][ y ]

* When dirS[ x ][ y ] is not equal 0, filtIdx[ x ][ y ] is modified as follows:

filtIdx[ x ][ y ] += ( ( ( dir1[ x ][ y ] & 0x1 )  <<  1 ) + dirS[ x ][ y ] ) \* 5 (1498)

#### Coding tree block filtering process for chroma samples

Inputs of this process are:

* a reconstructed chroma picture sample array recPicture prior to the adaptive loop filtering process,
* a filtered reconstructed chroma picture sample array alfPicture,
* a chroma location ( xCtbC, yCtbC ) specifying the top-left sample of the current chroma coding tree block relative to the top left sample of the current picture,
* an alternative chroma filter index altIdx.

Output of this process is the modified filtered reconstructed chroma picture sample array alfPicture.

The width and height of the current chroma coding tree block ctbWidthC and ctbHeightC is derived as follows:

ctbWidthC = CtbSizeY / SubWidthC (1499)

ctbHeightC = CtbSizeY / SubHeightC (1500)

For the derivation of the filtered reconstructed chroma samples alfPicture[ x ][ y ], each reconstructed chroma sample inside the current chroma coding tree block recPicture[ x ][ y ] is filtered as follows with x = 0..ctbWidthC − 1, y = 0..ctbHeightC − 1:

* + The locations ( hx + i, vy + j ) for each of the corresponding chroma samples ( x, y ) inside the given array recPicture of chroma samples with i, j = −2..2 are derived as follows:

hx + i = Clip3( 0, pps\_pic\_width\_in\_luma\_samples / SubWidthC − 1, xCtbC + x + i ) (1501)

vy + j = Clip3( 0, pps\_pic\_height\_in\_luma\_samples / SubHeightC − 1, yCtbC + y + j ) (1502)

* + The variables clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag are derived by invoking the ALF boundary position derivation process as specified in clause 8.8.5.5 with ( xCtbC \* SubWidthC, yCtbC \* SubHeightC ) and ( x \* SubWidthC, y \*SubHeightC ) as inputs.
  + The variables hx + i and vy + j are modified by invoking the ALF sample padding process as specified in clause 8.8.5.6 with ( xCtb, yCtb ), ( hx + i, vy + j ), the variable isChroma set equal to 1, 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 the picture and pps\_pic\_height\_in\_luma\_samples − ( yCtbC \* SubHeightC ) < CtbSizeY − 4, applyAlfLineBufBoundary is set equal to 0.
  + Otherwise, applyAlfLineBufBoundary is set equal to 1.
  + The vertical sample position offsets y1, y2 and the variable alfShiftC are specified in Table 45 according to the vertical chroma sample position y and applyAlfLineBufBoundary.
  + The variable curr is derived as follows:

curr = recPicture[ hx ][ vy ] (1503)

* + 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 ] = AlfCoeffC[ sh\_alf\_aps\_id\_chroma ][ altIdx ][ j ] (1504)

c[ j ] = AlfClipC[ sh\_alf\_aps\_id\_chroma ][ altIdx ][ j ] (1505)

* + The variable sum is derived as follows:

sum = f[ 0 ] \* (  Clip3( −c[ 0 ], c[ 0 ], recPicture[ hx ][ vy + y2 ] − curr ) +  
 Clip3( −c[ 0 ], c[ 0 ], recPicture[ hx ][ vy− y2 ] − curr ) ) +  
 f[ 1 ] \* ( Clip3( −c[ 1 ], c[ 1 ], recPicture[ hx + 1 ][ vy + y1 ] − curr ) +  
 Clip3( −c[ 1 ], c[ 1 ], recPicture[ hx − 1 ][ vy − y1 ] − curr ) ) +  
 f[ 2 ] \* ( Clip3( −c[ 2 ], c[ 2 ], recPicture[ hx ][ vy + y1 ] − curr ) +  
 Clip3( −c[ 2 ], c[ 2 ], recPicture[ hx ][ vy − y1 ] − curr ) ) + (1506)  
 f[ 3 ] \* ( Clip3( −c[ 3 ], c[ 3 ], recPicture[ hx − 1 ][ vy + y1 ] − curr ) +  
 Clip3( −c[ 3 ], c[ 3 ], recPicture[ hx + 1 ][ vy − y1 ] − curr ) ) +  
 f[ 4 ] \* ( Clip3( −c[ 4 ], c[ 4 ], recPicture[ hx + 2 ][ vy ] − curr ) +  
 Clip3( −c[ 4 ], c[ 4 ], recPicture[ hx − 2 ][ vy ] − curr ) ) +  
 f[ 5 ] \* ( Clip3( −c[ 5 ], c[ 5 ], recPicture[ hx + 1 ][ vy ] − curr ) +  
 Clip3( −c[ 5 ], c[ 5 ], recPicture[ hx − 1 ][ vy ] − curr ) )

sum = curr + ( ( sum + ( 1  <<  ( alfShiftC − 1) ) )  >>  alfShiftC ) (1507)

* + The modified filtered reconstructed chroma picture sample alfPicture[ xCtbC + x ][ yCtbC + y ] is derived as follows:

alfPicture[ xCtbC + x ][ yCtbC + y ] = Clip3( 0, ( 1  <<  BitDepth ) − 1, sum ) (1508)

Table 46 – Specification of y1, y2 and alfShiftC according to the vertical chroma sample position y and applyAlfLineBufBoundary

|  |  |  |  |
| --- | --- | --- | --- |
| Condition | alfShiftC | y1 | y2 |
| ( y = = ctbHeightC − 2 | | y = = ctbHeightC − 3 ) &&  ( applyAlfLineBufBoundary = = 1 ) | 10 | 0 | 0 |
| ( y = = ctbHeightC − 1 | | y = = ctbHeightC − 4 ) &&  ( applyAlfLineBufBoundary = = 1 ) | 7 | 1 | 1 |
| Otherwise | 7 | 1 | 2 |

#### ALF boundary position derivation process

Inputs of this process are:

* + a luma location ( xCtb, yCtb ) specifying the top-left sample of the current luma coding tree block relative to the top left sample of the current picture,
  + a luma location ( x, y ) specifying the current sample relative to the top-left sample of the current luma coding tree block.

Output of this process are:

* + the left vertical boundary position clipLeftPos,
  + the right vertical boundary position clipRightPos,
  + the above horizontal boundary position clipTopPos,
  + the below horizontal boundary position clipBottomPos,
  + the top left boundary flag clipTopLeftFlag,
  + the bottom right boundary flag clipBotRightFlag.

The variables clipLeftPos, clipRightPos, clipTopPos and clipBottomPos are set equal to −128.

The variables clipTopLeftFlag and clipBotRightFlag are both set equal to 0.

The variable clipTopPos is modified as follows:

* + If y − ( CtbSizeY − 4 ) is greater than or equal to 0, the variable clipTopPos is set equal to yCtb + CtbSizeY − 4.
  + Otherwise, if VirtualBoundariesPresentFlag is equal to 1, and yCtb + y − VirtualBoundaryPosY[ n ] is greater than or equal to 0 and less than 3 for any n = 0..NumHorVirtualBoundaries − 1, the following applies:

clipTopPos = VirtualBoundaryPosY[ n ] (1509)

* + Otherwise, if y is less than 3 and one or more of the following conditions are true, the variable clipTopPos is set equal to yCtb:
    - The top boundary of the current coding tree block is the top boundary of the tile, and pps\_loop\_filter\_across\_tiles\_enabled\_flag is equal to 0.
    - The top boundary of the current coding tree block is the top boundary of the slice, and pps\_loop\_filter\_across\_slices\_enabled\_flag is equal to 0.
    - The top boundary of the current coding tree block is the top boundary of the subpicture, and sps\_loop\_filter\_across\_subpic\_enabled\_flag[ CurrSubpicIdx ] is equal to 0.

The variable clipBottomPos is modified as follows:

* + If VirtualBoundariesPresentFlag is equal to 1, VirtualBoundaryPosY[ n ] is not equal to pps\_pic\_height\_in\_luma\_samples − 1 or 0, and VirtualBoundaryPosY[ n ] − yCtb − y is greater than 0 and less than 5 for any n = 0..NumHorVirtualBoundaries − 1, the following applies:

clipBottomPos = VirtualBoundaryPosY[ n ] (1510)

* + Otherwise, if CtbSizeY − 4 − y is greater than 0 and is less than 5, the variable clipBottomPos is set equal to yCtb + CtbSizeY − 4.
  + Otherwise, if CtbSizeY − y is less than 5, and one or more of the following conditions are true, the variable clipBottomPos is set equal to yCtb + CtbSizeY:
    - The bottom boundary of the current coding tree block is the bottom boundary of the tile, and pps\_loop\_filter\_across\_tiles\_enabled\_flag is equal to 0.
    - The bottom boundary of the current coding tree block is the bottom boundary of the slice, and pps\_loop\_filter\_across\_slices\_enabled\_flag is equal to 0.
    - The bottom boundary of the current coding tree block is the bottom boundary of the subpicture, and sps\_loop\_filter\_across\_subpic\_enabled\_flag[ CurrSubpicIdx ] is equal to 0.

The variable clipLeftPos is modified as follows:

* + If VirtualBoundariesPresentFlag is equal to 1, and xCtb + x − VirtualBoundaryPosX[ n ] is greater than or equal to 0 and less than 3 for any n = 0..NumVerVirtualBoundaries − 1, the following applies:

clipLeftPos = VirtualBoundaryPosX[ n ] (1511)

* + 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:

clipRightPos = VirtualBoundaryPosX[ n ] (1512)

* + Otherwise, if CtbSizeY − x is less than 5, and one or more of the following conditions are true, the variable clipRightPos is set equal to xCtb + CtbSizeY:
    - The right boundary of the current coding tree block is the right boundary of the tile, and loop\_filter\_across\_tiless\_enabled\_flag is equal to 0.
    - The right boundary of the current coding tree block is the right boundary of the slice, and pps\_loop\_filter\_across\_slices\_enabled\_flag is equal to 0.
    - The right boundary of the current coding tree block is the right boundary of the subpicture, and sps\_loop\_filter\_across\_subpic\_enabled\_flag[ CurrSubpicIdx ] is equal to 0.

The variable clipTopLeftFlag and clipBotRightFlag are modified as following:

* + 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.

#### 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 luma location ( x, y ) specifying the neighbouring sample relative to the top-left sample of the current picture,
  + a flag isChroma specifiying whether the colour componenet is chroma component or not,
  + the left vertical boundary position clipLeftPos,
  + the right vertical boundary position clipRightPos,
  + the above horizontal boundary position clipTopPos,
  + the below horizontal boundary position clipBottomPos,
  + the top left boundary flag clipTopLeftFlag,
  + the bottom right boundary flag clipBotRightFlag.

Outputs of this process are:

* + modified luma location ( x, y ) specifying the neighbouring sample relative to the top-left sample of the current picture,

The variables picWidth, picHeight, xCtbCur, yCtbCur, CtbSizeHor, CtbSizeVer, topBry, botBry, leftBry and rightBry are derived as follows:

picWidth = isChroma ? pps\_pic\_width\_in\_luma\_samples / SubWidthC :  
 pps\_pic\_width\_in\_luma\_samples (1513)

picHeight = isChroma ? pps\_pic\_height\_in\_luma\_samples / SubHeightC :  
 pps\_pic\_height\_in\_luma\_samples (1514)

xCtbCur = isChroma ? xCtb / SubWidthC : xCtb (1515)

yCtbCur = isChroma ? yCtb / SubHeightC : yCtb (1516)

ctbSizeHor = isChroma ? CtbSizeY / SubWidthC : CtbSizeY (1517)

ctbSizeVer = isChroma ? CtbSizeY / SubHeightC : CtbSizeY (1518)

topBryPos = isChroma ? clipTopPos / SubHeightC : clipTopPos (1519)

botBryPos = isChroma ? clipBottomPos / SubHeightC : clipBottomPos (1520)

leftBryPos = isChroma ? clipLeftPos / SubWidthC : clipLeftPos (1521)

rightBryPos = isChroma ? clipRightPos / SubWidthC : clipRightPos (1522)

The variables ( x , y ) is modified as follows:

* + When topBryPos is not less than 0, the following applies:

y = Clip3( topBryPos, picHeight − 1, y ) (1523)

* + When botBryPos is not less than 0, the following applies:

y = Clip3( 0, botBryPos − 1, y ) (1524)

* + When leftBryPos is not less than 0, the following applies:

x = Clip3( leftBryPos, picWidth − 1,  x ) (1525)

* + When rightBryPos is not less than 0, the following applies:

x = Clip3( 0, rightBryPos − 1, x ) (1526)

* + ( x, y ) is set equal to ( xCtbCur, y ) if all of the followig conditions are true:
    - 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
  + ( x, y ) is set equal to ( xCtbCur + CtbSizeHor − 1, y ) if all of the followig conditions are true:
    - clipBotRightFlag is equal to true
    - botBryPos is less than 0 and rightBryPos is less than 0
    - x is greater than xCtbCur + CtbSizeHor − 1 and y is greater than yCtbCur + CtbSizeVer − 1

#### Cross-component filtering process

Inputs of this process are:

* a reconstructed luma picture sample array recPictureL prior to the luma adaptive loop filtering process,
* a filtered reconstructed chroma picture sample array alfPictureC,
* a chroma location ( xCtbC, yCtbC ) specifying the top-left sample of the current chroma coding tree block relative to the top-left sample of the current picture,
* a CTB width ccAlfWidth in chroma samples,
* a CTB height ccAlfHeight in chroma samples,
* cross-component filter coefficients CcAlfCoeff[ j ], with j = 0..6.

Output of this process is the modified filtered reconstructed chroma picture sample array ccAlfPicture.

For the derivation of the filtered reconstructed chroma samples ccAlfPicture[ xCtbC + x ][ yCtbC + y ], each reconstructed chroma sample inside the current chroma block of samples alfPictureC[ xCtbC + x ][ yCtbC + y ] with x = 0..ccAlfWidth − 1, y = 0..ccAlfHeight − 1, is filtered as follows:

* + The luma location ( xL, yL ) corresponding to the current chroma sample at chroma location ( xCtbC + x, yCtbC + y ) is set equal to ( ( xCtbC + x ) \* SubWidthC, ( yCtbC + y ) \* SubHeightC ).
  + The luma locations ( hx + i, vy + j ) with i = −1..1, j = −1..2 inside the array recPictureL are derived as follows:

hx + i = Clip3( 0, pps\_pic\_width\_in\_luma\_samples − 1, xL + i ) (1527)

vy + j = Clip3( 0, pps\_pic\_height\_in\_luma\_samples − 1, yL + j ) (1528)

* The variables clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag are derived by invoking the ALF boundary position derivation process as specified in clause 8.8.5.5 with ( xCtbC \* SubWidthC, yCtbC \* SubHeightC ) and ( x \* SubWidthC, y \*SubHeightC ) as inputs.
* The variables hx + i and vy + j are modified by invoking the ALF sample padding process as specified in clause 8.8.5.6 with ( xCtbC \* SubWidthC, yCtbC \* SubHeightC ), ( hx + i, vy + j ), the variable isChroma set equal to 0, clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag as input.
* The variable applyAlfLineBufBoundary is derived as follows:
  + If the bottom boundary of the current coding tree block is the bottom boundary of current picture and pps\_pic\_height\_in\_luma\_samples − yCtbC \* SubHeightC is less then 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 = alfPictureC[ xCtbC + x][ yCtbC + y ] (1529)

* + The array of cross-component filter coefficients f[ j ] is derived as follows with j = 0..6:

f[ j ] = CcAlfCoeff[ j ] (1530)

* + The variable sum is derived as follows:

sum =  f[ 0 ] \* ( recPictureL[ hx ][ vy - yP1 ] − recPictureL[ hx ][ vy ] ) +  
 f[ 1 ] \* ( recPictureL[ hx-1 ][ vy ] − recPictureL[ hx ][ vy ] ) +  
 f[ 2 ] \* ( recPictureL[ hx+1 ][ vy ] − recPictureL[ hx ][ vy ] ) + (1531) f[ 3 ] \* ( recPictureL[ hx-1 ][ vy+yP1 ] − recPictureL[ hx ][ vy ] ) +  
 f[ 4 ] \* ( recPictureL[ hx ][ vy+yP1 ] − recPictureL[ hx ][ vy ] ) +  
 f[ 5 ] \* ( recPictureL[ hx+1 ][ vy+yP1 ] − recPictureL[ hx ][ vy ] ) +   
 f[ 6 ] \* ( recPictureL[ hx ][ vy + yP2 ] − recPictureL[ hx ][ vy ] )

scaledSum = Clip3( −( 1 << ( BitDepth − 1 ) ), ( 1 << ( BitDepth  − 1 ) ) − 1, ( sum + 64 ) >> 7) (1532)

sum = ( SubHeightC  = =  1  &&  ( y  = =  CtbSizeY − 3  | |  y  = =  CtbSizeY − 4 ) ) ?  
 curr : curr + scaledSum (1533)

* + The modified filtered reconstructed chroma picture sample ccAlfPicture[ xCtbC + x ][ yCtbC + y ] is derived as follows:

ccAlfPicture[ xCtbC + x ][ yCtbC + y ] = Clip3( 0, ( 1 << BitDepth ) − 1, sum ) (1534)

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 |

# Parsing process

## General

Inputs to this process are bits from the RBSP.

Outputs of this process are syntax element values.

This process is invoked when the descriptor of a syntax element in the syntax tables is equal to ue(v), se(v), or ae(v) (see clause 9.3).

## Parsing process for k-th order Exp-Golomb codes

### General

This process is invoked when the descriptor of a syntax element in the syntax tables is equal to ue(v) or se(v).

Inputs to this process are bits from the RBSP.

Outputs of this process are syntax element values.

Syntax elements coded as ue(v) or se(v) are Exp-Golomb-coded with order k equal to 0. The parsing process for these syntax elements begins with reading the bits starting at the current location in the bitstream up to and including the first non-zero bit, and counting the number of leading bits that are equal to 0. This process is specified as follows:

leadingZeroBits = −1  
for( b = 0; !b; leadingZeroBits++ ) (1535)  
 b = read\_bits( 1 )

The variable codeNum is then assigned as follows:

codeNum = ( 2leadingZeroBits − 1 ) \* 2k + read\_bits( leadingZeroBits + k ) (1536)

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 as specified above 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 xi in Table 48, with i in the range of 0 to leadingZeroBits − 1, inclusive. Each xi is equal to either 0 or 1.

Table 48 – Bit strings with "prefix" and "suffix" bits and assignment to codeNum ranges (informative)

|  |  |
| --- | --- |
| **Bit string form** | **Range of codeNum** |
| 1 | 0 |
| 0 1 x0 | 1..2 |
| 0 0 1 x1 x0 | 3..6 |
| 0 0 0 1 x2 x1 x0 | 7..14 |
| 0 0 0 0 1 x3 x2 x1 x0 | 15..30 |
| 0 0 0 0 0 1 x4 x3 x2 x1 x0 | 31..62 |
| ... | ... |

Table 49 illustrates explicitly the assignment of bit strings to codeNum values.

Table 49 – Exp-Golomb bit strings and codeNum in explicit form and used as ue(v) (informative)

|  |  |
| --- | --- |
| **Bit string** | **codeNum** |
| 1 | 0 |
| 0 1 0 | 1 |
| 0 1 1 | 2 |
| 0 0 1 0 0 | 3 |
| 0 0 1 0 1 | 4 |
| 0 0 1 1 0 | 5 |
| 0 0 1 1 1 | 6 |
| 0 0 0 1 0 0 0 | 7 |
| 0 0 0 1 0 0 1 | 8 |
| 0 0 0 1 0 1 0 | 9 |
| ... | ... |

Depending on the descriptor, the value of a syntax element is derived as follows:

* If the syntax element is coded as ue(v), the value of the syntax element is equal to codeNum.
* Otherwise (the syntax element is coded as se(v)), the value of the syntax element is derived by invoking the mapping process for signed Exp-Golomb codes as specified in clause 9.2.2 with codeNum as input.

### Mapping process for signed Exp-Golomb codes

Input to this process is codeNum as specified in clause 9.2.

Output of this process is a value of a syntax element coded as se(v).

The syntax element is assigned to the codeNum by ordering the syntax element by its absolute value in increasing order and representing the positive value for a given absolute value with the lower codeNum. Table 50 provides the assignment rule.

Table 50 – Assignment of syntax element to codeNum for signed Exp-Golomb coded syntax elements se(v)

|  |  |
| --- | --- |
| **codeNum** | **syntax element value** |
| 0 | 0 |
| 1 | 1 |
| 2 | −1 |
| 3 | 2 |
| 4 | −2 |
| 5 | 3 |
| 6 | −3 |
| k | (−1)k + 1 Ceil( k ÷ 2 ) |

## CABAC parsing process for slice data

### General

Inputs to this process are a request for a value of a syntax element and values of prior parsed syntax elements.

Output of this process is the value of the syntax element.

The initialization process as specified in clause 9.3.2 is invoked when starting the parsing of the CTU syntax specified in clause 7.3.10.2 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 clause 7.3.10.2, sps\_entropy\_coding\_sync\_enabled\_flag is equal to 1, and CtbAddrX is equal to CtbToTileColBd[ CtbAddrX ], the storage process for context variables as specified in clause 9.3.2.3 is invoked with TableStateIdx0Wpp, TableStateIdx1Wpp, and TableMpsValWpp as outputs.

When sps\_palette\_enabled\_flag is equal to 1, the storage process for palette predictor is applied as follows:

* When ending the parsing of the CTU syntax in clause 7.3.10.2 and the decoding process of the last CU in the CTU in clause 8.1.2, sps\_entropy\_coding\_sync\_enabled\_flag is equal to 1 and CtbAddrX is equal to CtbToTileColBd[ CtbAddrX ], the storage process for palette predictor as speficied in clause 9.3.2.6 is invoked.

The whole CABAC parsing process for a syntax element synEl is illustrated in Figure 11.



Figure 11 – Illustration of CABAC parsing process for a syntax element synEl (informative)

### Initialization process

#### General

Outputs of this process are initialized CABAC internal variables.



Figure 12 – Spatial neighbour T that is used to invoke the CTB availability derivation process relative to the current CTB (informative)

The context variables of the arithmetic decoding engine are initialized as follows:

– If the CTU is the first CTU in a slice or tile, the initialization process for context variables is invoked as specified in clause 9.3.2.2 and the array PredictorPaletteSize[ chType ], with chType = 0, 1, is initialized to 0.

– Otherwise, if sps\_entropy\_coding\_sync\_enabled\_flag is equal to 1 and CtbAddrX is equal to CtbToTileColBd[ CtbAddrX ], the following applies:

– The location ( xNbT, yNbT ) of the top-left luma sample of the spatial neighbouring block T (Figure 12) is derived using the location ( x0, y0 ) of the top-left luma sample of the current CTB as follows:

( xNbT, yNbT ) = ( x0, y0 − CtbSizeY ) (1537)

– The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location ( xCurr, yCurr ) set equal to ( x0, y0 ), the neighbouring location ( xNbY, yNbY ) set equal to ( xNbT, yNbT ), checkPredModeY set equal to FALSE, and cIdx set equal to 0 as inputs, and the output is assigned to availableFlagT.

– The synchronization process for context variables is invoked as follows:

– If availableFlagT is equal to 1, the following applies:

– The synchronization process for context variables as specified in clause 9.3.2.4 is invoked with TableStateIdx0Wpp, TableStateIdx1Wpp, TableMpsValWpp as inputs.

– When sps\_palette\_enabled\_flag is equal to 1, the synchronization process for palette predictor as specified in clause 9.3.2.7 is invoked.

– Otherwise, the initialization process for context variables is invoked as specified in clause 9.3.2.2 and the array PredictorPaletteSize[ chType ], with chType = 0, 1, is initialized to 0.

– Otherwise, the initialization process for context variables is invoked as specified in clause 9.3.2.2 and the array PredictorPaletteSize[ chType ], with chType = 0, 1, is initialized to 0. [Ed. (YK): Check whether this can be merged with the condition "If the CTU is the first CTU in a brick", as the operation is the same. This is also the case in HEVC.]

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 the flowchart of Figure 13.



Figure 13 – Illustration of CABAC initialization process (informative)

#### 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 125 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 7.3.10.1 through 7.3.10.11, except end\_of\_slice\_one\_bit, end\_of\_tile\_one\_bit, and end\_of\_subset\_one\_bit.
* From the 6 bit table entry initValue, the two 3 bit variables slopeIdx and offsetIdx are derived as follows:

slopeIdx = initValue >> 3  
offsetIdx = initValue & 7 (1538)

* The variables m and n, used in the initialization of context variables, are derived from slopeIdx and offsetIdx as follows:

m = slopeIdx − 4  
n = ( offsetIdx \* 18 ) + 1 (1539)

* The two values assigned to pStateIdx0 and pStateIdx1 for the initialization are derived from SliceQpY, which is derived in Equation 144. Given the variables m and n, the initialization is specified as follows:

preCtxState = Clip3( 1, 127, ( ( m \* ( Clip3( 0, 51, SliceQpY ) − 16 ) ) >> 1 ) + n ) (1540)

* The two values assigned to pStateIdx0 and pStateIdx1 for the initialization are derived as follows:

pStateIdx0 = preCtxState << 3  
pStateIdx1 = preCtxState << 7 (1541)

NOTE 1 – The variables pStateIdx0 and pStateIdx1 correspond to probability state indices as further described in subclause 9.3.4.3.

In Table 51, the ctxIdx for which initialization is needed for each of the three initialization types, specified by the variable initType, are listed. Also listed is the table number that includes the values of initValue needed for the initialization for each value of ctxIdx. For P and B slice types, the derivation of initType depends on the value of the sh\_cabac\_init\_flag syntax element. The variable initType is derived as follows:

if( sh\_slice\_type = = I )  
 initType = 0  
else if( sh\_slice\_type = = P )  
 initType = sh\_cabac\_init\_flag ? 2 : 1 (1542)  
else  
 initType = sh\_cabac\_init\_flag ? 1 : 2

| Table 51 – Association of ctxIdx and syntax elements for each initializationType in the initialization process | | | | | |
| --- | --- | --- | --- | --- | --- |
| **Syntax structure** | **Syntax element** | **ctxTable** | **initType** | | | |
| **0** | **1** | **2** | |
| coding\_tree\_unit( ) | alf\_ctb\_flag[ ][ ][ ] | Table 52 | 0..8 | 9..17 | 18..26 | |
| alf\_use\_aps\_flag | Table 53 | 0 | 1 | 2 | |
| alf\_ctb\_filter\_alt\_idx[ ][ ][ ] | Table 56 | 0..1 | 2..3 | 4..5 | |
| alf\_ctb\_cc\_cb\_idc[ ][ ] | Table 54 | 0..2 | 3..5 | 6..8 | |
| alf\_ctb\_cc\_cr\_idc[ ][ ] | Table 55 | 0..2 | 3..5 | 6..8 | |
| sao( ) | sao\_merge\_left\_flag sao\_merge\_up\_flag | Table 57 | 0 | 1 | 2 | |
| sao\_type\_idx\_luma sao\_type\_idx\_chroma | Table 58 | 0 | 1 | 2 | |
| coding\_tree( ) | split\_cu\_flag | Table 59 | 0..8 | 9..17 | 18..26 | |
| split\_qt\_flag | Table 60 | 0..5 | 6..11 | 12..17 | |
| mtt\_split\_cu\_vertical\_flag | Table 61 | 0..4 | 5..9 | 10..14 | |
| mtt\_split\_cu\_binary\_flag | Table 62 | 0..3 | 4..7 | 8..11 | |
| mode\_constraint\_flag | Table 63 | 0..1 | 2..3 | 4..5 | |
| coding\_unit( ) | cu\_skip\_flag[ ][ ] | Table 64 | 0..2 | 3..5 | 6..8 | |
| pred\_mode\_ibc\_flag | Table 65 | 0..3 | 4..7 | 8..11 | |
| 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\_bdpcm\_chroma\_flag | Table 76 | 0 | 1 | 2 | |
| intra\_bdpcm\_ chroma\_dir\_flag | Table 77 | 0 | 1 | 2 | |
| cclm\_mode\_flag | Table 78 | 0 | 1 | 2 | |
| cclm\_mode\_idx | Table 79 | 0 | 1 | 2 | |
| intra\_chroma\_pred\_mode | Table 80 | 0 | 1 | 2 | |
| general\_merge\_flag[ ][ ] | Table 81 | 0 | 1 | 2 | |
| inter\_pred\_idc[ x0 ][ y0 ] | Table 82 |  | 0..5 | 6..11 | |
| inter\_affine\_flag[ ][ ] | Table 83 |  | 0..2 | 3..5 | |
| cu\_affine\_type\_flag[ ][ ] | Table 84 |  | 0 | 1 | |
| sym\_mvd\_flag[ ][ ] | Table 85 |  | 0 | 1 | |
| ref\_idx\_l0[ ][ ], ref\_idx\_l1[ ][ ] | Table 86 |  | 0..1 | 2..3 | |
| mvp\_l0\_flag[ ][ ], mvp\_l1\_flag[ ][ ] | Table 87 | 0..1 | 2..3 | 4..5 | |
| amvr\_flag[ ][ ] | Table 88 |  | 0..1 | 2..3 | |
| amvr\_precision\_idx[ ][ ] | Table 89 | 0..1 | 2..3 | 4..5 | |
| bcw\_idx[ ][ ] | Table 90 |  | 0 | 1 | |
| cu\_coded\_flag | Table 91 | 0 | 1 | 2 | |
| cu\_sbt\_flag | Table 92 |  | 0..1 | 2..3 | |
| cu\_sbt\_quad\_flag | Table 93 |  | 0 | 1 | |
| cu\_sbt\_horizontal\_flag | Table 94 |  | 0..2 | 3..5 | |
| cu\_sbt\_pos\_flag | Table 95 |  | 0 | 1 | |
| lfnst\_idx | Table 96 | 0..2 | 3..4 | 6..8 | |
| mts\_idx | Table 97 | 0..3 | 4..7 | 8..11 | |
| palette\_coding( ) | copy\_above\_palette\_indices\_flag | Table 98 | 0..1 | 2..3 | 4..5 | |
| palette\_transpose\_flag | Table 99 | 0..1 | 2..3 | 4..5 | |
| run\_copy\_flag | Table 100 | 0..7 | 8..15 | 16..23 | |
| merge\_data( ) | regular\_merge\_flag[ ][ ] | Table 101 |  | 0..1 | 2..3 | |
| mmvd\_merge\_flag[ ][ ] | Table 102 |  | 0 | 1 | |
| mmvd\_cand\_flag[ ][ ] | Table 103 |  | 0 | 1 | |
| mmvd\_distance\_idx[ ][ ] | Table 104 |  | 0 | 1 | |
| ciip\_flag[ ][ ] | Table 105 |  | 0 | 1 | |
| merge\_subblock\_flag[ ][ ] | Table 106 |  | 0..2 | 3..5 | |
| merge\_subblock\_idx[ ][ ] | Table 107 |  | 0 | 1 | |
| merge\_idx[ ][ ]  merge\_gpm\_idx0[ ][ ] merge\_gpm\_idx1[ ][ ] | Table 108 | 0 | 1 | 2 | |
| mvd\_coding( ) | abs\_mvd\_greater0\_flag[ ] | Table 109 | 0 | 1 | 2 | |
| abs\_mvd\_greater1\_flag[ ] | Table 110 | 0 | 1 | 2 | |
| transform\_unit( ) | tu\_y\_coded\_flag[ ][ ] | Table 111 | 0..3 | 4..7 | 8..11 | |
| tu\_cb\_coded\_flag[ ][ ] | Table 112 | 0..1 | 2..3 | 4..5 | |
| tu\_cr\_coded\_flag[ ][ ] | Table 113 | 0..2 | 3..5 | 6..8 | |
| cu\_qp\_delta\_abs | Table 114 | 0..1 | 2..3 | 4..5 | |
| cu\_chroma\_qp\_offset\_flag | Table 115 | 0 | 1 | 2 | |
| cu\_chroma\_qp\_offset\_idx | Table 116 | 0 | 1 | 2 | |
| transform\_skip\_flag[ ][ ][ ] | Table 117 | 0..1 | 2..3 | 4..5 | |
| tu\_joint\_cbcr\_residual\_flag[ ][ ] | Table 118 | 0..2 | 3..5 | 6..8 | |
| residual\_coding( ) | last\_sig\_coeff\_x\_prefix | Table 119 | 0..22 | 23..45 | 46..68 | |
| last\_sig\_coeff\_y\_prefix | Table 120 | 0..22 | 23..45 | 46..68 | |
| sb\_coded\_flag[ ][ ] | Table 121 | 0..7 | 8..15 | 16..23 | |
| sig\_coeff\_flag[ ][ ] | Table 122 | 0..62 | 63..125 | 126..188 | |
| par\_level\_flag[ ] | Table 123 | 0..32 | 33..65 | 66..98 | |
| abs\_level\_gtx\_flag[ ][ ] | Table 124 | 0..73 | 74..147 | 148..220 | |
| coeff\_sign\_flag[ ] | Table 125 | 0..5 | 6..11 | 12..17 | |

Table 52 – Specification of initValue and shiftIdx for ctxIdx of alf\_ctb\_flag

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of alf\_ctb\_flag** | | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 53 – Specification of initValue and shiftIdx for ctxIdx of alf\_use\_aps\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of alf\_use\_aps\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 54 – Specification of initValue and shiftIdx for ctxIdx of alf\_ctb\_cc\_cb\_idc

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of alf\_ctb\_cc\_cb\_idc** | | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 55 – Specification of initValue and shiftIdx for ctxIdx of alf\_ctb\_cc\_cr\_idc

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of alf\_ctb\_cc\_cr\_idc** | | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 56 – Specification of initValue and shiftIdx for ctxIdx of alf\_ctb\_filter\_alt\_idx

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of alf\_ctb\_filter\_alt\_idx** | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** |
| **initValue** | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 |

Table 57 – Specification of initValue and shiftIdx for ctxIdx of sao\_merge\_left\_flag and sao\_merge\_up\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of sao\_merge\_left\_flag and sao\_merge\_up\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 58 – Specification of initValue and shiftIdx for ctxInc 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** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 59 – Specification of initValue and shiftIdx for ctxInc 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** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** |  |  |  |  |  |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |  |  |  |  |  |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |  |  |  |

Table 60 – Specification of initValue and shiftIdx for ctxInc of split\_qt\_flag

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of split\_qt\_flag** | | | | | | | | | | | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 61 – Specification of initValue and shiftIdx for ctxInc 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** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 62 – Specification of initValue and shiftIdx for ctxInc 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** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 63 – Specification of initValue and shiftIdx for ctxIdx of mode\_constraint\_flag

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of mode\_constraint\_flag** | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** |
| **initValue** | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 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** | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 4 | 0 | 0 | 1 | 0 | 0 | 1 |

Table 65 – Specification of initValue and shiftIdx for ctxInc of pred\_mode\_ibc\_flag

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of pred\_mode\_ibc\_flag** | | | | | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

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** | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 |

Table 67 – Specification of initValue and shiftIdx for ctxInc of pred\_mode\_plt\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of pred\_mode\_plt\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 68 – Specification of initValue and shiftIdx for ctxInc of cu\_act\_enabled\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of cu\_act\_enabled\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 69 – Specification of initValue and shiftIdx for ctxInc of intra\_bdpcm\_luma\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of intra\_bdpcm\_luma\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 70 – Specification of initValue and shiftIdx for ctxInc of intra\_bdpcm\_luma\_dir\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of intra\_bdpcm\_luma\_dir\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 71 – Specification of initValue and shiftIdx for ctxInc of intra\_mip\_flag

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of intra\_mip\_flag** | | | | | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

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** | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 |

Table 73 – Specification of initValue and shiftIdx for ctxInc of intra\_subpartitions\_mode\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of intra\_subpartitions\_mode\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 74 – Specification of initValue and shiftIdx for ctxInc of intra\_subpartitions\_split\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of intra\_subpartitions\_split\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 75 – Specification of initValue and shiftIdx for ctxInc of intra\_luma\_mpm\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of intra\_luma\_mpm\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 76 – Specification of initValue and shiftIdx for ctxInc of intra\_bdpcm\_chroma\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of intra\_bdpcm\_chroma\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 77 – Specification of initValue and shiftIdx for ctxInc of intra\_bdpcm\_chroma\_dir\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of intra\_bdpcm\_chroma\_dir\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 78 – Specification of initValue and shiftIdx for ctxInc of cclm\_mode\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of cclm\_mode\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 79 – Specification of initValue and shiftIdx for ctxInc of cclm\_mode\_idx

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of cclm\_mode\_idx** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 80 – Specification of initValue and shiftIdx for ctxInc of intra\_chroma\_pred\_mode

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of intra\_chroma\_pred\_mode** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 81 – Specification of initValue and shiftIdx for ctxInc of general\_merge\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of general\_merge\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 82 – Specification of initValue and shiftIdx for ctxInc of inter\_pred\_idc

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of inter\_pred\_idc** | | | | | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 83 – 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** | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 |

Table 84 – Specification of initValue and shiftIdx for ctxInc of cu\_affine\_type\_flag

|  |  |  |
| --- | --- | --- |
| **Initialization variable** | **ctxIdx of cu\_affine\_type\_flag** | |
| **0** | **1** |
| **initValue** | EP | EP |
| **shiftIdx** | 0 | 0 |

Table 85 – Specification of initValue and shiftIdx for ctxInc of sym\_mvd\_flag

|  |  |  |
| --- | --- | --- |
| **Initialization variable** | **ctxIdx of sym\_mvd\_flag** | |
| **0** | **1** |
| **initValue** | EP | EP |
| **shiftIdx** | 0 | 0 |

Table 86 – 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** | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 |

Table 87 – 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** | **3** | **4** | **5** |
| **initValue** | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 |

Table 88 – Specification of initValue and shiftIdx for ctxIdx of amvr\_flag

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of amvr\_flag** | | | |
| **0** | **1** | **2** | **3** |
| **initValue** | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 |

Table 89 – Specification of initValue and shiftIdx for ctxIdx of amvr\_precision\_idx

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of amvr\_precision\_idx** | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** |
| **initValue** | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 |

Table 90 – Specification of initValue and shiftIdx for ctxInc of bcw\_idx

|  |  |  |
| --- | --- | --- |
| **Initialization variable** | **ctxIdx of bcw\_idx** | |
| **0** | **1** |
| **initValue** | EP | EP |
| **shiftIdx** | 0 | 0 |

Table 91 – Specification of initValue and shiftIdx for ctxInc of cu\_coded\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of cu\_coded\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 92 – Specification of initValue and shiftIdx for ctxIdx of cu\_sbt\_flag

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of cu\_sbt\_flag** | | | |
| **0** | **1** | **2** | **3** |
| **initValue** | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 |

Table 93 – Specification of initValue and shiftIdx for ctxInc of cu\_sbt\_quad\_flag

|  |  |  |
| --- | --- | --- |
| **Initialization variable** | **ctxIdx of cu\_sbt\_quad\_flag** | |
| **0** | **1** |
| **initValue** | EP | EP |
| **shiftIdx** | 0 | 0 |

Table 94 – 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** | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 4 | 0 | 0 | 1 |

Table 95 – Specification of initValue and shiftIdx for ctxInc of cu\_sbt\_pos\_flag

|  |  |  |
| --- | --- | --- |
| **Initialization variable** | **ctxIdx of cu\_sbt\_pos\_flag** | |
| **0** | **1** |
| **initValue** | EP | EP |
| **shiftIdx** | 0 | 0 |

Table 96 – Specification of initValue and shiftIdx for ctxIdx of lfnst\_idx

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of lfnst\_idx** | | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 97 – Specification of initValue and shiftIdx for ctxInc of mts\_idx

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of mts\_idx** | | | | | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 98 – 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** | **3** | **4** | **5** |
| **initValue** | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 |

Table 99 – Specification of initValue and shiftIdx for ctxIdx of palette\_transpose\_flag

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of palette\_transpose\_flag** | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** |
| **initValue** | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 |

Table 100 – Specification of initValue and shiftIdx for ctxInc 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** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** |  |  |  |  |  |  |  |  |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP |  |  |  |  |  |  |  |  |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |  |  |  |  |  |  |

Table 101 – Specification of initValue and shiftIdx for ctxIdx of regular\_merge\_flag

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of regular\_merge\_flag** | | | |
| **0** | **1** | **2** | **3** |
| **initValue** | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 |

Table 102 – Specification of initValue and shiftIdx for ctxInc of mmvd\_merge\_flag

|  |  |  |
| --- | --- | --- |
| **Initialization variable** | **ctxIdx of mmvd\_merge\_flag** | |
| **0** | **1** |
| **initValue** | EP | EP |
| **shiftIdx** | 0 | 0 |

Table 103 – Specification of initValue and shiftIdx for ctxInc of mmvd\_cand\_flag

|  |  |  |
| --- | --- | --- |
| **Initialization variable** | **ctxIdx of mmvd\_cand\_flag** | |
| **0** | **1** |
| **initValue** | EP | EP |
| **shiftIdx** | 0 | 0 |

Table 104 – Specification of initValue and shiftIdx for ctxInc of mmvd\_distance\_idx

|  |  |  |
| --- | --- | --- |
| **Initialization variable** | **ctxIdx of mmvd\_distance\_idx** | |
| **0** | **1** |
| **initValue** | EP | EP |
| **shiftIdx** | 0 | 0 |

Table 105 – Specification of initValue and shiftIdx for ctxInc of ciip\_flag

|  |  |  |
| --- | --- | --- |
| **Initialization variable** | **ctxIdx of ciip\_flag** | |
| **0** | **1** |
| **initValue** | EP | EP |
| **shiftIdx** | 0 | 0 |

Table 106 – Specification of initValue and shiftIdx for ctxIdx of merge\_subblock\_flag

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of merge\_subblock\_flag** | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** |
| **initValue** | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 4 | 0 | 0 | 1 |

Table 107 – Specification of initValue and shiftIdx for ctxInc of merge\_subblock\_idx

|  |  |  |
| --- | --- | --- |
| **Initialization variable** | **ctxIdx of merge\_subblock\_idx** | |
| **0** | **1** |
| **initValue** | EP | EP |
| **shiftIdx** | 0 | 0 |

Table 108 – Specification of initValue and shiftIdx for ctxInc of merge\_idx, merge\_gpm\_idx0, and merge\_gpm\_idx1.

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of merge\_idx, merge\_gpm\_idx0, merge\_gpm\_idx1** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 109 – Specification of initValue and shiftIdx for ctxInc of abs\_mvd\_greater0\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of abs\_mvd\_greater0\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 110 – Specification of initValue and shiftIdx for ctxInc of abs\_mvd\_greater1\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of abs\_mvd\_greater1\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 111 – Specification of initValue and shiftIdx for ctxInc of tu\_y\_coded\_flag

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of tu\_y\_coded\_flag** | | | | | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 112 – Specification of initValue and shiftIdx for ctxInc of tu\_cb\_coded\_flag

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of tu\_cb\_coded\_flag** | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** |
| **initValue** | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 |

Table 113 – Specification of initValue and shiftIdx for ctxIdx of tu\_cr\_coded\_flag

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of tu\_cr\_coded\_flag** | | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 114 – Specification of initValue and shiftIdx for ctxIdx of cu\_qp\_delta\_abs

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of cu\_qp\_delta\_abs** | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** |
| **initValue** | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 |

Table 115 – Specification of initValue and shiftIdx for ctxInc of cu\_chroma\_qp\_offset\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of cu\_chroma\_qp\_offset\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 116 – Specification of initValue and shiftIdx for ctxInc of cu\_chroma\_qp\_offset\_idx

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of cu\_chroma\_qp\_offset\_idx** | | |
| **0** | **1** | **2** |
| **initValue** | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 |

Table 117 – Specification of initValue and shiftIdx for ctxInc of transform\_skip\_flag

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of transform\_skip\_flag** | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** |
| **initValue** | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 |

Table 118 – Specification of initValue and shiftIdx for ctxIdx of tu\_joint\_cbcr\_residual\_flag

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of tu\_joint\_cbcr\_residual\_flag** | | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 4 | 0 | 0 | 1 | 0 | 0 | 1 |

Table 119 – Specification of initValue and shiftIdx for ctxInc of last\_sig\_coeff\_x\_prefix

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of last\_sig\_coeff\_x\_prefix** | | | | | | | | | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **32** | **33** | **34** | **35** | **36** | **37** | **38** | **39** | **40** | **41** | **42** | **43** | **44** | **45** | **46** | **47** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **48** | **49** | **50** | **51** | **52** | **53** | **54** | **55** | **56** | **57** | **58** | **59** | **60** | **61** | **62** | **63** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **64** | **65** | **66** | **67** | **68** |  |  |  |  |  |  |  |  |  |  |  |
| **initValue** | EP | EP | EP | EP | EP |  |  |  |  |  |  |  |  |  |  |  |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 |  |  |  |  |  |  |  |  |  |  |  |

Table 120 – Specification of initValue and shiftIdx for ctxInc of last\_sig\_coeff\_y\_prefix

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of last\_sig\_coeff\_y\_prefix** | | | | | | | | | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **32** | **33** | **34** | **35** | **36** | **37** | **38** | **39** | **40** | **41** | **42** | **43** | **44** | **45** | **46** | **47** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **48** | **49** | **50** | **51** | **52** | **53** | **54** | **55** | **56** | **57** | **58** | **59** | **60** | **61** | **62** | **63** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **64** | **65** | **66** | **67** | **68** |  |  |  |  |  |  |  |  |  |  |  |
| **initValue** | EP | EP | EP | EP | EP |  |  |  |  |  |  |  |  |  |  |  |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 |  |  |  |  |  |  |  |  |  |  |  |

Table 121 – Specification of initValue and shiftIdx for ctxInc of sb\_coded\_flag

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of sb\_coded\_flag** | | | | | | | | | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP |  |  |  |  |  |  |  |  |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |  |  |  |  |  |  |

Table 122 – Specification of initValue and shiftIdx for ctxInc of sig\_coeff\_flag

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of sig\_coeff\_flag** | | | | | | | | | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **32** | **33** | **34** | **35** | **36** | **37** | **38** | **39** | **40** | **41** | **42** | **43** | **44** | **45** | **46** | **47** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **48** | **49** | **50** | **51** | **52** | **53** | **54** | **55** | **56** | **57** | **58** | **59** | **60** | **61** | **62** | **63** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **64** | **65** | **66** | **67** | **68** | **69** | **70** | **71** | **72** | **73** | **74** | **75** | **76** | **77** | **78** | **79** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **80** | **81** | **82** | **83** | **84** | **85** | **86** | **87** | **88** | **89** | **90** | **91** | **92** | **93** | **94** | **95** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **96** | **97** | **98** | **99** | **100** | **101** | **102** | **103** | **104** | **105** | **106** | **107** | **108** | **109** | **110** | **111** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **112** | **113** | **114** | **115** | **116** | **117** | **118** | **119** | **120** | **121** | **122** | **123** | **124** | **125** | **126** | **127** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **128** | **129** | **130** | **131** | **132** | **133** | **134** | **135** | **136** | **137** | **138** | **139** | **140** | **141** | **142** | **143** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **128** | **129** | **130** | **131** | **132** | **133** | **134** | **135** | **136** | **137** | **138** | **139** | **140** | **141** | **142** | **143** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **160** | **161** | **162** | **163** | **164** | **165** | **166** | **167** | **168** | **169** | **170** | **171** | **172** | **173** | **174** | **175** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **176** | **177** | **178** | **179** | **180** | **181** | **182** | **183** | **184** | **185** | **186** | **187** | **188** |  |  |  |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |  |  |  |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |  |

Table 123 – Specification of initValue and shiftIdx for ctxInc of par\_level\_flag

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of par\_level\_flag** | | | | | | | | | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **32** | **33** | **34** | **35** | **36** | **37** | **38** | **39** | **40** | **41** | **42** | **43** | **44** | **45** | **46** | **47** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **48** | **49** | **50** | **51** | **52** | **53** | **54** | **55** | **56** | **57** | **58** | **59** | **60** | **61** | **62** | **63** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **64** | **65** | **66** | **67** | **68** | **69** | **70** | **71** | **72** | **73** | **74** | **75** | **76** | **77** | **78** | **79** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **80** | **81** | **82** | **83** | **84** | **85** | **86** | **87** | **88** | **89** | **90** | **91** | **92** | **93** | **94** | **95** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **96** | **97** | **98** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **initValue** | EP | EP | EP |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 124 – Specification of initValue and shiftIdx for ctxInc of abs\_level\_gtx\_flag

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of abs\_level\_gtx\_flag** | | | | | | | | | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **32** | **33** | **34** | **35** | **36** | **37** | **38** | **39** | **40** | **41** | **42** | **43** | **44** | **45** | **46** | **47** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **48** | **49** | **50** | **51** | **52** | **53** | **54** | **55** | **56** | **57** | **58** | **59** | **60** | **61** | **62** | **63** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **64** | **65** | **66** | **67** | **68** | **69** | **70** | **71** | **72** | **73** | **74** | **75** | **76** | **77** | **78** | **79** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **80** | **81** | **82** | **83** | **84** | **85** | **86** | **87** | **88** | **89** | **90** | **91** | **92** | **93** | **94** | **95** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **96** | **97** | **98** | **99** | **100** | **101** | **102** | **103** | **104** | **105** | **106** | **107** | **108** | **109** | **110** | **111** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **112** | **113** | **114** | **115** | **116** | **117** | **118** | **119** | **120** | **121** | **122** | **123** | **124** | **125** | **126** | **127** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **128** | **129** | **130** | **131** | **132** | **133** | **134** | **135** | **136** | **137** | **138** | **139** | **140** | **141** | **142** | **143** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **128** | **129** | **130** | **131** | **132** | **133** | **134** | **135** | **136** | **137** | **138** | **139** | **140** | **141** | **142** | **143** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **160** | **161** | **162** | **163** | **164** | **165** | **166** | **167** | **168** | **169** | **170** | **171** | **172** | **173** | **174** | **175** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **176** | **177** | **178** | **179** | **180** | **181** | **182** | **183** | **184** | **185** | **186** | **187** | **188** | **189** | **190** | **191** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **192** | **193** | **194** | **195** | **196** | **197** | **198** | **199** | **200** | **201** | **202** | **203** | **204** | **205** | **206** | **207** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | **208** | **209** | **210** | **211** | **212** | **213** | **214** | **215** | **216** | **217** | **218** | **219** | **220** |  |  |  |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |  |  |  |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |  |

Table 125 – Specification of initValue and shiftIdx for ctxInc of coeff\_sign\_flag

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of coeff\_sign\_flag** | | | | | | | | | | | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** |
| **initValue** | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP | EP |
| **shiftIdx** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

#### Storage process for context variables

Inputs to this process are:

– The CABAC context variables indexed by ctxTable and ctxIdx.

Outputs of this process are:

– The variables tableStateSync0, tableStateSync1, and tableMPSSync containing the values of the variables pStateIdx0, pStateIdx1, and valMps used in the initialization process of context variables that are assigned to all syntax elements in clauses 7.3.10.1 through 7.3.10.11, except end\_of\_slice\_one\_bit, end\_of\_tile\_one\_bit, and end\_of\_subset\_one\_bit.

For each context variable, the corresponding entries pStateIdx0, pStateIdx1, and valMps of tables tableStateSync0, tableStateSync1, and tableMPSSync are initialized to the corresponding pStateIdx0, pStateIdx1, and valMps.

The storage process for context variables is illustrated in the flowchart of Figure 14.



Figure 14 – Illustration of CABAC storage process (informative)

#### Synchronization process for context variables

Inputs to this process are:

– The variables tableStateSync0, tableStateSync1, and tableMPSSync containing the values of the variables pStateIdx0, pStateIdx1, and valMps used in the storage process of context variables that are assigned to all syntax elements in clauses 7.3.10.1 through 7.3.10.11, except end\_of\_slice\_one\_bit, end\_of\_tile\_one\_bit, and end\_of\_subset\_one\_bit.

Outputs of this process are:

– The initialized CABAC context variables indexed by ctxTable and ctxIdx.

For each context variable, the corresponding context variables pStateIdx0, pStateIdx1, and valMps are initialized to the corresponding entries pStateIdx0, pStateIdx1, and valMps of tables tableStateSync0, tableStateSync1, and tableMPSSync.

#### Initialization process for the arithmetic decoding engine

Outputs of this process are the initialized decoding engine registers ivlCurrRange and ivlOffset both in 16 bit register precision.

The status of the arithmetic decoding engine is represented by the variables ivlCurrRange and ivlOffset. In the initialization procedure of the arithmetic decoding process, ivlCurrRange is set equal to 510 and ivlOffset is set equal to the value returned from read\_bits( 9 ) interpreted as a 9 bit binary representation of an unsigned integer with the most significant bit written first.

The bitstream shall not contain data that result in a value of ivlOffset being equal to 510 or 511.

NOTE – The description of the arithmetic decoding engine in this Specification utilizes 16 bit register precision. However, a minimum register precision of 9 bits is required for storing the values of the variables ivlCurrRange and ivlOffset after invocation of the arithmetic decoding process (DecodeBin) as specified in subclause 9.3.4.3. The arithmetic decoding process for a binary decision (DecodeDecision) as specified in subclause 9.3.4.3.2 and the decoding process for a binary decision before termination (DecodeTerminate) as specified in subclause 9.3.4.3.5 require a minimum register precision of 9 bits for the variables ivlCurrRange and ivlOffset. The bypass decoding process for binary decisions (DecodeBypass) as specified in subclause 9.3.4.3.4 requires a minimum register precision of 10 bits for the variable ivlOffset and a minimum register precision of 9 bits for the variable ivlCurrRange.

#### Storage process for palette predictor

This process stores the values of the arrays PredictorPaletteSize and PredictorPaletteEntries in the arrays TablePaletteSizeWpp and TablePaletteEntriesWpp as follows:

for( cIdx = 0; cIdx < 3; cIdx++ ) {  
 chType = cIdx = = 0 ? 0 : 1  
 TablePaletteSizeWpp[ chType ] = PredictorPaletteSize[ chType ] (1543)  
 for( i = 0; i < PredictorPaletteSize[ chType ]; i++ )   
 TablePaletteEntriesWpp[ cIdx ][ i ] = PredictorPaletteEntries[ cIdx ][ i ]  
}

#### Synchronization process for palette predictor

This process synchronizes the values of the arrays PredictorPaletteSize and PredictorPaletteEntries in the arrays TablePaletteSizeWpp and TablePaletteEntriesWpp as follows:

for( cIdx = 0; cIdx < 3; cIdx++ ) {  
 chType = cIdx = = 0 ? 0 : 1  
 PredictorPaletteSize[ chType ] = TablePaletteSizeWpp[ chType ] (1544)  
 for( i = 0; i < PredictorPaletteSize[ chType ]; i++ )   
 PredictorPaletteEntries[ cIdx ][ i ] = TablePaletteEntriesWpp[ cIdx ][ i ]  
}

### Binarization process

#### General

Input to this process is a request for a syntax element.

Output of this process is the binarization of the syntax element.

Table 126 specifies the type of binarization process associated with each syntax element and corresponding inputs.

The specification of the truncated Rice (TR) binarization process, the truncated binary (TB) binarization process, the k-th order Exp-Golomb (EGk) binarization process and the fixed-length (FL) binarization process are given in clauses 9.3.3.3 through 9.3.3.7, respectively.

| Table 126 – Syntax elements and associated binarizations | | | |
| --- | --- | --- | --- |
| **Syntax structure** | **Syntax element** | **Binarization** | |
| **Process** | **Input parameters** |
| slice\_data( ) | end\_of\_slice\_one\_bit | FL | cMax = 1 |
| end\_of\_tile\_one\_bit | FL | cMax = 1 |
| end\_of\_subset\_one\_bit | FL | cMax = 1 |
| coding\_tree\_unit( ) | alf\_ctb\_flag[ ][ ][ ] | FL | cMax = 1 |
| alf\_use\_aps\_flag | FL | cMax = 1 |
| alf\_luma\_fixed\_filter\_idx | TB | cMax = 15 |
| alf\_luma\_prev\_filter\_idx | TB | cMax = sh\_num\_alf\_aps\_ids\_luma − 1 |
| alf\_ctb\_filter\_alt\_idx[ ][ ][ ] | TR | cMax = alf\_chroma\_num\_alt\_filters\_minus1, cRiceParam = 0 |
| alf\_ctb\_cc\_cb\_idc[ ][ ] | TR | cMax = ( alf\_cc\_cb\_filters\_signalled\_minus1 + 1 ), cRiceParam = 0 |
| alf\_ctb\_cc\_cr\_idc[ ][ ] | TR | cMax = ( alf\_cc\_cr\_filters\_signalled\_minus1 + 1 ), cRiceParam = 0 |
| sao( ) | sao\_merge\_left\_flag | FL | cMax = 1 |
| sao\_merge\_up\_flag | FL | cMax = 1 |
| sao\_type\_idx\_luma | TR | cMax = 2, cRiceParam = 0 |
| sao\_type\_idx\_chroma | TR | cMax = 2, cRiceParam = 0 |
| sao\_offset\_abs[ ][ ][ ][ ] | TR | cMax = ( 1  <<  ( Min( BitDepth, 10 ) − 5 ) ) − 1, cRiceParam = 0 |
| sao\_offset\_sign\_flag[ ][ ][ ][ ] | FL | cMax = 1 |
| sao\_band\_position[ ][ ][ ] | FL | cMax = 31 |
| sao\_eo\_class\_luma | FL | cMax = 3 |
| sao\_eo\_class\_chroma | FL | cMax = 3 |
| coding\_tree( ) | split\_cu\_flag | FL | cMax = 1 |
| split\_qt\_flag | FL | cMax = 1 |
| mtt\_split\_cu\_vertical\_flag | FL | cMax = 1 |
| mtt\_split\_cu\_binary\_flag | FL | cMax = 1 |
| mode\_constraint\_flag | FL | cMax = 1 |
| cod  ing\_unit( ) | cu\_skip\_flag[ ][ ] | FL | cMax = 1 |
| pred\_mode\_ibc\_flag | FL | cMax = 1 |
| pred\_mode\_plt\_flag | FL | cMax = 1 |
| cu\_act\_enabled\_flag | FL | cMax = 1 |
| pred\_mode\_flag | FL | cMax = 1 |
| intra\_bdpcm\_luma\_flag | FL | cMax = 1 |
| intra\_bdpcm\_luma\_dir\_flag | FL | cMax = 1 |
| intra\_mip\_flag[ ][ ] | FL | cMax = 1 |
| intra\_mip\_transposed\_flag[ ][ ] | FL | cMax = 1 |
| intra\_mip\_mode[ ][ ] | TB | cMax = (cbWidth = = 4 && cbHeight = = 4) ? 15 : ( ( (cbWith = = 4  | |  cbHeight = = 4) | | (cbWith = = 8  &&  cbHeight = = 8) )? 7 : 5) |
| intra\_luma\_ref\_idx[ ][ ] | TR | cMax = 2, cRiceParam = 0 |
| intra\_subpartitions\_mode\_flag | FL | cMax = 1 |
| intra\_subpartitions\_split\_flag | FL | cMax = 1 |
| intra\_luma\_mpm\_flag[ ][ ] | FL | cMax = 1 |
| intra\_luma\_not\_planar\_flag[ ][ ] | FL | cMax = 1 |
| intra\_luma\_mpm\_idx[ ][ ] | TR | cMax = 4, cRiceParam = 0 |
| intra\_luma\_mpm\_remainder[ ][ ] | TB | cMax = 60 |
| intra\_bdpcm\_chroma\_flag | FL | cMax = 1 |
| intra\_bdpcm\_chroma\_dir\_flag | FL | cMax = 1 |
| cclm\_mode\_flag | FL | cMax = 1 |
| cclm\_mode\_idx | TR | cMax = 2, cRiceParam = 0 |
| intra\_chroma\_pred\_mode | 9.3.3.8 | - |
| general\_merge\_flag[ ][ ] | FL | cMax = 1 |
| inter\_pred\_idc[ x0 ][ y0 ] | 9.3.3.9 | cbWidth, cbHeight |
| inter\_affine\_flag[ ][ ] | FL | cMax = 1 |
| cu\_affine\_type\_flag[ ][ ] | FL | cMax = 1 |
| sym\_mvd\_flag[ ][ ] | FL | cMax = 1 |
| ref\_idx\_l0[ ][ ] | TR | cMax = NumRefIdxActive[ 0 ] − 1, cRiceParam = 0 |
| mvp\_l0\_flag[ ][ ] | FL | cMax = 1 |
| ref\_idx\_l1[ ][ ] | TR | cMax = NumRefIdxActive[ 1 ] − 1, cRiceParam = 0 |
| mvp\_l1\_flag[ ][ ] | FL | cMax = 1 |
| amvr\_flag[ ][ ] | FL | cMax = 1 |
| amvr\_precision\_idx[ ][ ] | TR | cMax = ( inter\_affine\_flag  = =  0 && CuPredMode[ 0 ][ x0 ][ y0 ]  !=  MODE\_IBC ) ? 2 : 1, cRiceParam = 0 |
| bcw\_idx[ ][ ] | TR | cMax = NoBackwardPredFlag ? 4: 2, cRiceParam = 0 |
| cu\_coded\_flag | FL | cMax = 1 |
| cu\_sbt\_flag | FL | cMax = 1 |
| cu\_sbt\_quad\_flag | FL | cMax = 1 |
| cu\_sbt\_horizontal\_flag | FL | cMax = 1 |
| cu\_sbt\_pos\_flag | FL | cMax = 1 |
| lfnst\_idx | TR | cMax = 2, cRiceParam = 0 |
| mts\_idx | TR | cMax = 4, cRiceParam = 0 |
| palette\_coding( ) | palette\_predictor\_run | EG0 | - |
| num\_signalled\_palette\_entries | EG0 | - |
| new\_palette\_entries | FL | cMax = ( 1  <<  BitDepth ) − 1 |
| palette\_escape\_val\_present\_flag | FL | cMax = 1 |
| palette\_idx\_idc | 9.3.3.13 | MaxPaletteIndex |
| palette\_transpose\_flag | FL | cMax = 1 |
| copy\_above\_palette\_indices\_flag | FL | cMax = 1 |
| run\_copy\_flag | FL | cMax = 1 |
| palette\_escape\_val | EG5 |  |
| merge\_data( ) | regular\_merge\_flag[ ][ ] | FL | cMax = 1 |
| mmvd\_merge\_flag[ ][ ] | FL | cMax = 1 |
| mmvd\_cand\_flag[ ][ ] | FL | cMax = 1 |
| mmvd\_distance\_idx[ ][ ] | TR | cMax = 7, cRiceParam = 0 |
| mmvd\_direction\_idx[ ][ ] | FL | cMax = 3 |
| ciip\_flag[ ][ ] | FL | cMax = 1 |
| merge\_subblock\_flag[ ][ ] | FL | cMax = 1 |
| merge\_subblock\_idx[ ][ ] | TR | cMax = MaxNumSubblockMergeCand − 1, cRiceParam = 0 |
| merge\_gpm\_partition\_idx[ ][ ] | FL | cMax = 63 |
| merge\_gpm\_idx0[ ][ ] | TR | cMax = MaxNumGpmMergeCand − 1, cRiceParam = 0 |
| merge\_gpm\_idx1[ ][ ] | TR | cMax = MaxNumGpmMergeCand − 2, cRiceParam = 0 |
| merge\_idx[ ][ ] | TR | cMax = ( CuPredMode[ 0 ][ x0 ][ y0 ]  !=  MODE\_IBC ? MaxNumMergeCand : MaxNumIbcMergeCand ) − 1, cRiceParam = 0 |
| mvd\_coding( ) | abs\_mvd\_greater0\_flag[ ] | FL | cMax = 1 |
| abs\_mvd\_greater1\_flag[ ] | FL | cMax = 1 |
| abs\_mvd\_minus2[ ] | 9.3.3.14 | - |
| mvd\_sign\_flag[ ] | FL | cMax = 1 |
| transform\_unit( ) | tu\_y\_coded\_flag[ ][ ] | FL | cMax = 1 |
| tu\_cb\_coded\_flag[ ][ ] | FL | cMax = 1 |
| tu\_cr\_coded\_flag[ ][ ] | FL | cMax = 1 |
| cu\_qp\_delta\_abs | 9.3.3.10 | - |
| cu\_qp\_delta\_sign\_flag | FL | cMax = 1 |
| cu\_chroma\_qp\_offset\_flag | FL | cMax = 1 |
| cu\_chroma\_qp\_offset\_idx | TR | cMax = pps\_chroma\_qp\_offset\_list\_len\_minus1, cRiceParam = 0 |
| transform\_skip\_flag[ ][ ][ ] | FL | cMax = 1 |
| tu\_joint\_cbcr\_residual\_flag[ ][ ] | FL | cMax = 1 |
| residual\_coding( ) | last\_sig\_coeff\_x\_prefix | TR | cMax = ( log2ZoTbWidth << 1 ) − 1, cRiceParam = 0 |
| last\_sig\_coeff\_y\_prefix | TR | cMax = ( log2ZoTbHeight << 1 ) − 1, cRiceParam = 0 |
| last\_sig\_coeff\_x\_suffix | FL | cMax = ( 1  <<  ( ( last\_sig\_coeff\_x\_prefix  >>  1 ) − 1 ) − 1 ) |
| last\_sig\_coeff\_y\_suffix | FL | cMax = ( 1  <<  ( ( last\_sig\_coeff\_y\_prefix  >>  1 ) − 1 ) − 1 ) |
| sb\_coded\_flag[ ][ ] | FL | cMax = 1 |
| sig\_coeff\_flag[ ][ ] | FL | cMax = 1 |
| par\_level\_flag[ ] | FL | cMax = 1 |
| abs\_level\_gtx\_flag[ ][ ] | FL | cMax = 1 |
| abs\_remainder[ ] | 9.3.3.11 | cIdx, current sub-block index i, x0, y0, xC, yC, log2TbWidth, log2TbHeight |
| dec\_abs\_level[ ] | 9.3.3.12 | cIdx, x0, y0, xC, yC, log2TbWidth, log2TbHeight |
| coeff\_sign\_flag[ ] | FL | cMax = 1 |

#### Rice parameter derivation process for abs\_remainder[ ] and dec\_abs\_level[ ]

Inputs to this process are the base level baseLevel, the colour component index cIdx, the luma location ( x0, y0 ) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture, the current coefficient scan location ( xC, yC ), the binary logarithm of the transform block width log2TbWidth, and the binary logarithm of the transform block height log2TbHeight.

Output of this process is the Rice parameter cRiceParam.

Given the array AbsLevel[ x ][ y ] for the transform block with component index cIdx and the top-left luma location ( x0, y0 ), the variable locSumAbs is derived as specified by the following pseudo code:

locSumAbs = 0  
if( xC < (1 << log2TbWidth) − 1 ) {  
 locSumAbs += AbsLevel[ xC + 1 ][ yC ]  
 if( xC < (1 << log2TbWidth) − 2 )  
 locSumAbs += AbsLevel[ xC + 2 ][ yC ]  
 if( yC < (1 << log2TbHeight) − 1 )  
 locSumAbs += AbsLevel[ xC + 1 ][ yC + 1 ] (1545)  
}  
if( yC < (1 << log2TbHeight) − 1 ) {  
 locSumAbs += AbsLevel[ xC ][ yC + 1 ]  
 if( yC < (1 << log2TbHeight) − 2 )  
 locSumAbs += AbsLevel[ xC ][ yC + 2 ]  
}   
locSumAbs = Clip3( 0, 31, locSumAbs − baseLevel \* 5 )

Given the variable locSumAbs, the Rice parameter cRiceParam is derived as specified in Table 127.

When baseLevel is equal to 0, the variable ZeroPos[ n ] is derived as follows:

ZeroPos[ n ] = ( QState < 2 ? 1 : 2 )  <<  cRiceParam (1546)

Table 127 – Specification of cRiceParam based on locSumAbs

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **locSumAbs** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| cRiceParam | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 |
| **locSumAbs** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| cRiceParam | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 |

#### Truncated Rice binarization process

Input to this process is a request for a truncated Rice (TR) binarization, cMax and cRiceParam.

Output of this process is the TR binarization associating each value symbolVal with a corresponding bin string.

A TR bin string is a concatenation of a prefix bin string and, when present, a suffix bin string.

For the derivation of the prefix bin string, the following applies:

* The prefix value of symbolVal, prefixVal, is derived as follows:

prefixVal = symbolVal  >>  cRiceParam (1547)

* The prefix of the TR bin string is specified as follows:
* If prefixVal is less than cMax  >>  cRiceParam, the prefix bin string is a bit string of length prefixVal + 1 indexed by binIdx. The bins for binIdx less than prefixVal are equal to 1. The bin with binIdx equal to prefixVal is equal to 0. Table 128 illustrates the bin strings of this unary binarization for prefixVal.
* Otherwise, the bin string is a bit string of length cMax  >>  cRiceParam with all bins being equal to 1.

Table 128 – Bin string of the unary binarization (informative)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **prefixVal** | **Bin string** | | | | | |
| 0 | 0 |  |  |  |  |  |
| 1 | 1 | 0 |  |  |  |  |
| 2 | 1 | 1 | 0 |  |  |  |
| 3 | 1 | 1 | 1 | 0 |  |  |
| 4 | 1 | 1 | 1 | 1 | 0 |  |
| 5 | 1 | 1 | 1 | 1 | 1 | 0 |
| ... |  |  |  |  |  |  |
| binIdx | 0 | 1 | 2 | 3 | 4 | 5 |

When cMax is greater than symbolVal and cRiceParam is greater than 0, the suffix of the TR bin string is present and it is derived as follows:

* The suffix value suffixVal is derived as follows:

suffixVal = symbolVal − ( ( prefixVal )  <<  cRiceParam ) (1548)

* The suffix of the TR bin string is specified by invoking the fixed-length (FL) binarization process as specified in clause 9.3.3.7 for suffixVal with a cMax value equal to ( 1  <<  cRiceParam ) − 1.

NOTE – For the input parameter cRiceParam = 0, the TR binarization is exactly a truncated unary binarization and it is always invoked with a cMax value equal to the largest possible value of the syntax element being decoded.

#### Truncated Binary (TB) binarization process

Input to this process is a request for a TB binarization for a syntax element with value synVal and cMax. Output of this process is the TB binarization of the syntax element.The bin string of the TB binarization process of a syntax element synVal is specified as follows:

n = cMax + 1  
k = Floor( Log2( n ) ) (1549)  
u = ( 1  <<  ( k + 1) ) − n

* If synVal is less than u, the TB bin string is derived by invoking the FL binarization process specified in clause 9.3.3.7 for synVal with a cMax value equal to ( 1  <<  k ) − 1.
* Otherwise (synVal is greater than or equal to u), the TB bin string is derived by invoking the FL binarization process specified in clause 9.3.3.7 for ( synVal + u ) with a cMax value equal to ( 1  <<  ( k + 1) ) − 1.

#### k-th order Exp-Golomb binarization process

Inputs to this process is a request for a k-th order Exp-Golomb (EGk) binarization.

Output of this process is the EGk binarization associating each value symbolVal with a corresponding bin string.

The bin string of the EGk binarization process for each value symbolVal is specified as follows, where each call of the function put( X ), with X being equal to 0 or 1, adds the binary value X at the end of the bin string:

absV = Abs( symbolVal )  
stopLoop = 0  
do  
 if( absV >= ( 1 << k ) ) {  
 put( 1 )  
 absV = absV − ( 1 << k )  
 k++  
 } else {  
 put( 0 ) (1550)  
 while( k− − )  
 put( ( absV >> k ) & 1 )  
 stopLoop = 1  
 }  
while( !stopLoop )

NOTE – The specification for the k-th order Exp-Golomb (EGk) code uses 1's and 0's in reverse meaning for the unary part of the Exp-Golomb code of k-th order as specified in clause 9.2.

#### Limited k-th order Exp-Golomb binarization process

Inputs to this process is a request for a limited k-th order Exp-Golomb (EGk) binarization, the Rice parameter riceParam, the variables log2TransformRange and maxPreExtLen.

Output of this process is the limited EGk binarization associating each value symbolVal with a corresponding bin string.

The bin string of the limited EGk binarization process for each value symbolVal is specified as follows, where each call of the function put( X ), with X being equal to 0 or 1, adds the binary value X at the end of the bin string:

codeValue = symbolVal >> riceParam  
preExtLen = 0  
while( ( preExtLen < maxPreExtLen ) && ( codeValue > ( ( 2 << preExtLen ) − 2 ) ) ) {  
 preExtLen++  
 put( 1 )  
}  
if( preExtLen = = maxPreExtLen ) (1551)  
 escapeLength = log2TransformRange  
else {  
 escapeLength = preExtLen + riceParam  
 put( 0 )   
}  
symbolVal = symbolVal − ( ( ( 1 << preExtLen ) − 1 ) << riceParam )  
while( ( escapeLength− − ) > 0 )  
 put( ( symbolVal >> escapeLength ) & 1 )

#### Fixed-length binarization process

Inputs to this process are a request for a fixed-length (FL) binarization and cMax.

Output of this process is the FL binarization associating each value symbolVal with a corresponding bin string.

FL binarization is constructed by using the fixedLength‑bit unsigned integer bin string of the symbol value symbolVal, where fixedLength = Ceil( Log2( cMax + 1 ) ). The indexing of bins for the FL binarization is such that the binIdx = 0 relates to the most significant bit with increasing values of binIdx towards the least significant bit.

#### Binarization process for intra\_chroma\_pred\_mode

Input to this process is a request for a binarization for the syntax element intra\_chroma\_pred\_mode.

Output of this process is the binarization of the syntax element.

The binarization for the syntax element intra\_chroma\_pred\_mode is specified in Table 129.

Table 129 – Binarization for intra\_chroma\_pred\_mode

|  |  |
| --- | --- |
| **Value of intra\_chroma\_pred\_mode** | **Bin string** |
| 0 | 100 |
| 1 | 101 |
| 2 | 110 |
| 3 | 111 |
| 4 | 0 |

#### Binarization process for inter\_pred\_idc

Input to this process is a request for a binarization for the syntax element inter\_pred\_idc, the current luma coding block width cbWidth and the current luma coding block height cbHeight.

Output of this process is the binarization of the syntax element.

The binarization for the syntax element inter\_pred\_idc is specified in Table 130.

Table 130 – Binarization for inter\_pred\_idc

|  |  |  |  |
| --- | --- | --- | --- |
| **Value of inter\_pred\_idc** | **Name of inter\_pred\_idc** | **Bin string** | |
| ( cbWidth + cbHeight )  >  12 | ( cbWidth + cbHeight )  = =  12 |
| 0 | PRED\_L0 | 00 | 0 |
| 1 | PRED\_L1 | 01 | 1 |
| 2 | PRED\_BI | 1 | - |

#### Binarization process for cu\_qp\_delta\_abs

Input to this process is a request for a binarization for the syntax element cu\_qp\_delta\_abs.

Output of this process is the binarization of the syntax element.

The binarization of the syntax element cu\_qp\_delta\_abs is a concatenation of a prefix bin string and (when present) a suffix bin string.

For the derivation of the prefix bin string, the following applies:

* The prefix value of cu\_qp\_delta\_abs, prefixVal, is derived as follows:

prefixVal = Min( cu\_qp\_delta\_abs, 5 ) (1552)

* The prefix bin string is specified by invoking the TR binarization process as specified in clause 9.3.3.3 for prefixVal with cMax = 5 and cRiceParam = 0.

When prefixVal is greater than 4, the suffix bin string is present and it is derived as follows:

* The suffix value of cu\_qp\_delta\_abs, suffixVal, is derived as follows:

suffixVal = cu\_qp\_delta\_abs − 5 (1553)

* The suffix bin string is specified by invoking the k-th order EGk binarization process as specified in clause 9.3.3.5 for suffixVal with the Exp-Golomb order k set equal to 0.

#### Binarization process for abs\_remainder[ ]

Input to this process is a request for a binarization for the syntax element abs\_remainder[ n ], the colour component cIdx, the current sub-block index i, and the luma location ( x0, y0 ) specifying the top-left sample of the current luma transform block relative to the top-left luma sample of the picture, the current coefficient scan location ( xC, yC ), the binary logarithm of the transform block width log2TbWidth, and the binary logarithm of the transform block height log2TbHeight.

Output of this process is the binarization of the syntax element.

The variables lastAbsRemainder and lastRiceParam are derived as follows:

* If this process is invoked for the first time for the current sub-block index i, lastAbsRemainder and lastRiceParam are both set equal to 0.
* Otherwise (this process is not invoked for the first time for the current sub-block index i), lastAbsRemainder and lastRiceParam are set equal to the values of abs\_remainder[ n ] and cRiceParam, respectively, that have been derived during the last invocation of the binarization process for the syntax element abs\_remainder[ n ] as specified in this clause.

The rice parameter cRiceParam is derived as follows:

* If transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] is equal to 1 and sh\_ts\_residual\_coding\_disabled\_flag is equal to 0, the Rice parameter cRiceParam is set equal to 1.
* Otherwise, the rice parameter cRiceParam is derived by invoking the rice parameter derivation process for abs\_remainder[] as specified in clause 9.3.3.2 with the variable baseLevel set equal to 4, the colour component index cIdx, the luma location ( x0, y0 ), the current coefficient scan location ( xC, yC ), the binary logarithm of the transform block width log2TbWidth, and the binary logarithm of the transform block height log2TbHeight as inputs.

The variable cMax is derived from cRiceParam as:

cMax = 6  <<  cRiceParam (1554)

The binarization of the syntax element abs\_remainder[ n ] is a concatenation of a prefix bin string and (when present) a suffix bin string.

For the derivation of the prefix bin string, the following applies:

* The prefix value of abs\_remainder[ n ], prefixVal, is derived as follows:

prefixVal = Min( cMax, abs\_remainder[ n ] ) (1555)

* The prefix bin string is specified by invoking the TR binarization process as specified in clause 9.3.3.3 for prefixVal with the variables cMax and cRiceParam as inputs.

When the prefix bin string is equal to the bit string of length 6 with all bits equal to 1, the suffix bin string is present and it is derived as follows:

* The suffix value of abs\_remainder[ n ], suffixVal, is derived as follows:

suffixVal = abs\_remainder[ n ] − cMax (1556)

* The suffix bin string is specified by invoking the limited k-th order EGk binarization process as specified in clause 9.3.3.6 for the binarization of suffixVal with the Exp-Golomb order k set equal to cRiceParam + 1, variable cRiceParam, variable log2TransformRange set equal to 15 and variable maxPreExtLen set equal to 11 as input.

#### Binarization process for dec\_abs\_level[ ]

Input to this process is a request for a binarization of the syntax element dec\_abs\_level[ n ], the colour component cIdx, the luma location ( x0, y0 ) specifying the top-left sample of the current transform block relative to the top-left luma sample of the picture, the current coefficient scan location ( xC, yC ), the binary logarithm of the transform block width log2TbWidth, and the binary logarithm of the transform block height log2TbHeight.

Output of this process is the binarization of the syntax element.

The rice parameter cRiceParam is derived by invoking the rice parameter derivation process for dec\_abs\_level[] as specified in clause 9.3.3.2 with the variable baseLevel set equal to 0, the colour component index cIdx, the luma location ( x0, y0 ), the current coefficient scan location ( xC, yC ), the binary logarithm of the transform block width log2TbWidth, and the binary logarithm of the transform block height log2TbHeight as inputs.

The variable cMax is derived from cRiceParam as:

cMax = 6  <<  cRiceParam (1557)

The binarization of dec\_abs\_level[ n ] is a concatenation of a prefix bin string and (when present) a suffix bin string.

For the derivation of the prefix bin string, the following applies:

* The prefix value of dec\_abs\_level[ n ], prefixVal, is derived as follows:

prefixVal = Min( cMax, dec\_abs\_level[ n ] ) (1558)

* The prefix bin string is specified by invoking the TR binarization process as specified in clause 9.3.3.3 for prefixVal with the variables cMax and cRiceParam as inputs.

When the prefix bin string is equal to the bit string of length 6 with all bits equal to 1, the suffix bin string is present and it is derived as follows:

* The suffix value of dec\_abs\_level[ n ], suffixVal, is derived as follows:

suffixVal = dec\_abs\_level[ n ] − cMax (1559)

* The suffix bin string is specified by invoking the limited k-th order EGk binarization process as specified in clause 9.3.3.6 for the binarization of suffixVal with the Exp-Golomb order k set equal to cRiceParam + 1, variable cRiceParam, variable log2TransformRange set equal to 15 and variable maxPreExtLen set equal to 11 as input.

#### Binarization process for palette\_idx\_idc

Input to this process is a request for a binarization for the syntax element palette\_idx\_idc and the variable MaxPaletteIndex.

Output of this process is the binarization of the syntax element.

The variable cMax is derived as follows:

* If this process is invoked for the first time for the current block, cMax is set equal to MaxPaletteIndex.
* Otherwise (this process is not invoked for the first time for the current block), cMax is set equal to MaxPaletteIndex minus 1.

The binarization for the palette\_idx\_idc is derived by invoking the TB binarization process specified in clause 9.3.3.4 with cMax.

#### Binarization process for abs\_mvd\_minus2

Input to this process is a request for a binarization for the syntax element abs\_mvd\_minus2.

Output of this process is the binarization of the syntax element.

The abs\_mvd\_minus bin string is specified by invoking the limited k-th order EGk binarization process as pecified in clause 9.3.3.6 with variable k set equal to 1, variable riceParam set equal to 1, variable log2TransformRange set equal to 17 and variable maxPreExtLen set equal to 15 as inputs.

NOTE – Binarization scheme is equivalent to representing abs\_mvd\_minus2 value less than equal to 217 − 3 is represented by EG1 binarization. The largest value 217 − 2 is represented as 0xffff0000.

### Decoding process flow

#### General

Inputs to this process are all bin strings of the binarization of the requested syntax element as specified in clause 9.3.3.

Output of this process is the value of the syntax element.

This process specifies how each bin of a bin string is parsed for each syntax element. After parsing each bin, the resulting bin string is compared to all bin strings of the binarization of the syntax element and the following applies:

– If the bin string is equal to one of the bin strings, the corresponding value of the syntax element is the output.

– Otherwise (the bin string is not equal to one of the bin strings), the next bit is parsed.

While parsing each bin, the variable binIdx is incremented by 1 starting with binIdx being set equal to 0 for the first bin.

The parsing of each bin is specified by the following two ordered steps:

1. The derivation process for ctxTable, ctxIdx, and bypassFlag as specified in clause 9.3.4.2 is invoked with binIdx as input and ctxTable, ctxIdx and bypassFlag as outputs.

2. The arithmetic decoding process as specified in clause 9.3.4.3 is invoked with ctxTable, ctxIdx and bypassFlag as inputs and the value of the bin as output.

#### Derivation process for ctxTable, ctxIdx and bypassFlag

##### General

Input to this process is the position of the current bin within the bin string, binIdx.

Outputs of this process are ctxTable, ctxIdx and bypassFlag.

The values of ctxTable, ctxIdx and bypassFlag are derived as follows based on the entries for binIdx of the corresponding syntax element in Table 131:

* If the entry in Table 131 is not equal to "bypass", "terminate" or "na", the values of binIdx are decoded by invoking the DecodeDecision process as specified in clause 9.3.4.3.2 and the following applies:
* ctxTable is specified in Table 51
* The variable ctxInc is specified by the corresponding entry in Table 131 and when more than one value is listed in Table 131 for a binIdx, the assignment process for ctxInc for that binIdx is further specified in the clauses given in parenthesis.
* The variable ctxIdxOffset is specified in Table 51 depending on the current value of initType.
* ctxIdx is set equal to the sum of ctxInc and ctxIdxOffset.
* bypassFlag is set equal to 0.
* Otherwise, if the entry in Table 131 is equal to "bypass", the values of binIdx are decoded by invoking the DecodeBypass process as specified in clause 9.3.4.3.4 and the following applies:
* ctxTable is set equal to 0.
* ctxIdx is set equal to 0.
* bypassFlag is set equal to 1.a
* Otherwise, if the entry in Table 131 is equal to "terminate", the values of binIdx are decoded by invoking the DecodeTerminate process as specified in clause 9.3.4.3.5 and the following applies:
* ctxTable is set equal to 0.
* ctxIdx is set equal to 0.
* bypassFlag is set equal to 0.
* Otherwise (the entry in Table 131 is equal to "na"), the values of binIdx do not occur for the corresponding syntax element.

| Table 131 – Assignment of ctxInc to syntax elements with context coded bins | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Syntax element** | **binIdx** | | | | | |
| **0** | **1** | **2** | **3** | **4** | **>= 5** |
| end\_of\_slice\_one\_bit | terminate | na | na | na | na | na |
| end\_of\_tile\_one\_bit | terminate | na | na | na | na | na |
| end\_of\_subset\_one\_bit | terminate | na | na | na | na | na |
| alf\_ctb\_flag[ ][ ][ ] | 0..8 (clause 9.3.4.2.2) | na | na | na | na | na |
| alf\_use\_aps\_flag | 0 | na | na | na | na | na |
| alf\_luma\_fixed\_filter\_idx | bypass | bypass | bypass | bypass | bypass | bypass |
| alf\_luma\_prev\_filter\_idx | bypass | bypass | bypass | bypass | bypass | bypass |
| alf\_ctb\_filter\_alt\_idx[ 0 ][ ][ ] | 0 | 0 | 0 | 0 | 0 | 0 |
| alf\_ctb\_filter\_alt\_idx[ 1 ][ ][ ] | 1 | 1 | 1 | 1 | 1 | 1 |
| alf\_ctb\_cc\_cb\_idc[ ][ ] | 0..2 (clause 9.3.4.2.2) | bypass | bypass | bypass | bypass | bypass |
| alf\_ctb\_cc\_cr\_idc[ ][ ] | 0..2 (clause 9.3.4.2.2) | bypass | bypass | bypass | bypass | bypass |
| sao\_merge\_left\_flag | 0 | na | na | na | na | na |
| sao\_merge\_up\_flag | 0 | na | na | na | na | na |
| sao\_type\_idx\_luma | 0 | bypass | na | na | na | na |
| sao\_type\_idx\_chroma | 0 | bypass | na | na | na | na |
| sao\_offset\_abs[ ][ ][ ][ ] | bypass | bypass | bypass | bypass | bypass | na |
| sao\_offset\_sign\_flag[ ][ ][ ][ ] | bypass | na | na | na | na | na |
| sao\_band\_position[ ][ ][ ] | bypass | bypass | bypass | bypass | bypass | bypass |
| sao\_eo\_class\_luma | bypass | bypass | na | na | na | na |
| sao\_eo\_class\_chroma | bypass | bypass | na | na | na | na |
| split\_cu\_flag | 0..8 (clause 9.3.4.2.2) | na | na | na | na | na |
| split\_qt\_flag | 0..5 (clause 9.3.4.2.2) | na | na | na | na | na |
| mtt\_split\_cu\_vertical\_flag | 0..4 (clause 9.3.4.2.3) | na | na | na | na | na |
| mtt\_split\_cu\_binary\_flag | ( 2 \* mtt\_split\_cu\_vertical\_flag ) + ( mttDepth < = 1 ? 1 : 0 ) | na | na | na | na | na |
| mode\_constraint\_flag | 0,1 (clause 9.3.4.2.2) | na | na | na | na | na |
| cu\_skip\_flag[ ][ ] | 0,1,2 (clause 9.3.4.2.2) | na | na | na | na | na |
| pred\_mode\_flag | 0,1 (clause 9.3.4.2.2) | na | na | na | na | na |
| pred\_mode\_ibc\_flag | 0,1,2 (clause 9.3.4.2.2) | na | na | na | na | na |
| pred\_mode\_plt\_flag | 0 | na | na | na | na | na |
| cu\_act\_enabled\_flag | 0 | na | na | na | na | na |
| intra\_bdpcm\_luma\_flag | 0 | na | na | na | na | na |
| intra\_bdpcm\_luma\_dir\_flag | 0 | na | na | na | na | na |
| intra\_mip\_flag[ ][ ] | (Abs( Log2(cbWidth) − Log2(cbHeight) ) > 1) ?  3 : ( 0,1,2 (clause 9.3.4.2.2) ) [Ed. (YK): Something is wrong here.] | na | na | na | na | na |
| intra\_mip\_transposed\_flag[ ][ ] | bypass | na | na | na | na | na |
| intra\_mip\_mode[ ][ ] | bypass | bypass | bypass | bypass | bypass | na |
| intra\_luma\_ref\_idx[ ][ ] | 0 | 1 | na | na | na | na |
| intra\_subpartitions\_mode\_flag | 0 | na | na | na | na | na |
| intra\_subpartitions\_split\_flag | 0 | na | na | na | na | na |
| intra\_luma\_mpm\_flag[ ][ ] | 0 | na | na | na | na | na |
| intra\_luma\_not\_planar\_flag[ ][ ] | intra\_subpartitions\_mode\_flag | na | na | na | na | na |
| intra\_luma\_mpm\_idx[ ][ ] | bypass | bypass | bypass | bypass | na | na |
| intra\_luma\_mpm\_remainder[ ][ ] | bypass | bypass | bypass | bypass | bypass | bypass |
| intra\_bdpcm\_chroma\_flag | 0 | na | na | na | na | na |
| intra\_bdpcm\_chroma\_dir\_flag | 0 | na | na | na | na | na |
| cclm\_mode\_flag | 0 | na | na | na | na | na |
| cclm\_mode\_idx | 0 | bypass | na | na | na | na |
| intra\_chroma\_pred\_mode | 0 | bypass | bypass | na | na | na |
| palette\_predictor\_run | bypass | bypass | bypass | bypass | bypass | bypass |
| num\_signalled\_palette\_entries | bypass | bypass | bypass | bypass | bypass | bypass |
| new\_palette\_entries | bypass | bypass | bypass | bypass | bypass | bypass |
| palette\_escape\_val\_present\_flag | bypass | na | na | na | na | na |
| palette\_transpose\_flag | 0 | na | na | na | na | na |
| palette\_idx\_idc | bypass | bypass | bypass | bypass | bypass | bypass |
| copy\_above\_palette\_indices\_flag | 0 | na | na | na | na | na |
| run\_copy\_flag | 0..7 (clause 9.3.4.2.11) | na | na | na | na | na |
| palette\_escape\_val | bypass | bypass | bypass | bypass | bypass | bypass |
| general\_merge\_flag[ ][ ] | 0 | na | na | na | na | na |
| regular\_merge\_flag[ ][ ] | cu\_skip\_flag[ ][ ] ? 0 : 1 | na | na | na | na | na |
| mmvd\_merge\_flag[ ][ ] | 0 | na | na | na | na | na |
| mmvd\_cand\_flag[ ][ ] | 0 | na | na | na | na | na |
| mmvd\_distance\_idx[ ][ ] | 0 | bypass | bypass | bypass | bypass | bypass |
| mmvd\_direction\_idx[ ][ ] | bypass | bypass | na | na | na | na |
| merge\_subblock\_flag[ ][ ] | 0,1,2 (clause 9.3.4.2.2) | na | na | na | na | na |
| merge\_subblock\_idx[ ][ ] | 0 | bypass | bypass | bypass | bypass | na |
| ciip\_flag[ ][ ] | 0 | na | na | na | na | na |
| merge\_idx[ ][ ] | 0 | bypass | bypass | bypass | bypass | na |
| merge\_gpm\_partition\_idx[ ][ ] | bypass | bypass | bypass | bypass | bypass | bypass |
| merge\_gpm\_idx0[ ][ ] | 0 | bypass | bypass | bypass | bypass | na |
| merge\_gpm\_idx1[ ][ ] | 0 | bypass | bypass | bypass | na | na |
| inter\_pred\_idc[ x0 ][ y0 ] | ( cbWidth + cbHeight ) > 12 ? 7 − ( ( 1 +  Log2( cbWidth ) + Log2( cbHeight ) ) >> 1 )   : 5 | 5 | na | na | na | na |
| inter\_affine\_flag[ ][ ] | 0,1,2 (clause 9.3.4.2.2) | na | na | na | na | na |
| cu\_affine\_type\_flag[ ][ ] | 0 | na | na | na | na | na |
| sym\_mvd\_flag[ ][ ] | 0 | na | na | na | na | na |
| ref\_idx\_l0[ ][ ] | 0 | 1 | bypass | bypass | bypass | bypass |
| ref\_idx\_l1[ ][ ] | 0 | 1 | bypass | bypass | bypass | bypass |
| mvp\_l0\_flag[ ][ ] | 0 | na | na | na | na | na |
| mvp\_l1\_flag[ ][ ] | 0 | na | na | na | na | na |
| amvr\_flag[ ][ ] | inter\_affine\_flag[ ][ ] ? 1 : 0 | na | na | na | na | na |
| amvr\_precision\_idx[ ][ ] | 0 | 1 | na | na | na | na |
| bcw\_idx[ ][ ] NoBackwardPredFlag = = 0 | 0 | bypass | na | na | na | na |
| bcw\_idx[ ][ ]  NoBackwardPredFlag = = 1 | 0 | bypass | bypass | bypass | na | na |
| cu\_coded\_flag | 0 | na | na | na | na | na |
| cu\_sbt\_flag | ( cbWidth \*  cbHeight <= 256 ) ? 1 : 0 | na | na | na | na | na |
| cu\_sbt\_quad\_flag | 0 | na | na | na | na | na |
| cu\_sbt\_horizontal\_flag | ( cbWidth  = = cbHeight ) ? 0 : ( cbWidth < cbHeight ) ? 1 : 2 | na | na | na | na | na |
| cu\_sbt\_pos\_flag | 0 | na | na | na | na | na |
| lfnst\_idx | ( treeType != SINGLE\_TREE ) ? 1 : 0 | 2 | na | na | na | na |
| mts\_idx | 0 | 1 | 2 | 3 | na | na |
| abs\_mvd\_greater0\_flag[ ] | 0 | na | na | na | na | na |
| abs\_mvd\_greater1\_flag[ ] | 0 | na | na | na | na | na |
| abs\_mvd\_minus2[ ] | bypass | bypass | bypass | bypass | bypass | bypass |
| mvd\_sign\_flag[ ] | bypass | na | na | na | na | na |
| tu\_y\_coded\_flag[ ][ ] | 0,1,2,3 (clause 9.3.4.2.5) | na | na | na | na | na |
| tu\_cb\_coded\_flag[ ][ ] | intra\_bdpcm\_chroma\_flag ? 1 : 0 | na | na | na | na | na |
| tu\_cr\_coded\_flag[ ][ ] | intra\_bdpcm\_chroma\_flag ? 2 : tu\_cb\_coded\_flag[ ][ ] | na | na | na | na | na |
| cu\_qp\_delta\_abs | 0 | 1 | 1 | 1 | 1 | bypass |
| cu\_qp\_delta\_sign\_flag | bypass | na | na | na | na | na |
| cu\_chroma\_qp\_offset\_flag | 0 | na | na | na | na | na |
| cu\_chroma\_qp\_offset\_idx | 0 | 0 | 0 | 0 | 0 | na |
| transform\_skip\_flag[ ][ ][ cIdx ] | cIdx = = 0 ? 0 : 1 | na | na | na | na | na |
| tu\_joint\_cbcr\_residual\_flag[ ][ ] | 2\*tu\_cb\_coded\_flag[ ][ ] + tu\_cr\_coded\_flag[ ][ ] − 1 | na | na | na | na | na |
| last\_sig\_coeff\_x\_prefix | 0..22 (clause 9.3.4.2.4) | | | | | |
| last\_sig\_coeff\_y\_prefix | 0..22 (clause 9.3.4.2.4) | | | | | |
| last\_sig\_coeff\_x\_suffix | bypass | bypass | bypass | bypass | bypass | bypass |
| last\_sig\_coeff\_y\_suffix | bypass | bypass | bypass | bypass | bypass | bypass |
| sb\_coded\_flag[ ][ ] | 0..7 (clause 9.3.4.2.6) | na | na | na | na | na |
| sig\_coeff\_flag[ ][ ] | 0..62 (clause 9.3.4.2.8) | na | na | na | na | na |
| par\_level\_flag[ ] | 0..32 (clause 9.3.4.2.9) | na | na | na | na | na |
| abs\_level\_gtx\_flag[ ][ ] | 0..73 (clause 9.3.4.2.9) | na | na | na | na | na |
| abs\_remainder[ ] | bypass | bypass | bypass | bypass | bypass | bypass |
| dec\_abs\_level[ ] | bypass | bypass | bypass | bypass | bypass | bypass |
| coeff\_sign\_flag[ ]  transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] = = 0 | | RemCcbs = = 0 | | sh\_ts\_residual\_coding\_disabled\_flag | bypass | na | na | na | na | na |
| coeff\_sign\_flag[ ] transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] = = 1 && RemCcbs >= 0 && !sh\_ts\_residual\_coding\_disabled\_flag | 0..5 (clause 9.3.4.2.10) | na | na | na | na | na |

##### Derivation process of ctxInc using left and above syntax elements

Input to this process is the luma location ( x0, y0 ) specifying the top-left luma sample of the current luma block relative to the top-left sample of the current picture, the colour component cIdx, the current coding quadtree depth cqtDepth, the dual tree channel type chType, the width and the height of the current coding block in luma samples cbWidth and cbHeight, and the variables allowSplitBtVer, allowSplitBtHor, allowSplitTtVer, allowSplitTtHor, and allowSplitQt as derived in the coding tree semantics in clause 7.4.11.4.

Output of this process is ctxInc.

The location ( xNbL, yNbL ) is set equal to ( x0 − 1, y0 ) and the derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location ( xCurr, yCurr ) set equal to ( x0, y0 ), the neighbouring location ( xNbY, yNbY ) set equal to ( xNbL, yNbL ), checkPredModeY set equal to FALSE, and cIdx as inputs, and the output is assigned to availableL.

The location ( xNbA, yNbA ) is set equal to ( x0, y0 − 1 ) and the derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location ( xCurr, yCurr ) set equal to ( x0, y0 ), the neighbouring location ( xNbY, yNbY ) set equal to ( xNbA, yNbA ), checkPredModeY set equal to FALSE, and cIdx as inputs, and the output is assigned to availableA.

The location ( xCtb, yCtb ) is set equal to ( x0 >> CtbLog2SizeY , y0 >> CtbLog2SizeY ), the location ( xCtbA, yCtbA ) is set equal to ( xNbA >> CtbLog2SizeY , yNbA >> CtbLog2SizeY ), and the location ( xCtbL, yCtbL ) is set equal to ( xNbL >> CtbLog2SizeY , yNbL >> CtbLog2SizeY ).

The assignment of ctxInc is specified as follows with condL and condA specified in Table 132:

* For the syntax elements alf\_ctb\_flag[ cIdx ][ xCtb ][ yCtb ], alf\_ctb\_cc\_cb\_idc[ xCtb ][ yCtb ], alf\_ctb\_cc\_cr\_idc[ xCtb ][ yCtb ], split\_qt\_flag, split\_cu\_flag, cu\_skip\_flag[ x0 ][ y0 ], pred\_mode\_ibc\_flag[ x0 ][ y0 ], intra\_mip\_flag[ x0 ][ y0 ], inter\_affine\_flag[ x0 ][ y0 ] and merge\_subblock\_flag[ x0 ][ y0 ]:

ctxInc = ( condL  &&  availableL ) + ( condA  &&  availableA ) + ctxSetIdx \* 3 (1560)

* For the syntax elements pred\_mode\_flag[ x0 ][ y0 ] and mode\_constraint\_flag:

ctxInc = ( condL  &&  availableL ) | | ( condA  &&  availableA ) (1561)

Table 132 – Specification of ctxInc using left and above syntax elements

|  |  |  |  |
| --- | --- | --- | --- |
| **Syntax element** | **condL** | **condA** | **ctxSetIdx** |
| alf\_ctb\_flag[ cIdx ][ xCtb ][ yCtb ] | alf\_ctb\_flag[ cIdx ][ xCtbL ][ yCtbL ] | alf\_ctb\_flag[cIdx ][ xCtbA ][ yCtbA ] | cIdx |
| alf\_ctb\_cc\_cb\_idc[ xCtb ][ yCtb ] | alf\_ctb\_cc\_cb\_idc[ xCtbL ][ yCtbL ] | alf\_ctb\_cc\_cb\_idc[ xCtbA ][ yCtbA ] | 0 |
| alf\_ctb\_cc\_cr\_idc[ xCtb ][ yCtb ] | alf\_ctb\_cc\_cr\_idc[ xCtbL ][ yCtbL ] | alf\_ctb\_cc\_cr\_idc[ xCtbA  ][ yCtbA ] | 0 |
| split\_qt\_flag | CqtDepth[ chType ][ xNbL ][ yNbL ] > cqtDepth | CqtDepth[ chType ][ xNbA ][ yNbA ] > cqtDepth | cqtDepth >= 2 |
| split\_cu\_flag | CbHeight[ chType ][ xNbL ][ yNbL ] < cbHeight | CbWidth[ chType ][ xNbA ][ yNbA ] < cbWidth | ( allowSplitBtVer +   allowSplitBtHor +   allowSplitTtVer +   allowSplitTtHor +  2 \* allowSplitQt − 1 )  / 2 |
| mode\_constraint\_flag | CuPredMode[ chType ][ xNbL ][ yNbL ] = = MODE\_INTRA | CuPredMode[ chType ][ xNbA ][ yNbA ] = = MODE\_INTRA | 0 |
| cu\_skip\_flag[ x0 ][ y0 ] | cu\_skip\_flag[ xNbL ][ yNbL ] | cu\_skip\_flag[ xNbA ][ yNbA ] | 0 |
| pred\_mode\_flag[ x0 ][ y0 ] | CuPredMode[ chType ][ xNbL ][ yNbL ] = = MODE\_INTRA | CuPredMode[ chType ][ xNbA ][ yNbA ] = = MODE\_INTRA | 0 |
| pred\_mode\_ibc\_flag[ x0 ][ y0 ] | CuPredMode[ chType ][ xNbL ][ yNbL ] = = MODE\_IBC | CuPredMode[ chType ][ xNbA ][ yNbA ] = = MODE\_IBC | 0 |
| intra\_mip\_flag[ x0 ][ y0 ] | intra\_mip\_flag[ xNbL ][ yNbL ] | intra\_mip\_flag[ xNbA ][ yNbA ] | 0 |
| merge\_subblock\_flag[ x0 ][ y0 ] | merge\_subblock\_flag[ xNbL ][ yNbL ] | | inter\_affine\_flag[ xNbL ][ yNbL ] | merge\_subblock\_flag[ xNbA ][ yNbA ] | | inter\_affine\_flag[ xNbA ][ yNbA ] | 0 |
| inter\_affine\_flag [ x0 ][ y0 ] | merge\_subblock\_flag[ xNbL ][ yNbL ] | | inter\_affine\_flag[ xNbL ][ yNbL ] | merge\_subblock\_flag[ xNbA ][ yNbA ] | | inter\_affine\_flag[ xNbA ][ yNbA ] | 0 |

##### Derivation process of ctxIncfor the syntax element mtt\_split\_cu\_vertical\_flag

Input to this process is the luma location ( x0, y0 ) specifying the top-left luma sample of the current luma block relative to the top-left sample of the current picture, the dual tree channel type chType, the width and the height of the current coding block in luma samples cbWidth and cbHeight, and the variables allowSplitBtVer, allowSplitBtHor, allowSplitTtVer, allowSplitTtHor, and allowSplitQt as derived in the coding tree semantics in clause 7.4.11.4.

Output of this process is ctxInc.

The location ( xNbL, yNbL ) is set equal to ( x0 − 1, y0 ) and the derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location ( xCurr, yCurr ) set equal to ( x0, y0 ) and the neighbouring location ( xNbY, yNbY ) set equal to ( xNbL, yNbL ), checkPredModeY set equal to FALSE, and cIdx as inputs, and the output is assigned to availableL.

The location ( xNbA, yNbA ) is set equal to ( x0, y0 − 1 ) and tthe derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location ( xCurr, yCurr ) set equal to ( x0, y0 ), the neighbouring location ( xNbY, yNbY ) set equal to ( xNbA, yNbA ), checkPredModeY set equal to FALSE, and cIdx as inputs, and the output is assigned to availableA.

The assignment of ctxInc is specified as follows:

* If allowSplitBtVer + allowSplitBtHor is greater than allowSplitTtVer + allowSplitTtHor, ctxInc is set equal to 4.
* Otherwise, if allowSplitBtVer + allowSplitBtHor is less than allowSplitTtVer + allowSplitTtHor, ctxInc is set equal to 4.
* Otherwise, the following applies:
* The variables dA and dL are derived as follows

dA = cbWidth / ( availableA ? CbWidth[ chType ][ xNbA ][ yNbA ] : 1 ) (1562)

dL = cbHeight / ( availableL ? CbHeight[ chType ][ xNbL ][ yNbL ] : 1 ) (1563)

* If any of the following conditions is true, ctxInc is set equal to 0:
* dA is equal to dL,
* availableA is equal to FALSE,
* availableL is equal to FALSE.
* Otherwise, if dA is less then dL, ctxInc is set equal to 1.
* Otherwise, ctxInc is set equal to 0.

##### Derivation process of ctxInc for the syntax elements last\_sig\_coeff\_x\_prefix and last\_sig\_coeff\_y\_prefix

Inputs to this process are the variable binIdx, the colour component index cIdx, the binary logarithm of the transform block width log2TbWidth and the transform block height log2TbHeight.

Output of this process is the variable ctxInc.

The variable log2TbSize is derived as follows:

* If the syntax element to be parsed is last\_sig\_coeff\_x\_prefix, log2TbSize is set equal to log2TbWidth.
* Otherwise (the syntax element to be parsed is last\_sig\_coeff\_y\_prefix), log2TbSize is set equal to log2TbHeight.

The variables ctxOffset and ctxShift are derived as follows:

* If cIdx is equal to 0, ctxOffset is set equal to offsetY[ log2TbSize − 2 ] and ctxShift is set equal to ( log2TbSize + 1 )  >>  2 with the list offsetY specified as follows:

offsetY[] = {0, 3, 6, 10, 15} (1564)

* Otherwise (cIdx is greater than 0), ctxOffset is set equal to 20 and ctxShift is set equal to Clip3( 0, 2, 2log2TbSize >> 3 ).

The variable ctxInc is derived as follows:

ctxInc = ( binIdx >> ctxShift ) + ctxOffset (1565)

##### Derivation process of ctxInc for the syntax element tu\_y\_coded\_flag

Inputs to this process are the variable binIdx, the luma location ( x0, y0 ) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture.

Output of this process is the variable ctxInc.

The variable ctxInc is derived as follows:

* If BdpcmFlag[ x0 ][ y0 ][ 0 ] is equal to 1, ctxInc is set equal to 1.
* Otherwise, if IntraSubpartitionsSplitType is equal to ISP\_NO\_SPLIT, ctxInc is set equal to 0.
* Otherwise ( BdpcmFlag[ x0 ][ y0 ][ 0 ] is equal to 0 and IntraSubpartitionsSplitType is not equal to ISP\_NO\_SPLIT ), the following applies:
* The variable prevTuCbfY is derived as follows:
* If the current transform unit is the first one to be parsed in a coding unit, prevTuCbfY is set equal to 0.
* Otherwise, prevTuCbfY is set equal to the value of tu\_y\_coded\_flag of the previous luma transform unit in the current coding unit.
* The variable ctxInc is derived as follows:

ctxInc = 2 + prevTuCbfY (1566)

##### Derivation process of ctxInc for the syntax element sb\_coded\_flag

Inputs to this process are the colour component index cIdx, the luma location ( x0, y0 ) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture, the current sub-block scan location ( xS, yS ), the previously decoded bins of the syntax element sb\_coded\_flag and the binary logarithm of the transform block width log2TbWidth and the transform block height log2TbHeight.

Output of this process is the variable ctxInc.

The variable csbfCtx is derived using the current location ( xS, yS ), two previously decoded bins of the syntax element sb\_coded\_flag in scan order, log2TbWidth and log2TbHeight, as follows:

* The variables log2SbWidth and log2SbHeight are dervied as follows:

log2SbWidth = ( Min( log2TbWidth, log2TbHeight ) < 2 ? 1 : 2 ) (1567)

log2SbHeight = log2SbWidth (1568)

* The variables log2SbWidth and log2SbHeight are modified as follows:
* If log2TbWidth is less than 2 and cIdx is equal to 0, the following applies

log2SbWidth = log2TbWidth (1569)

log2SbHeight = 4 − log2SbWidth (1570)

* Otherwise, if log2TbHeight is less than 2 and cIdx is equal to 0, the following applies

log2SbHeight = log2TbHeight (1571)

log2SbWidth = 4 − log2SbHeight (1572)

* The variable csbfCtx is initialized with 0 and modified as follows:
* If transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] is equal to 1 and sh\_ts\_residual\_coding\_disabled\_flag is equal to 0, the following applies:
* When xS is greater than 0, csbfCtx is modified as follows:

csbfCtx += sb\_coded\_flag[ xS − 1 ][ yS ] (1573)

* When yS is greater than 0, csbfCtx is modified as follows:

csbfCtx += sb\_coded\_flag[ xS ][ yS − 1 ] (1574)

* Otherwise (transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] is equal to 0 or sh\_ts\_residual\_coding\_disabled\_flag is equal to 1), the following applies:
* When xS is less than ( 1  <<  ( log2TbWidth − log2SbWidth ) ) − 1, csbfCtx is modified as follows:

csbfCtx += sb\_coded\_flag[ xS + 1 ][ yS ] (1575)

* When yS is less than ( 1  <<  ( log2TbHeight − log2SbHeight ) ) − 1, csbfCtx is modified as follows:

csbfCtx += sb\_coded\_flag[ xS ][ yS + 1 ] (1576)

The context index increment ctxInc is derived using the colour component index cIdx and csbfCtx as follows:

* If transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] is equal to 1 and sh\_ts\_residual\_coding\_disabled\_flag is equal to 0, ctxInc is derived as follows:

ctxInc = 4 + csbfCtx (1577)

* Otherwise (transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] is equal to 0 or sh\_ts\_residual\_coding\_disabled\_flag is equal to 1), ctxInc is derived as follows:
* If cIdx is equal to 0, the following applies:

ctxInc = Min( csbfCtx, 1 ) (1578)

* Otherwise (cIdx is greater than 0), ctxInc is derived as follows:

ctxInc = 2 + Min( csbfCtx, 1 ) (1579)

##### Derivation process for the variables locNumSig, locSumAbsPass1

Inputs to this process are the colour component index cIdx, the luma location ( x0, y0 ) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture, the current coefficient scan location ( xC, yC ), the binary logarithm of the transform block width log2TbWidth, and the binary logarithm of the transform block height log2TbHeight.

Outputs of this process are the variables locNumSig and locSumAbsPass1.

Given the syntax elements sig\_coeff\_flag[ x ][ y ] and the array AbsLevelPass1[ x ][ C ] for the transform block with component index cIdx and the top-left luma location ( x0, y0 ), the variables locNumSig and locSumAbsPass1 are derived as specified by the following pseudo code:

locNumSig = 0  
locSumAbsPass1 = 0  
if( transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] && !sh\_ts\_residual\_coding\_disabled\_flag ) {  
 if( xC > 0 ) {  
 locNumSig += sig\_coeff\_flag[ xC − 1 ][ yC ]  
 locSumAbsPass1 += AbsLevelPass1[ xC − 1 ][ yC ]  
 }  
 if( yC > 0 ) {  
 locNumSig += sig\_coeff\_flag[ xC ][ yC − 1 ]   
 locSumAbsPass1 += AbsLevelPass1[ xC ][ yC − 1 ]  
 }  
} else {  
 if( xC < ( 1 << log2TbWidth ) − 1 ) {  
 locNumSig += sig\_coeff\_flag[ xC + 1 ][ yC ]  
 locSumAbsPass1 += AbsLevelPass1[ xC + 1 ][ yC ]  
 if( xC < ( 1 << log2TbWidth ) − 2 ) {  
 locNumSig += sig\_coeff\_flag[ xC + 2 ][ yC ]   
 locSumAbsPass1 += AbsLevelPass1[ xC + 2 ][ yC ]  
 }  
 if( yC < ( 1 << log2TbHeight ) − 1 ) {  
 locNumSig += sig\_coeff\_flag[ xC + 1 ][ yC + 1 ] (1580)  
 locSumAbsPass1 += AbsLevelPass1[ xC + 1 ][ yC + 1 ]  
 }  
 }  
 if( yC < ( 1 << log2TbHeight ) − 1 ) {  
 locNumSig += sig\_coeff\_flag[ xC ][ yC + 1 ]   
 locSumAbsPass1 += AbsLevelPass1[ xC ][ yC + 1 ]  
 if( yC < ( 1 << log2TbHeight ) − 2 ) {  
 locNumSig += sig\_coeff\_flag[ xC ][ yC + 2 ]   
 locSumAbsPass1 += AbsLevelPass1[ xC ][ yC + 2 ]  
 }  
 }  
}

##### Derivation process of ctxInc for the syntax element sig\_coeff\_flag

Inputs to this process are the colour component index cIdx, the luma location ( x0, y0 ) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture, the current coefficient scan location ( xC, yC ), the binary logarithm of the transform block width log2TbWidth, and the binary logarithm of the transform block height log2TbHeight.

Output of this process is the variable ctxInc.

The variable locSumAbsPass1 is derived by invoking the derivation process for the variables locNumSig and locSumAbsPass1 specifies in clause 9.3.4.2.7 with colour component index cIdx, the luma location ( x0, y0), the current coefficient scan location (xC, yC ), the binary logarithm of the transform block width log2TbWidth, and the binary logarithm of the transform block height log2TbHeight as input.

The variable d is set equal to xC + yC.

The variable ctxInc is derived as follows:

* If transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] is equal to 1 and sh\_ts\_residual\_coding\_disabled\_flag is equal to 0, the following applies:

ctxInc = 60 + locNumSig (1581)

* Otherwise (transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] is equal to 0 or sh\_ts\_residual\_coding\_disabled\_flag is equal to 1), the following applies:
* If cIdx is equal to 0, ctxInc is derived as follows:

ctxInc = 12 \* Max( 0, QState − 1) + Min( ( locSumAbsPass1 + 1 )  >>  1, 3 ) + (1582)  
 ( d < 2 ? 8 : ( d < 5 ? 4 : 0 ) )

* Otherwise (cIdx is greater than 0), ctxInc is derived as follows:

ctxInc = 36 + 8 \* Max( 0, QState − 1) + Min( ( locSumAbsPass1 + 1 )  >>  1, 3 ) + ( d < 2 ? 4 : 0 ) (1583)

##### Derivation process of ctxInc for the syntax elements par\_level\_flag and abs\_level\_gtx\_flag

Inputs to this process are the colour component index cIdx, the luma location ( x0, y0 ) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture, the current coefficient scan location ( xC, yC ), the binary logarithm of the transform block width log2TbWidth, and the binary logarithm of the transform block height log2TbHeight.

Output of this process is the variable ctxInc.

The variable ctxInc is derived as follows:

* If transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] is equal to 1 and sh\_ts\_residual\_coding\_disabled\_flag is equal to 0, the following applies:
* If the syntax element is par\_level\_flag, the following applies:

ctxInc = 33 (1584)

* Otherwise, if the syntax element is abs\_level\_gtx\_flag[ n ][ 0 ], the following applies:
* If BdpcmFlag[ x0 ][ y0 ][ cIdx ] is equal to 1, ctxInc is derived as follows:

ctxInc = 68 (1585)

* Otherwise, if xC is greater than 0 and yC is greater than 0, ctxInc is derived as follows:

ctxInc = 65 + sig\_coeff\_flag[ xC − 1 ][ yC ] + sig\_coeff\_flag[ xC ][ yC − 1 ] (1586)

* Otherwise, if xC is greater than 0, ctxInc is derived as follows:

ctxInc = 65 + sig\_coeff\_flag[ xC − 1 ][ yC ] (1587)

* Otherwise, if yC is greater than 0, ctxInc is derived as follows:

ctxInc = 65 + sig\_coeff\_flag[ xC ][ yC − 1 ] (1588)

* Otherwise, ctxInc is derived as follows:

ctxInc = 65 (1589)

* Otherwise, if the syntax element is abs\_level\_gtx\_flag[ n ][ j ] with j > 0, the following applies:

ctxInc = 68 + j (1590)

* Otherwise (transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] is equal to 0 or sh\_ts\_residual\_coding\_disabled\_flag is equal to 1), the following applies:
* The variable locNumSig and locSumAbsPass1 is derived by invoking the derivation process for the variables locNumSig and locSumAbsPass1 specifies in clause 9.3.4.2.7 with colour component index cIdx, the luma location ( x0, y0), the current coefficient scan location (xC, yC ), the binary logarithm of the transform block width log2TbWidth, and the binary logarithm of the transform block height log2TbHeight as input.
* The variable ctxOffset is set equal to Min( locSumAbsPass1 − locNumSig, 4 ).
* The variable d is set equal to xC + yC.
* If xC is equal to LastSignificantCoeffX and yC is equal to LastSignificantCoeffY, ctxInc is derived as follows:

ctxInc = ( cIdx  = =  0 ? 0 : 21 ) (1591)

* Otherwise, if cIdx is equal to 0, ctxInc is derived as follows:

ctxInc = 1 + ctxOffset + ( d  = =  0 ? 15 : ( d < 3 ? 10 : ( d < 10 ? 5 : 0 ) ) ) (1592)

* Otherwise (cIdx is greater than 0), ctxInc is derived as follows:

ctxInc = 22 + ctxOffset + ( d  = =  0 ? 5 : 0 ) (1593)

##### Derivation process of ctxInc for the syntax element coeff\_sign\_flag for transform skip mode

Inputs to this process are the colour component index cIdx, the luma location ( x0, y0 ) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture, the current coefficient scan location ( xC, yC )

Output of this process is the variable ctxInc.

The variables leftSign and aboveSign are derived as follows:

leftSign = ( xC = = 0 ) ? 0 : CoeffSignLevel[ xC − 1 ][ yC ] (1594)

aboveSign = ( yC = = 0 ) ? 0 : CoeffSignLevel[ xC ][ yC − 1 ] (1595)

The variable ctxInc is derived as follows:

* If leftSign is equal to 0 and aboveSign is equal to 0, or if leftSign is equal to −aboveSign, the following applies:

ctxInc = ( BdpcmFlag[ x0 ][ y0 ][ cIdx ]  = =  0 ? 0 : 3 ) (1596)

* Otherwise, if leftSign is greater than or equal to 0 and aboveSign is greater than or equal to 0, the following applies:

ctxInc = ( BdpcmFlag[ x0 ][ y0 ][ cIdx ] ? 1 : 4 ) (1597)

* Otherwise, the following applies:

ctxInc = ( BdpcmFlag[ x0 ][ y0 ][ cIdx ] ? 2 : 5 ) (1598)

##### Derivation process of ctxInc for the syntax element run\_copy\_flag

Inputs to this process are the variables PreviousRunType, PreviousRunPosition, and the current scan position curPos.

Output of this process is the variable ctxInc.

The variable binDist is set equal to curPos − PreviousRunPosition − 1.

The variable ctxInc is provided by Table 133 depending on binDist and PreviousRunType.

Table 133 – Specification of ctxInc depending on binDist and PreviousRunType

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **binDist** | **0** | **1** | **2** | **3** | **>=4** |
| PreviousRunType = = 1 | 5 | 6 | 6 | 7 | 7 |
| PreviousRunType = = 0 | 0 | 1 | 2 | 3 | 4 |

#### Arithmetic decoding process

##### General

Inputs to this process are ctxTable, ctxIdx, and bypassFlag, as derived in subclause 9.3.4.2, and the state variables ivlCurrRange and ivlOffset of the arithmetic decoding engine.

Output of this process is the value of the bin.

Figure 15 illustrates the whole arithmetic decoding process for a single bin. For decoding the value of a bin, the context index table ctxTable, the ctxIdx and the bypassFlag are passed to the arithmetic decoding process DecodeBin( ctxTable, ctxIdx, bypassFlag ), which is specified as follows:

– If bypassFlag is equal to 1, DecodeBypass( ) as specified in subclause 9.3.4.3.4 is invoked.

– Otherwise, if bypassFlag is equal to 0, ctxTable is equal to 0, and ctxIdx is equal to 0, DecodeTerminate( ) as specified in subclause 9.3.4.3.5 is invoked.

– Otherwise (bypassFlag is equal to 0 and ctxTable is not equal to 0), DecodeDecision( ctxTable, ctxIdx ) as specified in subclause 9.3.4.3.2 is invoked.



Figure 15 – Overview of the arithmetic decoding process for a single bin (informative)

NOTE – Arithmetic coding is based on the principle of recursive interval subdivision. Given a probability estimation p( 0 ) and p( 1 ) = 1 − p( 0 ) of a binary decision ( 0, 1 ), an initially given code sub-interval with the range ivlCurrRange will be subdivided into two sub-intervals having range p( 0 ) \* ivlCurrRange and ivlCurrRange − p( 0 ) \* ivlCurrRange, respectively. Depending on the decision, which has been observed, the corresponding sub-interval will be chosen as the new code interval, and a binary code string pointing into that interval will represent the sequence of observed binary decisions. It is useful to distinguish between the most probable symbol(MPS) and the least probable symbol(LPS), so that binary decisions have to be identified as either MPS or LPS, rather than 0 or 1. Given this terminology, each context is specified by the probability pLPS of the LPS and the value of MPS (valMps), which is either 0 or 1. The arithmetic core engine in this Specification has three distinct properties:

– The probability estimation is performed by means of a finite-state machine with a table-based transition process between 64 different representative probability states { pLPS( pStateIdx ) | 0  <=  pStateIdx < 64 } for the LPS probability pLPS. The numbering of the states is arranged in such a way that the probability state with indexpStateIdx = 0 corresponds to an LPS probability value of 0.5, with decreasing LPS probability towards higher state indices.

– The range ivlCurrRange representing the state of the coding engine is quantized to a small set {Q1,…,Q4} of pre-set quantization values prior to the calculation of the new interval range. Storing a table containing all 64×4 pre-computed product values of Qi \* pLPS( pStateIdx ) allows a multiplication-free approximation of the product ivlCurrRange \* pLPS( pStateIdx ).

– For syntax elements or parts thereof for which an approximately uniform probability distribution is assumed to be given a separate simplified encoding and decoding bypass process is used.

##### Arithmetic decoding process for a binary decision

###### General

Inputs to this process are the variables ctxTable, ctxIdx, ivlCurrRange, and ivlOffset.

Outputs of this process are the decoded value binVal, and the updated variables ivlCurrRange and ivlOffset.

Figure 16 shows the flowchart for decoding a single decision (DecodeDecision):

1. The value of the variable ivlLpsRange is derived as follows:

– Given the current value of ivlCurrRange, the variable qRangeIdx is derived as follows:

qRangeIdx = ivlCurrRange >> 5 (1599)

– Given qRangeIdx, pStateIdx0 and pStateIdx1 associated with ctxTable and ctxIdx, valMps and ivlLpsRange are derived as follows:

pState = pStateIdx1 + 16 \* pStateIdx0  
valMps = pState >> 14  
ivlLpsRange = ( qRangeIdx \* ( (valMps ? 32767 − pState : pState ) >> 9 ) >> 1 ) + 4 (1600)

1. The variable ivlCurrRange is set equal to ivlCurrRange − ivlLpsRange and the following applies:

– If ivlOffset is greater than or equal to ivlCurrRange, the variable binVal is set equal to 1 − valMps, ivlOffset is decremented by ivlCurrRange, and ivlCurrRange is set equal to ivlLpsRange.

– Otherwise, the variable binVal is set equal to valMps.

Given the value of binVal, the state transition isperformed as specified in subclause 9.3.4.3.2.2. Depending on the current value of ivlCurrRange, renormalization is performed as specified in subclause 9.3.4.3.3.

###### State transition process

Inputs to this process are the current pStateIdx0 and pStateIdx1, and the decoded value binVal.

Outputs of this process are the updated pStateIdx0 and pStateIdx1 of the context variable associated with ctxIdx.

The variable ctxIdxOffset is specified in Table 51 depending on the current value of initType and ctxInc set equal to ctxIdx − ctxIdxOffset.

The variables shift0 and shift1 are derived from the shiftIdx value associated with ctxTable and ctxInc in clause 9.3.2.2.

shift0 = (shiftIdx >> 2) + 2   
shift1 = (shiftIdx & 3) + 3 + shift0 (1601)

Depending on the decoded value binVal, the update of the two variables pStateIdx0 and pStateIdx1 associated with ctxIdx is derived as follows:

pStateIdx0 = pStateIdx0 − (pStateIdx0 >> shift0) + (1023 \* binVal >> shift0)  
pStateIdx1 = pStateIdx1 − (pStateIdx1 >> shift1) + (16383 \* binVal >> shift1) (1602)



Figure 16 – Flowchart for decoding a decision

##### Renormalization process in the arithmetic decoding engine

Inputs to this process are bits from slice data and the variables ivlCurrRange and ivlOffset.

Outputs of this process are the updated variables ivlCurrRange and ivlOffset.

A flowchart of the renormalization is shown in Figure 17. The current value of ivlCurrRange is first compared to 256 and further steps are specified as follows:

– If ivlCurrRange is greater than or equal to 256, no renormalization is needed and the RenormD process is finished;

– Otherwise (ivlCurrRange is less than 256), the renormalization loop is entered. Within this loop, the value of ivlCurrRange is doubled, i.e. left-shifted by 1 and a single bit is shifted into ivlOffset by using read\_bits( 1 ).

The bitstream shall not contain data that result in a value of ivlOffset being greater than or equal to ivlCurrRange upon completion of this process.



Figure 17 – Flowchart of renormalization

##### Bypass decoding process for binary decisions

Inputs to this process are bits from slice data and the variables ivlCurrRange and ivlOffset.

Outputs of this process are the updated variable ivlOffset and the decoded value binVal.

The bypass decoding process is invoked when bypassFlag is equal to 1. Figure 18 shows a flowchart of the corresponding process.

First, the value of ivlOffset is doubled, i.e. left-shifted by 1 and a single bit is shifted into ivlOffset by using read\_bits( 1 ). Then, the value of ivlOffset is compared to the value of ivlCurrRange and further steps are specified as follows:

– If ivlOffset is greater than or equal to ivlCurrRange, the variable binVal is set equal to 1 and ivlOffset is decremented by ivlCurrRange.

– Otherwise (ivlOffset is less than ivlCurrRange), the variable binVal is set equal to 0*.*

The bitstream shall not contain data that result in a value of ivlOffset being greater than or equal to ivlCurrRange upon completion of this process.



Figure 18 – Flowchart of bypass decoding process

##### Decoding process for binary decisions before termination

Inputs to this process are bits from slice data and the variables ivlCurrRange and ivlOffset.

Outputs of this process are the updated variables ivlCurrRange and ivlOffset, and the decoded value binVal.

This decoding process applies to decoding of end\_of\_slice\_one\_bit, end\_of\_tile\_one\_bit, and end\_of\_subset\_one\_bit corresponding to ctxTable equal to 0 and ctxIdx equal to 0. Figure 19 shows the flowchart of the corresponding decoding process, which is specified as follows:

First, the value of ivlCurrRange is decremented by 2. Then, the value of ivlOffset is compared to the value of ivlCurrRange and further steps are specified as follows:

– If ivlOffset is greater than or equal to ivlCurrRange, the variable binVal is set equal to 1, no renormalization is carried out, and CABAC decoding is terminated. The last bit inserted in register ivlOffset is equal to 1. When decoding end\_of\_slice\_one\_bit, this last bit inserted in register ivlOffset is interpreted as rbsp\_stop\_one\_bit. When decoding end\_of\_tile\_one\_bit or end\_of\_subset\_one\_bit, this last bit inserted in register ivlOffset is interpreted as alignment\_bit\_equal\_to\_one.

– Otherwise (ivlOffset is less than ivlCurrRange), the variable binVal is set equal to 0 and renormalization is performed as specified in subclause 9.3.4.3.3.

NOTE – This procedure may also be implemented using DecodeDecision( ctxTable, ctxIdx, bypassFlag ) with ctxTable = 0, ctxIdx = 0 and bypassFlag = 0. In the case where the decoded value is equal to 1, seven more bits would be read by DecodeDecision( ctxTable, ctxIdx, bypassFlag ) and a decoding process would have to adjust its bitstream pointer accordingly to properly decode following syntax elements.



Figure 19 – Flowchart of decoding a decision before termination

1. Annex A  
     
   Profiles, tiers and levels

(This annex forms an integral part of this Recommendation | International Standard.)

* 1. Overview of profiles, tiers and levels

Profiles, tiers and levels specify restrictions on bitstreams and hence limits on the capabilities needed to decode the bitstreams. Profiles, tiers and levels may also be used to indicate interoperability points between individual decoder implementations.

Each profile specifies a subset of algorithmic features and limits that shall be supported by all decoders conforming to that profile.

NOTE 1 – Encoders are not required to make use of any particular subset of features supported in a profile.

Each level of a tier specifies a set of limits on the values that may be taken by the syntax elements of this Specification. The same set of tier and level definitions is usually used with all profiles, but individual implementations may support a different tier and within a tier a different level for each supported profile. For any given profile, a level of a tier generally corresponds to a particular decoder processing load and memory capability.

In this annex, phrases like "the bitstream" should be intepreted as "the bitstream of the operation point", and phrases like "AU n" and "a layer" should be interpreted as "AU n in the bitstream of the operation point" and "a layer in the bitstream of the operation point", respectively, where "the operation point" is the operation point with which the profile, tier, or level is associated with.

For each operation point identified by TargetOlsIdx and Htid, the profile, tier, and level information is indicated through general\_profile\_idc and general\_tier\_flag in the VPS, and sublayer\_level\_idc[ Htid ] found in or derived from the VPS.

When no VPS is available, the profile and tier information is indicated through general\_profile\_idc and general\_tier\_flag in the SPS, and the level information is indicated as follows:

* If Htid is provided by external means indicating the highest TemporalId of any NAL unit in the bitstream, level information is indicated through sublayer\_level\_idc[ Htid ] found in or derived from the SPS.
* Otherwise (Htid is not provided by external means), level information is indicated through general\_level\_idc in the SPS.

NOTE 2 – Decoders are not required to extract a subset of the bitstream; any such extraction process that may be a part of the system is considered to be outside of the scope of the decoding process. The values TargetOlsIdx and Htid, are not necessary for the operation of the decoding process. Values of TargetOlsIdx and Htid may be provided by external means and may be used to check the conformance of the bitstream.

* 1. Requirements on video decoder capability

Capabilities of video decoders conforming to this Specification are specified in terms of the ability to decode video streams conforming to the constraints of profiles, tiers and levels specified in this annex and other annexes. When expressing the capabilities of a decoder for a specified profile, the tier and level supported for that profile should also be expressed.

Specific values are specified in this annex for the syntax elements general\_profile\_idc, general\_tier\_flag, general\_level\_idc, and sublayer\_level\_idc[ i ]. All other values of general\_profile\_idc, general\_level\_idc, and sublayer\_level\_idc[ i ] are reserved for future use by ITU-T | ISO/IEC.

NOTE 3– Decoders should not infer that a reserved value of general\_profile\_idc between the values specified in this Specification indicates intermediate capabilities between the specified profiles, as there are no restrictions on the method to be chosen by ITU-T | ISO/IEC for the use of such future reserved values. However, decoders should infer that a reserved value of general\_level\_idc or sublayer\_level\_idc[ i ] associated with a particular value of general\_tier\_flag between the values specified in this Specification indicates intermediate capabilities between the specified levels of the tier.

* 1. Profiles
     1. Main 10 and Main 10 Still Picture profiles

Bitstreams conforming to the Main 10 or Main 10 Still Picture profile shall obey the following constraints:

* In a bitstream conforming to the Main 10 Still Picture profile, the bitstream shall contain only one picture.
* Referenced SPSs shall have sps\_chroma\_format\_idc equal to 0 or 1.
* Referenced SPSs shall have sps\_bit\_depth\_minus8 in the range of 0 to 2, inclusive.
* In a bitstream conforming to the Main 10 Still Picture profile, the referenced SPS shall have max\_dec\_pic\_buffering\_minus1[ sps\_max\_sublayers\_minus1 ] equal to 0. [Ed. (YK): Maybe it's better not to have this constraint for the Main 10 Still Picture profile, such that when extracting an intra picture from a Main 10 bitstream to form a Main 10 Still Picure bitstream, the extractor/"encoder" does not have to change the value of max\_dec\_pic\_buffering\_minus1[ ] in the SPS.]
* Referenced SPSs shall have sps\_palette\_enabled\_flag equal to 0.
* In a bitstream conforming to the Main 10 profile that do not conform to the Main 10 Still Picture profile, general\_level\_idc and sublayer\_level\_idc[ i ] for all values of i in the referenced VPS (when available) and in the referenced SPSs shall not be equal to 255 (which indicates level 15.5).
* The tier and level constraints specified for the Main 10 or Main 10 Still Picture profile in clause A.4, as applicable, shall be fulfilled.

Conformance of a bitstream to the Main 10 profile is indicated by general\_profile\_idc being equal to 1.

Conformance of a bitstream to the Main 10 Still Picture profile is indicated by general\_one\_picture\_only\_constraint\_flag being equal to 1 together with general\_profile\_idc being equal to 1.

NOTE – When the conformance of a bitstream to the Main 10 Still Picture profile is indicated as specified above, and the indicated level is not level 15.5, the conditions for indication of the conformance of the bitstream to the Main 10 profile are also fulfilled.

Decoders conforming to the Main 10 profile at a specific level of a specific tier shall be capable of decoding all bitstreams for which all of the following conditions apply:

* The bitstream is indicated to conform to the Main 10 or Main 10 Still Picture profile.
* The bitstream is indicated to conform to a tier that is lower than or equal to the specified tier.
* The bitstream is indicated to conform to a level that is not level 15.5 and is lower than or equal to the specified level.

Decoders conforming to the Main 10 Still Picture profile at a specific level of a specific tier shall be capable of decoding all bitstreams for which all of the following conditions apply:

* The bitstream is indicated to conform to the Main 10 Still Picture profile.
* The bitstream is indicated to conform to a tier that is lower than or equal to the specified tier.
* The bitstream is indicated to conform to a level that is not level 15.5 and is lower than or equal to the specified level.
  + 1. Main 4:4:4 10 and Main 4:4:4 10 Still Picture profiles

Bitstreams conforming to the Main 4:4:4 10 or Main 4:4:4 10 Still Picture profile shall obey the following constraints:

* In a bitstream conforming to the Main 4:4:4 10 Still Picture profile, the bitstream shall contain only one picture.
* Referenced SPSs shall have sps\_chroma\_format\_idc in the range of 0 to 3, inclusive.
* Referenced SPSs shall have sps\_bit\_depth\_minus8 in the range of 0 to 2, inclusive.
* In a bitstream conforming to the Main 4:4:4 10 Still Picture profile, the referenced SPS shall have max\_dec\_pic\_buffering\_minus1[ sps\_max\_sublayers\_minus1 ] equal to 0. [Ed. (YK): Maybe it's better not to have this constraint for the Main 4:4:4 10 Still Picture profile, such that when extracting an intra picture from a Main 4:4:4 10 bitstream to form a Main 4:4:4 10 Still Picure bitstream, the extractor/"encoder" does not have to change the value of max\_dec\_pic\_buffering\_minus1[ ] in the SPS.]
* In a bitstream conforming to the Main 4:4:4 10 profile that do not conform to the Main 4:4:4 10 Still Picture profile, general\_level\_idc and sublayer\_level\_idc[ i ] for all values of i in the referenced VPS (when available) and in the referenced SPSs shall not be equal to 255 (which indicates level 15.5).
* The tier and level constraints specified for the Main 4:4:4 10 or Main 4:4:4 10 Still Picture profile in clause A.4, as applicable, shall be fulfilled.

Conformance of a bitstream to the Main 4:4:4 10 profile is indicated by general\_profile\_idc being equal to 2.

Conformance of a bitstream to the Main 4:4:4 10 Still Picture profile is indicated by general\_one\_picture\_only\_constraint\_flag being equal to 1 together with general\_profile\_idc being equal to 2.

NOTE – When the conformance of a bitstream to the Main 10 4:4:4 Still Picture profile is indicated as specified above, and the indicated level is not level 15.5, the conditions for indication of the conformance of the bitstream to the Main 10 4:4:4 profile are also fulfilled.

Decoders conforming to the Main 4:4:4 10 profile at a specific level of a specific tier shall be capable of decoding all bitstreams for which all of the following conditions apply:

* The bitstream is indicated to conform to the Main 4:4:4 10, Main 10, Main 4:4:4 10 Still Picture, or Main 10 Still Picture profile.
* The bitstream is indicated to conform to a tier that is lower than or equal to the specified tier.
* The bitstream is indicated to conform to a level that is not level 15.5 and is lower than or equal to the specified level.

Decoders conforming to the Main 4:4:4 10 Still Picture profile at a specific level of a specific tier shall be capable of decoding all bitstreams for which all of the following conditions apply:

* The bitstream is indicated to conform to the Main 4:4:4 10 Still Picture or Main 10 Still Picture profile.
* The bitstream is indicated to conform to a tier that is lower than or equal to the specified tier.
* The bitstream is indicated to conform to a level that is not level 15.5 and is lower than or equal to the specified level.
  1. Tiers and levels
     1. General tier and level limits

For purposes of comparison of tier capabilities, the tier with general\_tier\_flag equal to 0 is considered to be a lower tier than the tier with general\_tier\_flag equal to 1.

For purposes of comparison of level capabilities, a particular level of a specific tier is considered to be a lower level than some other level of the same tier when the value of the general\_level\_idc or sublayer\_level\_idc[ i ] of the particular level is less than that of the other level.

The following is specified for expressing the constraints in this annex:

* Let AU n be the n-th AU in decoding order, with the first AU being AU 0 (i.e., the 0-th AU).
* For an OLS with OLS index TargetOlsIdx, the variables PicWidthMaxInSamplesY, PicHeightMaxInSamplesY, and PicSizeMaxInSamplesY, and the applicable dpb\_parameters( ) syntax structure are derived as follows:
  + If NumLayersInOls[ TargetOlsIdx ] is equal to 1, PicWidthMaxInSamplesY is set equal to sps\_pic\_width\_max\_in\_luma\_samples, PicHeightMaxInSamplesY is set equal to sps\_pic\_height\_max\_in\_luma\_samples, PicSizeMaxInSamplesY is set equal to PicWidthMaxInSamplesY \* PicHeightMaxInSamplesY, where sps\_pic\_width\_max\_in\_luma\_samples and sps\_pic\_height\_max\_in\_luma\_samples are found in the SPS referred to by the layer in the OLS, and the applicable dpb\_parameters( ) syntax structure is also found in that SPS.
  + Otherwise (NumLayersInOls[ TargetOlsIdx ] is greater than 1), PicWidthMaxInSamplesY is set equal to vps\_ols\_dpb\_pic\_width[ MultiLayerOlsIdx[ TargetOlsIdx ] ], PicHeightMaxInSamplesY is set equal to vps\_ols\_dpb\_pic\_height[ MultiLayerOlsIdx[ TargetOlsIdx ] ], PicSizeMaxInSamplesY is set equal to PicWidthMaxInSamplesY \* PicHeightMaxInSamplesY, and the applicable dpb\_parameters( ) syntax structure is identified by vps\_ols\_dpb\_params\_idx[ MultiLayerOlsIdx[ TargetOlsIdx ] ] found in the VPS.

When the specified level is not level 15.5, bitstreams conforming to a profile at a specified tier and level shall obey the following constraints for each bitstream conformance test as specified in Annex C:

1. PicSizeMaxInSamplesY shall be less than or equal to MaxLumaPs, where MaxLumaPs is specified in Table A.1.
2. The value of PicWidthMaxInSamplesY shall be less than or equal to Sqrt( MaxLumaPs \* 8 ).
3. The value of PicHeightMaxInSamplesY shall be less than or equal to Sqrt( MaxLumaPs \* 8 ).
4. For each referenced PPS, the value of NumTileColumns shall be less than or equal to MaxTileCols and the value of NumTileRows shall be less than or equal to MaxTileRows, where MaxTileCols and MaxTileRows are specified in Table A.1.
5. For the VCL HRD parameters, CpbSize[Htid][ i ] shall be less than or equal to CpbVclFactor \* MaxCPB for at least one value of i in the range of 0 to hrd\_cpb\_cnt\_minus1, inclusive, where CpbSize[Htid][ i ] is specified in clause 7.4.6.3 based on parameters selected as specified in clause C.1, CpbVclFactor is specified in Table A.3 and MaxCPB is specified in Table A.1 in units of CpbVclFactor bits.
6. For the NAL HRD parameters, CpbSize[Htid][ i ] shall be less than or equal to CpbNalFactor \* MaxCPB for at least one value of i in the range of 0 to hrd\_cpb\_cnt\_minus1, inclusive, where CpbSize[Htid][ i ] is specified in clause 7.4.6.3 based on parameters selected as specified in clause C.1, CpbNalFactor is specified in Table A.3, and MaxCPB is specified in Table A.1 in units of CpbNalFactor bits.

Table A.1 specifies the limits for each level of each tier for levels other than level 15.5.

A tier and level to which a bitstream conforms are indicated by the syntax elements general\_tier\_flag and general\_level\_idc, and a level to which a sublayer representation conforms are indicated by the syntax element sublayer\_level\_idc[ i ], as follows:

– If the specified level is not level 15.5, general\_tier\_flag equal to 0 indicates conformance to the Main tier, general\_tier\_flag equal to 1 indicates conformance to the High tier, according to the tier constraints specified in Table A.1 and general\_tier\_flag shall be equal to 0 for levels below level 4 (corresponding to the entries in Table A.1 marked with "-"). Otherwise (the specified level is level 15.5), it is a requirement of bitstream conformance that general\_tier\_flag shall be equal to 1 and the value 0 for general\_tier\_flag is reserved for future use by ITU-T | ISO/IEC and decoders shall ignore the value of general\_tier\_flag.

– general\_level\_idc and sublayer\_level\_idc[ i ] shall be set equal to a value of general\_level\_idc for the level number specified in Table A.1.

Table A.1 – General tier and level limits

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Level** | **general\_level\_idc value\*** | **Max luma picture size MaxLumaPs (samples)** | **Max CPB size MaxCPB (CpbVclFactor or CpbNalFactor bits)** | | **Max slices per picture MaxSlicesPerPicture** | **Max # of tile rows MaxTileRows** | **Max # of tile columns MaxTileCols** |
| **Main tier** | **High tier** |
| **1.0** | 16 | 36 864 | 350 | - | 16 | 1 | 1 |
| **2.0** | 32 | 122 880 | 1 500 | - | 16 | 1 | 1 |
| **2.1** | 35 | 245 760 | 3 000 | - | 20 | 1 | 1 |
| **3.0** | 48 | 552 960 | 6 000 | - | 30 | 2 | 2 |
| **3.1** | 51 | 983 040 | 10 000 | - | 40 | 3 | 3 |
| **4.0** | 64 | 2 228 224 | 12 000 | 30 000 | 75 | 5 | 5 |
| **4.1** | 67 | 2 228 224 | 20 000 | 50 000 | 75 | 5 | 5 |
| **5.0** | 80 | 8 912 896 | 25 000 | 100 000 | 200 | 11 | 10 |
| **5.1** | 83 | 8 912 896 | 40 000 | 160 000 | 200 | 11 | 10 |
| **5.2** | 86 | 8 912 896 | 60 000 | 240 000 | 200 | 11 | 10 |
| **6.0** | 96 | 35 651 584 | 80 000 | 240 000 | 600 | 22 | 20 |
| **6.1** | 99 | 35 651 584 | 120 000 | 480 000 | 600 | 22 | 20 |
| **6.2** | 102 | 35 651 584 | 180 000 | 800 000 | 600 | 22 | 20 |
| \*For the level number in the form of major.minor, the value of general\_level\_idc for each of the above-lisetd levels is equal to major \* 16 + minor \* 3. | | | | | | | |

* + 1. Profile-specific level limits

The following is specified for expressing the constraints in this annex:

* Let the variable fR be set equal to 1 ÷ 300.

The variable HbrFactor is defined as follows:

* If the bitstream is indicated to conform to the Main 10 profile or the Main 4:4:4 10 profile, HbrFactor is set equal to 1.

The variable BrVclFactor, which represents the VCL bit rate scale factor, is set equal to CpbVclFactor \* HbrFactor.

The variable BrNalFactor, which represents the NAL bit rate scale factor, is set equal to CpbNalFactor \* HbrFactor.

The variable MinCr is set equal to MinCrBase \* MinCrScaleFactor ÷ HbrFactor.

When the specified level is not level 15.5, the value of max\_dec\_pic\_buffering\_minus1[ Htid ] + 1 shall be less than or equal to MaxDpbSize, which is derived as follows:

if( PicSizeMaxInSamplesY <= ( MaxLumaPs >> 2 ) )  
 MaxDpbSize = Min( 4 \* maxDpbPicBuf, 16 )  
else if( PicSizeMaxInSamplesY <= ( MaxLumaPs >> 1 ) )  
 MaxDpbSize = Min( 2 \* maxDpbPicBuf, 16 ) (A.1)  
else if( PicSizeMaxInSamplesY <= ( ( 3 \* MaxLumaPs ) >> 2 ) )  
 MaxDpbSize = Min( ( 4 \* maxDpbPicBuf ) / 3, 16 )  
else  
 MaxDpbSize = maxDpbPicBuf

where MaxLumaPs is specified in Table A.1, maxDpbPicBuf is equal to 8, and max\_dec\_pic\_buffering\_minus1[ Htid ] is found in or derived from the applicable dpb\_parameters( ) syntax structure.

Let numDecPics be the number of pictures in AU n. The variable AuSizeMaxInSamplesY[ n ] is set equal to PicSizeMaxInSamplesY \* numDecPics.

Bitstreams conforming to the Main 10 or Main 4:4:4 10 profile at a specified tier and level shall obey the following constraints for each bitstream conformance test as specified in Annex C:

1. The nominal removal time of AU n (with n greater than 0) from the CPB, as specified in clause C.2.3, shall satisfy the constraint that AuNominalRemovalTime[ n ] − AuCpbRemovalTime[ n − 1 ] is greater than or equal to Max( AuSizeMaxInSamplesY[ n − 1 ] ÷ MaxLumaSr, fR ), where MaxLumaSr is the value specified in that applies to AU n − 1.
2. The difference between consecutive output times of pictures of different AUs from the DPB, as specified in clause C.3.3, shall satisfy the constraint that DpbOutputInterval[ n ] is greater than or equal to Max( AuSizeMaxInSamplesY[ n  ÷ MaxLumaSr, fR ), where MaxLumaSr is the value specified in Table A.2 for AU n, provided that AU n has a picture that is output and AU n is not the last AU of the bitstream that has a picture that is output.
3. The removal time of AU 0 shall satisfy the constraint that the number of slices in each picture in AU 0 is less than or equal to Min( Max( 1, MaxSlicesPerPicture \* MaxLumaSr / MaxLumaPs \* ( AuCpbRemovalTime[ 0 ] − AuNominalRemovalTime[ 0 ] ) + MaxSlicesPerPicture \* PicSizeMaxInSamplesY / MaxLumaPs ), MaxSlicesPerPicture ), for the value of PicSizeMaxInSamplesY of picture 0, where MaxSlicesPerPicture, MaxLumaPs and MaxLumaSr are the values specified in Table A.1 and Table A.2, respectively, that apply to AU 0. [Ed (YK). Further study on whether MaxSlicesPerPicture used herein should be specified as MaxSlicesPerAu, as the other variables involved in the calculation are all at AU level.]
4. The difference between consecutive CPB removal times of AUs n and n − 1 (with n greater than 0) shall satisfy the constraint that the number of slices in each picture in AU n is less than or equal to Min( (Max( 1, MaxSlicesPerPicture \* MaxLumaSr / MaxLumaPs \* ( AuCpbRemovalTime[ n ] − AuCpbRemovalTime[ n − 1 ] ) ), MaxSlicesPerPicture ), where MaxSlicesPerPicture, MaxLumaPs and MaxLumaSr are the values specified in Table A.1 and Table A.2 that apply to AU n. [Ed (YK). Further study on whether MaxSlicesPerPicture used herein should be specified as MaxSlicesPerAu, as the other variables involved in the calculation are all at AU level.]
5. For the VCL HRD parameters, BitRate[Htid][ i ] shall be less than or equal to BrVclFactor \* MaxBR for at least one value of i in the range of 0 to hrd\_cpb\_cnt\_minus1, inclusive, where BitRate[Htid][ i ] is specified in clause 7.4.6.3 based on parameters selected as specified in clause C.1 and MaxBR is specified in Table A.2 in units of BrVclFactor bits/s.
6. For the NAL HRD parameters, BitRate[Htid][ i ] shall be less than or equal to BrNalFactor \* MaxBR for at least one value of i in the range of 0 to hrd\_cpb\_cnt\_minus1, inclusive, where BitRate[Htid][ i ] is specified in clause 7.4.6.3 based on parameters selected as specified in clause C.1 and MaxBR is specified in Table A.2 in units of BrNalFactor bits/s.
7. The sum of the NumBytesInNalUnit variables for AU 0 shall be less than or equal to FormatCapabilityFactor \* ( Max( AuSizeMaxInSamplesY[ 0 ], fR \* MaxLumaSr ) + MaxLumaSr \* ( AuCpbRemovalTime[ 0 ] − AuNominalRemovalTime[ 0 ] ) ) ÷ MinCr, where MaxLumaSr and FormatCapabilityFactor are the values specified in Table A.2 and Table A.3, respectively, that apply to AU 0.
8. The sum of the NumBytesInNalUnit variables for AU n (with n greater than 0) shall be less than or equal to FormatCapabilityFactor \* MaxLumaSr \* ( AuCpbRemovalTime[ n ] − AuCpbRemovalTime[ n − 1 ] ) ÷ MinCr, where MaxLumaSr and FormatCapabilityFactor are the values specified in Table A.2 and Table A.3 respectively, that apply to AU n.
9. The removal time of AU 0 shall satisfy the constraint that the number of tiles in each picture in AU 0 is less than or equal to Min( Max( 1, MaxTileCols \* MaxTileRows \* 120 \* ( AuCpbRemovalTime[ 0 ] − AuNominalRemovalTime[ 0 ] ) + MaxTileCols \* MaxTileRows \* AuSizeMaxInSamplesY[ 0 ] / MaxLumaPs ), MaxTileCols \* MaxTileRows ), where MaxTileCols and MaxTileRows are the values specified in Table A.1 that apply to AU 0. [Ed (YK). Further study on whether MaxTileCols and MaxTileRows used herein should be specified as for each AU instead of for each picture, as the other variables involved in the calculation are all at AU level.]
10. The difference between consecutive CPB removal times of AUs n and n − 1 (with n greater than 0) shall satisfy the constraint that the number of tiles in each picture in AU n is less than or equal to Min( Max( 1, MaxTileCols \* MaxTileRows \* 120 \* ( AuCpbRemovalTime[ n ] − AuCpbRemovalTime[ n − 1 ] ) ), MaxTileCols \* MaxTileRows ), where MaxTileCols and MaxTileRows are the values specified in Table A.1 that apply to AU n. [Ed (YK). Further study on whether MaxTileCols and MaxTileRows used herein should be specified as for each AU instead of for each picture, as the other variables involved in the calculation are all at AU level.]

Table A.2 – Tier and level limits for the video profiles

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Level** | **Max luma sample rate MaxLumaSr (samples/sec)** | **Max bit rate MaxBR (BrVclFactor or BrNalFactor bits/s)** | | **Min compression ratio MinCrBase** | |
| **Main tier** | **High tier** | **Main tier** | **High tier** |
| **1** | 552 960 | 128 | - | 2 | 2 |
| **2** | 3 686 400 | 1 500 | - | 2 | 2 |
| **2.1** | 7 372 800 | 3 000 | - | 2 | 2 |
| **3** | 16 588 800 | 6 000 | - | 2 | 2 |
| **3.1** | 33 177 600 | 10 000 | - | 2 | 2 |
| **4** | 66 846 720 | 12 000 | 30 000 | 4 | 4 |
| **4.1** | 133 693 440 | 20 000 | 50 000 | 4 | 4 |
| **5** | 267 386 880 | 25 000 | 100 000 | 6 | 4 |
| **5.1** | 534 773 760 | 40 000 | 160 000 | 8 | 4 |
| **5.2** | 1 069 547 520 | 60 000 | 240 000 | 8 | 4 |
| **6** | 1 069 547 520 | 60 000 | 240 000 | 8 | 4 |
| **6.1** | 2 139 095 040 | 120 000 | 480 000 | 8 | 4 |
| **6.2** | 4 278 190 080 | 240 000 | 800 000 | 8 | 4 |

Table A.3 – Specification of CpbVclFactor, CpbNalFactor, FormatCapabilityFactor and MinCrScaleFactor

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Profile** | **CpbVclFactor** | **CpbNalFactor** | **FormatCapabilityFactor** | **MinCrScaleFactor** |
| Main 10 | 1 000 | 1 100 | 1.875 | 1.00 |
| Main 4:4:4 10 | 2 500 | 2 750 | 3.750 | 0.75 |
| Main 10 Still Picture | 1 000 | 1 100 | 1.875 | 1.00 |
| Main 4:4:4 10 Still Picture | 2 500 | 2 750 | 3.750 | 0.75 |

Informative clause A.4.3 shows the effect of these limits on picture rates for several example picture formats.

* + 1. Effect of level limits on picture rate (informative)

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

Informative Table A.4 and Table A.5 provide examples of maximum picture rates for the Main 10 and Main 4:4:4 10 profiles.

Table A.4 – Maximum picture rates (pictures per second) at level 1 to 4.1 for some example picture sizes  
when MinCbSizeY is equal to 64

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Level:** |  |  |  | **1** | **2** | **2.1** | **3** | **3.1** | **4** | **4.1** |
| **Max luma picture size (samples):** |  |  |  | 36 864 | 122 880 | 245 760 | 552 960 | 983 040 | 2 228 224 | 2 228 224 |
| **Max luma sample rate (samples/sec)** |  |  |  | 552 960 | 3 686 400 | 7 372 800 | 16 588 800 | 33 177 600 | 66 846 720 | 133 693 440 |
| **Format nickname** | **Luma width** | **Luma height** | **Luma picture size** |  |  |  |  |  |  |  |
| **SQCIF** | **128** | **96** | 16 384 | 33.7 | 225.0 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **QCIF** | **176** | **144** | 36 864 | 15.0 | 100.0 | 200.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **QVGA** | **320** | **240** | 81 920 | - | 45.0 | 90.0 | 202.5 | 300.0 | 300.0 | 300.0 |
| **525 SIF** | **352** | **240** | 98 304 | - | 37.5 | 75.0 | 168.7 | 300.0 | 300.0 | 300.0 |
| **CIF** | **352** | **288** | 122 880 | - | 30.0 | 60.0 | 135.0 | 270.0 | 300.0 | 300.0 |
| **525 HHR** | **352** | **480** | 196 608 | - | - | 37.5 | 84.3 | 168.7 | 300.0 | 300.0 |
| **625 HHR** | **352** | **576** | 221 184 | - | - | 33.3 | 75.0 | 150.0 | 300.0 | 300.0 |
| **Q720p** | **640** | **360** | 245 760 | - | - | 30.0 | 67.5 | 135.0 | 272.0 | 300.0 |
| **VGA** | **640** | **480** | 327 680 | - | - | - | 50.6 | 101.2 | 204.0 | 300.0 |
| **525 4SIF** | **704** | **480** | 360 448 | - | - | - | 46.0 | 92.0 | 185.4 | 300.0 |
| **525 SD** | **720** | **480** | 393 216 | - | - | - | 42.1 | 84.3 | 170.0 | 300.0 |
| **4CIF** | **704** | **576** | 405 504 | - | - | - | 40.9 | 81.8 | 164.8 | 300.0 |
| **625 SD** | **720** | **576** | 442 368 | - | - | - | 37.5 | 75.0 | 151.1 | 300.0 |
| **480p (16:9)** | **864** | **480** | 458 752 | - | - | - | 36.1 | 72.3 | 145.7 | 291.4 |
| **SVGA** | **800** | **600** | 532 480 | - | - | - | 31.1 | 62.3 | 125.5 | 251.0 |
| **QHD** | **960** | **540** | 552 960 | - | - | - | 30.0 | 60.0 | 120.8 | 241.7 |
| **XGA** | **1 024** | **768** | 786 432 | - | - | - | - | 42.1 | 85.0 | 170.0 |
| **720p HD** | **1 280** | **720** | 983 040 | - | - | - | - | 33.7 | 68.0 | 136.0 |
| **4VGA** | **1 280** | **960** | 1 228 800 | - | - | - | - | - | 54.4 | 108.8 |
| **SXGA** | **1 280** | **1 024** | 1 310 720 | - | - | - | - | - | 51.0 | 102.0 |
| **525 16SIF** | **1 408** | **960** | 1 351 680 | - | - | - | - | - | 49.4 | 98.9 |
| **16CIF** | **1 408** | **1 152** | 1 622 016 | - | - | - | - | - | 41.2 | 82.4 |
| **4SVGA** | **1 600** | **1 200** | 1 945 600 | - | - | - | - | - | 34.3 | 68.7 |
| **1080 HD** | **1 920** | **1 080** | 2 088 960 | - | - | - | - | - | 32.0 | 64.0 |
| **2K×1K** | **2 048** | **1 024** | 2 097 152 | - | - | - | - | - | 31.8 | 63.7 |
| **2K×1080** | **2 048** | **1 080** | 2 228 224 | - | - | - | - | - | 30.0 | 60.0 |
| **4XGA** | **2 048** | **1 536** | 3 145 728 | - | - | - | - | - | - | - |
| **16VGA** | **2 560** | **1 920** | 4 915 200 | - | - | - | - | - | - | - |
| **3616×1536 (2.35:1)** | **3 616** | **1 536** | 5 603 328 | - | - | - | - | - | - | - |
| **3672×1536 (2.39:1)** | **3 680** | **1 536** | 5 701 632 | - | - | - | - | - | - | - |
| **3840×2160 (4\*HD)** | **3 840** | **2 160** | 8 355 840 | - | - | - | - | - | - | - |
| **4K×2K** | **4 096** | **2 048** | 8 388 608 | - | - | - | - | - | - | - |
| **4096×2160** | **4 096** | **2 160** | 8 912 896 | - | - | - | - | - | - | - |
| **4096×2304 (16:9)** | **4 096** | **2 304** | 9 437 184 | - | - | - | - | - | - | - |
| **7680×4320** | **7 680** | **4 320** | 33 423 360 | - | - | - | - | - | - | - |
| **8192×4096** | **8 192** | **4 096** | 33 554 432 | - | - | - | - | - | - | - |
| **8192×4320** | **8 192** | **4 320** | 35 651 584 | - | - | - | - | - | - | - |

Table A.5 – Maximum picture rates (pictures per second) at level 5 to 6.2 for some example picture sizes  
when MinCbSizeY is equal to 64

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Level:** |  |  |  | **5** | **5.1** | **5.2** | **6** | **6.1** | **6.2** |
| **Max luma picture size (samples):** |  |  |  | 8 912 896 | 8 912 896 | 8 912 896 | 35 651 584 | 35 651 584 | 35 651 584 |
| **Max luma sample rate (samples/sec)** |  |  |  | 267 386 880 | 534 773 760 | 1 069 547 520 | 1 069 547 520 | 2 139 095 040 | 4 278 190 080 |
| **Format nickname** | **Luma width** | **Luma height** | **Luma picture size** |  |  |  |  |  |  |
| **SQCIF** | **128** | **96** | 16 384 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **QCIF** | **176** | **144** | 36 864 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **QVGA** | **320** | **240** | 81 920 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **525 SIF** | **352** | **240** | 98 304 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **CIF** | **352** | **288** | 122 880 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **525 HHR** | **352** | **480** | 196 608 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **625 HHR** | **352** | **576** | 221 184 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **Q720p** | **640** | **360** | 245 760 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **VGA** | **640** | **480** | 327 680 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **525 4SIF** | **704** | **480** | 360 448 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **525 SD** | **720** | **480** | 393 216 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **4CIF** | **704** | **576** | 405 504 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **625 SD** | **720** | **576** | 442 368 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **480p (16:9)** | **864** | **480** | 458 752 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **SVGA** | **800** | **600** | 532 480 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **QHD** | **960** | **540** | 552 960 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **XGA** | **1 024** | **768** | 786 432 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **720p HD** | **1 280** | **720** | 983 040 | 272.0 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **4VGA** | **1 280** | **960** | 1 228 800 | 217.6 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **SXGA** | **1 280** | **1 024** | 1 310 720 | 204.0 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **525 16SIF** | **1 408** | **960** | 1 351 680 | 197.8 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **16CIF** | **1 408** | **1 152** | 1 622 016 | 164.8 | 300.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **4SVGA** | **1 600** | **1 200** | 1 945 600 | 137.4 | 274.8 | 300.0 | 300.0 | 300.0 | 300.0 |
| **1080 HD** | **1 920** | **1 080** | 2 088 960 | 128.0 | 256.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **2K×1K** | **2 048** | **1 024** | 2 097 152 | 127.5 | 255.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **2K×1080** | **2 048** | **1 080** | 2 228 224 | 120.0 | 240.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **4XGA** | **2 048** | **1 536** | 3 145 728 | 85.0 | 170.0 | 300.0 | 300.0 | 300.0 | 300.0 |
| **16VGA** | **2 560** | **1 920** | 4 915 200 | 54.4 | 108.8 | 217.6 | 217.6 | 300.0 | 300.0 |
| **3616×1536 (2.35:1)** | **3 616** | **1 536** | 5 603 328 | 47.7 | 95.4 | 190.8 | 190.8 | 300.0 | 300.0 |
| **3672×1536 (2.39:1)** | **3 680** | **1 536** | 5 701 632 | 46.8 | 93.7 | 187.5 | 187.5 | 300.0 | 300.0 |
| **3840×2160 (4\*HD)** | **3 840** | **2 160** | 8 355 840 | 32.0 | 64.0 | 128.0 | 128.0 | 256.0 | 300.0 |
| **4K×2K** | **4 096** | **2 048** | 8 388 608 | 31.8 | 63.7 | 127.5 | 127.5 | 255.0 | 300.0 |
| **4096×2160** | **4 096** | **2 160** | 8 912 896 | 30.0 | 60.0 | 120.0 | 120.0 | 240.0 | 300.0 |
| **4096×2304 (16:9)** | **4 096** | **2 304** | 9 437 184 | - | - | - | 113.3 | 226.6 | 300.0 |
| **4096×3072** | **4 096** | **3 072** | 12 582 912 | - | - | - | 85.0 | 170.0 | 300.0 |
| **7680×4320** | **7 680** | **4 320** | 33 423 360 | - | - | - | 32.0 | 64.0 | 128.0 |
| **8192×4096** | **8 192** | **4 096** | 33 554 432 | - | - | - | 31.8 | 63.7 | 127.5 |
| **8192×4320** | **8 192** | **4 320** | 35 651 584 | - | - | - | 30.0 | 60.0 | 120.0 |

The following should be noted in regard to the examples shown in Table A.4 and Table A.5:

– This is a variable-picture-size Specification. The specific listed picture sizes are illustrative examples only.

– The example luma picture sizes were computed by rounding up the luma width and luma height to multiples of 64 before computing the product of these quantities, to reflect the potential use of MinCbSizeY equal to 64 for these picture sizes, as pps\_pic\_width\_in\_luma\_samples and pps\_pic\_height\_in\_luma\_samples are each required to be a multiple of MinCbSizeY. For some illustrated values of luma width and luma height, a somewhat higher number of pictures per second can be supported when MinCbSizeY is less than 64.

– In cases where the maximum picture rate value is not an integer multiple of 0.1 pictures per second, the given maximum picture rate values have been rounded down to the largest integer multiple of 0.1 frames per second that does not exceed the exact value. For example, for level 3.1, the maximum picture rate for 720p HD has been rounded down to 33.7 from an exact value of 33.75.

– As used in the examples, "525" refers to typical use for environments using 525 analogue scan lines (of which approximately 480 lines contain the visible picture region) and "625" refers to environments using 625 analogue scan lines (of which approximately 576 lines contain the visible picture region).

– XGA is also known as (aka) XVGA, 4SVGA aka UXGA, 16XGA aka 4Kx3K, CIF aka 625 SIF, 625 HHR aka 2CIF aka half 625 D-1, aka half 625 ITU-R BT.601, 525 SD aka 525 D-1 aka 525 ITU-R BT.601, 625 SD aka 625 D-1 aka 625 ITU-R BT.601.

1. Annex B  
     
   Byte stream format

(This annex forms an integral part of this Recommendation | International Standard.)

* 1. General

This annex specifies syntax and semantics of a byte stream format specified for use by applications that deliver some or all of the NAL unit stream as an ordered stream of bytes or bits within which the locations of NAL unit boundaries need to be identifiable from patterns in the data, such as Rec. ITU-T H.222.0 | ISO/IEC 13818-1 systems or Rec. ITU‑T H.320 systems. For bit-oriented delivery, the bit order for the byte stream format is specified to start with the MSB of the first byte, proceed to the LSB of the first byte, followed by the MSB of the second byte, etc.

The byte stream format consists of a sequence of byte stream NAL unit syntax structures. Each byte stream NAL unit syntax structure contains one start code prefix followed by one nal\_unit( NumBytesInNalUnit ) syntax structure. It may (and under some circumstances, it shall) also contain an additional zero\_byte syntax element. It may also contain one or more additional trailing\_zero\_8bits syntax elements. When it is the first byte stream NAL unit in the bitstream, it may also contain one or more additional leading\_zero\_8bits syntax elements.

* 1. **Byte stream NAL unit syntax and semantics**
     1. **Byte stream NAL unit syntax**

|  |  |
| --- | --- |
| byte\_stream\_nal\_unit( NumBytesInNalUnit ) { | **Descriptor** |
| while( next\_bits( 24 ) != 0x000001 && next\_bits( 32 ) != 0x00000001 ) |  |
| **leading\_zero\_8bits** /\* equal to 0x00 \*/ | f(8) |
| if( next\_bits( 24 ) != 0x000001 ) |  |
| **zero\_byte** /\* equal to 0x00 \*/ | f(8) |
| **start\_code\_prefix\_one\_3bytes** /\* equal to 0x000001 \*/ | f(24) |
| nal\_unit( NumBytesInNalUnit ) |  |
| while( more\_data\_in\_byte\_stream( ) && next\_bits( 24 ) != 0x000001 &&  next\_bits( 32 ) != 0x00000001 ) |  |
| **trailing\_zero\_8bits** /\* equal to 0x00 \*/ | f(8) |
| } |  |

* + 1. **Byte stream NAL unit semantics**

The order of byte stream NAL units in the byte stream shall follow the decoding order of the NAL units contained in the byte stream NAL units (see clause 7.4.2.4). The content of each byte stream NAL unit is associated with the same AU as the NAL unit contained in the byte stream NAL unit (see clause 7.4.2.4.4).

**leading\_zero\_8bits** is a byte equal to 0x00.

NOTE – The leading\_zero\_8bits syntax element can only be present in the first byte stream NAL unit of the bitstream, because (as shown in the syntax diagram of clause B.2.1) any bytes equal to 0x00 that follow a NAL unit syntax structure and precede the four-byte sequence 0x00000001 (which is to be interpreted as a zero\_byte followed by a start\_code\_prefix\_one\_3bytes) will be considered to be trailing\_zero\_8bits syntax elements that are part of the preceding byte stream NAL unit.

**zero\_byte** is a single byte equal to 0x00.

When one or more of the following conditions are true, the zero\_byte syntax element shall be present:

– The nal\_unit\_type within the nal\_unit( ) syntax structure is equal to DCI\_NUT, VPS\_NUT, SPS\_NUT, PPS\_NUT, PREFIX\_APS\_NUT, or SUFFIX\_APS\_NUT.

– The byte stream NAL unit syntax structure contains the first NAL unit of an AU in decoding order, as specified in clause 7.4.2.4.4.

**start\_code\_prefix\_one\_3bytes** is a fixed-value sequence of 3 bytes equal to 0x000001. This syntax element is called a start code prefix.

**trailing\_zero\_8bits** is a byte equal to 0x00.

* 1. **Byte stream NAL unit decoding process**

Input to this process consists of an ordered stream of bytes consisting of a sequence of byte stream NAL unit syntax structures.

Output of this process consists of a sequence of NAL unit syntax structures.

At the beginning of the decoding process, the decoder initializes its current position in the byte stream to the beginning of the byte stream. It then extracts and discards each leading\_zero\_8bits syntax element (when present), moving the current position in the byte stream forward one byte at a time, until the current position in the byte stream is such that the next four bytes in the bitstream form the four-byte sequence 0x00000001.

The decoder then performs the following step-wise process repeatedly to extract and decode each NAL unit syntax structure in the byte stream until the end of the byte stream has been encountered (as determined by unspecified means) and the last NAL unit in the byte stream has been decoded:

1. When the next four bytes in the bitstream form the four-byte sequence 0x00000001, the next byte in the byte stream (which is a zero\_byte syntax element) is extracted and discarded and the current position in the byte stream is set equal to the position of the byte following this discarded byte.
2. The next three-byte sequence in the byte stream (which is a start\_code\_prefix\_one\_3bytes) is extracted and discarded and the current position in the byte stream is set equal to the position of the byte following this three‑byte sequence.
3. NumBytesInNalUnit is set equal to the number of bytes starting with the byte at the current position in the byte stream up to and including the last byte that precedes the location of one or more of the following conditions:

– A subsequent byte-aligned three-byte sequence equal to 0x000000,

– A subsequent byte-aligned three-byte sequence equal to 0x000001,

– The end of the byte stream, as determined by unspecified means.

1. NumBytesInNalUnit bytes are removed from the bitstream and the current position in the byte stream is advanced by NumBytesInNalUnit bytes. This sequence of bytes is nal\_unit( NumBytesInNalUnit ) and is decoded using the NAL unit decoding process.
2. When the current position in the byte stream is not at the end of the byte stream (as determined by unspecified means) and the next bytes in the byte stream do not start with a three-byte sequence equal to 0x000001 and the next bytes in the byte stream do not start with a four byte sequence equal to 0x00000001, the decoder extracts and discards each trailing\_zero\_8bits syntax element, moving the current position in the byte stream forward one byte at a time, until the current position in the byte stream is such that the next bytes in the byte stream form the four-byte sequence 0x00000001 or the end of the byte stream has been encountered (as determined by unspecified means).
   1. **Decoder byte-alignment recovery (informative)**

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

Many applications provide data to a decoder in a manner that is inherently byte aligned, and thus have no need for the bit-oriented byte alignment detection procedure described in this clause.

A decoder is said to have byte alignment with a bitstream when the decoder has determined whether or not the positions of data in the bitstream are byte-aligned. When a decoder does not have byte alignment with the bitstream, the decoder may examine the incoming bitstream for the binary pattern '00000000 00000000 00000000 00000001' (31 consecutive bits equal to 0 followed by a bit equal to 1). The bit immediately following this pattern is the first bit of an aligned byte following a start code prefix. Upon detecting this pattern, the decoder will be byte-aligned with the bitstream and positioned at the start of a NAL unit in the bitstream.

Once byte aligned with the bitstream, the decoder can examine the incoming bitstream data for subsequent three-byte sequences 0x000001 and 0x000003.

When the three-byte sequence 0x000001 is detected, this is a start code prefix.

When the three-byte sequence 0x000003 is detected, the third byte (0x03) is an emulation\_prevention\_three\_byte to be discarded as specified in clause 7.4.2.

When an error in the bitstream syntax is detected (e.g., a non-zero value of the forbidden\_zero\_bit or one of the three‑byte or four-byte sequences that are prohibited in clause 7.4.2), the decoder may consider the detected condition as an indication that byte alignment may have been lost and may discard all bitstream data until the detection of byte alignment at a later position in the bitstream as described above in this clause.

1. Annex C  
     
   Hypothetical reference decoder

(This annex forms an integral part of this Recommendation | International Standard.)

* 1. General

This annex specifies the hypothetical reference decoder (HRD) and its use to check bitstream and decoder conformance.

Two types of bitstreams or bitstream subsets are subject to HRD conformance checking for this Specification. The first type, called a Type I bitstream, is a NAL unit stream containing only the VCL NAL units and NAL units with nal\_unit\_type equal to FD\_NUT (filler data NAL units) for all AUs in the bitstream. The second type, called a Type II bitstream, contains, in addition to the VCL NAL units and filler data NAL units for all AUs in the bitstream, at least one of the following:

– additional non-VCL NAL units other than filler data NAL units,

– all leading\_zero\_8bits, zero\_byte, start\_code\_prefix\_one\_3bytes and trailing\_zero\_8bits syntax elements that form a byte stream from the NAL unit stream (as specified in Annex B).

NOTE 1 – Decoders conforming to profiles specified in Annex A do not use NAL units with nuh\_layer\_id greater than vps\_layer\_id[ 0 ] (e.g., AU delimiter NAL units with nuh\_layer\_id greater than vps\_layer\_id[ 0 ]) for AU boundary detection, except for identification of whether a NAL unit is a VCL or non-VCL NAL unit. Consequently, hypothetical reference decoder (HRD) parameters carried in BP, PT and DU information SEI messages apply to AUs that are identified based on such AU boundary detection.

Figure C.1 shows the types of bitstream conformance points checked by the HRD.



Figure C.1 – Structure of byte streams and NAL unit streams for HRD conformance checks

The syntax elements of non-VCL NAL units (or their default values for some of the syntax elements), required for the HRD, are specified in the semantic clauses of clause 7 and Annex D. When the VUI parameters or any SEI message specified in ITU-T H.SEI | ISO/IEC 23002-7 is included in a non-VCL NAL unit specified in this Specification, those syntax elements are specified in the semantics clauses of ITU-T H.SEI | ISO/IEC 23002-7.

Two sets of HRD parameters (NAL HRD parameters and VCL HRD parameters) are used. The HRD parameters are signalled through the general\_hrd\_parameters( ) syntax structure and the ols\_hrd\_parameters( ) syntax structure, which are either part of the VPS or part of the SPS.

A set of bitstream conformance tests is needed for checking the conformance of a bitstream, which is referred to as the entire bitstream, denoted as entireBitstream. The set of bitstream conformance tests are for testing the conformance of each OP of each OLS specified by the VPS, and also for testing the conformance of each subpicture sequence specified by the subpicture level information SEI message. [Ed. (YK): Update the HRD text, particularly the part on describing the conformance tests, such that it does not appear that VPS would be needed even when the bitstream only has a single layer.]

For each test, the following ordered steps apply in the order listed, followed by the processes described after these steps in this clause:

1. An operation point under test, denoted as targetOp, is selected by selecting a target OLS with OLS index opOlsIdx, a highest TemporalId value opTid, and optionally, a list of target subpicture index values opSubpicIdxList[ ], one for each layer. The value of opOlsIdx is in the range of 0 to TotalNumOlss − 1, inclusive. The value of opTid is in the range of 0 to vps\_max\_sublayers\_minus1, inclusive.

If opSubpicIdxList[ ] is not present, targetOp consists of pictures, and each pair of selected values of opOlsIdx and opTid shall be such that the sub-bitstream BitstreamToDecode that is the output by invoking the sub-bitstream extraction process as specified in clause C.6 with entireBitstream, opOlsIdx, and opTid as inputs satisify the following condition:

– There is at least one VCL NAL unit with TemporalId equal to opTid in BitstreamToDecode.

Otherwise (opSubpicIdxList[ ] is present), targetOp consists of subpictures, and the sub-bitstream that is the output by invoking the subpicture sub-bitstream extraction process as specified in clause C.7 with entireBitstream, opOlsIdx, opTid, and opSubpicIdxList[ ] as inputs satisify the following conditions:

– There is at least one VCL NAL unit with TemporalId equal to opTid in BitstreamToDecode.

– There is at least one VCL NAL unit with nuh\_layer\_id equal to LayerIdInOls[ opOlsIdx ][ j ] and with sh\_subpic\_id equal to SubpicIdVal[ opSubpicIdxList[ j ] ] for each j in the range of 0 to NumLayersInOls[ opOlsIdx ] − 1.

NOTE 2 – Regardless of whether opSubpicIdxList[ ] is present, due to that each IRAP or GDR AU needs to be complete, there is at least one VCL NAL unit with nuh\_layer\_id equal to each of the nuh\_layer\_id values in LayerIdInOls[ opOlsIdx ] in BitstreamToDecode.

1. If opSubpicIdxList[ ] is not present the following applies:

– If the layers in targetOp include all layers in entireBitstream and opTid is equal to the highest TemporalId value among all NAL units in entireBitstream, BitstreamToDecode is set to be identical to entireBitstream.

– Otherwise, BitstreamToDecode is set to be the output by invoking the sub-bitstream extraction process as specified in clause C.6 with entireBitstream, opOlsIdx, and opTid as inputs.

Otherwise (opSubpicIdxList[ ] is present), BitstreamToDecode is set to be the output by invoking the subpicture sub-bitstream extraction process as specified in clause C.7 with entireBitstream, opOlsIdx, opTid and opSubpicIdxList[ ] as inputs.

1. The values of TargetOlsIdx and Htid are set equal to opOlsIdx and opTid, respectively, of targetOp.
2. The general\_hrd\_parameters( ) syntax structure, the ols\_hrd\_parameters( ) syntax structure, and the sublayer\_hrd\_parameters( ) syntax structure applicable to BitstreamToDecode are selected as follows:

– If NumLayersInOls[ TargetOlsIdx ] is equal to 1, the general\_hrd\_parameters( ) syntax structure and the ols\_hrd\_parameters( ) syntax structure in the SPS referenced by the layer in BitstreamToDecode (or provided through an external means not specified in this Specification) are selected. Otherwise, the general\_hrd\_parameters( ) syntax structure and the vps\_ols\_hrd\_idx[ MultiLayerOlsIdx[ TargetOlsIdx ] ]-th ols\_hrd\_parameters( ) syntax structure in the VPS (or provided through an external means not specified in this Specification) are selected.

– Within the selected ols\_hrd\_parameters( ) syntax structure, for testing of the Type I bitstream conformance piont, the sublayer\_hrd\_parameters( Htid ) syntax structure that immediately follows the condition "if( general\_vcl\_hrd\_params\_present\_flag )" is selected and the variable NalHrdModeFlag is set equal to 0, and for testing of the Type II bitstream conformance piont, the sublayer\_hrd\_parameters( Htid ) syntax structure that immediately follows the condition "if( general\_nal\_hrd\_params\_present\_flag )" is selected and the variable NalHrdModeFlag is set equal to 1. When BitstreamToDecode is a Type II bitstream and NalHrdModeFlag is equal to 0, all non-VCL NAL units except filler data NAL units, and all leading\_zero\_8bits, zero\_byte, start\_code\_prefix\_one\_3bytes and trailing\_zero\_8bits syntax elements that form a byte stream from the NAL unit stream (as specified in Annex B), when present, are discarded from BitstreamToDecode and the remaining bitstream is assigned to BitstreamToDecode.

1. An AU associated with a BP SEI message (present in BitstreamToDecode or available through external means not specified in this Specification) applicable to TargetOp is selected as the HRD initialization point and referred to as AU 0.
2. When general\_decoding\_unit\_hrd\_params\_present\_flag in the selected general\_hrd\_parameters( ) syntax structure is equal to 1, the CPB is scheduled to operate either at the AU level (in which case the variable DecodingUnitHrdFlag is set equal to 0) or at the DU level (in which case the variable DecodingUnitHrdFlag is set equal to 1). Otherwise, DecodingUnitHrdFlag is set equal to 0 and the CPB is scheduled to operate at the AU level.
3. For each AU in BitstreamToDecode starting from AU 0, the BP SEI message (present in BitstreamToDecode or available through external means not specified in this Specification) that is associated with the AU and applies to TargetOlsIdx is selected, and the PT SEI message (present in BitstreamToDecode or available through external means not specified in this Specification) that is associated with the AU and applies to TargetOlsIdx is selected, and when DecodingUnitHrdFlag is equal to 1 and du\_cpb\_params\_in\_pic\_timing\_sei\_flag is equal to 0, the DU information SEI messages (present in BitstreamToDecode or available through external means not specified in this Specification) that are associated with DUs in the AU and apply to TargetOlsIdx are selected.
4. A value of ScIdx is selected. The selected ScIdx shall be in the range of 0 to hrd\_cpb\_cnt\_minus1, inclusive.
5. When the BP SEI message associated with AU 0 has cpb\_alt\_timing\_info\_present\_flag equal to 1, either of the following applies for selection of the initial CPB removal delay and delay offset:

– If NalHrdModeFlag is equal to 1, the default initial CPB removal delay and delay offset represented by nal\_initial\_cpb\_removal\_delay[ Htid ][ ScIdx ] and nal\_initial\_cpb\_removal\_offset[ Htid ][ ScIdx ], respectively, in the selected BP SEI message are selected. Otherwise, the default initial CPB removal delay and delay offset represented by vcl\_initial\_cpb\_removal\_delay[ Htid ][ ScIdx ] and vcl\_initial\_cpb\_removal\_offset[ Htid ][ ScIdx ], respectively, in the selected BP SEI message are selected. The variable DefaultInitCpbParamsFlag is set equal to 1.

– If NalHrdModeFlag is equal to 1, the alternative initial CPB removal delay and delay offset represented by nal\_initial\_cpb\_removal\_delay[ Htid ][ ScIdx ] and nal\_initial\_cpb\_removal\_offset[ Htid ][ ScIdx ], respectively, in the selected BP SEI message and nal\_cpb\_alt\_initial\_removal\_delay\_delta[ Htid ][ ScIdx ] and nal\_cpb\_alt\_initial\_removal\_offset\_delta[ Htid ][ ScIdx ], respectively, in the PT SEI message associated with the AU following AU 0 in decoding order are selected. Otherwise, the alternative initial CPB removal delay and delay offset represented by vcl\_initial\_cpb\_removal\_delay[ Htid ][ ScIdx ] and vcl\_initial\_cpb\_removal\_offset[ Htid ][ ScIdx ], respectively, in the selected BP SEI message and vcl\_cpb\_alt\_initial\_removal\_delay\_delta[ Htid ][ ScIdx ] and vcl\_cpb\_alt\_initial\_removal\_offset\_delta[ Htid ][ ScIdx ], respectively, in the PT SEI message associated with the AU following AU 0 in decoding order are selected. The variable DefaultInitCpbParamsFlag is set equal to 0, and one of the following applies:

– The RASL AUs that contain RASL pictures with pps\_mixed\_nalu\_types\_in\_pic\_flag equal to 0 and are associated with CRA pictures contained in AU 0 are discarded from BitstreamToDecode and the remaining bitstream is assigned to BitstreamToDecode.

– All AUs following AU 0 in decoding order up to an AU associated with a DRAP indication SEI message are discarded from BitstreamToDecode and the remaining bitstream is assigned to BitstreamToDecode.

Each conformance test consists of a combination of one option in each of the above steps. When there is more than one option for a step, for any particular conformance test only one option is chosen. All possible combinations of all the steps form the entire set of conformance tests. For each operation point under test, the number of bitstream conformance tests to be performed is equal to n0 \* n1 \* ( 2 \* n2 + n3 + n4 + n5), where the values of n0, n1, n2, n3, n4, and n5 are specified as follows:

– n0 is set equal to 2.

– n1 is equal to hrd\_cpb\_cnt\_minus1 + 1.

– n2 is the number of AUs in BitstreamToDecode that each is associated with a BP SEI message applicable to TargetOlsIdx and for which all of the following conditions are true:

– nal\_unit\_type is equal to CRA\_NUT.

– The associated BP SEI message has bp\_alt\_cpb\_params\_present\_flag equal to 1.

– There is at least one RASL picture with pps\_mixed\_nalu\_types\_in\_pic\_flag equal to 0 associated with the AU.

– n3 is the number of IRAP or GDR AUs in BitstreamToDecode that each is associated with a BP SEI message applicable to TargetOlsIdx and for which at least one the following conditions is false:

– nal\_unit\_type is equal to CRA\_NUT.

– The associated BP SEI message has bp\_alt\_cpb\_params\_present\_flag equal to 1.

– There is at least one RASL picture with pps\_mixed\_nalu\_types\_in\_pic\_flag equal to 0 associated with the AU.

– n4 is the number of AUs in BitstreamToDecode that each is associated with a DRAP indication SEI message applicable to TargetOlsIdx and for each of which the associated PT SEI message has cpb\_alt\_timing\_info\_present\_flag equal to 1.

– n5 is derived as follows:

– If general\_decoding\_unit\_hrd\_params\_present\_flag in the selected general\_hrd\_parameters( ) syntax structure is equal to 0, n5 is equal to 1.

– Otherwise, n5 is equal to 2.

NOTE 3 – n0 corresponds to conformance tests for Type I bitstream conformance and Type II bitstream conformance. n1 corresponds to conformance tests for each CPB delivery schedule. n2 corresponds to conformance tests for bitstreams starting at each CRA pictures with associated RASL pictures and alternative initial CPB removal delay and delay offset present. These test are carried twice: once for bitstream keeping and once for bitstreams removing RASL pictures associated with the CRA. n3 corresponds to conformance tests for bitstreams starting at each IRAP or GDR AU that is not a CRA with associated RASL pictures and alternative initial CPB removal delay and delay offset present. n4 corresponds to conformance tests for bitstreams starting with each IRAP with associated DRAP pictures with alternative timing information present and result of removing all access units between a DRAP picture with alternative timing information present and the preceding IRAP. n5 corresponds to conformance tests for AU based conformance and DU-based when general\_decoding\_unit\_hrd\_params\_present\_flag is equal to 1.

When BitstreamToDecode is a Type II bitstream, the following applies:

– If the sublayer\_hrd\_parameters( Htid ) syntax structure that immediately follows the condition "if( general\_vcl\_hrd\_params\_present\_flag )" is selected, the test is conducted at the Type I conformance point shown in Figure C.1, and only VCL and filler data NAL units are counted for the input bit rate and CPB storage.

– Otherwise (the sublayer\_hrd\_parameters( Htid ) syntax structure that immediately follows the condition "if( general\_nal\_hrd\_params\_present\_flag )" is selected), the test is conducted at the Type II conformance point shown in Figure C.1, and all bytes of the Type II bitstream, which may be a NAL unit stream or a byte stream, are counted for the input bit rate and CPB storage.

NOTE 4 – NAL HRD parameters established by a value of ScIdx for the Type II conformance point shown in Figure C.1 are sufficient to also establish VCL HRD conformance for the Type I conformance point shown in Figure C.1 for the same values of InitCpbRemovalDelay[ Htid ][ ScIdx ], BitRate[ Htid ][ ScIdx ] and CpbSize[ Htid ][ ScIdx ] for the variable bit rate (VBR) case (cbr\_flag[ Htid ][ ScIdx ] equal to 0). This is because the data flow into the Type I conformance point is a subset of the data flow into the Type II conformance point and because, for the VBR case, the CPB is allowed to become empty and stay empty until the time a next picture is scheduled to begin to arrive.

All DCI NAL units, when available, all VPSs, SPSs, PPSs, and APSs referred to in the VCL NAL units, and appropriate BP, PT, and DU information SEI messages shall be conveyed to the HRD, in a timely manner, either in the bitstream (by non-VCL NAL units), or by other means not specified in this Specification.

In Annexes C and D, the specification for "presence" of non-VCL NAL units that contain DCI NAL units, VPSs, SPSs, PPSs, APSs, BP SEI messages, PT SEI messages, or DU information SEI messages, is also satisfied when those NAL units (or just some of them) are conveyed to decoders (or to the HRD) by other means not specified in this Specification. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

NOTE 5 – As an example, synchronization of such a non-VCL NAL unit, conveyed by means other than presence in the bitstream, with the NAL units that are present in the bitstream, can be achieved by indicating two points in the bitstream, between which the non‑VCL NAL unit would have been present in the bitstream, had the encoder decided to convey it in the bitstream.

When the content of such a non-VCL NAL unit is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the non-VCL NAL unit is not required to use the same syntax as specified in this Specification.

NOTE 6 – When HRD information is contained within the bitstream, it is possible to verify the conformance of a bitstream to the requirements of this clause based solely on information contained in the bitstream. When the HRD information is not present in the bitstream, as is the case for all "stand-alone" Type I bitstreams, conformance can only be verified when the HRD data are supplied by some other means not specified in this Specification.

The HRD contains a bitstream extractor (optionally present), a CPB, an instantaneous decoding process, a DPB, and output cropping as shown in Figure C.2.



Figure C.2 – HRD buffer model

For each bitstream conformance test, the CPB size (number of bits) is CpbSize[ Htid ][ ScIdx ] as specified in clause 7.4.6.3, where ScIdx and the HRD parameters are specified above in this clause, and the DPB parameters max\_dec\_pic\_buffering\_minus1[ Htid ], max\_num\_reorder\_pics[ Htid ], and MaxLatencyPictures[ Htid ] are found in or derived from the dpb\_parameters( ) syntax structure that applies to the target OLS as follows:

– If NumLayersInOls[ TargetOlsIdx ] is equal to 1, the dpb\_parameters( ) syntax structure is found in the SPS referred to be the layer in the target OLS, and the variables PicWidthMaxInSamplesY, PicHeightMaxInSamplesY, MaxChromaFormat, and MaxBitDepthMinus8 are set equal to sps\_pic\_width\_max\_in\_luma\_samples, sps\_pic\_height\_max\_in\_luma\_samples, sps\_chroma\_format\_idc, and sps\_bit\_depth\_minus8, respectively, found in the SPS referred to by the layer in the target OLS.

– Otherwise (NumLayersInOls[ TargetOlsIdx ] is greater than 1), the dpb\_parameters( ) is identified by vps\_ols\_dpb\_params\_idx[ MultiLayerOlsIdx[ TargetOlsIdx ] ] found in the VPS, and the variables PicWidthMaxInSamplesY, PicHeightMaxInSamplesY, MaxChromaFormat, and MaxBitDepthMinus8 are set equal to vps\_ols\_dpb\_pic\_width[ MultiLayerOlsIdx[ TargetOlsIdx ] ], vps\_ols\_dpb\_pic\_height[ MultiLayerOlsIdx[ TargetOlsIdx ] ], vps\_ols\_dpb\_chroma\_format[ MultiLayerOlsIdx[ TargetOlsIdx ] ], and vps\_ols\_dpb\_bitdepth\_minus8[ MultiLayerOlsIdx[ TargetOlsIdx ] ], respectively, found in the VPS.

If DecodingUnitHrdFlag is equal to 0, the HRD operates at AU level and each DU is an AU. Otherwise the HRD operates at DU level and each DU is a subset of an AU.

NOTE 7 – If the HRD operates at AU level, each time when some bits are removed from the CPB, a DU that is an entire AU is removed from the CPB. Otherwise (the HRD operates at DU level), each time when some bits are removed from the CPB, a DU that is a subset of an AU is removed from the CPB. Regardless of whether the HRD operates at access unt level or DU level, each time when some picture is output from the DPB, an entire decoded picture is output from the DPB, though the picture output time is derived based on the differently derived CPB removal times and the differently signalled DPB output delays.

The following is specified for expressing the constraints in this annex:

– Each AU is referred to as AU n, where the number n identifies the particular AU. AU 0 is selected per step 5 above. The value of n is incremented by 1 for each subsequent AU in decoding order.

– Each DU is referred to as DU m, where the number m identifies the particular DU. The first DU in decoding order in AU 0 is referred to as DU 0. The value of m is incremented by 1 for each subsequent DU in decoding order.

NOTE 8 – The numbering of DUs is relative to the first DU in AU 0.

– Picture n refers to the coded picture or the decoded picture of AU n.

The HRD operates as follows:

– The HRD is initialized at DU 0, with both the CPB and the DPB being set to be empty (the DPB fullness is set equal to 0).

NOTE 9 – After initialization, the HRD is not initialized again by subsequent BP SEI messages.

– Data associated with DUs that flow into the CPB according to a specified arrival schedule are delivered by the hypothetical stream scheduler (HSS).

– The data associated with each DU are removed and decoded instantaneously by the instantaneous decoding process at the CPB removal time of the DU.

– Each decoded picture is placed in the DPB.

– A decoded picture is removed from the DPB when it becomes no longer needed for inter prediction reference and no longer needed for output.

For each bitstream conformance test, the operation of the CPB is specified in clause C.2, the instantaneous decoder operation is specified in clauses 2 through 9, the operation of the DPB is specified in clause C.3 and the output cropping is specified in clauses C.3.3 and C.5.2.2.

HSS and HRD information concerning the number of enumerated delivery schedules and their associated bit rates and buffer sizes is specified in clauses 7.3.5.1 and 7.4.6.1. The HRD is initialized as specified by the BP SEI message (specified in clause D.3). The removal timing of DUs from the CPB and output timing of decoded pictures from the DPB is specified using information in PT SEI messages (specified in clause D.4) or in DU information SEI messages (specified in clause D.5). All timing information relating to a specific DU shall arrive prior to the CPB removal time of the DU.

The requirements for bitstream conformance are specified in clause C.4 and the HRD is used to check conformance ofbitstreams as specified above in this clause and to check conformance of decoders as specified in clause C.5.

NOTE 10 – While conformance is guaranteed under the assumption that all picture-rates and clocks used to generate the bitstream match exactly the values signalled in the bitstream, in a real system each of these may vary from the signalled or specified value.

All the arithmetic in this annex is performed with real values, so that no rounding errors can propagate. For example, the number of bits in a CPB just prior to or after removal of a DU is not necessarily an integer.

The variable ClockTick is derived as follows and is called a clock tick:

ClockTick = num\_units\_in\_tick ÷ time\_scale (C.1)

The variable ClockSubTick is derived as follows and is called a clock sub-tick:

ClockSubTick = ClockTick ÷ ( tick\_divisor\_minus2 + 2 ) (C.2)

* 1. Operation of the CPB
     1. General

The specifications in this clause apply independently to each set of CPB parameters that is present and to both the Type I and Type II conformance points shown in Figure C.1 and the set of CPB parameters is selected as specified in clause C.1.

* + 1. Timing of DU arrival

If DecodingUnitHrdFlag is equal to 0, the variable decodingUnitParamsFlag is set equal to 0 and the process specified in the remainder of this clause is invoked with a DU being considered as an AU, for derivation of the initial and final CPB arrival times for AU n.

Otherwise (DecodingUnitHrdFlag is equal to 1), the process specified in the remainder of this clause is first invoked with the variable decodingUnitParamsFlag set equal to 0 and a DU being considered as an AU, for derivation of the initial and final CPB arrival times for AU n, and then invoked with decodingUnitParamsFlag set equal to 1 and a DU being considered as a subset of an AU, for derivation of the initial and final CPB arrival times for the DUs in AU n.

The process specified in the remainder of this clause is invoked for derivation of the initial and final CPB arrival times for AU n.

The variables InitCpbRemovalDelay[ Htid ][ ScIdx ] and InitCpbRemovalDelayOffset[ Htid ][ ScIdx ] are derived as follows:

– If one or more of the following conditions are true, InitCpbRemovalDelay[ Htid ][ ScIdx ] and InitCpbRemovalDelayOffset[ Htid ][ ScIdx ] are set equal to the values of the BP SEI message syntax elements nal\_initial\_cpb\_removal\_delay[ Htid ][ ScIdx ] and nal\_initial\_cpb\_removal\_offset[ Htid ][ ScIdx ] minus the values of the PT SEI message syntax elements nal\_cpb\_alt\_initial\_removal\_delay\_delta[ Htid ][ ScIdx ] and nal\_cpb\_alt\_initial\_removal\_offset\_delta[ Htid ][ ScIdx ] of AU 1, respectively, when NalHrdModeFlag is equal to 1, or vcl\_initial\_cpb\_removal\_delay[ Htid ][ ScIdx ] and vcl\_initial\_cpb\_removal\_offset[ Htid ][ ScIdx ] minus the values of the PT SEI message syntax elements vcl\_cpb\_alt\_initial\_removal\_delay\_delta[ Htid ][ ScIdx ] and vcl\_cpb\_alt\_initial\_removal\_offset\_delta[ Htid ][ ScIdx ] of AU 1, respectively, when NalHrdModeFlag is equal to 0, where the BP SEI message containing the syntax elements is selected as specified in clause C.1:

– UseAltCpbParamsFlag for AU 0 is equal to 1.

– DefaultInitCpbParamsFlag is equal to 0.

– Otherwise, if the value of decodingUnitParamsFlag is equal to 1, InitCpbRemovalDelay[ Htid ][ ScIdx ] and InitCpbRemovalDelayOffset[ Htid ][ ScIdx ] are set equal to the values of the BP SEI message syntax elements nal\_initial\_alt\_cpb\_removal\_delay[ Htid ][ ScIdx ] and nal\_initial\_alt\_cpb\_removal\_offset[ Htid ][ ScIdx ], respectively, when NalHrdModeFlag is equal to 1 or vcl\_initial\_alt\_cpb\_removal\_delay[ Htid ][ ScIdx ] and vcl\_initial\_alt\_cpb\_removal\_offset[ Htid ][ ScIdx ], respectively, when NalHrdModeFlag is equal to 0, where the BP SEI message syntax elements are selected as specified in clause C.1.

– Otherwise (DecodingUnitHrdFlag is equal to 0), InitCpbRemovalDelay[ Htid ][ ScIdx ] and InitCpbRemovalDelayOffset[ Htid ][ ScIdx ] are set equal to the values of the BP SEI message syntax elements nal\_initial\_cpb\_removal\_delay[ Htid ][ ScIdx ] and nal\_initial\_cpb\_removal\_offset[ Htid ][ ScIdx ], respectively, when NalHrdModeFlag is equal to 1, or vcl\_initial\_cpb\_removal\_delay[ Htid ][ ScIdx ] and vcl\_initial\_cpb\_removal\_offset[ Htid ][ ScIdx ], respectively, when NalHrdModeFlag is equal to 0, where the BP SEI message syntax elements are selected as specified in clause C.1.

The variables DpbDelayOffset and CpbDelayOffset are derived as follows with k being the AU associated with the BP SEI message:

– If one or more of the following conditions are true, DpbDelayOffset is set equal to the value of the PT SEI message syntax element nal\_dpb\_delay\_offset[ Htid ] (when NalHrdModeFlag is equal to 1) or vcl\_dpb\_delay\_offset[ Htid ] (when NalHrdModeFlag is equal to 0) of AU k + 1, and CpbDelayOffset is set equal to the value of the PT SEI message syntax element nal\_cpb\_delay\_offset[ Htid ] (when NalHrdModeFlag is equal to 1) or vcl\_cpb\_delay\_offset[ Htid ] (when NalHrdModeFlag is equal to 0) of AU k + 1, where the PT SEI message containing the syntax elements is selected as specified in clause C.1:

– UseAltCpbParamsFlag for AU 0 is equal to 1.

– DefaultInitCpbParamsFlag is equal to 0.

– Otherwise, DpbDelayOffset and CpbDelayOffset are set equal to 0.

The time at which the first bit of DU m begins to enter the CPB is referred to as the initial arrival time initArrivalTime[ m ].

The initial arrival time of DU m is derived as follows:

– If the AU is DU 0 (i.e., when m is equal to 0), initArrivalTime[ 0 ] is set equal to 0.

– Otherwise (the DU is DU m with m > 0), the following applies:

– If cbr\_flag[ ScIdx ] is equal to 1, the initial arrival time for DU m is equal to the final arrival time (which is derived below) of DU m − 1, i.e.,

if( !decodingUnitParamsFlag )  
 initArrivalTime[ m ] = AuFinalArrivalTime[ m − 1 ] (C.3)  
else  
 initArrivalTime[ m ] = DuFinalArrivalTime[ m − 1 ]

– Otherwise (cbr\_flag[ ScIdx ] is equal to 0), the initial arrival time for DU m is derived as follows:

if( !decodingUnitParamsFlag )  
 initArrivalTime[ m ] = Max( AuFinalArrivalTime[ m − 1 ], initArrivalEarliestTime[ m ] ) (C.4)  
else  
 initArrivalTime[ m ] = Max( DuFinalArrivalTime[ m − 1 ], initArrivalEarliestTime[ m ] )

where initArrivalEarliestTime[ m ] is derived as follows:

– The variable tmpNominalRemovalTime is derived as follows:

if( !decodingUnitParamsFlag )  
 tmpNominalRemovalTime = AuNominalRemovalTime[ m ] (C.5)   
else  
 tmpNominalRemovalTime = DuNominalRemovalTime[ m ]

where NominalRemovalTime[ m ] and DuNominalRemovalTime[ m ] are the nominal CPB removal time of AU m and DU m, respectively, as specified in clause C.2.3.

– If DU m is not the first DU of a subsequent BP, initArrivalEarliestTime[ m ] is derived as follows:

initArrivalEarliestTime[ m ] = tmpNominalRemovalTime −  
 ( InitCpbRemovalDelay[ Htid ][ ScIdx ]  
 + InitCpbRemovalDelayOffset[ Htid ][ ScIdx ] ) ÷ 90000 (C.6)

– Otherwise (DU m is the first DU of a subsequent BP), initArrivalEarliestTime[ m ] is derived as follows:

initArrivalEarliestTime[ m ] = tmpNominalRemovalTime −  
 ( InitCpbRemovalDelay[ Htid ][ ScIdx ] ÷ 90000 ) (C.7)

The final arrival time for DU m is derived as follows:

if( !decodingUnitParamsFlag )  
 AuFinalArrivalTime[ m ] = initArrivalTime[ m ] + sizeInbits[ m ] ÷ BitRate[ Htid ][ ScIdx ] (C.8)   
else  
 DuFinalArrivalTime[ m ] = initArrivalTime[ m ] + sizeInbits[ m ] ÷ BitRate[ Htid ][ ScIdx ]

where sizeInbits[ m ] is the size in bits of DU m, counting the bits of the VCL NAL units and the filler data NAL units for the Type I conformance point or all bits of the Type II bitstream for the Type II conformance point, where the Type I and Type II conformance points are as shown in Figure C.1.

The values of ScIdx, BitRate[ Htid ][ ScIdx ] and CpbSize[ Htid ][ ScIdx ] are constrained as follows:

– If the content of the selected general\_hrd\_parameters( ) syntax structures for the AU containing AU m and the previous AU differ, the HSS selects a value ScIdx1 of ScIdx from among the values of ScIdx provided in the selected general\_hrd\_parameters( ) syntax structures for the AU containing AU m that results in a BitRate[ Htid ][ ScIdx1 ] or CpbSize[ Htid ][ ScIdx1 ] for the AU containing AU m. The value of BitRate[ Htid ][ ScIdx1 ] or CpbSize[ Htid ][ ScIdx1 ] may differ from the value of BitRate[ Htid ][ ScIdx0 ] or CpbSize[ Htid ][ ScIdx0 ] for the value ScIdx0 of ScIdx that was in use for the previous AU.

– Otherwise, the HSS continues to operate with the previous values of ScIdx, BitRate[ Htid ][ ScIdx ] and CpbSize[ Htid ][ ScIdx ].

When the HSS selects values of BitRate[ Htid ][ ScIdx ] or CpbSize[ Htid ][ ScIdx ] that differ from those of the previous AU, the following applies:

– The variable BitRate[ Htid ][ ScIdx ] comes into effect at the initial CPB arrival time of the current AU.

– The variable CpbSize[ Htid ][ ScIdx ] comes into effect as follows:

– If the new value of CpbSize[ Htid ][ ScIdx ] is greater than the old CPB size, it comes into effect at the initial CPB arrival time of the current AU.

– Otherwise, the new value of CpbSize[ Htid ][ ScIdx ] comes into effect at the CPB removal time of the current AU.

* + 1. Timing of DU removal and decoding of DU

The variables InitCpbRemovalDelay[ Htid ][ ScIdx ], and InitCpbRemovalDelayOffset[ Htid ][ ScIdx ], are derived as follows: [Ed. (VD): InitCpbRemovalDelayOffset[ Htid ][ ScIdx ] is derived here but never used in this clause. (YK): Hmm, this is simialr as in HEVC. I don’t know why it was like this in HEVC; maybe inherited from AVC. We should study whether the derivation of InitCpbRemovalDelayOffset[ Htid ][ ScIdx ] is really not needed herein, and if not, we can just remove it. Probably also for HEVC.]

– If one or more of the following conditions are true, InitCpbRemovalDelay[ Htid ][ ScIdx ] and InitCpbRemovalDelayOffset[ Htid ][ ScIdx ] are set equal to the values of the BP SEI message syntax elements nal\_initial\_cpb\_removal\_delay[ Htid ][ ScIdx ] and nal\_initial\_cpb\_removal\_offset[ Htid ][ ScIdx ] minus the values of the PT SEI message syntax elements cpb\_alt\_initial\_removal\_delay\_delta[ Htid ][ ScIdx ] and cpb\_alt\_initial\_removal\_offset\_delta[ Htid ][ ScIdx ] of AU n + 1, respectively, when NalHrdModeFlag is equal to 1, or vcl\_initial\_cpb\_removal\_delay[ Htid ][ ScIdx ] and vcl\_initial\_cpb\_removal\_offset[ Htid ][ ScIdx ] minus the values of the PT SEI message syntax elements cpb\_alt\_initial\_removal\_delay\_delta[ Htid ][ ScIdx ] and cpb\_alt\_initial\_removal\_offset\_delta[ Htid ][ ScIdx ] of AU n + 1, respectively, when NalHrdModeFlag is equal to 0, where the BP SEI message containing the syntax elements is selected as specified in clause C.1:

– UseAltCpbParamsFlag for AU n is equal to 1.

– DefaultInitCpbParamsFlag is equal to 0.

– Otherwise, if the value of decodingUnitParamsFlag is equal to 1, InitCpbRemovalDelay[ Htid ][ ScIdx ] and InitCpbRemovalDelayOffset[ Htid ][ ScIdx ] are set equal to the values of the BP SEI message syntax elements nal\_initial\_alt\_cpb\_removal\_delay[ Htid ][ ScIdx ] and nal\_initial\_alt\_cpb\_removal\_offset[ Htid ][ ScIdx ], respectively, when NalHrdModeFlag is equal to 1, or vcl\_initial\_alt\_cpb\_removal\_delay[ Htid ][ ScIdx ] and vcl\_initial\_alt\_cpb\_removal\_offset[ Htid ][ ScIdx ], respectively, when NalHrdModeFlag is equal to 0, where the BP SEI message containing the syntax elements is selected as specified in clause C.1.

– Otherwise (DecodingUnitHrdFlag is equal to 0) [Ed. (VD/YK): Something is wrong here, at least for the mixed use of decodingUnitParamsFlag, which is but should not be a local variable, and DecodingUnitHrdFlag.], InitCpbRemovalDelay[ Htid ][ ScIdx ] and InitCpbRemovalDelayOffset[ Htid ][ ScIdx ] are set equal to the values of the BP SEI message syntax elements nal\_initial\_cpb\_removal\_delay[ Htid ][ ScIdx ] and nal\_initial\_cpb\_removal\_offset[ Htid ][ ScIdx ], respectively, when NalHrdModeFlag is equal to 1, or vcl\_initial\_cpb\_removal\_delay[ Htid ][ ScIdx ] and vcl\_initial\_cpb\_removal\_offset[ Htid ][ ScIdx ], respectively, when NalHrdModeFlag is equal to 0, where the BP SEI message containing the syntax elements is selected as specified in clause C.1.

The nominal removal time of the AU n from the CPB is specified as follows:

– If AU n is the AU with n equal to 0 (the AU that initializes the HRD), the nominal removal time of the AU from the CPB is specified by:

AuNominalRemovalTime[ 0 ] = InitCpbRemovalDelay[ Htid ][ ScIdx ] ÷ 90000 (C.9)  
[Ed. (YK): It's probably bettter to conistently either use an Htid-index for both InitCpbRemovalDelay and NominalRemovalTime, or not use this index for both, the latter of which means to drop the Htid-index in many other HRD parameters/variables as well, like BitRate, CpbSize, etc.]

– Otherwise, the following applies: [Ed. (VD/YK): Throughout the document, carefully and consistently apply the prefixes "Au" and "Du" to variables FinalArrivalTime, NominalRemovalTime, CpbRemovalTime, etc.]

– When AU n is the first AU of a BP that does not initialize the HRD, the following applies:

The nominal removal time of the AU n from the CPB is specified by:

if( !concatenationFlag ) {  
 baseTime = AuNominalRemovalTime[ firstPicInPrevBuffPeriod ]  
 tmpCpbRemovalDelay = AuCpbRemovalDelayVal  
} else {  
 baseTime1 = AuNominalRemovalTime[ prevNonDiscardablePic ]  
 tmpCpbRemovalDelay1 = ( auCpbRemovalDelayDeltaMinus1 + 1 )  
 baseTime2 = AuNominalRemovalTime[ n − 1 ]  
 tmpCpbRemovalDelay2 = (C.10)  
 Ceil( ( InitCpbRemovalDelay[ Htid ][ ScIdx ] ÷ 90000 +  
 AuFinalArrivalTime[ n − 1 ] − AuNominalRemovalTime[ n − 1 ] ) ÷ ClockTick )  
 if( baseTime1 + ClockTick \* tmpCpbRemovalDelay1 <  
 baseTime2 + ClockTick \* tmpCpbRemovalDelay2 ) {  
 baseTime = baseTime2  
 tmpCpbRemovalDelay = tmpCpbRemovalDelay2  
 } else {  
 baseTime = baseTime1  
 tmpCpbRemovalDelay = tmpCpbRemovalDelay1  
 }  
}  
AuNominalRemovalTime[ n ] = baseTime + ( ClockTick \* tmpCpbRemovalDelay − CpbDelayOffset )

where AuNominalRemovalTime[ firstPicInPrevBuffPeriod ] is the nominal removal time of the first AU of the previous BP, AuNominalRemovalTime[ prevNonDiscardablePic ] is the nominal removal time of the preceding picture in decoding order with TemporalId equal to 0 that is not a RASL or RADL picture, AuCpbRemovalDelayVal is the value of CpbRemovalDelayVal[ Htid ] derived according to cpb\_removal\_delay\_minus1[ Htid ] and cpb\_removal\_delay\_delta\_idx[ Htid ] in the PT SEI message, and cpb\_removal\_delay\_delta[ cpb\_removal\_delay\_delta\_idx[ Htid ] ] in the BP SEI message, selected as specified in clause C.1, associated with AU n and concatenationFlag and CpbRemovalDelayDeltaMinus1 are the values of the syntax elements concatenation\_flag and cpb\_removal\_delay\_delta\_minus1, respectively, in the BP SEI message, selected as specified in clause C.1, associated with AU n.

After the derivation of the nominal CPB removal time and before the derivation of the DPB output time of access unit n, the variables DpbDelayOffset and CpbDelayOffset are derived as:

– If one or more of the following conditions are true, DpbDelayOffset is set equal to the value of the PT SEI message syntax element dpb\_delay\_offset[ Htid ] of AU n + 1, and CpbDelayOffset is set equal to the value of the PT SEI message syntax element cpb\_delay\_offset[ Htid ] of AU n + 1, where the PT SEI message containing the syntax elements is selected as specified in clause C.1:

– UseAltCpbParamsFlag for AU n is equal to 1.

– DefaultInitCpbParamsFlag is equal to 0.

– Otherwise, DpbDelayOffset and CpbDelayOffset are both set equal to 0.

– When AU n is not the first AU of a BP, the nominal removal time of the AU n from the CPB is specified by:

AuNominalRemovalTime[ n ] = AuNominalRemovalTime[ firstPicInCurrBuffPeriod ] +  
 ClockTick \* ( AuCpbRemovalDelayVal − CpbDelayOffset ) (C.11)

where AuNominalRemovalTime[ firstPicInCurrBuffPeriod ] is the nominal removal time of the first AU of the current BP and AuCpbRemovalDelayVal is the value of CpbRemovalDelayVal[ OpTid ] derived according to cpb\_removal\_delay\_minus1[ OpTid ] and cpb\_removal\_delay\_delta\_idx[ OpTid ] in the PT SEI message, and cpb\_removal\_delay\_delta[ cpb\_removal\_delay\_delta\_idx[ OpTid ] ] in the BP SEI message, selected as specified in clause C.1, associated with AU n.

When DecodingUnitHrdFlag is equal to 1, the following applies:

– When num\_decoding\_units\_minus1 is greater than 0, the variable duCpbRemovalDelayInc is derived as follows:

– If du\_cpb\_params\_in\_pic\_timing\_sei\_flag is equal to 0, duCpbRemovalDelayInc is set equal to the value of du\_spt\_cpb\_removal\_delay\_increment[ i ] in the DU information SEI message, selected as specified in clause C.1, associated with DU m.

– Otherwise, if du\_common\_cpb\_removal\_delay\_flag is equal to 0, duCpbRemovalDelayInc is set equal to the value of du\_cpb\_removal\_delay\_increment\_minus1[ i ][ Htid ] + 1 for DU m in the PT SEI message, selected as specified in clause C.1, associated with AU n, where the value of i is 0 for the first num\_nalus\_in\_du\_minus1[ 0 ] + 1 consecutive NAL units in the AU that contains DU m, 1 for the subsequent num\_nalus\_in\_du\_minus1[ 1 ] + 1 NAL units in the same AU, 2 for the subsequent num\_nalus\_in\_du\_minus1[ 2 ] + 1 NAL units in the same AU, etc.

– Otherwise, duCpbRemovalDelayInc is set equal to the value of du\_common\_cpb\_removal\_delay\_increment\_minus1[ Htid ] + 1 in the PT SEI message, selected as specified in clause C.1, associated with AU n.

– The nominal removal time of DU m from the CPB is specified as follows, where AuNominalRemovalTime[ n ] is the nominal removal time of AU n:

– If DU m is the last DU in AU n, the nominal removal time of DU m DuNominalRemovalTime[ m ] is set equal to AuNominalRemovalTime[ n ].

– Otherwise (DU m is not the last DU in AU n), the nominal removal time of DU m DuNominalRemovalTime[ m ] is derived as follows:

if( du\_cpb\_params\_in\_pic\_timing\_sei\_flag )  
 DuNominalRemovalTime[ m ] = DuNominalRemovalTime[ m + 1 ] −  
 ClockSubTick \* duCpbRemovalDelayInc (C.12)  
else  
 DuNominalRemovalTime[ m ] = AuNominalRemovalTime[ n ] −  
 ClockSubTick \* duCpbRemovalDelayInc

If DecodingUnitHrdFlag is equal to 0, the removal time of AU n from the CPB is specified as follows, where FinalArrivalTime[ n ] and NominalRemovalTime[ n ] are the final CPB arrival time and nominal CPB removal time, respectively, of AU n:

if( !low\_delay\_hrd\_flag[ Htid ] | | AuNominalRemovalTime[ n ] >= AuFinalArrivalTime[ n ] )  
 AuCpbRemovalTime[ n ] = AuNominalRemovalTime[ n ]  
else (C.13)  
 AuCpbRemovalTime[ n ] = AuNominalRemovalTime[ n ] + ClockTick \*  
 Ceil( ( AuFinalArrivalTime[ n ] − AuNominalRemovalTime[ n ] ) ÷ ClockTick )

NOTE 1 – When low\_delay\_hrd\_flag[ Htid ] is equal to 1 and AuNominalRemovalTime[ n ] is less than AuFinalArrivalTime[ n ], the size of AU n is so large that it prevents removal at the nominal removal time.

Otherwise (DecodingUnitHrdFlag is equal to 1), the removal time of DU m from the CPB is specified as follows:

if( !low\_delay\_hrd\_flag[ Htid ] | | DuNominalRemovalTime[ m ] >= DuFinalArrivalTime[ m ] )  
 DuCpbRemovalTime[ m ] = DuNominalRemovalTime[ m ]  
else (C.14)  
 DuCpbRemovalTime[ m ] = DuFinalArrivalTime[ m ]

NOTE 2 – When low\_delay\_hrd\_flag[ Htid ] is equal to 1 and DuNominalRemovalTime[ m ] is less than DuFinalArrivalTime[ m ], the size of DU m is so large that it prevents removal at the nominal removal time.

If DecodingUnitHrdFlag is equal to 0, at the CPB removal time of AU n, the AU is instantaneously decoded.

Otherwise (DecodingUnitHrdFlag is equal to 1), at the CPB removal time of DU m, the DU is instantaneously decoded, and when DU m is the last DU of AU n, the following applies:

– Picture n is considered as decoded.

– The final CPB arrival time of AU n, i.e., AuFinalArrivalTime[ n ], is set equal to the final CPB arrival time of the last DU in AU n, i.e., DuFinalArrivalTime[ m ].

– The nominal CPB removal time of AU n, i.e., AuNominalRemovalTime[ n ], is set equal to the nominal CPB removal time of the last DU in AU n, i.e., DuNominalRemovalTime[ m ].

– The CPB removal time of AU n, i.e., AuCpbRemovalTime[ m ], is set equal to the CPB removal time of the last DU in AU n, i.e., DuCpbRemovalTime[ m ].

* 1. Operation of the DPB
     1. General

The specifications in this clause apply independently to each set of DPB parameters selected as specified in clause C.1.

The DPB contains picture storage buffers for storage of decoded pictures. Each of the picture storage buffers may contain a decoded picture that is marked as "used for reference" or is held for future output. The processes specified in clauses C.3.2, C.3.3, and C.3.4 are sequentially applied as specified below, and are separately applied for each layer, starting from the lowest layer in the OLS, in increasing order of nuh\_layer\_id values of the layers in the OLS.

NOTE – In the operation of output timing DPB, decoded pictures with PicOutputFlag equal to 1 in the same AU are output consecutively in ascending order of the nuh\_layer\_id values of the decoded pictures.

Let picture n and the current picture be the coded picture or decoded picture of the AU n for a particular value of nuh\_layer\_id, wherein n is a non-negative integer number.

* + 1. Removal of pictures from the DPB before decoding of the current picture

The removal of pictures from the DPB before decoding of the current picture (but after parsing the slice header of the first slice of the current picture) happens instantaneously at the CPB removal time of the first DU of AU n (containing the current picture) and proceeds as follows:

– The decoding process for reference picture list construction as specified in clause 8.3.2 is invoked and the decoding process for reference picture marking as specified in clause 8.3.3 is invoked.

– When the current AU is a CVSS AU that is not AU 0, the following ordered steps are applied:

1. The variable NoOutputOfPriorPicsFlag is derived for the decoder under test as follows:

– If the value of PicWidthMaxInSamplesY, PicHeightMaxInSamplesY, MaxChromaFormat, MaxBitDepthMinus8, or max\_dec\_pic\_buffering\_minus1[ Htid ] derived for the current AU is different from the value of PicWidthMaxInSamplesY, PicHeightMaxInSamplesY, MaxChromaFormat, MaxBitDepthMinus8, or max\_dec\_pic\_buffering\_minus1[ Htid ], respectively, derived for the preceding AU in decoding order, NoOutputOfPriorPicsFlag may (but should not) be set equal to 1 by the decoder under test, regardless of the value of ph\_no\_output\_of\_prior\_pics\_flag of the current AU.

NOTE – Although setting NoOutputOfPriorPicsFlag equal to ph\_no\_output\_of\_prior\_pics\_flag of the current AU is preferred under these conditions, the decoder under test is allowed to set NoOutputOfPriorPicsFlag equal to 1 in this case.

– Otherwise, NoOutputOfPriorPicsFlag is set equal to ph\_no\_output\_of\_prior\_pics\_flag of the current AU.

2. The value of NoOutputOfPriorPicsFlag derived for the decoder under test is applied for the HRD, such that when the value of NoOutputOfPriorPicsFlag is equal to 1, all picture storage buffers in the DPB are emptied without output of the pictures they contain, and the DPB fullness is set equal to 0.

– When both of the following conditions are true for any pictures k in the DPB, all such pictures k in the DPB are removed from the DPB:

– picture k is marked as "unused for reference".

– picture k has PictureOutputFlag equal to 0 or its DPB output time is less than or equal to the CPB removal time of the first DU (denoted as DU m) of the current picture n; i.e., DpbOutputTime[ k ] is less than or equal to DuCpbRemovalTime[ m ].

– For each picture that is removed from the DPB, the DPB fullness is decremented by one.

* + 1. Picture output

The processes specified in this clause happen instantaneously at the CPB removal time of AU n, CpbRemovalTime[ n ].

When picture n has PictureOutputFlag equal to 1, its DPB output time DpbOutputTime[ n ] is derived as follows, where the variable firstPicInBufferingPeriodFlag is equal to 1 if AU n is the first AU of a BP and 0 otherwise:

if( !DecodingUnitHrdFlag ) {  
 DpbOutputTime[ n ] = CpbRemovalTime[ n ] + ClockTick \* ( picDpbOutputDelay – (C.15)  
 picDpbOutputDelta )  
 if( firstPicInBufferingPeriodFlag )  
 DpbOutputTime[ n ] −= ClockTick \* DpbDelayOffset  
} else  
 DpbOutputTime[ n ] = AuCpbRemovalTime[ n ] + ClockSubTick \* picSptDpbOutputDuDelay

where picDpbOutputDelay is the value of dpb\_output\_delay and picDpbOutputDelta is the value of picDpbOutputDelta[ Htid ] derived according to cpb\_removal\_delay\_minus1[ Htid ], and cpb\_removal\_delay\_delta\_idx[ Htid ] in the PT SEI message associated with AU n, and cpb\_removal\_delay\_delta[ cpb\_removal\_delay\_delta\_idx[ Htid ] ] and dpb\_output\_tid\_offset[ Htid ] in the BP SEI message associated with AU n, and picSptDpbOutputDuDelay is the value of pic\_spt\_dpb\_output\_du\_delay, when du\_dpb\_params\_in\_pic\_timing\_sei\_flag is equal to 0, in the DU information SEI messages associated with AU n, or the value of pic\_dpb\_output\_du\_delay in the PT SEI message associated with AU n when du\_dpb\_params\_in\_pic\_timing\_sei\_flag is equal top 1.

NOTE – When the syntax element pic\_spt\_dpb\_output\_du\_delay is not present in any DU information SEI message associated with AU n, the value is inferred to be equal to pic\_dpb\_output\_du\_delay in the PT SEI message associated with AU n.

The output of the current picture is specified as follows:

– If PictureOutputFlag is equal to 1 and DpbOutputTime[ n ] is equal to CpbRemovalTime[ n ], the current picture is output.

– Otherwise, if PictureOutputFlag is equal to 0, the current picture is not output, but will be stored in the DPB as specified in clause C.3.4.

– Otherwise (PictureOutputFlag is equal to 1 and DpbOutputTime[ n ] is greater than CpbRemovalTime[ n ] ), the current picture is output later and will be stored in the DPB (as specified in clause C.3.4) and is output at time DpbOutputTime[ n ] unless indicated not to be output by NoOutputOfPriorPicsFlag equal to 1.

When output, the picture is cropped, using the conformance cropping window for the picture.

When picture n is a picture that is output and is not the last picture of the bitstream that is output, the value of the variable DpbOutputInterval[ n ] is derived as follows:

DpbOutputInterval[ n ] = DpbOutputTime[ nextPicInOutputOrder ] − DpbOutputTime[ n ] (C.16)

where nextPicInOutputOrder is the picture that follows picture n in output order and has PictureOutputFlag equal to 1.

* + 1. Current decoded picture marking and storage

The current decoded picture is stored in the DPB in an empty picture storage buffer, the DPB fullness is incremented by one, and the current picture is marked as "used for short-term reference".

NOTE – Unless more memory than required by the level limit is available for storage of decoded pictures, decoders should start storing decoded parts of the current picture into the DPB when the first slice is decoded and continue storing more decoded samples as the decoding process proceeds.

* 1. Bitstream conformance

A bitstream of coded data conforming to this Specification shall fulfil all requirements specified in this clause.

The bitstream shall be constructed according to the syntax, semantics and constraints specified in this Specification outside of this annex.

The first coded picture in a bitstream shall be an IRAP picture (i.e., an IDR picture or a CRA picture) or a GDR picture.

The bitstream is tested by the HRD for conformance as specified in clause C.1.

Let currPicLayerId be equal to the nuh\_layer\_id of the current picture.

For each current picture, let the variables maxPicOrderCnt and minPicOrderCnt be set equal to the maximum and the minimum, respectively, of the PicOrderCntVal values of the following pictures with nuh\_layer\_id equal to currPicLayerId:

– The current picture.

– The previous picture in decoding order that has TemporalId equal to 0 and is not a RASL or RADL picture.

– The STRPs referred to by all entries in RefPicList[ 0 ] and all entries in RefPicList[ 1 ] of the current picture.

– All pictures n that have PictureOutputFlag equal to 1, CpbRemovalTime[ n ] less than CpbRemovalTime[ currPic ] and DpbOutputTime[ n ] greater than or equal to CpbRemovalTime[ currPic ], where currPic is the current picture.

All of the following conditions shall be fulfilled for each of the bitstream conformance tests:

1. For each AU n, with n greater than 0, associated with a BP SEI message, let the variable deltaTime90k[ n ] be specified as follows:

deltaTime90k[ n ] = 90000 \* ( NominalRemovalTime[ n ] − FinalArrivalTime[ n − 1 ] ) (C.17)

The value of InitCpbRemovalDelay[ Htid ][ ScIdx ] is constrained as follows:

– If cbr\_flag[ ScIdx ] is equal to 0, the following condition shall be true:

InitCpbRemovalDelay[ Htid ][ ScIdx ] <= Ceil( deltaTime90k[ n ] ) (C.18)

– Otherwise (cbr\_flag[ ScIdx ] is equal to 1), the following condition shall be true:

Floor( deltaTime90k[ n ] ) <= InitCpbRemovalDelay[ Htid ][ ScIdx ] <= Ceil( deltaTime90k[ n ] ) (C.19)

NOTE 1 – The exact number of bits in the CPB at the removal time of each picture may depend on which BP SEI message is selected to initialize the HRD. Encoders must take this into account to ensure that all specified constraints must be obeyed regardless of which BP SEI message is selected to initialize the HRD, as the HRD may be initialized at any one of the BP SEI messages.

1. A CPB overflow is specified as the condition in which the total number of bits in the CPB is greater than the CPB size. The CPB shall never overflow.
2. When low\_delay\_hrd\_flag[ Htid ] is equal to 0, the CPB shall never underflow. A CPB underflow is specified as follows:

– If DecodingUnitHrdFlag is equal to 0, a CPB underflow is specified as the condition in which the nominal CPB removal time of AU n NominalRemovalTime[ n ] is less than the final CPB arrival time of AU n FinalArrivalTime[ n ] for at least one value of n.

– Otherwise (DecodingUnitHrdFlagis equal to 1), a CPB underflow is specified as the condition in which the nominal CPB removal time of DU m DuNominalRemovalTime[ m ] is less than the final CPB arrival time of DU m DuFinalArrivalTime[ m ] for at least one value of m.

1. When DecodingUnitHrdFlag is equal to 1, low\_delay\_hrd\_flag[ Htid ] is equal to 1 and the nominal removal time of a DU m of AU n is less than the final CPB arrival time of DU m (i.e., DuNominalRemovalTime[ m ] < DuFinalArrivalTime[ m ]), the nominal removal time of AU n shall be less than the final CPB arrival time of AU n (i.e., AuNominalRemovalTime[ n ] < AuFinalArrivalTime[ n ]).
2. The nominal removal times of pictures from the CPB (starting from the second picture in decoding order) shall satisfy the constraints on NominalRemovalTime[ n ] and CpbRemovalTime[ n ] expressed in clauses A.4.1 through A.4.2.
3. For each current picture, after invocation of the process for removal of pictures from the DPB as specified in clause C.3.2, the number of decoded pictures in the DPB, including all pictures n that are marked as "used for reference", or that have PictureOutputFlag equal to 1 and CpbRemovalTime[ n ] less than CpbRemovalTime[ currPic ], where currPic is the current picture, shall be less than or equal to max\_dec\_pic\_buffering\_minus1[ Htid ].
4. All reference pictures shall be present in the DPB when needed for prediction. Each picture that has PictureOutputFlag equal to 1 shall be present in the DPB at its DPB output time unless it is removed from the DPB before its output time by one of the processes specified in clause C.3.
5. For each current picture that is not a CLVSS picture, the value of maxPicOrderCnt − minPicOrderCnt shall be less than MaxPicOrderCntLsb / 2.
6. The value of DpbOutputInterval[ n ] as given by Equation C.16, which is the difference between the output times of a picture and the first picture following it in output order and having PictureOutputFlag equal to 1, shall satisfy the constraint expressed in clause A.4.1 for the profile, tier and level specified in the bitstream using the decoding process specified in clauses 2 through 9.
7. For each current picture, when du\_cpb\_params\_in\_pic\_timing\_sei\_flag is equal to 1, let tmpCpbRemovalDelaySum be derived as follows:

tmpCpbRemovalDelaySum = 0  
for( i = 0; i < num\_decoding\_units\_minus1; i++ ) (C.20)  
 tmpCpbRemovalDelaySum += du\_cpb\_removal\_delay\_increment\_minus1[ i ][ Htid ] + 1

The value of ClockSubTick \* tmpCpbRemovalDelaySum shall be equal to the difference between the nominal CPB removal time of the current AU and the nominal CPB removal time of the first DU in the current AU in decoding order.

1. For any two pictures m and n in the same CVS, when DpbOutputTime[ m ] is greater than DpbOutputTime[ n ], the PicOrderCntVal of picture m shall be greater than the PicOrderCntVal of picture n.

NOTE 2 – All pictures of an earlier CVS in decoding order that are output are output before any pictures of a later CVS in decoding order. Within any particular CVS, the pictures that are output are output in increasing PicOrderCntVal order.

1. The DPB output times derived for all pictures in any particular AU shall be the same.
   1. Decoder conformance
      1. General

A decoder conforming to this Specification shall fulfil all requirements specified in this clause.

A decoder claiming conformance to a specific profile, tier and level shall be able to successfully decode all bitstreams that conform to the bitstream conformance requirements specified in clause C.4, in the manner specified in Annex A, provided that all DCI NAL units, when available, all VPSs, SPSs, PPSs and APSs referred to in the VCL NAL units, and appropriate BP, PT, and DU information SEI messages are conveyed to the decoder, in a timely manner, either in the bitstream (by non-VCL NAL units), or by external means not specified in this Specification.

When a bitstream contains syntax elements that have values that are specified as reserved and it is specified that decoders shall ignore values of the syntax elements or NAL units containing the syntax elements having the reserved values, and the bitstream is otherwise conforming to this Specification, a conforming decoder shall decode the bitstream in the same manner as it would decode a conforming bitstream and shall ignore the syntax elements or the NAL units containing the syntax elements having the reserved values as specified.

There are two types of conformance that can be claimed by a decoder: output timing conformance and output order conformance.

To check conformance of a decoder, test bitstreams conforming to the claimed profile, tier and level, as specified in clause C.4 are delivered by a hypothetical stream scheduler (HSS) both to the HRD and to the decoder under test (DUT). All cropped decoded pictures output by the HRD shall also be output by the DUT, each cropped decoded picture output by the DUT shall be a picture with PictureOutputFlag equal to 1, and, for each such cropped decoded picture output by the DUT, the values of all samples that are output shall be equal to the values of the samples produced by the specified decoding process.

For output timing decoder conformance, the HSS operates as described above, with delivery schedules selected only from the subset of values of ScIdx for which the bit rate and CPB size are restricted as specified in Annex A for the specified profile, tier and level or with "interpolated" delivery schedules as specified below for which the bit rate and CPB size are restricted as specified in Annex A. The same delivery schedule is used for both the HRD and the DUT.

When the HRD parameters and the BP SEI messages are present with hrd\_cpb\_cnt\_minus1 and bp\_cpb\_cnt\_minus1, respectively, greater than 0, the decoder shall be capable of decoding the bitstream as delivered from the HSS operating using an "interpolated" delivery schedule specified as having peak bit rate r, CPB size c( r ) and initial CPB removal delay ( f( r ) ÷ r ) as follows:

α = ( r − BitRate[ Htid ][ ScIdx − 1 ] ) ÷ ( BitRate[ Htid ][ ScIdx ] −  
 BitRate[ Htid ][ ScIdx − 1 ] ), (C.21)

c( r ) = α \* CpbSize[ Htid ][ ScIdx ] + ( 1 − α ) \* CpbSize[ Htid ][ ScIdx − 1 ], (C.22)

f( r ) = α \* InitCpbRemovalDelay[ Htid ][ ScIdx ] \* BitRate[ Htid ][ ScIdx ] +  
 ( 1 − α ) \* InitCpbRemovalDelay[ Htid ][ ScIdx − 1 ] \*  
 BitRate[ Htid ][ ScIdx − 1 ] (C.23)

for any ScIdx > 0 and r such that BitRate[ Htid ][ ScIdx − 1 ] <= r <= BitRate[ Htid ][ ScIdx ] such that r and c( r ) are within the limits as specified in Annex A for the maximum bit rate and buffer size for the specified profile, tier and level.

NOTE 1 – InitCpbRemovalDelay[ Htid ][ ScIdx ] can be different from one BP to another and have to be re-calculated.

For output timing decoder conformance, an HRD as described above is used and the timing (relative to the delivery time of the first bit) of picture output is the same for both the HRD and the DUT up to a fixed delay.

For output order decoder conformance, the following applies:

– The HSS delivers the bitstream BitstreamToDecode to the DUT "by demand" from the DUT, meaning that the HSS delivers bits (in decoding order) only when the DUT requires more bits to proceed with its processing.

NOTE 2 – This means that for this test, the CPB of the DUT could be as small as the size of the largest DU.

– A modified HRD as described below is used, and the HSS delivers the bitstream to the HRD by one of the schedules specified in the bitstream BitstreamToDecode such that the bit rate and CPB size are restricted as specified in Annex A. The order of pictures output shall be the same for both the HRD and the DUT.

– The HRD CPB size is given by CpbSize[ Htid ][ ScIdx ] as specified in clause 7.4.6.3, where ScIdx and the HRD parameters are selected as specified in clause C.1. The DPB size is given by max\_dec\_pic\_buffering\_minus1[ Htid ] + 1. Removal time from the CPB for the HRD is the final bit arrival time and decoding is immediate. The operation of the DPB of this HRD is as described in clauses C.5.2 through C.5.2.3.

* + 1. Operation of the output order DPB
       1. General

The specifications in this clause apply independently to each set of DPB parameters selected as specified in clause C.1.

The DPB contains picture storage buffers for storage of decoded pictures. Each of the picture storage buffers contains a decoded picture that is marked as "used for reference" or is held for future output.

The process for output and removal of pictures from the DPB before decoding of the current picture as specified in clause C.5.2.2 is invoked, followed by the invocation of the process for current decoded picture marking and storage as specified in clause C.3.4, and finally followed by the invocation of the process for additional bumping as specified in clause C.5.2.3. The "bumping" process is specified in clause C.5.2.4 and is invoked as specified in clauses C.5.2.2 and C.5.2.3.

These processes are separately applied for each layer, starting from the lowest layer in the OLS, in increasing order of the nuh\_layer\_id values of the layers in the OLS.

NOTE – In the operation of output order DPB, same as in the operation of output timing DPB, decoded pictures with PicOutputFlag equal to 1 in the same AU are also output consecutively in ascending order of the nuh\_layer\_id values of the decoded pictures.

Let picture n and the current picture be the coded picture or decoded picture of the AU n for a particular value of nuh\_layer\_id, wherein n is a non-negative integer number.

* + - 1. Output and removal of pictures from the DPB

The output and removal of pictures from the DPB before the decoding of the current picture (but after parsing the slice header of the first slice of the current picture) happens instantaneously when the first DU of the AU containing the current picture is removed from the CPB and proceeds as follows:

– The decoding process for reference picture list construction as specified in clause 8.3.2 and decoding process for reference picture marking as specified in clause 8.3.3 are invoked.

– If the current AU is a CVSS AU that is not AU 0, the following ordered steps are applied:

1. The variable NoOutputOfPriorPicsFlag is derived for the decoder under test as follows:

– If the value of PicWidthMaxInSamplesY, PicHeightMaxInSamplesY, MaxChromaFormat, MaxBitDepthMinus8, or max\_dec\_pic\_buffering\_minus1[ Htid ] derived for the current AU is different from the value of PicWidthMaxInSamplesY, PicHeightMaxInSamplesY, MaxChromaFormat, MaxBitDepthMinus8, or max\_dec\_pic\_buffering\_minus1[ Htid ], respectively, derived for the preceding AU in decoding order, NoOutputOfPriorPicsFlag may (but should not) be set equal to 1 by the decoder under test, regardless of the value of ph\_no\_output\_of\_prior\_pics\_flag of the current AU.

NOTE – Although setting NoOutputOfPriorPicsFlag equal to ph\_no\_output\_of\_prior\_pics\_flag of the current AU is preferred under these conditions, the decoder under test is allowed to set NoOutputOfPriorPicsFlag equal to 1 in this case.

– Otherwise, NoOutputOfPriorPicsFlag is set equal to ph\_no\_output\_of\_prior\_pics\_flag of the current AU.

2. The value of NoOutputOfPriorPicsFlag derived for the decoder under test is applied for the HRD as follows:

– If NoOutputOfPriorPicsFlag is equal to 1, all picture storage buffers in the DPB are emptied without output of the pictures they contain and the DPB fullness is set equal to 0.

– Otherwise (NoOutputOfPriorPicsFlag is equal to 0), all picture storage buffers containing a picture that is marked as "not needed for output" and "unused for reference" are emptied (without output) and all non-empty picture storage buffers in the DPB are emptied by repeatedly invoking the "bumping" process specified in clause C.5.2.4 and the DPB fullness is set equal to 0.

– Otherwise (the current AU is not a CVSS AU), all picture storage buffers containing a picture which are marked as "not needed for output" and "unused for reference" are emptied (without output). For each picture storage buffer that is emptied, the DPB fullness is decremented by one. When one or more of the following conditions are true, the "bumping" process specified in clause C.5.2.4 is invoked repeatedly while further decrementing the DPB fullness by one for each additional picture storage buffer that is emptied, until none of the following conditions are true:

* The number of pictures in the DPB that are marked as "needed for output" is greater than max\_num\_reorder\_pics[ Htid ].
* max\_latency\_increase\_plus1[ Htid ] is not equal to 0 and there is at least one picture in the DPB that is marked as "needed for output" for which the associated variable PicLatencyCount is greater than or equal to MaxLatencyPictures[ Htid ].
* The number of pictures in the DPB is greater than or equal to max\_dec\_pic\_buffering\_minus1[ Htid ] + 1.
  + - 1. Additional bumping

The processes specified in this clause happen instantaneously when the last DU of AU n containing the current picture is removed from the CPB.

When the current picture has PictureOutputFlag equal to 1, for each picture in the DPB that is marked as "needed for output" and follows the current picture in output order, the associated variable PicLatencyCount is set equal to PicLatencyCount + 1.

The following applies:

– If the current decoded picture has PictureOutputFlag equal to 1, it is marked as "needed for output" and its associated variable PicLatencyCount is set equal to 0.

– Otherwise (the current decoded picture has PictureOutputFlag equal to 0), it is marked as "not needed for output".

When one or more of the following conditions are true, the "bumping" process specified in clause C.5.2.4 is invoked repeatedly until none of the following conditions are true:

– The number of pictures in the DPB that are marked as "needed for output" is greater than max\_num\_reorder\_pics[ Htid ].

– max\_latency\_increase\_plus1[ Htid ] is not equal to 0 and there is at least one picture in the DPB that is marked as "needed for output" for which the associated variable PicLatencyCount that is greater than or equal to MaxLatencyPictures[ Htid ].

* + - 1. "Bumping" process

The "bumping" process consists of the following ordered steps:

1. The picture or pictures that are first for output are selected as the one having the smallest value of PicOrderCntVal of all pictures in the DPB marked as "needed for output".
2. Each of these pictures, in ascending nuh\_layer\_id order, is cropped, using the conformance cropping window for the picture, the cropped picture is output, and the picture is marked as "not needed for output".
3. Each picture storage buffer that contains a picture marked as "unused for reference" and was one of the pictures cropped and output is emptied and the fullness of the DPB is decremented by one.

NOTE – For any two pictures picA and picB that belong to the same CVS and are output by the "bumping process", when picA is output earlier than picB, one of the following conditions applies:

– The value of PicOrderCntVal of picA and the value of PicOrderCntVal of picB are the same and the nuh\_layer\_id of picA is less than the nuh\_layer\_id\_of picB.

– The value of PicOrderCntVal of picA is less than the value of PicOrderCntVal of picB.

* 1. Sub-bitstream extraction process

Inputs to this process are a bitstream inBitstream, a target OLS index targetOlsIdx, and a target highest TemporalId value tIdTarget.

Output of this process is a sub-bitstream outBitstream.

It is a requirement of bitstream conformance for the input bitstream that any output sub-bitstream that satisfies all of the following conditions shall be a conforming bitstream:

– The output sub-bitstream is the output of the process specified in this clause with the bitstream, targetOlsIdx equal to an index to the list of OLSs specified by the VPS, and tIdTarget equal to any value in the range of 0 to 6, inclusive, as inputs.

– The output sub-bitstream contains at least one VCL NAL unit with nuh\_layer\_id equal to each of the nuh\_layer\_id values in LayerIdInOls[ targetOlsIdx ].

– The output sub-bitstream contains at least one VCL NAL unit with TemporalId equal to tIdTarget.

NOTE – A conforming bitstream contains one or more coded slice NAL units with TemporalId equal to 0, but does not have to contain coded slice NAL units with nuh\_layer\_id equal to 0.

The output sub-bitstream OutBitstream is derived as follows:

– The bitstream outBitstream is set to be identical to the bitstream inBitstream.

– Remove from outBitstream all NAL units with TemporalId greater than tIdTarget.

– Remove from outBitstream all NAL units with nal\_unit\_type not equal to any of VPS\_NUT, DCI\_NUT, and EOB\_NUT and with nuh\_layer\_id not included in the list LayerIdInOls[ targetOlsIdx ].

– Remove from outBitstream all VCL NAL units for which all of the following conditions are true, and their associated non-VCL NAL units with nal\_unit\_type equal to PH\_NUT, FD\_NUT, SUFFIX\_SEI\_NUT, and PREFIX\_SEI\_NUT with PayloadType not equal to 0, 1, or 130:

– nal\_unit\_type is equal to TRAIL\_NUT, STSA\_NUT, RADL\_NUT, or RASL\_NUT, or nal\_unit\_type is equal to GDR\_NUT and the associated ph\_recovery\_poc\_cnt is not equal to 0.

– nuh\_layer\_id is equal to LayerIdInOls[ targetOlsIdx ][ j ] for a value of j in the range of 0 to NumLayersInOls[ targetOlsIdx ] − 1 inclusive.

– TemporalId is greater than or equal to NumSubLayersInLayerInOLS[ targetOlsIdx ][ GeneralLayerIdx[ nuh\_layer\_id ] ].

– Remove from outBitstream all SEI NAL units that contain a scalable nesting SEI message that has nesting\_ols\_flag equal to 1 and there is no value of i in the range of 0 to nesting\_num\_olss\_minus1, inclusive, such that NestingOlsIdx[ i ] is equal to targetOlsIdx.

– When LayerIdInOls[ targetOlsIdx ] does not include all values of nuh\_layer\_id in all NAL units in the bitstream, the following applies:

– Remove from outBitstream all SEI NAL units that contain a non-scalable-nested SEI message with payloadType equal to 0 (buffering period) or 130 (decoding unit information).

– When general\_same\_pic\_timing\_in\_all\_ols\_flag is equal to 0, remove from outBitstream all SEI NAL units that contain a non-scalable-nested SEI message with payloadType equal to 1 (picture timing).

– When outBitstream contains SEI NAL units that contain a scalable nesting SEI message with nesting\_ols\_flag equal to 1 and are applicable to outBitstream (NestingOlsIdx[ i ] is equal to targetOlsIdx), the following applies:

– If general\_same\_pic\_timing\_in\_all\_ols\_flag is equal to 0, extract appropriate non-scalable-nested SEI message with payloadType equal to 0 (buffering period), 1 (picture timing), or 130 (decoding unit information) from the scalable nesting SEI message and include those SEI messages in outBitstream.

– Otherwise (general\_same\_pic\_timing\_in\_all\_ols\_flag is equal to 1), extract appropriate non-scalable-nested SEI message with payloadType equal to 0 (buffering period) or 130 (decoding unit information) from the scalable nesting SEI message and include those SEI messages in outBitstream.

* 1. Subpicture sub-bitstream extraction process

Inputs to this process are a bitstream inBitstream, a target OLS index targetOlsIdx, a target highest TemporalId value tIdTarget, and an array of target subpicture index values for each layer subpicIdxTarget[ ].

Output of this process is a sub-bitstream outBitstream.

It is a requirement of bitstream conformance for the input bitstream that any output sub-bitstream that satisfies all of the following conditions shall be a conforming bitstream:

– The output sub-bitstream is the output of the process specified in this clause with the bitstream, targetOlsIdx equal to an index to the list of OLSs specified by the VPS, and subpicIdxTarget[ ] equal to a subpicture index present in the OLS, as inputs.

– The output sub-bitstream contains at least one VCL NAL unit with nuh\_layer\_id equal to each of the nuh\_layer\_id values in LayerIdInOls[ targetOlsIdx ].

– The output sub-bitstream contains at least one VCL NAL unit with TemporalId equal to tIdTarget.

NOTE – A conforming bitstream contains one or more coded slice NAL units with TemporalId equal to 0, but does not have to contain coded slice NAL units with nuh\_layer\_id equal to 0.

– The output sub-bitstream contains at least one VCL NAL unit with nuh\_layer\_id equal to LayerIdInOls[ targetOlsIdx ][ i ] and with sh\_subpic\_id equal to the value in SubpicIdVal[ subpicIdxTarget[ i ] ] for each i in the range of 0 to NumLayersInOls[ targetOlsIdx ] − 1, inclusive.

The output sub-bitstream outBitstream is derived as follows:

– The sub-bitstream extraction process, specified in Annex C.6, is invoked with inBitstream, targetOlsIdx, and tIdTarget as inputs and the output of the process is assigned to outBitstream.

– If some external means not specified in this Specification is available to provide replacement parameter sets for the sub-bitstream outBitstream, replace all parameter sets with the replacement parameter sets.

– Otherwise, when subpicture level information SEI messages are present in inBitstream, the following applies:

– The variable subpicIdx is set equal to the value of subpicIdxTarget[ [ NumLayersInOls[ targetOlsIdx ] − 1 ] ].

– Rewrite the value of general\_level\_idc in the vps\_ols\_ptl\_idx[ targetOlsIdx ]-th entry in the list of profile\_tier\_level() syntax structures in all the referenced VPS NAL units to be equal to SubpicSetLevelIdc derived in Equation D.11 for the set of subpictures consisting of the subpictures with subpicture index equal to subpicIdx.

– When VCL HRD parameters or NAL HRD parameters are present, rewrite the respective values of cpb\_size\_value\_minus1[ tIdTarget ][ j ] and bit\_rate\_value\_minus1[ tIdTarget ][ j ] of the j-th CPB in the vps\_ols\_hrd\_idx[ MultiLayerOlsIdx[ targetOlsIdx ] ]-th ols\_hrd\_parameters() syntax structure in all the referenced VPS NAL units and in the ols\_hrd\_parameters( ) syntax structures in all SPS NAL units referred to by the i-th layer, such that they correspond to SubpicCpbSizeVcl[ SubpicSetLevelIdx ][ subpicIdx ], and SubpicCpbSizeNal[ SubpicSetLevelIdx ][ subpicIdx ] as derived by Equations D.6 and D.7, respectively, SubpicBitrateVcl[ SubpicSetLevelIdx ][ subpicIdx ] and SubpicBitrateNal[ SubpicSetLevelIdx ][ subpicIdx ] as derived by Equations D.8 and D.9, respectively, where SubpicSetLevelIdx is derived by Equation D.11 for the subpicture with subpicture index equal to subpicIdx, j is in the range of 0 to hrd\_cpb\_cnt\_minus1, inclusive, and i is in the range of 0 to NumLayersInOls[ targetOlsIdx ] − 1, inclusive.

For the i-th layer with i in the range of 0 to NumLayersInOls[ targetOlsIdx ] − 1, the following applies.

– Rewrite the value of general\_level\_idc in the profile\_tier\_level() syntax structure in all the referenced SPS NAL units with sps\_ptl\_dpb\_hrd\_params\_present\_flag equal to 1 to be equal to SubpicSetLevelIdc derived by Equation D.11 for the set of subpictures consisting of the subpicture with subpicture index equal to subpicIdx.

– The variables subpicWidthInLumaSamples and subpicHeightInLumaSamples are derived as follows:

subpicWidthInLumaSamples = min( ( sps\_subpic\_ctu\_top\_left\_x[ subpicIdx ] + (C.24)  
 sps\_subpic\_width\_minus1[ subpicIdx ] + 1 ) \* CtbSizeY, pps\_pic\_width\_in\_luma\_samples ) −  
 sps\_subpic\_ctu\_top\_left\_x[ subpicIdx ] \* CtbSizeY

subpicHeightInLumaSamples = min( ( sps\_subpic\_ctu\_top\_left\_y[ subpicIdx ] + (C.25)  
 sps\_subpic\_height\_minus1[ subpicIdx ] + 1 ) \* CtbSizeY, pps\_pic\_height\_in\_luma\_samples ) −  
 sps\_subpic\_ctu\_top\_left\_y[ subpicIdx ] \* CtbSizeY

– Rewrite the values of the sps\_pic\_width\_max\_in\_luma\_samples and sps\_pic\_height\_max\_in\_luma\_samples in all the referenced SPS NAL units and the values of pps\_pic\_width\_in\_luma\_samples and pps\_pic\_height\_in\_luma\_samples in all the referenced PPS NAL unitsto be equal to subpicWidthInLumaSamples and subpicHeightInLumaSamples, respectively.

– Rewrite the value of sps\_num\_subpics\_minus1 in all the referenced SPS NAL units and pps\_num\_subpics\_minus1 in all the referenced PPS NAL unitsto 0.

– Rewrite the syntax elements sps\_subpic\_ctu\_top\_left\_x[ subpicIdx ] and sps\_subpic\_ctu\_top\_left\_y[ subpicIdx ], when present, in all the referenced SPS NAL units to 0.

– Remove the syntax elements sps\_subpic\_ctu\_top\_left\_x[ j ], sps\_subpic\_ctu\_top\_left\_y[ j ], sps\_subpic\_width\_minus1[ j ], sps\_subpic\_height\_minus1[ j ], sps\_subpic\_treated\_as\_pic\_flag[ j ], sps\_loop\_filter\_across\_subpic\_enabled\_flag[ j ], and sps\_subpic\_id[ j ] in all the referenced SPS NAL units and for each j that is not equal to subpicIdx.

– Rewrite the syntax elements in all the referenced PPS for signalling of tiles and slices to remove all tile rows, tile columns, and slices that are not associated with the subpicture with subpicture index equal to subpicIdx.

– The variables subpicConfWinLeftOffset, subpicConfWinRightOffset, subpicConfWinTopOffset and subpicConfWinBottomOffset are derived as follows:

subpicConfWinLeftOffset = sps\_subpic\_ctu\_top\_left\_x[ subpicIdx ] = = 0 ? (C.26)  
 sps\_conf\_win\_left\_offset : 0

subpicConfWinRightOffset = ( sps\_subpic\_ctu\_top\_left\_x[ subpicIdx ] + (C.27)  
 sps\_subpic\_width\_minus1[ subpicIdx ] + 1 ) \*CtbSizeY >=  
 sps\_pic\_width\_max\_in\_luma\_samples ? sps\_conf\_win\_right\_offset : 0

subpicConfWinTopOffset = sps\_subpic\_ctu\_top\_left\_y[ subpicIdx ] = = 0 ? (C.28)  
 sps\_conf\_win\_top\_offset : 0

subpicConfWinBottomOffset = ( sps\_subpic\_ctu\_top\_left\_y[ subpicIdx ] + (C.29)  
 sps\_subpic\_height\_minus1[ subpicIdx ] + 1 ) \* CtbSizeY >=  
 sps\_pic\_height\_max\_in\_luma\_samples ? sps\_conf\_win\_bottom\_offset : 0

– Rewrite the values of sps\_conf\_win\_left\_offset, sps\_conf\_win\_right\_offset, sps\_conf\_win\_top\_offset, and sps\_conf\_win\_bottom\_offset in all the referenced SPS NAL units and the values of pps\_conf\_win\_left\_offset, pps\_conf\_win\_right\_offset, pps\_conf\_win\_top\_offset, and pps\_conf\_win\_bottom\_offset in all the referenced PPS NAL units to be equal to subpicConfWinLeftOffset, subpicConfWinRightOffset, subpicConfWinTopOffset, and subpicConfWinBottomOffset, respectively.

– Remove from outBitstream all VCL NAL units with nuh\_layer\_id equal to the nuh\_layer\_id of the i-th layer and with sh\_subpic\_id not equal to SubpicIdVal[ subpicIdx ].

– When sli\_cbr\_constraint\_flag is equal to 1, remove all NAL units with nal\_unit\_type equal to FD\_NUT and filler payload SEI messages that are not associated with the VCL NAL units of a subpicture in subpicIdTarget[ ] and set cbr\_flag[ tIdTarget ][ j ] equal to 1 of the j-th CPB in the vps\_ols\_hrd\_idx[ MultiLayerOlsIdx[ targetOlsIdx ] ]-th ols\_hrd\_parameters() syntax structure in all the referenced VPS NAL units and SPS NAL units and j in the range of 0 to hrd\_cpb\_cnt\_minus1. Otherwise, (sli\_cbr\_constraint\_flag is equal to 0), remove all NAL units with nal\_unit\_type equal to FD\_NUT and filler payload SEI messages and set cbr\_flag[ tIdTarget ][ j ] equal to 0.

– When outBitstream contains SEI NAL units that contain a scalable nesting SEI message with nesting\_ols\_flag equal to 1 and nesting\_subpic\_flag equal to 1 that are applicable to outBitstream, extract appropriate non-scalable-nested SEI message with payloadType equal to 1 (picture timing), 130 (decoding unit information), or 132 (decoded picture hash) from the scalable nesting SEI message and place the extracted SEI messages into outBitstream.

1. Annex D  
     
   Supplemental enhancement information

(This annex forms an integral part of this Recommendation | International Standard.)

* 1. General

This annex specifies syntax and semantics for SEI message payloads for some SEI messages, and specifies the use of the SEI messages and VUI parameters for which the syntax and semantics are specified in ITU-T H.SEI | ISO/IEC 23002-7.

SEI messages assist in processes related to decoding, display or other purposes. However, SEI messages are not required for constructing the luma or chroma samples by the decoding process. Conforming decoders are not required to process this information for output order conformance to this Specification (see Annex C for the specification of conformance). Some SEI messages are required for checking bitstream conformance and for output timing decoder conformance. Other SEI messages are not required for check bitstream conformance.

In clause C.5.2, the specification for presence of SEI messages are also satisfied when those messages (or some subset of them) are conveyed to decoders (or to the HRD) by other means not specified in this Specification. When present in the bitstream, SEI messages shall obey the syntax and semantics specified in clause 7.3.6 and this annex or ITU-T H.SEI | ISO/IEC 23002-7. When the content of an SEI message is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the SEI message is not required to use the same syntax specified in this annex. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

* 1. General SEI payload
     1. General SEI message syntax

|  |  |
| --- | --- |
| sei\_payload( payloadType, payloadSize ) { | Descriptor |
| if( nal\_unit\_type = = PREFIX\_SEI\_NUT ) |  |
| if( payloadType = = 0 ) |  |
| buffering\_period( payloadSize ) |  |
| else if( payloadType = = 1 ) |  |
| pic\_timing( payloadSize ) |  |
| else if( payloadType = = 3 ) |  |
| filler\_payload( payloadSize ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| else if( payloadType = = 4 ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| user\_data\_registered\_itu\_t\_t35( payloadSize ) |  |
| else if( payloadType = = 5 ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| user\_data\_unregistered( payloadSize ) |  |
| else if( payloadType = = 19 ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| film\_grain\_characteristics( payloadSize ) |  |
| else if( payloadType = = 45 ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| frame\_packing\_arrangement( payloadSize ) |  |
| else if( payloadType = = 129 ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| parameter\_sets\_inclusion\_indication( payloadSize ) |  |
| else if( payloadType = = 130 ) |  |
| decoding\_unit\_info( payloadSize ) |  |
| else if( payloadType = = 133 ) |  |
| scalable\_nesting( payloadSize ) |  |
| else if( payloadType = = 137 ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| mastering\_display\_colour\_volume( payloadSize ) |  |
| else if( payloadType = = 144 ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| content\_light\_level\_info( payloadSize ) |  |
| else if( payloadType = = 145 ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| dependent\_rap\_indication( payloadSize ) |  |
| else if( payloadType = = 147 ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| alternative\_transfer\_characteristics( payloadSize ) |  |
| else if( payloadType = = 148 ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| ambient\_viewing\_environment( payloadSize ) |  |
| else if( payloadType = = 149 ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| content\_colour\_volume( payloadSize ) |  |
| else if( payloadType = = 150 ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| equirectangular\_projection( payloadSize ) |  |
| else if( payloadType = = 153 ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| generalized\_cubemap\_projection( payloadSize ) |  |
| else if( payloadType = = 154 ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| sphere\_rotation( payloadSize ) |  |
| else if( payloadType = = 155 ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| regionwise\_packing( payloadSize ) |  |
| else if( payloadType = = 156 ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| omni\_viewport( payloadSize ) |  |
| else if( payloadType = = 168 ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| frame\_field\_info( payloadSize ) |  |
| else if( payloadType = = 203 ) |  |
| subpic\_level\_info( payloadSize ) |  |
| else if( payloadType = = 204 ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| sample\_aspect\_ratio\_info( payloadSize ) |  |
| else /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| reserved\_message( payloadSize ) |  |
| else /\* nal\_unit\_type = = SUFFIX\_SEI\_NUT \*/ |  |
| if( payloadType = = 3 ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| filler\_payload( payloadSize ) |  |
| if( payloadType = = 132 ) /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| decoded\_picture\_hash( payloadSize ) |  |
| else if( payloadType = = 133 ) |  |
| scalable\_nesting( payloadSize ) |  |
| else /\* Specified in ITU-T H.SEI | ISO/IEC 23002-7 \*/ |  |
| reserved\_message( payloadSize ) |  |
| if( more\_data\_in\_payload( ) ) { |  |
| if( payload\_extension\_present( ) ) |  |
| **reserved\_payload\_extension\_data** | u(v) |
| **payload\_bit\_equal\_to\_one** /\* equal to 1 \*/ | f(1) |
| while( !byte\_aligned( ) ) |  |
| **payload\_bit\_equal\_to\_zero** /\* equal to 0 \*/ | f(1) |
| } |  |
| } |  |

* + 1. General SEI payload semantics

**reserved\_payload\_extension\_data** shall not be present in bitstreams conforming to this version of this Specification. However, decoders conforming to this version of this Specification shall ignore the presence and value of reserved\_payload\_extension\_data. When present, the length, in bits, of reserved\_payload\_extension\_data is equal to 8 \* payloadSize − nEarlierBits − nPayloadZeroBits − 1, where nEarlierBits is the number of bits in the sei\_payload( ) syntax structure that precede the reserved\_payload\_extension\_data syntax element, and nPayloadZeroBits is the number of payload\_bit\_equal\_to\_zero syntax elements at the end of the sei\_payload( ) syntax structure.

**payload\_bit\_equal\_to\_one** shall be equal to 1.

**payload\_bit\_equal\_to\_zero** shall be equal to 0.

NOTE 1 – SEI messages with the same value of payloadType are conceptually the same SEI message regardless of whether they are contained in prefix or suffix SEI NAL units.

NOTE 2 – For SEI messages with payloadType in the range of 0 to 47, inclusive, that are specified in this Specification, the payloadType values are aligned with similar SEI messages specified in Rec. ITU-T H.264 | ISO/IEC 14496-10.

The semantics and persistence scope for each SEI message are specified in the semantics specification for each particular SEI message.

NOTE 3 – Persistence information for SEI messages is informatively summarized in Table D.1.

Table D.1 – Persistence scope of SEI messages (informative)

|  |  |
| --- | --- |
| SEI message | Persistence scope |
| Buffering period | The remainder of the bitstream |
| Picture timing | The AU containing the SEI message |
| DU information | The AU containing the SEI message |
| Scalable nesting | Depending on the scalable-nested SEI messages. Each scalable-nested SEI message has the same persistence scope as if the SEI message was not scalable-nested |
| Subpicture level information | The CLVS containing the SEI message |

The list VclAssociatedSeiList is set to consist of the payloadType values 3, 19, 45, 129, 132, 137, 144, 145, 147 to 150, inclusive, 153 to 156, inclusive, 168, 203, and 204.

The list PicUnitRepConSeiList is set to consist of the payloadType values 0, 1, 19, 45, 129, 132, 133, 137, 147 to 150, inclusive, 153 to 156, inclusive, 168, 203, and 204.

NOTE 4 – VclAssociatedSeiList consists of the payloadType values of the SEI messages that, when non-scalable-nested and contained in an SEI NAL unit, infer constraints on the NAL unit header of the SEI NAL unit on the basis of the NAL unit header of the associated VCL NAL unit. PicUnitRepConSeiList consists of the payloadType values of the SEI messages that are subject to the restriction on 4 repetitions per PU.

It is a requirement of bitstream conformance that the following restrictions apply on containing of SEI messages in SEI NAL units:

– When an SEI NAL unit contains a non-scalable-nested buffering period SEI message, a non-scalable-nested picture timing SEI message, or a non-scalable-nested decoding unit information SEI message, the SEI NAL unit shall not contain any other SEI message with payloadType not equal to 0 (buffering period), 1 (picture timing), or 130 (decoding unit information).

– When an SEI NAL unit contains a scalable-nested buffering period SEI message, a scalable-nested picture timing SEI message, or a scalable-nested decoding unit information SEI message, the SEI NAL unit shall not contain any other SEI message with payloadType not equal to 0 (buffering period), 1 (picture timing), 130 (decoding unit information) or 133 (scalable nesting).

The following applies on the applicable OLSs or layers of non-scalable-nested SEI messages:

– For a non-scalable-nested SEI message, when payloadType is equal to 0 (buffering period), 1 (picture timing), or 130 (decoding unit information), the non-scalable-nested SEI message applies only to the 0-th OLS.

– For a non-scalable-nested SEI message, when payloadType is equal to any value among VclAssociatedSeiList, the non-scalable-nested SEI message applies only to the layer for which the VCL NAL units have nuh\_layer\_id equal to the nuh\_layer\_id of the SEI NAL unit containing the SEI message.

It is a requirement of bitstream conformance that the following restrictions apply on the value of nuh\_layer\_id of SEI NAL units:

– When a non-scalable-nested SEI message has payloadType equal to 0 (buffering period), 1 (picture timing), or 130 (decoding unit information), the SEI NAL unit containing the non-scalable-nested SEI message shall have nuh\_layer\_id equal to vps\_layer\_id[ 0 ].

– When a non-scalable-nested SEI message has payloadType equal to any value among VclAssociatedSeiList, the SEI NAL unit containing the non-scalable-nested SEI message shall have nuh\_layer\_id equal to the value of nuh\_layer\_id of the VCL NAL unit associated with the SEI NAL unit.

– An SEI NAL unit containing a scalable nesting SEI message shall have nuh\_layer\_id equal to the lowest value of nuh\_layer\_id of all layers to which the scalable-nested SEI messages apply (when nesting\_ols\_flag of the scalable nesting SEI message is equal to 0) or the lowest value of nuh\_layer\_id of all layers in the OLSs to which the scalable-nested SEI message apply (when nesting\_ols\_flag of the scalable nesting SEI message is equal to 1).

It is a requirement of bitstream conformance that the following restrictions apply on repetition of SEI messages:

– For each of the payloadType values included in PicUnitRepConSeiList, there shall be less than or equal to 4 identical sei\_payload( ) syntax structures within a PU.

– There shall be less than or equal to 4 identical sei\_payload( ) syntax structures with payloadType equal to 130 within a DU.

The following applies on the order of BP, PT, and DU information SEI messages:

– When a BP SEI message and a PT SEI message that apply to a particular OP are present within an AU, the BP SEI messages shall precede the PT SEI message in decoding order.

– When a BP SEI message and a DUI SEI message that apply to a particular OP are present within an AU, the BP SEI messages shall precede the DUI SEI message in decoding order.

– When a PT SEI message and a DUI SEI message that apply to a particular OP are present within an AU, the PT SEI messages shall precede the DUI SEI message in decoding order.

* 1. Buffering period SEI message
     1. Buffering period SEI message syntax

|  |  |
| --- | --- |
| buffering\_period( payloadSize ) { | **Descriptor** |
| **bp\_nal\_hrd\_params\_present\_flag** | u(1) |
| **bp\_vcl\_hrd\_params\_present\_flag** | u(1) |
| **initial\_cpb\_removal\_delay\_length\_minus1** | u(5) |
| **cpb\_removal\_delay\_length\_minus1** | u(5) |
| **dpb\_output\_delay\_length\_minus1** | u(5) |
| **alt\_cpb\_params\_present\_flag** | u(1) |
| **bp\_decoding\_unit\_hrd\_params\_present\_flag** | u(1) |
| if( bp\_decoding\_unit\_hrd\_params\_present\_flag ) { |  |
| **du\_cpb\_removal\_delay\_increment\_length\_minus1** | u(5) |
| **dpb\_output\_delay\_du\_length\_minus1** | u(5) |
| **du\_cpb\_params\_in\_pic\_timing\_sei\_flag** | u(1) |
| **du\_dpb\_params\_in\_pic\_timing\_sei\_flag** | u(1) |
| } |  |
| **concatenation\_flag** | u(1) |
| **additional\_concatenation\_info\_present\_flag** | u(1) |
| if( additional\_concatenation\_info\_present\_flag ) |  |
| **max\_initial\_removal\_delay\_for\_concatenation** | u(v) |
| **cpb\_removal\_delay\_delta\_minus1** | u(v) |
| **cpb\_removal\_delay\_deltas\_present\_flag** | u(1) |
| if( cpb\_removal\_delay\_deltas\_present\_flag ) { |  |
| **num\_cpb\_removal\_delay\_deltas\_minus1** | ue(v) |
| for( i = 0; i <= num\_cpb\_removal\_delay\_deltas\_minus1; i++ ) |  |
| **cpb\_removal\_delay\_delta**[ i ] | u(v) |
| } |  |
| **bp\_max\_sublayers\_minus1** | u(3) |
| **bp\_cpb\_cnt\_minus1** | ue(v) |
| **sublayer\_initial\_cpb\_removal\_delay\_present\_flag** | u(1) |
| for( i = ( sublayer\_initial\_cpb\_removal\_delay\_present\_flag ?  0 : bp\_max\_sublayers\_minus1 ); i <= bp\_max\_sublayers\_minus1; i++ ) { |  |
| if( bp\_nal\_hrd\_params\_present\_flag ) |  |
| for( j = 0; j < bp\_cpb\_cnt\_minus1 + 1; j++ ) { |  |
| **nal\_initial\_cpb\_removal\_delay**[ i ][ j ] | u(v) |
| **nal\_initial\_cpb\_removal\_offset**[ i ][ j ] | u(v) |
| if( bp\_decoding\_unit\_hrd\_params\_present\_flag) { |  |
| **nal\_initial\_alt\_cpb\_removal\_delay**[ i ][ j ] | u(v) |
| **nal\_initial\_alt\_cpb\_removal\_offset**[ i ][ j ] | u(v) |
| } |  |
| } |  |
| if( bp\_vcl\_hrd\_params\_present\_flag ) |  |
| for( j = 0; j < bp\_cpb\_cnt\_minus1 + 1; j++ ) { |  |
| **vcl\_initial\_cpb\_removal\_delay**[ i ][ j ] | u(v) |
| **vcl\_initial\_cpb\_removal\_offset**[ i ][ j ] | u(v) |
| if( bp\_decoding\_unit\_hrd\_params\_present\_flag ) { |  |
| **vcl\_initial\_alt\_cpb\_removal\_delay**[ i ][ j ] | u(v) |
| **vcl\_initial\_alt\_cpb\_removal\_offset**[ i **]**[ j ] | u(v) |
| } |  |
| } |  |
| } |  |
| **sublayer\_dpb\_output\_offsets\_present\_flag** | u(1) |
| if( sublayer\_dpb\_output\_offsets\_present\_flag ) |  |
| for( i = 0; i < bp\_max\_sublayers\_minus1; i++ ) |  |
| **dpb\_output\_tid\_offset**[ i ] | ue(v) |
| if( alt\_cpb\_params\_present\_flag ) |  |
| **use\_alt\_cpb\_params\_flag** | u(1) |
| } |  |

* + 1. Buffering period SEI message semantics

A BP SEI message provides initial CPB removal delay and initial CPB removal delay offset information for initialization of the HRD at the position of the associated AU in decoding order.

When the BP SEI message is present, a picture is said to be a notDiscardablePic picture when the picture has TemporalId equal to 0 and is not a RASL or RADL picture.

When the current picture is not the first picture in the bitstream in decoding order, let prevNonDiscardablePic be the preceding picture in decoding order with TemporalId equal to 0 that is not a RASL or RADL picture.

The presence of BP SEI messages is specified as follows:

– If NalHrdBpPresentFlag is equal to 1 or VclHrdBpPresentFlag is equal to 1, the following applies for each AU in the CVS:

– If the AU is an IRAP or GDR AU, a BP SEI message applicable to the operation point shall be associated with the AU.

– Otherwise, if the AU contains a notDiscardablePic, a BP SEI message applicable to the operation point may or may not be associated with the AU.

– Otherwise, the AU shall not be associated with a BP SEI message applicable to the operation point.

– Otherwise (NalHrdBpPresentFlag and VclHrdBpPresentFlag are both equal to 0), no AU in the CVS shall be associated with a BP SEI message.

NOTE 1 – For some applications, frequent presence of BP SEI messages may be desirable (e.g., for random access at an IRAP picture or a non-IRAP picture or for bitstream splicing).

**bp\_nal\_hrd\_params\_present\_flag** equal to 1 specifies that a list of syntax element pairs nal\_initial\_cpb\_removal\_delay[ i ][ j ] and nal\_initial\_cpb\_removal\_offset[ i ][ j ] are present in the BP SEI message. bp\_nal\_hrd\_params\_present\_flag equal to 0 specifies that no syntax element pairs nal\_initial\_cpb\_removal\_delay[ i ][ j ] and nal\_initial\_cpb\_removal\_offset[ i ][ j ] are present in the BP SEI message.

The value of bp\_nal\_hrd\_params\_present\_flag shall be equal to general\_nal\_hrd\_params\_present\_flag.

**bp\_vcl\_hrd\_params\_present\_flag** equal to 1 specifies that a list of syntax element pairs vcl\_initial\_cpb\_removal\_delay[ i ][ j ] and vcl\_initial\_cpb\_removal\_offset[ i ][ j ] are present in the BP SEI message. bp\_vcl\_hrd\_params\_present\_flag equal to 0 specifies that no syntax element pairs vcl\_initial\_cpb\_removal\_delay[ i ][ j ] and vcl\_initial\_cpb\_removal\_offset[ i ][ j ] are present in the BP SEI message.

The value of bp\_vcl\_hrd\_params\_present\_flag shall be equal to general\_vcl\_hrd\_params\_present\_flag.

bp\_vcl\_hrd\_params\_present\_flag and bp\_nal\_hrd\_params\_present\_flag in a BP SEI message shall not be both equal to 0.

**initial\_cpb\_removal\_delay\_length\_minus1** plus 1 specifies the length, in bits, of the syntax elements nal\_initial\_cpb\_removal\_delay[ i ][ j ], nal\_initial\_cpb\_removal\_offset[ i ][ j ], vcl\_initial\_cpb\_removal\_delay[ i ][ j ], and vcl\_initial\_cpb\_removal\_offset[ i ][ j ] of the BP SEI messages, and the syntax elements cpb\_alt\_initial\_removal\_delay\_delta[ i ][ j ] and cpb\_alt\_initial\_removal\_offset\_delta[ i ][ j ] in the PT SEI messages in the current buffering period. When not present, the value of initial\_cpb\_removal\_delay\_length\_minus1 is inferred to be equal to 23.

**cpb\_removal\_delay\_length\_minus1** plus 1 specifies the length, in bits, of the syntax elements cpb\_removal\_delay\_delta\_minus1 and cpb\_removal\_delay\_delta[ i ] in the BP SEI message and the syntax elements cpb\_removal\_delay\_minus1[ i ] and cpb\_delay\_offset[ i ] in the PT SEI messages in the current buffering period. When not present, the value of cpb\_removal\_delay\_length\_minus1 is inferred to be equal to 23.

**dpb\_output\_delay\_length\_minus1** plus 1 specifies the length, in bits, of the syntax elements dpb\_output\_delay and cpb\_delay\_offset[ i ] in the PT SEI messages in the current buffering period. When not present, the value of dpb\_output\_delay\_length\_minus1 is inferred to be equal to 23.

**alt\_cpb\_params\_present\_flag** equal to 1 specifies the presence of the syntax element use\_alt\_cpb\_params\_flag in the BP SEI message and the presence of the alternative timing information in the PT SEI messages in the current buffering period. When not present, the value of bp\_alt\_cpb\_params\_present\_flag is inferred to be equal to 0. When the associated picture is neither a CRA picture nor an IDR picture, the value of bp\_alt\_cpb\_params\_present\_flag shall be equal to 0.

**bp\_decoding\_unit\_hrd\_params\_present\_flag** equal to 1 specifies that DU level HRD parameters are present and the HRD may operate at AU level or DU level. bp\_decoding\_unit\_hrd\_params\_present\_flag equal to 0 specifies that DU level HRD parameters are not present and the HRD operates at AU level. When bp\_decoding\_unit\_hrd\_params\_present\_flag is not present, its value is inferred to be equal to 0.

The value of bp\_decoding\_unit\_hrd\_params\_present\_flag shall be equal to general\_decoding\_unit\_hrd\_params\_present\_flag.

**du\_cpb\_removal\_delay\_increment\_length\_minus1** plus 1 specifies the length, in bits, of the du\_cpb\_removal\_delay\_increment\_minus1[ ][ ] and du\_common\_cpb\_removal\_delay\_increment\_minus1[ ] syntax elements of the PT SEI messages in the current buffering period and the du\_spt\_cpb\_removal\_delay\_increment[ ] syntax element in the DU information SEI messages in the current buffering period. When not present, the value of du\_cpb\_removal\_delay\_increment\_length\_minus1 is inferred to be equal to 23.

**dpb\_output\_delay\_du\_length\_minus1** plus 1 specifies the length, in bits, of the pic\_dpb\_output\_du\_delay syntax element in the PT SEI messages in the current buffering period and the pic\_spt\_dpb\_output\_du\_delay syntax element in the DU information SEI messages in the current buffering period. When not present, the value of dpb\_output\_delay\_du\_length\_minus1 is inferred to be equal to 23.

**du\_cpb\_params\_in\_pic\_timing\_sei\_flag** equal to 1 specifies that DU level CPB removal delay parameters are present in PT SEI messages and no DU information SEI message is available (in the CVS or provided through external means not specified in this Specification). du\_cpb\_params\_in\_pic\_timing\_sei\_flag equal to 0 specifies that DU level CPB removal delay parameters are present in DU information SEI messages and PT SEI messages do not include DU level CPB removal delay parameters. When the du\_cpb\_params\_in\_pic\_timing\_sei\_flag syntax element is not present, it is inferred to be equal to 0.

**du\_dpb\_params\_in\_pic\_timing\_sei\_flag** equal to 1 specifies that DU level DPB output delay parameters are present in PT SEI messages and not in DU information SEI messages. du\_dpb\_params\_in\_pic\_timing\_sei\_flag equal to 0 specifies that DU level DPB output delay parameters are present in DU information SEI messages and not in PT SEI messages. When the du\_dpb\_params\_in\_pic\_timing\_sei\_flag syntax element is not present, it is inferred to be equal to 0.

**concatenation\_flag** indicates, when the current picture is not the first picture in the bitstream in decoding order, whether the nominal CPB removal time of the current picture is determined relative to the nominal CPB removal time of the preceding picture with a BP SEI message or relative to the nominal CPB removal time of the picture prevNonDiscardablePic.

**additional\_concatenation\_info\_present\_flag** equal to 1 specifies that the syntax element max\_initial\_removal\_delay\_for\_concatenation is present in the BP SEI message and the syntax element delay\_for\_concatenation\_ensured\_flag is present in the PT SEI messages. additional\_concatenation\_info\_present\_flag equal to 0 specifies that the syntax element max\_initial\_removal\_delay\_for\_concatenation is not present in the BP SEI message and the syntax element delay\_for\_concatenation\_ensured\_flag is not present in the PT SEI messages.

**max\_initial\_removal\_delay\_for\_concatenation** may be used together with delay\_for\_concatenation\_ensured\_flag in a PT SEI message to identify whether the nominal removal time from the CPB of the first AU of a following BP computed with cpb\_removal\_delay\_delta\_minus1 applies. The length of max\_initial\_removal\_delay\_for\_concatenation is initial\_cpb\_removal\_delay\_length\_minus1 + 1 bits.

**cpb\_removal\_delay\_delta\_minus1** plus 1, when the current picture is not the first picture in the bitstream in decoding order, specifies a CPB removal delay increment value relative to the nominal CPB removal time of the picture prevNonDiscardablePic. The length of this syntax element is cpb\_removal\_delay\_length\_minus1 + 1 bits.

When the current picture contains a BP SEI message and concatenation\_flag is equal to 0 and the current picture is not the first picture in the bitstream in decoding order, it is a requirement of bitstream conformance that the following constraint applies:

– If the picture prevNonDiscardablePic is not associated with a BP SEI message, the cpb\_removal\_delay\_minus1 of the current picture shall be equal to the cpb\_removal\_delay\_minus1 of prevNonDiscardablePic plus cpb\_removal\_delay\_delta\_minus1 + 1.

– Otherwise, cpb\_removal\_delay\_minus1 shall be equal to cpb\_removal\_delay\_delta\_minus1.

NOTE 2 – When the current picture contains a BP SEI message and concatenation\_flag is equal to 1, the cpb\_removal\_delay\_minus1 for the current picture is not used. The above-specified constraint can, under some circumstances, make it possible to splice bitstreams (that use suitably-designed referencing structures) by simply changing the value of concatenation\_flag from 0 to 1 in the BP SEI message for an IRAP or GDR picture at the splicing point. When concatenation\_flag is equal to 0, the above-specified constraint enables the decoder to check whether the constraint is satisfied as a way to detect the loss of the picture prevNonDiscardablePic.

**cpb\_removal\_delay\_deltas\_present\_flag** equal to 1 specifies that the BP SEI message contains CPB removal delay deltas. cpb\_removal\_delay\_deltas\_present\_flag equal to 0 specifies that no CPB removal delay deltas are present in the BP SEI message.

**num\_cpb\_removal\_delay\_deltas\_minus1** plus 1 specifies the number of syntax elements cpb\_removal\_delay\_delta[ i ] in the BP SEI message. The value of num\_cpb\_removal\_offsets\_minus1shall be in the range of 0 to 15, inclusive.

**cpb\_removal\_delay\_delta**[ i ] specifies the i-th CPB removal delay delta. The length of this syntax element is cpb\_removal\_delay\_length\_minus1 + 1 bits.

**bp\_max\_sublayers\_minus1** plus 1 specifies the maximum number of temporal sublayers for which CPB removal delay and CPB removal offset are indicated in the BP SEI message. The value of bp\_max\_sublayers\_minus1 shall be in the range of 0 to vps\_max\_sublayers\_minus1, inclusive.

**bp\_cpb\_cnt\_minus1** plus 1 specifies the number of syntax element pairs nal\_initial\_cpb\_removal\_delay[ i ][ j ] and nal\_initial\_cpb\_removal\_offset[ i ][ j ] of the i-th temporal sublayer when bp\_nal\_hrd\_params\_present\_flag is equal to 1, and the number of syntax element pairs vcl\_initial\_cpb\_removal\_delay[ i ][ j ] and vcl\_initial\_cpb\_removal\_offset[ i ][ j ] of the i-th temporal sublayer when bp\_vcl\_hrd\_params\_present\_flag is equal to 1. The value of bp\_cpb\_cnt\_minus1 shall be in the range of 0 to 31, inclusive.

The value of bp\_cpb\_cnt\_minus1 shall be equal to the value of hrd\_cpb\_cnt\_minus1.

**sublayer\_initial\_cpb\_removal\_delay\_present\_flag** equal to 1 specifies that initial CPB removal delay related syntax elements are present for temporal sublayer representation(s) in the range of 0 to bp\_max\_sublayers\_minus1, inclusive. sublayer\_initial\_cpb\_removal\_delay\_present\_flag equal to 0 specifies that initial CPB removal delay related syntax elements are present for the bp\_max\_sublayers\_minus1-th temporal sublayer representation. [Ed. (YK): Add a definition of "temporal sublayer representation" or avoid using the term in the spec.]

**nal\_initial\_cpb\_removal\_delay**[ i ][ j ] and **nal\_initial\_alt\_cpb\_removal\_delay**[ i ][ j ] specify the j-th default and alternative initial CPB removal delay for the NAL HRD in units of a 90 kHz clock of the i-th temporal sublayer. The length of nal\_initial\_cpb\_removal\_delay[ i ][ j ] and nal\_initial\_alt\_cpb\_removal\_delay[ i ][ j ] is initial\_cpb\_removal\_delay\_length\_minus1 + 1 bits. The value of nal\_initial\_cpb\_removal\_delay[ i ][ j ] and nal\_initial\_alt\_cpb\_removal\_delay[ i ][ j ] shall not be equal to 0 and shall be less than or equal to 90000 \* ( CpbSize[ i ][ j ] ÷ BitRate[ i ][ j ] ), the time-equivalent of the CPB size in 90 kHz clock units. When not present, the values of nal\_initial\_cpb\_removal\_delay[ i ][ j ] and nal\_initial\_alt\_cpb\_removal\_delay[ i ][ j ] are inferred to be equal to 90000 \* ( CpbSize[ i ][ j ] ÷ BitRate[ i ][ j ] ). [Ed. (YK): Add a clarification of CpbSize[ i ][ j ] ÷ BitRate[ i ][ j ] at the beginning of this clause.]

**nal\_initial\_cpb\_removal\_offset**[ i ][ j ] and **nal\_initial\_alt\_cpb\_removal\_offset**[ i ][ j ] specify the j-th default and alternative initial CPB removal offset of the i-th temporal sublayer for the NAL HRD in units of a 90 kHz clock. The length of nal\_initial\_cpb\_removal\_offset[ i ][ j ] and nal\_initial\_alt\_cpb\_removal\_offset[ i ][ j ] is initial\_cpb\_removal\_delay\_length\_minus1 + 1 bits. When not present, the values of nal\_initial\_cpb\_removal\_offset[ i ][ j ] and nal\_initial\_alt\_cpb\_removal\_offset[ i ][ j ] are inferred to be equal to 0.

Over the entire CVS, for each value pair of i and j, the sum of nal\_initial\_cpb\_removal\_delay[ i ][ j ] and nal\_initial\_cpb\_removal\_offset[ i ][ j ] shall be constant, and the sum of nal\_initial\_alt\_cpb\_removal\_delay[ i ][ j ] and nal\_initial\_alt\_cpb\_removal\_offset[ i ][ j ] shall be constant.

**vcl\_initial\_cpb\_removal\_delay**[ i ][ j ] and **vcl\_initial\_alt\_cpb\_removal\_delay**[ i ][ j ] specify the j-th default and alternative initial CPB removal delay of the i-th temporal sublayer for the VCL HRD in units of a 90 kHz clock. The length of vcl\_initial\_cpb\_removal\_delay[ i ][ j ] and vcl\_initial\_alt\_cpb\_removal\_delay[ i ][ j ] is initial\_cpb\_removal\_delay\_length\_minus1 + 1 bits. The value of vcl\_initial\_cpb\_removal\_delay[ i ][ j ] and vcl\_initial\_alt\_cpb\_removal\_delay[ i ][ j ] shall not be equal to 0 and shall be less than or equal to 90000 \* ( CpbSize[ i ][ j ] ÷ BitRate[ i ][ j ] ), the time-equivalent of the CPB size in 90 kHz clock units. When not present, the values of vcl\_initial\_cpb\_removal\_delay[ i ][ j ] and vcl\_initial\_alt\_cpb\_removal\_delay[ i ][ j ] are inferred to be equal to 90000 \* ( CpbSize[ i ][ j ] ÷ BitRate[ i ][ j ] ).

**vcl\_initial\_cpb\_removal\_offset**[ i ][ j ] and **vcl\_initial\_alt\_cpb\_removal\_offset**[ i ][ j ] specify the j-th default and alternative initial CPB removal offset of the i-th temporal sublayer for the VCL HRD in units of a 90 kHz clock. The length of vcl\_initial\_cpb\_removal\_offset[ i ] and vcl\_initial\_alt\_cpb\_removal\_offset[ i ][ j ] is initial\_cpb\_removal\_delay\_length\_minus1 + 1 bits. When not present, the values of vcl\_initial\_cpb\_removal\_offset[ i ][ j ] and vcl\_initial\_alt\_cpb\_removal\_offset[ i ][ j ] are inferred to be equal to 0.

Over the entire CVS, for each value pair of i and j the sum of vcl\_initial\_cpb\_removal\_delay[ i ][ j ] and vcl\_initial\_cpb\_removal\_offset[ i ][ j ] shall be constant, and the sum of vcl\_initial\_alt\_cpb\_removal\_delay[ i ][ j ] and vcl\_initial\_alt\_cpb\_removal\_offset[ i ][ j ] shall be constant.

**sublayer\_dpb\_output\_offsets\_present\_flag** equal to 1 specifies that DPB output time offsets are present for temporal sublayer representation(s) in the range of 0 to bp\_max\_sublayers\_minus1 − 1, inclusive. sublayer\_dpb\_output\_offsets\_present\_flag equal to 0 specified that no such DPB output time offsets are present.

**dpb\_output\_tid\_offset**[ i ] specifies the difference between the DPB output times for the i-th temporal sublayer representation and the bp\_max\_sublayers\_minus1-th temporal sublayer representation. When dpb\_output\_tid\_offset[ i ] is not present, it is inferred to be equal to 0.

**use\_alt\_cpb\_params\_flag** may be used to derive the value of UseAltCpbParamsFlag. When use\_alt\_cpb\_params\_flag is not present, it is inferred to be equal to 0.

When one or more of the following condtions apply, UseAltCpbParamsFlag is set equal to 1:

– use\_alt\_cpb\_params\_flag is equal to 1.

– When some external means not specified in this Specification is available to set UseAltCpbParamsFlag and the value of UseAltCpbParamsFlag is set equal to 1 by the external means.

* 1. Picture timing SEI message
     1. Picture timing SEI message syntax

|  |  |
| --- | --- |
| pic\_timing( payloadSize ) { | Descriptor |
| **cpb\_removal\_delay\_minus1**[ bp\_max\_sublayers\_minus1 ] | u(v) |
| if( alt\_cpb\_params\_present\_flag ) { |  |
| **cpb\_alt\_timing\_info\_present\_flag** | u(1) |
| if( cpb\_alt\_timing\_info\_present\_flag ) { |  |
| if( bp\_nal\_hrd\_params\_present\_flag ) { |  |
| for( i = ( sublayer\_initial\_cpb\_removal\_delay\_present\_flag ? 0 :  bp\_max\_sublayers\_minus1 ); i <= bp\_max\_sublayers\_minus1; i++ ) { |  |
| for( j = 0; j < bp\_cpb\_cnt\_minus1 + 1; j++ ) { |  |
| **nal\_cpb\_alt\_initial\_removal\_delay\_delta**[ i ][ j ] | u(v) |
| **nal\_cpb\_alt\_initial\_removal\_offset\_delta**[ i ][ j ] | u(v) |
| } |  |
| **nal\_cpb\_delay\_offset**[ i ] | u(v) |
| **nal\_dpb\_delay\_offset**[ i ] | u(v) |
| } |  |
| } |  |
| if( bp\_vcl\_hrd\_params\_present\_flag ) { |  |
| for( i = ( sublayer\_initial\_cpb\_removal\_delay\_present\_flag ? 0 :  bp\_max\_sublayers\_minus1 ); i <= bp\_max\_sublayers\_minus1; i++ ) { |  |
| for( j = 0; j < bp\_cpb\_cnt\_minus1 + 1; j++ ) { |  |
| **vcl\_cpb\_alt\_initial\_removal\_delay\_delta**[ i ][ j ] | u(v) |
| **vcl\_cpb\_alt\_initial\_removal\_offset\_delta**[ i ][ j ] | u(v) |
| } |  |
| **vcl\_cpb\_delay\_offset**[ i ] | u(v) |
| **vcl\_dpb\_delay\_offset**[ i ] | u(v) |
| } |  |
| } |  |
| } |  |
| } |  |
| for( i = TemporalId; i < bp\_max\_sublayers\_minus1; i++ ) { |  |
| **pt\_sublayer\_delays\_present\_flag**[ i ] | u(1) |
| if( pt\_sublayer\_delays\_present\_flag[ i ] ) { |  |
| if( cpb\_removal\_delay\_deltas\_present\_flag ) |  |
| **cpb\_removal\_delay\_delta\_enabled\_flag**[ i ] | u(1) |
| if( cpb\_removal\_delay\_delta\_enabled\_flag[ i ] ) |  |
| if( num\_cpb\_removal\_delay\_deltas\_minus1 > 0 ) |  |
| **cpb\_removal\_delay\_delta\_idx**[ i ] | u(v) |
| else |  |
| **cpb\_removal\_delay\_minus1**[ i ] | u(v) |
| } |  |
| } |  |
| **dpb\_output\_delay** | u(v) |
| if( bp\_decoding\_unit\_hrd\_params\_present\_flag &&  du\_dpb\_params\_in\_pic\_timing\_sei\_flag ) |  |
| **pic\_dpb\_output\_du\_delay** | u(v) |
| if( bp\_decoding\_unit\_hrd\_params\_present\_flag &&  du\_cpb\_params\_in\_pic\_timing\_sei\_flag ) { |  |
| **num\_decoding\_units\_minus1** | ue(v) |
| if( num\_decoding\_units\_minus1 > 0 ) { |  |
| **du\_common\_cpb\_removal\_delay\_flag** | u(1) |
| if( du\_common\_cpb\_removal\_delay\_flag ) |  |
| for( i = TemporalId; i <= bp\_max\_sublayers\_minus1; i++ ) |  |
| if( pt\_sublayer\_delays\_present\_flag[ i ] ) |  |
| **du\_common\_cpb\_removal\_delay\_increment\_minus1**[ i ] | u(v) |
| for( i = 0; i <= num\_decoding\_units\_minus1; i++ ) { |  |
| **num\_nalus\_in\_du\_minus1**[ i ] | ue(v) |
| if( !du\_common\_cpb\_removal\_delay\_flag && i < num\_decoding\_units\_minus1 ) |  |
| for( j = TemporalId; j <= bp\_max\_sublayers\_minus1; j++ ) |  |
| if( pt\_sublayer\_delays\_present\_flag[ j ] ) |  |
| **du\_cpb\_removal\_delay\_increment\_minus1**[ i ][ j ] | u(v) |
| } |  |
| } |  |
| } |  |
| if( additional\_concatenation\_info\_present\_flag ) |  |
| **delay\_for\_concatenation\_ensured\_flag** | u(1) |
| **pt\_display\_elemental\_periods\_minus1** | u(4) |
| } |  |

* + 1. Picture timing SEI message semantics

The PT SEI message provides CPB removal delay and DPB output delay information for the AU associated with the SEI message.

If bp\_nal\_hrd\_params\_present\_flag or bp\_vcl\_hrd\_params\_present\_flag of the BP SEI mesage applicable for the current AU is equal to 1, the variable CpbDpbDelaysPresentFlag is set equal to 1. Otherwise, CpbDpbDelaysPresentFlag is set equal to 0.

The presence of PT SEI messages is specified as follows:

– If CpbDpbDelaysPresentFlag is equal to 1, a PT SEI message shall be associated with the current AU.

– Otherwise (CpbDpbDelaysPresentFlag is equal to 0), there shall not be a PT SEI message associated with the current AU.

The TemporalId in the PT SEI message syntax is the TemporalId of the SEI NAL unit containing the PT SEI message.

**cpb\_removal\_delay\_minus1**[ i ] plus 1 is used to calculate the number of clock ticks between the nominal CPB removal times of the AU associated with the PT SEI message and the preceding AU in decoding order that contains a BP SEI message when Htid is equal to i. This value is also used to calculate an earliest possible time of arrival of AU data into the CPB for the HSS. The length of cpb\_removal\_delay\_minus1[ i ] is cpb\_removal\_delay\_length\_minus1 + 1 bits.

**cpb\_alt\_timing\_info\_present\_flag** equal to 1 specifies that the syntax elements nal\_cpb\_alt\_initial\_removal\_delay\_delta[ i ][ j ], nal\_cpb\_alt\_initial\_removal\_offset\_delta[ i ][ j ], nal\_cpb\_delay\_offset[ i ], nal\_dpb\_delay\_offset[ i ], vcl\_cpb\_alt\_initial\_removal\_delay\_delta[ i ][ j ], vcl\_cpb\_alt\_initial\_removal\_offset\_delta[ i ][ j ], vcl\_cpb\_delay\_offset[ i ], and vcl\_dpb\_delay\_offset[ i ] may be present in the PT SEI message. cpb\_alt\_timing\_info\_present\_flag equal to 1 specifies that these syntax elements are not present in the PT SEI message. When the associated picture is a RASL picture, the value of cpb\_alt\_timing\_info\_present\_flag shall be equal to 0.

NOTE 1 – The value of cpb\_alt\_timing\_info\_present\_flag might be equal to 1 for more than one AU following an IRAP picture in decoding order. However, the alternative timing is only applied to the first AU that has cpb\_alt\_timing\_info\_present\_flag equal to 1 and follows the IRAP picture in decoding order.

**nal\_cpb\_alt\_initial\_removal\_delay\_delta**[ i ][ j ] specifies the alternative initial CPB removal delay delta for the i-th sublayer for the j-th CPB for the NAL HRD in units of a 90 kHz clock. The length of nal\_cpb\_alt\_initial\_removal\_delay\_delta[ i ][ j ] is initial\_cpb\_removal\_delay\_length\_minus1 + 1 bits.

When cpb\_alt\_timing\_info\_present\_flag is equal to 1 and nal\_cpb\_alt\_initial\_removal\_delay\_delta[ i ][ j ] is not present for any value of i less than bp\_max\_sublayers\_minus1, its value is inferred to be equal to 0.

**nal\_cpb\_alt\_initial\_removal\_offset\_delta**[ i ][ j ] specifies the alternative initial CPB removal offset delta for the i-th sublayer for the j-th CPB for the NAL HRD in units of a 90 kHz clock. The length of nal\_cpb\_alt\_initial\_removal\_offset\_delta[ i ][ j ] is initial\_cpb\_removal\_delay\_length\_minus1 + 1 bits.

When cpb\_alt\_timing\_info\_present\_flag is equal to 1 and nal\_cpb\_alt\_initial\_removal\_offset\_delta[ i ][ j ] is not present for any value of i less than bp\_max\_sublayers\_minus1, its value is inferred to be equal to 0.

**nal\_cpb\_delay\_offset**[ i ] specifies, for the i-th sublayer for the NAL HRD, an offset to be used in the derivation of the nominal CPB removal times of the AU associated with the PT SEI message and of the AUs following in decoding order, when the AU associated with the PT SEI message directly follows in decoding order the AU associated with the BP SEI message. The length of nal\_cpb\_delay\_offset[ i ] is cpb\_removal\_delay\_length\_minus1 + 1 bits. When not present, the value of nal\_cpb\_delay\_offset[ i ] is inferred to be equal to 0.

**nal\_dpb\_delay\_offset**[ i ] specifies, for the i-th sublayer for the NAL HRD, an offset to be used in the derivation of the DPB output times of the IRAP AU associated with the BP SEI message when the AU associated with the PT SEI message directly follows in decoding order the IRAP AU associated with the BP SEI message. The length of nal\_dpb\_delay\_offset[ i ] is dpb\_output\_delay\_length\_minus1 + 1 bits. When not present, the value of nal\_dpb\_delay\_offset[ i ] is inferred to be equal to 0.

**vcl\_cpb\_alt\_initial\_removal\_delay\_delta**[ i ][ j ] specifies the alternative initial CPB removal delay delta for the i-th sublayer for the j-th CPB for the VCL HRD in units of a 90 kHz clock. The length of vcl\_cpb\_alt\_initial\_removal\_delay\_delta[ i ][ j ] is initial\_cpb\_removal\_delay\_length\_minus1 + 1 bits.

When cpb\_alt\_timing\_info\_present\_flag is equal to 1 and vcl\_cpb\_alt\_initial\_removal\_delay\_delta[ i ][ j ] is not present for any value of i less than bp\_max\_sublayers\_minus1, its value is inferred to be equal to 0.

**vcl\_cpb\_alt\_initial\_removal\_offset\_delta**[ i ][ j ] specifies the alternative initial CPB removal offset delta for the i-th sublayer for the j-th CPB for the VCL HRD in units of a 90 kHz clock. The length of vcl\_cpb\_alt\_initial\_removal\_offset\_delta[ i ][ j ] is initial\_cpb\_removal\_delay\_length\_minus1 + 1 bits.

When cpb\_alt\_timing\_info\_present\_flag is equal to 1 and vcl\_cpb\_alt\_initial\_removal\_offset\_delta[ i ][ j ] is not present for any value of i less than bp\_max\_sublayers\_minus1, its value is inferred to be equal to 0.

**vcl\_cpb\_delay\_offset**[ i ] specifies, for the i-th sublayer for the VCL HRD, an offset to be used in the derivation of the nominal CPB removal times of the AU associated with the PT SEI message and of the AUs following in decoding order, when the AU associated with the PT SEI message directly follows in decoding order the AU associated with the BP SEI message. The length of vcl\_cpb\_delay\_offset[ i ] is cpb\_removal\_delay\_length\_minus1 + 1 bits. When not present, the value of vcl\_cpb\_delay\_offset[ i ] is inferred to be equal to 0.

**vcl\_dpb\_delay\_offset**[ i ] specifies, for the i-th sublayer for the VCL HRD, an offset to be used in the derivation of the DPB output times of the IRAP AU associated with the BP SEI message when the AU associated with the PT SEI message directly follows in decoding order the IRAP AU associated with the BP SEI message. The length of vcl\_dpb\_delay\_offset[ i ] is dpb\_output\_delay\_length\_minus1 + 1 bits. When not present, the value of vcl\_dpb\_delay\_offset[ i ] is inferred to be equal to 0.

The variable BpResetFlag of the current picture is derived as follows:

– If the current picture is associated with a BP SEI message, BpResetFlag is set equal to 1.

– Otherwise, BpResetFlag is set equal to 0.

**pt\_sublayer\_delays\_present\_flag**[ i ] equal to 1 specifies that cpb\_removal\_delay\_delta\_idx[ i ] or cpb\_removal\_delay\_minus1[ i ], and du\_common\_cpb\_removal\_delay\_increment\_minus1[ i ] or du\_cpb\_removal\_delay\_increment\_minus1[ ][ ] are present for the sublayer with TemporalId equal to i. sublayer\_delays\_present\_flag[ i ] equal to 0 specifies that neither cpb\_removal\_delay\_delta\_idx[ i ] nor cpb\_removal\_delay\_minus1[ i ] and neither du\_common\_cpb\_removal\_delay\_increment\_minus1[ i ] nor du\_cpb\_removal\_delay\_increment\_minus1[ ][ ] are present for the sublayer with TemporalId equal to i. The value of pt\_sublayer\_delays\_present\_flag[ bp\_max\_sublayers\_minus1 ] is infered to be equal to 1. When not present, the value of pt\_sublayer\_delays\_present\_flag[ i ] for any i in the range of 0 to bp\_max\_sublayers\_minus1 − 1, inclusive, is infered to be equal to 0.

**cpb\_removal\_delay\_delta\_enabled\_flag**[ i ] equal to 1 specifies that cpb\_removal\_delay\_delta\_idx[ i ] is present in the PT SEI message. cpb\_removal\_delay\_delta\_enabled\_flag[ i ] equal to 0 specifies that cpb\_removal\_delay\_delta\_idx[ i ] is not present in the PT SEI message. When not present, the value of cpb\_removal\_delay\_delta\_enabled\_flag[ i ] is infered to be equal to 0.

**cpb\_removal\_delay\_delta\_idx**[ i ] specifies the index of the CPB removal delta that applies to Htid equal to i in the list of cpb\_removal\_delay\_delta[ j ] for j ranging from 0 to num\_cpb\_removal\_delay\_deltas\_minus1, inclusive. The length of cpb\_removal\_delay\_delta\_idx[ i ] is Ceil( Log2( num\_cpb\_removal\_delay\_deltas\_minus1 + 1 ) ) bits. When cpb\_removal\_delay\_delta\_idx[ i ] is not present and cpb\_removal\_delay\_delta\_enabled\_flag[ i ] is equal to 1, the value of cpb\_removal\_delay\_delta\_idx[ i ] is inferred to be equal to 0.

The variables CpbRemovalDelayMsb[ i ] and CpbRemovalDelayVal[ i ] of the current picture are derived as follows:

– If the current AU is the AU that initializes the HRD, CpbRemovalDelayMsb[ i ] and CpbRemovalDelayVal[ i ] are both set equal to 0, and the value of cpbRemovalDelayValTmp[ i ] is set equal to cpb\_removal\_delay\_minus1[ i ] + 1.

– Otherwise, let the picture prevNonDiscardablePic be the previous picture in decoding order that has TemporalId equal to 0 that is not a RASL or RADL, let prevCpbRemovalDelayMinus1[ i ], prevCpbRemovalDelayMsb[ i ], and prevBpResetFlag be set equal to the values of cpbRemovalDelayValTmp[ i ] − 1, CpbRemovalDelayMsb[ i ], and BpResetFlag, respectively, for the picture prevNonDiscardablePic, and the following applies:

– CpbRemovalDelayMsb[ i ] is derived as follows:

cpbRemovalDelayValTmp[ i ] = cpb\_removal\_delay\_delta\_enabled\_flag[ i ] ?  
 cpb\_removal\_delay\_minus1[ bp\_max\_sublayers\_minus1 ] + 1 +  
 cpb\_removal\_delay\_delta[ cpb\_removal\_delay\_delta\_idx[ i ] ] : cpb\_removal\_delay\_minus1[ i ] + 1  
if( prevBpResetFlag )  
 CpbRemovalDelayMsb[ i ] = 0  
else if( cpbRemovalDelayValTmp[ i ] < prevCpbRemovalDelayMinus1[ i ] )  
 CpbRemovalDelayMsb[ i ] = prevCpbRemovalDelayMsb[ i ] + 2cpb\_removal\_delay\_length\_minus1 + 1 (D.1)  
else  
 CpbRemovalDelayMsb[ i ] = prevCpbRemovalDelayMsb[ i ]

– CpbRemovalDelayVal is derived as follows:

if( pt\_sublayer\_delays\_present\_flag[ i ] )  
 CpbRemovalDelayVal[ i ] = CpbRemovalDelayMsb[ i ] + cpbRemovalDelayValTmp[ i ] (D.2)  
else  
 CpbRemovalDelayVal[ i ] = CpbRemovalDelayVal[ i + 1 ]

The value of CpbRemovalDelayVal[ i ] shall be in the range of 1 to 232, inclusive.

The variable picDpbOutputDelta[ i ] is derived as follows:

picDpbOutputDelta[ i ] = CpbRemovalDelayVal[ i ] −  
 ( cpb\_removal\_delay\_minus1[ bp\_max\_sublayers\_minus1 ] + 1 ) − (D.3)  
 ( i = = bp\_max\_sublayers\_minus1 ? 0 : dpb\_output\_tid\_offset[ i ] )

Where the value of dpb\_output\_tid\_offset[ i ] is found in the associated BP SEI message.

**dpb\_output\_delay** is used to compute the DPB output time of the picture. It specifies how many clock ticks to wait after removal of an AU from the CPB before the decoded picture is output from the DPB.

NOTE 2 – A picture is not removed from the DPB at its output time when it is still marked as "used for short-term reference" or "used for long-term reference".

The length of dpb\_output\_delay is dpb\_output\_delay\_length\_minus1 + 1 bits. When max\_dec\_pic\_buffering\_minus1[ Htid ] is equal to 0, the value of dpb\_output\_delay shall be equal to 0.

The output time derived from the dpb\_output\_delay of any picture that is output from an output timing conforming decoder shall precede the output time derived from the dpb\_output\_delay of all pictures in any subsequent CVS in decoding order.

The picture output order established by the values of this syntax element shall be the same order as established by the values of PicOrderCntVal.

For pictures that are not output by the "bumping" process because they precede, in decoding order, a CLVSS picture that has ph\_no\_output\_of\_prior\_pics\_flag equal to 1 or inferred to be equal to 1, the output times derived from dpb\_output\_delay shall be increasing with increasing value of PicOrderCntVal relative to all pictures within the same CVS.

**pic\_dpb\_output\_du\_delay** is used to compute the DPB output time of the picture when DecodingUnitHrdFlag is equal to 1. It specifies how many sub clock ticks to wait after removal of the last DU in an AU from the CPB before the decoded picture is output from the DPB.

The length of the syntax element pic\_dpb\_output\_du\_delay is given in bits by dpb\_output\_delay\_du\_length\_minus1 + 1.

The output time derived from the pic\_dpb\_output\_du\_delay of any picture that is output from an output timing conforming decoder shall precede the output time derived from the pic\_dpb\_output\_du\_delay of all pictures in any subsequent CVS in decoding order.

The picture output order established by the values of this syntax element shall be the same order as established by the values of PicOrderCntVal.

For pictures that are not output by the "bumping" process because they precede, in decoding order, a CLVSS picture that has ph\_no\_output\_of\_prior\_pics\_flag equal to 1 or inferred to be equal to 1, the output times derived from pic\_dpb\_output\_du\_delay shall be increasing with increasing value of PicOrderCntVal relative to all pictures within the same CVS.

For any two pictures in the CVS, the difference between the output times of the two pictures when DecodingUnitHrdFlag is equal to 1 shall be identical to the same difference when DecodingUnitHrdFlag is equal to 0.

**num\_decoding\_units\_minus1** plus 1 specifies the number of DUs in the AU the PT SEI message is associated with. The value of num\_decoding\_units\_minus1 shall be in the range of 0 to PicSizeInCtbsY − 1, inclusive.

**du\_common\_cpb\_removal\_delay\_flag** equal to 1 specifies that the syntax elements du\_common\_cpb\_removal\_delay\_increment\_minus1[ i ] are present. du\_common\_cpb\_removal\_delay\_flag equal to 0 specifies that the syntax elements du\_common\_cpb\_removal\_delay\_increment\_minus1[ i ] are not present. When not present du\_common\_cpb\_removal\_delay\_flag is inferred to be equal to 0.

**du\_common\_cpb\_removal\_delay\_increment\_minus1**[ i ] plus 1 specifies the duration, in units of clock sub-ticks (see clause C.1), between the nominal CPB removal times of any two consecutive DUs in decoding order in the AU associated with the PT SEI message when Htid is equal to i. This value is also used to calculate an earliest possible time of arrival of DU data into the CPB for the HSS, as specified in Annex C. The length of this syntax element is du\_cpb\_removal\_delay\_increment\_length\_minus1 + 1 bits.

When du\_common\_cpb\_removal\_delay\_increment\_minus1[ i ] is not present for any value of i less than bp\_max\_sublayers\_minus1, its value is inferred to be equal to du\_common\_cpb\_removal\_delay\_increment\_minus1[ bp\_max\_sublayers\_minus1 ].

**num\_nalus\_in\_du\_minus1**[ i ] plus 1 specifies the number of NAL units in the i-th DU of the AU the PT SEI message is associated with. The value of num\_nalus\_in\_du\_minus1[ i ] shall be in the range of 0 to PicSizeInCtbsY − 1, inclusive.

The first DU of the AU consists of the first num\_nalus\_in\_du\_minus1[ 0 ] + 1 consecutive NAL units in decoding order in the AU. The i-th (with i greater than 0) DU of the AU consists of the num\_nalus\_in\_du\_minus1[ i ] + 1 consecutive NAL units immediately following the last NAL unit in the previous DU of the AU, in decoding order. There shall be at least one VCL NAL unit in each DU. All non-VCL NAL units associated with a VCL NAL unit shall be included in the same DU as the VCL NAL unit.

**du\_cpb\_removal\_delay\_increment\_minus1**[ i ][ j ] plus 1 specifies the duration, in units of clock sub-ticks, between the nominal CPB removal times of the ( i + 1 )-th DU and the i-th DU, in decoding order, in the AU associated with the PT SEI message when Htid is equal to j. This value is also used to calculate an earliest possible time of arrival of DU data into the CPB for the HSS, as specified in Annex C. The length of this syntax element is du\_cpb\_removal\_delay\_increment\_length\_minus1 + 1 bits.

When du\_cpb\_removal\_delay\_increment\_minus1[ i ][ j ] is not present for any value of j less than bp\_max\_sublayers\_minus1, its value is inferred to be equal to du\_cpb\_removal\_delay\_increment\_minus1[ i ][ bp\_max\_sublayers\_minus1 ].

**delay\_for\_concatenation\_ensured\_flag** equal to 1 specifies that the difference between the final arrival time and the CPB removal time of the AU associated with the PT SEI message is such that when followed by an AU with a BP SEI message with concatenation\_flag equal to 1 and InitCpbRemovalDelay[ ][ ] [Ed. (YK): Check whether it's precise to use "InitCpbRemovalDelay[ Htid ][ ScIdx ]" herein.] less than or equal to the value of max\_initial\_removal\_delay\_for\_concatenation, the nominal removal time of the following AU from the CPB computed with cpb\_removal\_delay\_delta\_minus1 applies. delay\_for\_concatenation\_ensured\_flag equal to 0 specifies that the difference between the final arrival time and the CPB removal time of the AU associated with the PT SEI message may or may not exceed the value of max\_val\_initial\_removal\_delay\_for\_splicing.

**pt\_display\_elemental\_periods\_minus1** plus 1, when sps\_field\_seq\_flag is equal to 0 and fixed\_pic\_rate\_within\_cvs\_flag[ TemporalId ] is equal to 1, indicates the number of elemental picture period intervals that the current coded picture occupies for the display model.

When fixed\_pic\_rate\_within\_cvs\_flag[ TemporalId ] is equal to 0 or sps\_field\_seq\_flag is equal to 1, the value of pt\_display\_elemental\_periods\_minus1 shall be equal to 0.

When sps\_field\_seq\_flag is equal to 0 and fixed\_pic\_rate\_within\_cvs\_flag[ TemporalId ] is equal to 1, a value of pt\_display\_elemental\_periods\_minus1 greater than 0 may be used to indicate a frame repetition period for displays that use a fixed frame refresh interval equal to DpbOutputElementalInterval[ n ] as given by Equation 112.

* 1. DU information SEI message
     1. DU information SEI message syntax

|  |  |
| --- | --- |
| decoding\_unit\_info( payloadSize ) { | Descriptor |
| **decoding\_unit\_idx** | ue(v) |
| if( !du\_cpb\_params\_in\_pic\_timing\_sei\_flag ) |  |
| for( i = TemporalId; i <= bp\_max\_sublayers\_minus1; i++ ) { |  |
| if( i < bp\_max\_sublayers\_minus1 ) |  |
| **dui\_sublayer\_delays\_present\_flag**[ i ] | u(1) |
| if( dui\_sublayer\_delays\_present\_flag[ i ] ) |  |
| **du\_spt\_cpb\_removal\_delay\_increment**[ i ] | u(v) |
| } |  |
| if( !du\_dpb\_params\_in\_pic\_timing\_sei\_flag ) |  |
| **dpb\_output\_du\_delay\_present\_flag** | u(1) |
| if( dpb\_output\_du\_delay\_present\_flag ) |  |
| **pic\_spt\_dpb\_output\_du\_delay** | u(v) |
| } |  |

* + 1. DU information SEI message semantics

The DU information SEI message provides CPB removal delay information for the DU associated with the SEI message.

The following applies for the DU information SEI message syntax and semantics:

– The syntax elements bp\_decoding\_unit\_hrd\_params\_present\_flag, du\_cpb\_params\_in\_pic\_timing\_sei\_flag, du\_dpb\_params\_in\_pic\_timing\_sei\_flag, and dpb\_output\_delay\_du\_length\_minus1 are found in the BP SEI message that is applicable to at least one of the operation points to which the DU information SEI message applies.

– The bitstream (or a part thereof) refers to the bitstream subset (or a part thereof) associated with any of the operation points to which the DU information SEI message applies.

The presence of DU information SEI messages for an operation point is specified as follows:

– If CpbDpbDelaysPresentFlag is equal to 1, bp\_decoding\_unit\_hrd\_params\_present\_flag is equal to 1 and du\_cpb\_params\_in\_pic\_timing\_sei\_flag or du\_dpb\_params\_in\_pic\_timing\_sei\_flag is equal to 0, one or more DU information SEI messages applicable to the operation point shall be associated with each DU in the CVS.

– Otherwise, in the CVS there shall be no DU that is associated with a DU information SEI message applicable to the operation point.

The set of NAL units associated with a DU information SEI message consists, in decoding order, of the SEI NAL unit containing the DU information SEI message and all subsequent NAL units in the AU up to but not including any subsequent SEI NAL unit containing a DU information SEI message with a different value of decoding\_unit\_idx. Each DU shall include at least one VCL NAL unit. All non-VCL NAL units associated with a VCL NAL unit shall be included in the DU containing the VCL NAL unit.

The TemporalId in the DU information SEI message syntax is the TemporalId of the SEI NAL unit containing the DU information SEI message.

**decoding\_unit\_idx** specifies the index, starting from 0, to the list of DUs in the current AU, of the DU associated with the DU information SEI message. The value of decoding\_unit\_idx shall be in the range of 0 to PicSizeInCtbsY − 1, inclusive.

A DU identified by a particular value of duIdx includes and only includes all NAL units associated with all DU information SEI messages that have decoding\_unit\_idx equal to duIdx. Such a DU is also referred to as associated with the DU information SEI messages having decoding\_unit\_idx equal to duIdx.

For any two DUs duA and duB in one AU with decoding\_unit\_idx equal to duIdxA and duIdxB, respectively, where duIdxA is less than duIdxB, duA shall precede duB in decoding order.

A NAL unit of one DU shall not be present, in decoding order, between any two NAL units of another DU.

**dui\_sublayer\_delays\_present\_flag**[ i ] equal to 1 specifies that du\_spt\_cpb\_removal\_delay\_increment[ i ] is present for the sublayer with TemporalId equal to i. dui\_sublayer\_delays\_present\_flag[ i ] equal to 0 specifies that du\_spt\_cpb\_removal\_delay\_increment[ i ] is not present for the sublayer with TemporalId equal to i.

When not present, the value of dui\_sublayer\_delays\_present\_flag[ i ] is infered to be as follows:

– If du\_cpb\_params\_in\_pic\_timing\_sei\_flag is equal to 0 and i is equal to bp\_max\_sublayers\_minus1, the value of dui\_sublayer\_delays\_present\_flag[ i ] is inferred to be equal to 1.

– Otherwise, the value of dui\_sublayer\_delays\_present\_flag[ i ] is inferred to be equal to 0.

**du\_spt\_cpb\_removal\_delay\_increment**[ i ] specifies the duration, in units of clock sub-ticks, between the nominal CPB times of the last DU in decoding order in the current AU and the DU associated with the DU information SEI message when Htid is equal to i. This value is also used to calculate an earliest possible time of arrival of DU data into the CPB for the HSS, as specified in Annex C. The length of this syntax element is du\_cpb\_removal\_delay\_increment\_length\_minus1 + 1. When the DU associated with the DU information SEI message is the last DU in the current AU, the value of du\_spt\_cpb\_removal\_delay\_increment[ i ] shall be equal to 0. When du\_spt\_cpb\_removal\_delay\_increment[ i ] is not present for any value of i less than bp\_max\_sublayers\_minus1, its value is inferred to be equal to du\_spt\_cpb\_removal\_delay\_increment[ bp\_max\_sublayers\_minus1 ].

**dpb\_output\_du\_delay\_present\_flag** equal to 1 specifies the presence of the pic\_spt\_dpb\_output\_du\_delay syntax element in the DU information SEI message. dpb\_output\_du\_delay\_present\_flag equal to 0 specifies the absence of the pic\_spt\_dpb\_output\_du\_delay syntax element in the DU information SEI message. When not present, the value of dpb\_output\_du\_delay\_present\_flag is inferred to be equal to 0.

**pic\_spt\_dpb\_output\_du\_delay** is used to compute the DPB output time of the picture when DecodingUnitHrdFlag is equal to 1 and du\_dpb\_params\_in\_pic\_timing\_sei\_flag is equal to 0. It specifies how many sub clock ticks to wait after removal of the last DU in an AU from the CPB before the decoded picture is output from the DPB. When not present, the value of pic\_spt\_dpb\_output\_du\_delay is inferred to be equal to pic\_dpb\_output\_du\_delay. The length of the syntax element pic\_spt\_dpb\_output\_du\_delay is given in bits by dpb\_output\_delay\_du\_length\_minus1 + 1.

It is a requirement of bitstream conformance that all DU information SEI messages that are associated with the same AU, apply to the same operation point, and have du\_dpb\_params\_in\_pic\_timing\_sei\_flag equal to 0 shall have the same value of pic\_spt\_dpb\_output\_du\_delay.

The output time derived from the pic\_spt\_dpb\_output\_du\_delay of any picture that is output from an output timing conforming decoder shall precede the output time derived from the pic\_spt\_dpb\_output\_du\_delay of all pictures in any subsequent CVS in decoding order.

The picture output order established by the values of this syntax element shall be the same order as established by the values of PicOrderCntVal.

For pictures that are not output by the "bumping" process because they precede, in decoding order, a CLVSS picture that has ph\_no\_output\_of\_prior\_pics\_flag equal to 1 or inferred to be equal to 1, the output times derived from pic\_spt\_dpb\_output\_du\_delay shall be increasing with increasing value of PicOrderCntVal relative to all pictures within the same CVS.

For any two pictures in the CVS, the difference between the output times of the two pictures when DecodingUnitHrdFlag is equal to 1 shall be identical to the same difference when DecodingUnitHrdFlag is equal to 0.

* 1. Scalable nesting SEI message
     1. Scalable nesting SEI message syntax

|  |  |
| --- | --- |
| scalable\_nesting( payloadSize ) { | **Descriptor** |
| **nesting\_ols\_flag** | u(1) |
| **nesting\_subpic\_flag** | u(1) |
| if( nesting\_ols\_flag ) { |  |
| **nesting\_num\_olss\_minus1** | ue(v) |
| for( i = 0; i <= nesting\_num\_olss\_minus1; i++ ) |  |
| **nesting\_ols\_idx\_delta\_minus1**[ i ] | ue(v) |
| } else { |  |
| **nesting\_all\_layers\_flag** | u(1) |
| if( !nesting\_all\_layers\_flag ) { |  |
| **nesting\_num\_layers\_minus1** | ue(v) |
| for( i = 1; i <= nesting\_num\_layers\_minus1; i++ ) |  |
| **nesting\_layer\_id**[ i ] | u(6) |
| } |  |
| } |  |
| if( nesting\_subpic\_flag ) { |  |
| **nesting\_num\_subpics\_minus1** | ue(v) |
| **sei\_subpic\_id\_len\_minus1** | ue(v) |
| for( i = 0; i <= nesting\_num\_subpics\_minus1; i++ ) |  |
| **nesting\_subpic\_id**[ i ] | u(v) |
| } |  |
| **nesting\_num\_seis\_minus1** | ue(v) |
| while( !byte\_aligned( ) ) |  |
| **nesting\_zero\_bit** /\* equal to 0 \*/ | u(1) |
| for( i = 0; i <= nesting\_num\_seis\_minus1; i++ ) |  |
| sei\_message( ) |  |
| } |  |

* + 1. Scalable nesting SEI message semantics

The scalable nesting SEI message provides a mechanism to associate SEI messages with specific OLSs or with specific layers and also associate SEI messages with specific sets of subpictures.

A scalable nesting SEI message contains one or more SEI messages. The SEI messages contained in the scalable nesting SEI message are also referred to as the scalable-nested SEI messages.

It is a requirement of bitstream conformance that the following restrictions apply on containing of SEI messages in a scalable nesting SEI message:

– An SEI message that has payloadType equal to 132 (decoded picture hash) shall only be contained in a scalable nesting SEI message with nesting\_subpic\_flag equal to 1.

– An SEI message that has payloadType equal to 133 (scalable nesting) shall not be contained in a scalable nesting SEI message.

– When a scalable nesting SEI message contains a buffering period, picture timing, or decoding unit information SEI message, the scalable nesting SEI message shall not contain any other SEI message with payloadType not equal to 0 (buffering period), 1 (picture timing), or 130 (decoding unit information).

It is a requirement of bitstream conformance that the following restriction applies on the value of the nal\_unit\_type of the SEI NAL unit containing a scalable nesting SEI message:

– When a scalable nesting SEI message contains an SEI message that has payloadType equal to 0 (buffering period), 1 (picture timing), 130 (decoding unit information), 145 (DRAP indication), or 168 (frame-field information), the SEI NAL unit containing the scalable nesting SEI message shall have nal\_unit\_type equal to PREFIX\_SEI\_NUT.

**nesting\_ols\_flag** equal to 1 specifies that the scalable-nested SEI messages apply to specific OLSs. nesting\_ols\_flag equal to 0 specifies that the scalable-nested SEI messages apply to specific layers.

It is a requirement of bitstream conformance that the following restrictions apply on the value of nesting\_ols\_flag:

– When the scalable nesting SEI message contains an SEI message that has payloadType equal to 0 (buffering period), 1 (picture timing), or 130 (decoding unit information), the value of nesting\_ols\_flag shall be equal to 1.

– When the scalable nesting SEI message contains an SEI message that has payloadType equal to a value in VclAssociatedSeiList, the value of nesting\_ols\_flag shall be equal to 0.

**nesting\_subpic\_flag** equal to 1 specifies that the scalable-nested SEI messages that apply to specified OLSs or layers apply only to specific subpictures of the specified OLSs or layers. nesting\_subpic\_flag equal to 0 specifies that the scalable-nested SEI messages that apply to specific OLSs or layers apply to all subpictures of the specified OLSs or layers.

**nesting\_num\_olss\_minus1** plus 1 specifies the number of OLSs to which the scalable-nested SEI messages apply. The value of nesting\_num\_olss\_minus1 shall be in the range of 0 to TotalNumOlss − 1, inclusive.

**nesting\_ols\_idx\_delta\_minus1**[ i ] is used to derive the variable NestingOlsIdx[ i ] that specifies the OLS index of the i-th OLS to which the scalable-nested SEI messages apply when nesting\_ols\_flag is equal to 1. The value of nesting\_ols\_idx\_delta\_minus1[ i ] shall be in the range of 0 to TotalNumOlss − 2, inclusive, inclusive.

The variable NestingOlsIdx[ i ] is derived as follows:

if( i = = 0 )  
 NestingOlsIdx[ i ] = nesting\_ols\_idx\_delta\_minus1[ i ] (D.4)  
else  
 NestingOlsIdx[ i ] = NestingOlsIdx[ i − 1 ] + nesting\_ols\_idx\_delta\_minus1[ i ] + 1

**nesting\_all\_layers\_flag** equal to 1 specifies that the scalable-nested SEI messages apply to all layers that have nuh\_layer\_id greater than or equal to the nuh\_layer\_id of the current SEI NAL unit. nesting\_all\_layers\_flag equal to 0 specifies that the scalable-nested SEI messages may or may not apply to all layers that have nuh\_layer\_id greater than or equal to the nuh\_layer\_id of the current SEI NAL unit.

**nesting\_num\_layers\_minus1** plus 1 specifies the number of layers to which the scalable-nested SEI messages apply. The value of nesting\_num\_layers\_minus1 shall be in the range of 0 to vps\_max\_layers\_minus1 − GeneralLayerIdx[ nuh\_layer\_id ], inclusive, where nuh\_layer\_id is the nuh\_layer\_id of the current SEI NAL unit.

**nesting\_layer\_id**[ i ] specifies the nuh\_layer\_id value of the i-th layer to which the scalable-nested SEI messages apply when nesting\_all\_layers\_flag is equal to 0. The value of nesting\_layer\_id[ i ] shall be greater than nuh\_layer\_id, where nuh\_layer\_id is the nuh\_layer\_id of the current SEI NAL unit.

When nesting\_ols\_flag is equal to 0, the variable nestingNumLayers, specifying the nubmer of layer to which the scalable-nested SEI messages apply, and the list nestingLayerId[ i ] for i in the range of 0 to nestingNumLayers − 1, inclusive, specifying the list of nuh\_layer\_id value of the layers to which the scalable-nested SEI messages apply, are derived as follows, where nuh\_layer\_id is the nuh\_layer\_id of the current SEI NAL unit:

if( nesting\_all\_layers\_flag ) {  
 nestingNumLayers = vps\_max\_layers\_minus1 + 1 − GeneralLayerIdx[ nuh\_layer\_id ]   
 for( i = 0; i < nestingNumLayers; i ++)  
 nestingLayerId[ i ] = vps\_layer\_id[ GeneralLayerIdx[ nuh\_layer\_id ] + i ] (D.5)  
} else {  
 nestingNumLayers = nesting\_num\_layers\_minus1 + 1  
 for( i = 0; i < nestingNumLayers; i ++)  
 nestingLayerId[ i ] = ( i = = 0 ) ? nuh\_layer\_id : nesting\_layer\_id[ i ]  
}

**nesting\_num\_subpics\_minus1** plus 1 specifies the number of subpictures to which the scalable-nested SEI messages apply. The value of nesting\_num\_subpics\_minus1 shall be less than or equal to the value of sps\_num\_subpics\_minus1 in the SPS referred to by the pictures in the CLVS.

**sei\_subpic\_id\_len\_minus1** plus 1 specifies the number of bits used to represent the syntax element nesting\_subpic\_id[ i ]. The value of sei\_subpic\_id\_len\_minus1 shall be in the range of 0 to 15, inclusive.

It is a requirement of bitstream conformance that the value of sei\_subpic\_id\_len\_minus1 shall be the same for all scalable nesting SEI messages that are present in a CLVS.

**nesting\_subpic\_id**[ i ] indicates the i-th subpicture ID associated with the scalable-nested SEI messages. The length of the nesting\_subpic\_id[ i ] syntax element is sei\_subpic\_id\_len\_minus1 + 1 bits.

**nesting\_num\_seis\_minus1** plus 1 specifies the number of scalable-nested SEI messages. The value of nesting\_num\_seis\_minus1 shall be in the range of 0 to 63, inclusive.

**nesting\_zero\_bit** shall be equal to 0.

* 1. Subpicture level information SEI message
     1. Subpicture level information SEI message syntax

|  |  |
| --- | --- |
| subpic\_level\_info( payloadSize ) { | Descriptor |
| **num\_ref\_levels\_minus1** | u(3) |
| **sli\_cbr\_constraint\_flag** | u(1) |
| **explicit\_fraction\_present\_flag** | u(1) |
| if( explicit\_fraction\_present\_flag ) |  |
| **sli\_num\_subpics\_minus1** | ue(v) |
| while( !byte\_aligned( ) ) |  |
| **sli\_alignment\_zero\_bit** | f(1) |
| for( i = 0; i <= num\_ref\_levels\_minus1; i++ ) { |  |
| **ref\_level\_idc**[ i ] | u(8) |
| if( explicit\_fraction\_present\_flag ) |  |
| for( j = 0; j <= sli\_num\_subpics\_minus1; j++ ) |  |
| **ref\_level\_fraction\_minus1**[ i ][ j ] | u(8) |
| } |  |
| } |  |

* + 1. Subpicture level information SEI message semantics

The subpicture level information SEI message contains information about the level that subpicture sequences in the bitstream conform to when testing the conformance of the extracted bitstreams containing the subpicture sequences according to Annex A.

When a subpicture level information SEI message is present for any picture of a CLVS, a subpicture level information SEI message shall be present for the first picture of the CLVS. The subpicture level information SEI message persists for the current layer in decoding order from the current picture until the end of the CLVS. All subpicture level information SEI messages that apply to the same CLVS shall have the same content. A subpicture sequence consists of all subpictures within a CLVS that have the same value of subpicture index.

It is a requirement of bitstream conformance that, when a subpicture level information SEI message is present for a CLVS, the value of sps\_subpic\_treated\_as\_pic\_flag[ i ] shall be equal to 1 for each value of i in the range of 0 to sps\_num\_subpics\_minus1, inclusive.

**num\_ref\_levels\_minus1** plus 1 specifies the number of reference levels signalled for each of the sps\_num\_subpics\_minus1 + 1 subpictures.

**sli\_cbr\_constraint\_flag** equal to 0 specifies that to decode the sub-bitstreams resulting from extraction of any subpicture of the bitstream according to clause C.7 by using the HRD using any CPB specification in the extracted sub-bitstream, the hypothetical stream scheduler (HSS) operates in an intermittent bit rate mode. sli\_cbr\_constraint\_flag equal to 1 specifies that the HSS operates in a constant bit rate (CBR) mode.

**explicit\_fraction\_present\_flag** equal to 1 specifies that the syntax elements ref\_level\_fraction\_minus1[ i ] are present. explicit\_fraction\_present\_flag equal to 0 specifies that the syntax elements ref\_level\_fraction\_minus1[ i ] are not present.

**sli\_num\_subpics\_minus1** plus 1 specifies the number of subpictures in the pictures of the CLVS. When present, the value of sli\_num\_subpics\_minus1 shall be equal to the value of sps\_num\_subpics\_minus1 in the SPS referred to by the pictures in the CLVS.

**sli\_alignment\_zero\_bit** shall be equal to 0.

**ref\_level\_idc**[ i ] indicates a level to which each subpicture conforms as specified in Annex A. Bitstreams shall not contain values of ref\_level\_idc other than those specified in Annex A. Other values of ref\_level\_idc[ i ] are reserved for future use by ITU-T | ISO/IEC. It is a requirement of bitstream conformance that the value of ref\_level\_idc[ 0 ] shall be equal to the value of general\_level\_idc of the bitstream and that the value of ref\_level\_idc[ i ] shall be less than or equal to ref\_level\_idc[ k ] for any value of i greater than 0 and k greater than i.

**ref\_level\_fraction\_minus1**[ i ][ j ] plus 1 specifies the fraction of the level limits associated with ref\_level\_idc[ i ] that the j-th subpicture conforms to as specified in clause A.4.1.

The variable SubpicSizeY[ j ] is set equal to ( sps\_subpic\_width\_minus1[ j ] + 1 ) \* CtbSizeY \* ( sps\_subpic\_height\_minus1[ j ] + 1 ) \* CtbSizeY.

When not present, the value of ref\_level\_fraction\_minus1[ i ][ j ] is inferred to be equal to Ceil( 256 \* SubpicSizeY[ j ] ÷ PicSizeInSamplesY \* MaxLumaPs( general\_level\_idc ) ÷ MaxLumaPs( ref\_level\_idc[ i ] ) − 1. [Ed. (HD): Shouldn't PicSizeInSamplesY here be PicSizeMaxInSamplesY as specified in clause A.4.2?]

The variable RefLevelFraction[ i ][ j ] is set equal to ref\_level\_fraction\_minus1[ i ][ j ] + 1.

The variables SubpicCpbSizeVcl[ i ][ j ] and SubpicCpbSizeNal[ i ][ j ] are derived as follows:

SubpicCpbSizeVcl[ i ][ j ] = Floor( CpbVclFactor \* MaxCPB \* RefLevelFraction[ i ][ j ] ÷ 256) (D.6)

SubpicCpbSizeNal[ i ][ j ] = Floor( CpbNalFactor \* MaxCPB \* RefLevelFraction[ i ][ j ] ÷ 256) (D.7)

with MaxCPB derived from ref\_level\_idc[ i ] as specified in clause A.4.2.

The variables SubpicBitRateVcl[ i ][ j ] and SubpicBitRateNal[ i ][ j ] are derived as follows:

SubpicBitRateVcl[ i ][ j ] = Floor( CpbVclFactor \* ValBR \* RefLevelFraction[ i ][ j ] ÷ 256) (D.8)

SubpicBitRateNal[ i ][ j ] = Floor( CpbNalFactor \* ValBR \* RefLevelFraction[ i ][ j ] ÷ 256) (D.9)

Where the value of ValBR is derived as follows:

– When bit\_rate\_value\_minus1[ Htid ][ ScIdx ] is available in the respective HRD parameters in the VPS or SPS, ValBR is set equal to ( bit\_rate\_value\_minus1[ Htid ][ ScIdx ] + 1) \* 2( 6 + bit\_rate\_scale ), where Htid is the considered sublayer index and ScIdx is the considered schedule index.

– Otherwise, ValBR is set equal to MaxBR derived from ref\_level\_idc[ 0 ] as specified in clause A.4.2.

NOTE 1 – When a subpicture is extracted, the resulting bitstream has a CpbSize (either indicated in the SPS or inferred) that is greater than or equal to SubpicCpbSizeVcl[ i ][ j ] and SubpicCpbSizeNal[ i ][ j ] and a BitRate (either indicated in the SPS or inferred) that is greater than or equal to SubpicBitRateVcl[ i ][ j ] and SubpicBitRateNal[ i ][ j ].

It is a requirement of bitstream conformance that the bitstreams resulting from extracting the j-th subpicture for j in the range of 0 to sps\_num\_subpics\_minus1, inclusive, and conforming to a profile with general\_tier\_flag equal to 0 and level equal to ref\_level\_idc[ i ] for i in the range of 0 to num\_ref\_level\_minus1, inclusive, shall obey the following constraints for each bitstream conformance test as specified in Annex C:

– Ceil( 256 \* SubpicSizeY[ j ] ÷ RefLevelFraction[ i ][ j ] ) shall be less than or equal to MaxLumaPs, where MaxLumaPs is specified in Table A.1 for level ref\_level\_idc[ i ].

– The value of Ceil( 256 \* ( sps\_subpic\_width\_minus1[ j ] + 1 ) \* CtbSizeY ÷ RefLevelFraction[ i ][ j ] ) shall be less than or equal to Sqrt( MaxLumaPs \* 8 ).

– The value of Ceil( 256 \* ( sps\_subpic\_height\_minus1[ j ] + 1 ) \* CtbSizeY ÷ RefLevelFraction[ i ][ j ] ) shall be less than or equal to Sqrt( MaxLumaPs \* 8 ).

– The value of SubpicWidthInTiles[ j ] shall be less than or equal to MaxTileCols and of SubpicHeightInTiles[ j ] shall be less than or equal to MaxTileRows, where MaxTileCols and MaxTileRows are specified in Table A.1 for level ref\_level\_idc[ i ].

– The value of SubpicWidthInTiles[ j ] \* SubpicHeightInTiles[ j ] shall be less than or equal to MaxTileCols \* MaxTileRows \* RefLevelFraction[ i ][ j ], where MaxTileCols and MaxTileRows are specified in Table A.1 for level ref\_level\_idc[ i ].

– The sum of the NumBytesInNalUnit variables for AU 0 corresponding to the j-th subpicture shall be less than or equal to FormatCapabilityFactor \* ( Max(SubpicSizeY[ j ], fR \* MaxLumaSr \* RefLevelFraction[ i ][ j ] ÷ 256 ) + MaxLumaSr \* ( AuCpbRemovalTime[ 0 ] − AuNominalRemovalTime[ 0 ] ) \* RefLevelFraction[ i ][ j ] ) ÷ ( 256 \* MinCr ) for the value of SubpicSizeInSamplesY of AU 0, where MaxLumaSr and FormatCapabilityFactor are the values specified in Table A.2 and Table A.3, respectively, that apply to AU 0, at level ref\_level\_idc[ i ], and MinCr is derived as indicated in A.4.2. [Ed. (RS): fR is a profile-specific variable defined in A.4.2.]

– The sum of the NumBytesInNalUnit variables for AU n (with n greater than 0) corresponding to the j-th subpicture shall be less than or equal to FormatCapabilityFactor \* MaxLumaSr \* ( AuCpbRemovalTime[ n ] − AuCpbRemovalTime[ n − 1 ] ) \* RefLevelFraction[ i ][ j ] ÷ ( 256 \* MinCr ), where MaxLumaSr and FormatCapabilityFactor are the values specified in Table A.2 and Table A.3 respectively, that apply to AU n, at level ref\_level\_idc[ i ], and MinCr is derived as indicated in A.4.2.

For any subpicture set containing one ore more subpictures and constisting of a list of subpicture indices SubpicSetIndices and a number of subpictures in the subpicture set NumSubpicsInSet, the level information of the subpicture set is derived.

The variable SubpicSetAccLevelFraction[ i ] for the total level fraction with respect to the reference level ref\_level\_idc[ i ], and the variables SubpicSetCpbSizeVcl[ i ], SubpicSetCpbSizeNal[ i ], SubpicSetBitRateVcl[ i ], and SubpicSetBitRateNal[ i ] of the subpicture set, are derived as follows:

for (i = 0; i <= num\_ref\_level\_minus1; i ++) {  
 SubpicSetAccLevelFraction[ i ] = 0  
 SubpicSetCpbSizeVcl[ i ] = 0  
 SubpicSetCpbSizeNal[ i ] = 0  
 SubpicSetNumTiles[ i ] = 0  
 for (j = 0; j < NumSubpicsInSet; j ++) {  
 CurrSubpicIdx = SubpicSetIndices[ j ]  
 SubpicSetAccLevelFraction[ i ] += RefLevelFraction[ i ][ CurrSubpicIdx ] (D.10)  
 SubpicSetCpbSizeVcl[ i ] += SubpicSetCpbSizeVcl[ i ][ CurrSubpicIdx ]  
 SubpicSetCpbSizeNal[ i ] += SubpicSetCpbSizeNal[ i ][ CurrSubpicIdx ]  
 SubpicSetBitRateVcl[ i ] += SubpicSetBitRateVcl[ i ][ CurrSubpicIdx ]  
 SubpicSetBitRateNal[ i ] += SubpicSetBitRateNal[ i ][ CurrSubpicIdx ]  
 SubpicSetNumTiles[ i ] += SubpicWidthInTiles[ CurrSubpicIdx ] \*   
 SubpicHeightInTiles[ CurrSubpicIdx ]  
 }  
}

The value of the subpicture set sequence level indicator, SubpicSetLevelIdc, is derived as follows:

SubpicSetLevelIdc = general\_level\_idc   
SubpicSetLevelIdx = 0  
for (i = num\_ref\_level\_minus1; i >= 1; i− − )  
 if( SubpicSetNumTiles[ i ] <= ( MaxTileCols \* MaxTileRows ) && (D.11)  
 SubpicSetAccLevelFraction[ i ] <= 256 ) {  
 SubpicSetLevelIdc = ref\_level\_idc[ i ]   
 SubpicSetLevelIdx = i  
 }

where MaxTileCols and MaxTileRows are specified in Table A.1 for ref\_level\_idc[ i ].

The subpicture set bitstream conforming to a profile with general\_tier\_flag equal to 0 and a level equal to SubpicSetLevelIdc shall obey the following constraints for each bitstream conformance test as specified in Annex C:

– For the VCL HRD parameters, SubpicSetCpbSizeVcl[ i ] shall be less than or equal to CpbVclFactor \* MaxCPB, where CpbVclFactor is specified in Table A.3 and MaxCPB is specified in Table A.1 in units of CpbVclFactor bits.

– For the NAL HRD parameters, SubpicSetCpbSizeNal[ i ] shall be less than or equal to CpbNalFactor \* MaxCPB, where CpbNalFactor is specified in Table A.3, and MaxCPB is specified in Table A.1 in units of CpbNalFactor bits.

– For the VCL HRD parameters, SubpicSetBitRateVcl[ i ] shall be less than or equal to CpbVclFactor \* MaxBR, where CpbVclFactor is specified in Table A.3 and MaxBR is specified in Table A.1 in units of CpbVclFactor bits.

– For the NAL HRD parameters, SubpicSetBitRateNal[ i ] shall be less than or equal to CpbNalFactor \* MaxCR, where CpbNalFactor is specified in Table A.3, and MaxBR is specified in Table A.1 in units of CpbNalFactor bits.

NOTE 2 – When a subpicture set is extracted, the resulting bitstream has a CpbSize (either indicated in the SPS or inferred) that is greater than or equal to SubpicSetCpbSizeVcl[ i ][ j ] and SubpicSetCpbSizeNal[ i ][ j ] and a BitRate (either indicated in the SPS or inferred) that is greater than or equal to SubpicSetBitRateVcl[ i ][ j ] and SubpicSetBitRateNal[ i ][ j ].

* 1. Use of ITU-T H.SEI | ISO/IEC 23002-7 VUI parameters

The VUI parameters specified in ITU-T H.SEI | ISO/IEC 23002-7 may be used together with bitstreams specified by this Specification.

The vui\_parameters( ) syntax structure as specified in ITU-T H.SEI | ISO/IEC 23002-7 may be included in the SPS syntax as specified in clause 7.3.2.3.

* 1. Use of ITU-T H.SEI | ISO/IEC 23002-7 SEI messages
     1. General

The SEI messages having syntax structures identified in subclause D.2.1 that are specified in ITU-T H.SEI | ISO/IEC 23002-7 may be used together with bitstreams specified by this Specification.

When any particular ITU-T H.SEI | ISO/IEC 23002-7 SEI message is included in a bitstream specified by this Specification, the SEI payload syntax shall be included into the sei\_payload( ) syntax structure as specified in clause D.2.1, shall use the payloadType value specified in clause D.2.1, and, additionally, any SEI-message-specific constraints specified in this annex for that particular SEI message shall apply.

* + 1. Use of the film grain characteristics SEI message

For purposes of interpretation of the film grain characteristics SEI message, the following variables are specified:

– PicWidthInLumaSamples and PicHeightInLumaSamples are set equal to pps\_pic\_width\_in\_luma\_samples and pps\_pic\_height\_in\_luma\_samples, respectively.

– ChromaFormatIdc is set equal to sps\_chroma\_format\_idc.

– BitDepthY and BitDepthC are both set equal to BitDepth.

* + 1. Use of the decoded picture hash SEI message

For purposes of interpretation of the decoded picture hash SEI message, the following variables are specified:

– PicWidthInLumaSamples and PicHeightInLumaSamples are set equal to pps\_pic\_width\_in\_luma\_samples and pps\_pic\_height\_in\_luma\_samples, respectively.

– ChromaFormatIdc is set equal to sps\_chroma\_format\_idc.

– BitDepthY and BitDepthC are both set equal to BitDepth.

– ComponentSamples[ cIdx ][ i ] is [Ed. (GJS): The ith sample of the cIdx’th component of the current decoded picture, which should probably have different indexing and perhaps should not have a plural name.]

* + 1. Use of the dependent random access point (DRAP) indication SEI message

A picture that is associated with a DRAP indication SEI message is referred to as a DRAP picture.

The following constraints apply to a DRAP picture:

– The VCL NAL units of the DRAP picture shall have nal\_unit\_type equal to TRAIL\_NUT.

– The DRAP picture shall have TemporalId equal to 0.

* + 1. Use of the equirectangular projection, generalized cubemap projection, and region-wise packing SEI messages

For purposes of interpretation of the equirectangular projection, generalized cubemap projection, and region-wise packing SEI message, the following variable is specified:

– ChromaFormatIdc is set equal to sps\_chroma\_format\_idc.

* + 1. Use of the frame-field information SEI message

For purposes of interpretation of the frame-field information SEI message, the following variables are specified:

– FixedPicRateWithinCvsFlag is set equal to fixed\_pic\_rate\_general\_flag[ Htid ]. [Ed. (GJS): Check/correct/clarify.]

– DisplayElementalPeriods is set equal to ElementalOutputPeriods.

When general\_progressive\_source\_flag and general\_interlaced\_source\_flag in the vui\_parameters( ) syntax structure are both equal to 1, for each picture associated with the vui\_parameters( ) syntax structure, a frame-field information SEI message associated with the picture shall be present.

When a frame-field information SEI message is present, the following constraints apply:

– The value of field\_pic\_flag shall be equal to sps\_field\_seq\_flag.

– If a PT SEI message is present for picture n, elementalOutputPeriods is equal to the value of pt\_display\_elemental\_periods\_minus1 + 1.

– When general\_progressive\_source\_flag is equal to 0 or general\_interlaced\_source\_flag is equal to 0, the value of source\_scan\_type shall be constrained as follows:

– If general\_progressive\_source\_flag is equal to 0 and general\_interlaced\_source\_flag is equal to 1, source\_scan\_type shall be equal to 0 (interlaced).

– Otherwise, if general\_progressive\_source\_flag is equal to 1 and general\_interlaced\_source\_flag is equal to 0, source\_scan\_type shall be equal to 1 (progressive).

– Otherwise (general\_progressive\_source\_flag is equal to 0 and general\_interlaced\_source\_flag is equal to 0), source\_scan\_type shall be equal to 2 (unknown or unspecified).

**Bibliography**

1. Rec. ITU-T H.222.0 (in force), *Information technology – Generic coding of moving pictures and associated audio information: Systems.*

ISO/IEC 13818-1(in force), *Information technology* – *Generic coding of moving pictures and associated audio information* – *Part 1: Systems.*

1. Rec. ITU-T H.264 (in force), Advanced video coding for generic audiovisual services.

ISO/IEC 14496-10: (in force), *Information technology* – *Coding of audio-visual objects* – *Part 10: Advanced Video Coding*.

1. Rec. ITU-T H.320 (in force), Narrow-band visual telephone systems and terminal equipment.
2. Rec. ITU-R BT.2100 (in force), *Image parameter values for high dynamic range television for use in production and international programme exchange.*