Joint Collaborative Team on Video Coding (JCT-VC)
of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11
11th Meeting: Shanghai, CN, 10–19 October 2012

Document: JCTVC-K1003_v13

Title: **High Efficiency Video Coding (HEVC) text specification draft 9**

Status: Output Document of JCT-VC

Purpose: Draft of HEVC

Author(s) or Contact(s):

Benjamin Bross Fraunhofer HHI	Email:	benjamin.bross@hhi.fraunhofer.de
Woo-Jin Han Gachon University	Email:	hurumi@gmail.com
Jens-Rainer Ohm RWTH Aachen	Email:	ohm@ient.rwth-aachen.de
Gary J. Sullivan Microsoft	Email:	garysull@microsoft.com
Thomas Wiegand Fraunhofer HHI / TU Berlin	Email:	thomas.wiegand@hhi.fraunhofer.de

Source: Editor

Abstract

Draft 9 of High efficiency video coding.

Ed. Notes (D9):

- General editorial cleanup and consistency improvements (including some section restructuring)
- Cleanup and correction of colour space specification aspects (e.g. w.r.t. UHDTV)
- Improvement of Annex A examples for level limits on picture sizes and frame rates
- Correction of missing condition on LongTermRefPic in spatial MVP derivation relative to action recorded for JCTVC-J0071 and JCTVC-J0121 (#647)
- Improved definition of "inter prediction" (#342)
- Correcting usage of "disable_deblocking_filter_flag" (#654)
- Corrected use of Log2CtbSize versus Log2CtbSizeY in Annex A (#655)
- Corrected use of I_PCM mode (#634)
- Removed apparently-unnecessary checks for "&& NumPocTotalCurr > 1" in ref_pic_list_modification( ) syntax, as it does not seem to make sense for the encoder to set ref_pic_list_modification_flag_lX to 1 when NumPocTotalCurr is equal to 1
- Modifying cbf_cb[x0][y0][trafoDepth] and cbf_cr[x0][y0][trafoDepth] semantics in regard to luma versus chroma (TB versus TU) location specification (Dzung Hoang JCT-VC experts reflector email 3 Aug 2012)
- Fixed typos/trivial issues #697, #667, #658, #652, #726, #725, #722, #694, #688, #682, #674, #721, #719, #718, #708, #705, #695, #693, #691, #689, #683, #677, #672, #663, #706, #668, #656, #653
- Fixed spatial merge candidate horizontal position xBk modification (#702)
- Fixed 8 bit WP chroma offset (#699)
- Fixed scaling_list_dc_coef_minus8 has no default value (#679)
- Fixed missing condition on constrained_intra_pred_flag (#665)
- Fixed typos/trivial issues #698, #741, #752, #469, #764, #765, #766, #767, #769, #773, #774, #775
- Renamed ref_pic_list_modification( ) to ref_pic_lists_modification( )
- Replaced Sign2( ) by Sign3( ) and renamed Sign3( ) to Sign( )
- Fixed NOTE formating by using Note 2 style for notes in definitions and Note 1 style for all other notes
- Renamed CtbAddrRS and CtbAddrTS in CtbAddrInRS and CtbAddrInTS to not be prefix of the arrays CtbAddrRStoTS and CtbAddrTStoRS anymore.
- Added default values for num_tile_columns_minus1, num_tile_rows_minus1 and added tiles_enable_flag to the constraint (#649)
- video_format table added to VUI parameters semantics (#730)
- fixed chroma deblocking grid (#711)

- Clarified binarization of cu_qp_delta_abs with prefix and suffix (#759)
- Fixed part_mode ctx derivation for binIdx equal to 2 (#720)
- Improved chroma_weight_l1_flag parsing (#779)
- Fixed default scaling list data issue (#762)
- Fixed using chroma bit depth in chroma edge deblocking (#744)
- Improved semantics of loop_filter_across_slices_enabled_flag and slice_loop_filter_across_slices_enabled_flag (#772)
- Added description of quantization group (#736)
- Fixed qP_{Y_PREV} derivation (#742)
- Replaced abs(·) with Abs(·)
- Added editorial comments to parts that need to be updated from AVC spec.
- Insert line breaks in the nal_unit_type table to align the number and the names.
- Fix for RPS restriction of TSA to reflect the original intent.
- Fixed the problem with the precedence in the derivation / usage of NumPocTotalCurr.
- Editorial fix for the marking process of the bumping decoder in Annex C.
- Renamed TFD and DLP pictures as random access skipped leading (RASL) picture and random access decodable leading (RADL) picture, respectively.
- Incorporated cleanup and complexity reduction for spatial merge candidate derivation according to JCTVC-K0197.
- Incorporated MV line buffer cleanup (JCTVC-K0101).
- Incorporated the contouring artefact fix (JCTVC-K0139)
- Incorporated inverse ctb raster scan fix and grouping of pcm sps syntax elements (JCTVC-K0217)
- Changed Table A1, A3 and A4 for Level limits in Level 6, 6.1 and 6.2 (JCTVC-K0377)
- Modified number of slice constraints (JCTVC-K0201)
- Added number of tiles constraints (JCTVC-K0202)
- Incorporated CTU bit size limit (JCTVC-K0176)
- Delete the 2/3 case of MaxDpbSize derivation (JCTVC-K0189)
- Incorporated Main 10 profile (JCTVC-K0109)
- Removed num_subsequent_pcm (JCTVC-K0258)
- Remove entropy slices (JCTVC-K0288)
- Add Main Still Picture profile (as agreed jointly in VCEG+MPEG+JPEG discussion).
- Incorporated 4x4 default scaling matrix as flat (JCTVC-K0203)
- Incorporated CABAC termination for the end of tile/WPP (JCTVC-K0182)
- Incorporated CABAC cleanup and fixes
- Fixed PCM mode definition (#796)
- Fixed infinite loop in intra mode parsing (#793)
- Disabled luma DC, hor. and ver. intra filtering for 32x32 blocks (JCTVC-K0380)
- Incorporated use of picture-level chroma QP offsets in deblocking (JCTVC-K0220)
- Dependent slices clarification, including the use of the term "slice segment" (JCTVC-K0314)
- Replaced the chroma subsampling figures, fixed and added CABAC decoding flowcharts
- Removed the slice (segment) extension data (JCTVC-K0361)
- Integrated re-allocation of NAL unit types per decisions noted under JCTVC-K0159
- Integrated the following decisions noted under JCTVC-K0120:
    - Move sps_temporal_nesting_flag to an earlier place to replace sps_reserved_zero_bit (later cleanup #864)
    - Clarification of semantics of end of sequence RBSP
    - Specify activation of VPS and SPS by APS SEI message and not specify SPS activation by the SPS ID in the BP SEI message
    - 2^24 constraint on POC difference for LTRP per BoG report (JCTVC-K0339)
    - Alignment of the range restriction on SPS ID (make it 16) and PPS ID (make it 64) for semantics and profile specification. We can simply remove this constraint in the profile specification
    - Removal of constraint on position of persistent SEI messages from 7.4.1.4.2
    - Specify a suffix SEI message NUT – with payloadType = 132 for the decoded picture hash SEI message
    - Added an informative table describing the scope of each type of SEI message
    - POC temporal relationship syntax based on JCTVC-K0343. timing_info_present_flag should have an inferred value of 0 when not present.
- Integrated SEI payload extension mechanism noted under JCTVC-K0371
- Integrated changes related to CRA and BLA pictures noted under JCTVC-K0122
- Integrated RPS changes noted under JCTVC-K0123
- Integrated VPS changes noted under JCTVC-K0125
- Integrated operation-point changes relating to JCTVC-K0204
- Integrated TL0 index SEI message under JCTVC-K0205

- Integrated addition of extra reserved slice header bits relating to JCTVC-K0210
- Integrated changes relating to temporal ID nesting (vps_temporal_id_nesting_flag) relating to JCTVC-K0173 (w/ later cleanup)
- Integrated VPS reserved bits changes relating to JCTVC-K0254
- Integrated sub-layering presence change for VPS and SPS and note on VPS discarding relating to JCTVC-K0330
- Integrated change of dependent slice segments flag locations relating to JCTVC-K0184
- Integrated moving of restricted_ref_pic_lists_flag and lists_modification_present_flag relating to JCTVC-K0170
- Integrated changes to ref_pic_list_modification( ) relating to proposal #1 of JCTVC-K0224
- Integrated change to not send inter_ref_pic_set_prediction_flag for index 0 relating to JCTVC-K0136
- Integrated change to slice_temporal_mvp_enable_flag in slice header noted under JCTVC-K0251 (w/ later cleanup)
- Fixed several issues (#808, #809, #811, #812, #814, #815, #823, #825, #839, #842, #844, #846).
- Fixed several issues (#802, #685, #847, #849, #850, #851, #852, #853, #855, #856, #857, #783).
- Integrated decisions on FPA SEI messages and display/crop windows relating to JCTVC-K0382 (w/ later cleanup).
- Fixed several issues (#858, #859, #860, #863).
- Integrated decoder parallelism indication relating to JCTVC-K0236.
- Added 8192x4096 and 4*HD example resolutions in Annex A (#824).
- Integrated frame/field coding indication changes relating to JCTVC-K0165.
- Fixed TMVP process to match HM and fixed several minor issues in that process (#733, #757, #771).
- Fixed 2D matrix notation and several issues (#687, #868, #873, #876).
- Integrated VPS changes, scalable nesting SEI message, and HRD changes relating to JCTVC-K0180 and JCTVC-K0126.
- Integrated sub-picture CPB size noted under JCTVC-K0221 (w/ later cleanup).
- Integrated fixed_pic_rate_within_cvs_flag[i] as noted under JCTVC-K0140.
- Integrated region refresh information SEI message under JCTVC-K0128.
- Updating UHDTV with reference to BT.2020 and aligning number assignment with CICP and JPEG XR.
- Fixed several issues (#893, #897, #898, #899, #902, #905, #908).
- Fixed several issues (#896, #915, #916, #917, #918, #686).

Draft 8 of High efficiency video coding.

Ed. Notes (D8):

- prev_intra_pred_flag should be prev_intra_luma_pred_flag (#624)
- fixed level 2 MaxBR and Max CPB Size (#625)
- Revised the Annex A tables based on JCTVC-J0558, JCTVC-J0154, JCTVC-J0151.
- Renamed no_residual_data_flag to no_residual_syntax_flag (editorial only). (JCTVC-J0336)
- Added limit to the minimum CTB size to 32x32 for level 5 and higher while still requiring decoding of lower level bitstreams (JCTVC-J0334)
- Integrated step wise increase of MaxDpbSize in spirit of JCTVC-J0496
- Bug fix for recovery point SEI so that it can point to a point before the recovery point
- Remove rap_pic_id and move no_output_of_prior_pic_flag to the top before any ue(v) (JCTVC-J0108)
- Added editorial suggestions from JCTVC-J0345
- Added picture timing SEI message syntax changes and corresponding text to Annex C CPB operation (JCTVC-J0306 and JCTVC-J0136)
- Added text for sub-picture based CPB removal timing (JCTVC-J0569)
- Added GTLA (STSA) related text (JCTVC-J0305)
- Added TFD related constraints on the RPS that are missing (JCTVC-J0229)
- Integrated RAP NUT according to (JCTVC-J0344) and GTLA (STSA) NUT according to (JCTVC-J0305)
- Remove nal_ref_flag and related text, added 3 new non-reference NUT and the RPS constraint (JCTVC-J0549)
- prev_intra_pred_flag should be prev_intra_luma_pred_flag (#624)
- Fixed level 2 MaxBR and Max CPB Size (#625)
- Integrated Inter-RPS complexity reduction (JCTVC-J0234) and restriction on delta_idx_minus1 (JCTVC-J0185r2)
- Reverted definition of prevRefPic to have TemporalId equal to 0 (JCTVC-J0248)
- Integrated changes of NAL unit header, VPS, HRD parameters and other changes per adoptions noted under JCTVC-J0550, JCTVC-J0548, and JCTVC-J0562 (JCTVC-J0550, JCTVC-J0548, and JCTVC-J0562)
- Incorporated removal of zigzag scan from scaling list coding (JCTVC-J0150)
- Incorporated removal of the 8x2/2x8 coefficient groups (JCTVC-J0256)
- Incorporated SAO syntax changes (JCTVC-J0563)
- Incorporated greater1 and greater2 counter removal (JCTVC-J0408)
- Incorporated split_transform_flag ctx derivation cleanup (JCTVC-J0133)
- Incorporated bypass bins for reference index coding (JCTVC-J0098)
- Incorporated bin reduction for delta QP coding (JCTVC-J0089)

- Incorporated coeff_abs_level_remaining bin reduction (JCTVC-J0142)
- Incorporated simplification on context derivation of cbf_luma syntax element (JCTVC-J0303)
- Incorporated chroma QP range extension (JCTVC-J0342)
- More flexible syntax for tile and WPP combination using tiles_enabled_flag, entropy_coding_sync_enabled_flag, and entropy_slice_enabled_flag (JCTVC-J0558).
- Incorporated end of bitstream and end of sequence NUTs, with end of bitstream indicating HRD discontinuity (response to JCTVC-J0290 and JCTVC-J0347)
- Removed ALF
- Removed LM Chroma
- Removed fine-granularity slices
- Removed NSRQT
- Incorporated transform simplification for 4x4 luma intra transform block (JCTVC-J0021)
- Moved transform_skip_enabled_flag to PPS (JCTVC-J0184)
- Moved seq_loop_filter_across_slices_enabled_flag to PPS (JCTVC-J0288)
- Incorporated inter transform skipping changes (JCTVC-J0237)
- Use of one bit from profile_space for indication of the level tier
- A clarification of intended tolerance of decoders for reserved values (JCTVC-J0112)
- POC value range to 32 bits (JCTVC-J0084)
- Modification of POC definition in Clause 3 and the POC constraint in Annex C.4 (JCTVC-J0110)
- Changing SEI NAL unit to be allowed to follow the first VCL NAL unit in an AU, added sub-picture timing SEI message, and clarified hat existing SEI messages that have whole-picture scope shall appear before the first slice in the picture (JCTVC-J0255)
- Modification of the definitions of reference pictures and related (related to JCTVC-J0118)
- Clarification of semantics of vps_temporal_id_nesting_flag and sps_temporal_id_nesting_flag (JCTVC-J0183)
- Slice header syntax clean-up (JCTVC-J0300)
- Change of definitions of slice and tile scan (JCTVC-J0209)
- Integrated long-term reference pictures in SPS (JCTVC-J0116)
- Support of UHDTV colorimetry (JCTVC-J0577, JCTVC-J0477)
- Change on signalling of luminance dynamic range in tone mapping information SEI (JCTVC-J0149)
- Change on CU QP delta enabling syntax (JCTVC-J0220)
- Inclusion of motion related hooks (JCTVC-J0568, JCTVC-J0071, JCTVC-J0121)
- Modified sub-bitstream extraction process (JCTVC-J0074)
- Modified prediction weight table syntax and semantics for slice header parsing overhead reduction and improved value range for chroma weight offset (JCTVC-J0571 and JCTVC-J0221)
- Changed example time interval for t_c to a frame-based example rather than a field coding example (JCTVC-J0136)
- Changed semantics of restricted_ref_pic_lists_flag (JCTVC-J0290)
- Incorporated disallow bi-predictive mode for 8x4 and 4x8 inter PUs (JCTVC-J0086)
- Incorporated various limits and modulo MV interpretation (response to JCTVC-J0579 BoG report)

Draft 7 of High efficiency video coding.

Ed. Notes (D7):
- RQT related issues (#459)
- Log2MaxTrafoSize constraint (#348)
- log2TrafoWidth1 and log2TrafoHeight1 calculation for NSQT (#458)
- non_square_quadtree_enabled_flag (#403)
- Wrong 'else if...' for 'if (skip_flag)' in CU syntax (#452)
- use_delta_flag[j] is not decoded when used_by_curr_pic_flag[j] is 1 (#451)
- combined_inter_pred_ref_idx does not exist anymore (#448)
- Typos in reference picture list combination (#446)
- Text cleanup of QP prediction / derivation (#492)
- Equation (7-59) and (7-60) for RPS derivation do not match HM6.0 (#445)
- Chroma NSRQT fixes (#505,#506)
- Missing decoding process for AMP fixed (#361)
- Deblocking processing order fixed (#412)
- Derivation of coefficient group scan fixed (#372)
- Phrase "one of the following conditions is true" fixed (#540)
- Non-normative aspects in scaling process removed (#544)
- scaling_list_present_flag issues fixed (#407)
- CTB,CB,PB,TB/CTU,CU,PU,TU defined
- beta_offset_div2 and tc_offset_div2 semantics fixed (#353)

- pixel replaced by sample (#406)
- references in HRD Annex C fixed (#557)
- sao_type_idx binarization fixed (#367)
- raster scan to tile scan order fixed (#376)
- error in tileId[] derivation loop fixed (#558)
- CtbAddrRS update in slice data syntax fixed (#345)
- SAO process issues fixed (#504)
- SAO clipping added (#517)
- fixed-length (FL) binarization process fixed (#518)
- intra transform skipping shift with high bit-depth fixed (#560)
- missing syntax element sao_type_idx added in sao_param syntax table (#576)
- residual coding semantics typo fixed (#333)
- PCM alignment at PU syntax fixed (#346)
- non-break hyphen and minus fixed (#347)
- uneccessary shared_pps_info_enabled_flag semantics removed (#351)
- skip_flag semantics fixed (#355)
- POC usage description fixed (#359)
- replaced coding tree depth variables cuDepth and cbDepth with ctDepth (#385)
- PicWidthInSamples fixed (#443)
- RefPicList naming fixed (#478)
- References to StCurr0/1 changed to StCurrBefore/After (#482)
- two references in 8.1 fixed (#479)
- one reference in 8.1 fixed (#509)
- replaced 'nal_ref_idc' by 'nal_ref_flag' in subclause 8.3.6 (#510)
- removed non-breaking space between log2_min_coding_block_size_minus and 3 (#386)
- removed the "+1" in clause C in description of DPB size (#522)
- moved the restriction on the relationship between tiles and slices to subclause 6.3 (#331)
- Subclause 0.7 text reference issue fixed (#572)
- cu_qp_delta binarization fixed (#588)
- typo fixed (#480)
- indices of nonsquare scaling matrix fixed (#535)
- intra_chroma_pred_mode bypass-coded bin fixed (#570)
- typos fixed and definition of picture added (#378)
- uneccesary input paramter in luma block edge deblocking removed (#583)
- missing index i for sao_offset_abs in sao_param syntax added (#592)
- fixed the Sign( ) function for SAO edgeIdx calculation (#591)
- picture constrution process added and embedded properly (#364)
- removed SignalledAsChromaDC because it not used anymore (#371)
- CABAC cross references fixed (#462)
- recovery_frame_cnt semantics removed (#600)
- nal_unit_type values updated in temporal_id semantics (#601)
- bugs, wordings, typos fixed in inter prediction process (#433)
- removed redundant equations in WP process (#416)
- fixed clipping values in MV scaling (#341)
- fixed 14-bit issues in WP (#594)
- fixed significant_coeff_group_flag for 2x8 and 8x2 CGs (#609)
- fixed inter_pred_idc binarization (#585)
- clarified condition in "6.4.5 Up-right diagonal scanning array initialization process" (#612)
- revision of weighted prediction process (#606)
- added transform_skip_enabled_flag semantics (#613)
- removed old alf aps syntax (#615)
- fixed decoding process for intra blocks for chroma (#330)
- fixed filtering process for neighbouring samples for Intra_FromLuma (#404)
- IntraPredMode derivation improvement (#360)
- vui_parameters_present_flag semantics added (#477)
- fixed Cr pcm_sample_chroma location in PCM intra decoding (#526)
- renamed unused intraPreModeN to candIntraPreModeN (#581)
- fixed sig flag semantics for 8x2 / 2x8 coefficient groups (#617)
- fixed out of bounds index computation in Angular Intra prediction (#339)

- fixed table for invAngle (#363)
- intra decoding process cleanup and fixes (#417)
- fixed %32 error for luma intra prediction mode derivation process (#493)
- generalized angular prediction process (#537)
- fixed aps_extension_flag semantics (#621)
- fixed ctxIdxInc for cbf_cb and cbf_cr overflow (#619)
- fixed mismatch between WD and HM on end_of_slice_flag (#598)
- fixed typo in 8.3.2 Decoding process for reference picture set (#622)
- fixed coeff_abs_level_greater1/2_flag ctx derivation (#384)
- updated CABAC init values to match HM (#473)
- fixed cRiceParam mismatch (#636)
- fixed transform issues (#524)
- fixed typos in 9.3 CABAC parsing process (#637)
- fixed hor and ver intra filtering (#639)

- Incorporated CBF coding without derivation process (JCTVC-I0152)
- Incorporated Unified CBFU and CBFV Coding in RQT (JCTVC-I0332)
- Incorporated BoG on I_PCM / lossless deblocking unification (JCTVC-I0586)
- Incorporated transform and quantization bypass (JCTVC-I0529)
- Incorporated intra 4x4 transform skipping (JCTVC-I0408)
- Incorporated modifed deblocking threshold derivation table (JCTVC-I0258)
- Incorporated constrained motion data compression (JCTVC-I0182)
- Removed SAO parameters from APS (JCTVC-I0021)
- Incorporated SAO offset signaling with magnitude and sign (JCTVC-I0168)
- Incorporated SAO offset magnitude TU binarization (JCTVC-I0066)
- Incorporated no SAO merge at tile boundaries (JCTVC-I0172)
- Incorporated reordering of slice type values (JCTVC-I0500)
- Incorporated having tile syntax only in PPS and reordering of pic_parameter_set_id in slice header to solve the slice header parsing issue (JCTVC-I0113)
- Incorporated moving list scaling syntax as well as deblocking filter parameters from APS to SPS and PPS (JCTVC-I0465)
- Incorporated a note on the presense of required parameter sets for random access (JCTVC-I0067)
- Incorporated an additional constraint on RPS for TLA (TSA) pictures (JCTVC-I0236)
- Incorporated changing the derivation of the variable prevRefPic used in derivation of picture order count (JCTVC-I0345)
- Incorporated high-level syntax clean-ups on TMVP enabling as well as signalling of collocated picture, CU QP delta, entropy slice header, and slice header syntax (JCTVC-I0127, JCTVC-I0266, and JCTVC-I0420)
- Incorporated mandating nal_ref_flag to be 1 for CRA pictures (JCTVC-I0143)
- Incorporated broken link access (BLA) pictures, signaling of leading/TFD pictures, signalling of presence of leading/TFD pictures, and allocation of NAL unit types (JCTVC-I0275, JCTVC-I0278, JCTVC-I0404 and JCTVC-I0607)
- Incorporated removal of combined list (JCTVC-I0125)
- Incorporated entropy slice enabling in PPS, support of dependent slice, a constraint on prevRefPic, slice header byte alignment, entropy slice header, relationship between TLA (TSA) and temporal_id_nesting_flag, tile and WPP byte alginment, semantics of num_reorder_pics[i], and semantics of temporal_id (JCTVC-I0138, JCTVC-I0229, JCTVC-I0330 and JCTVC-I0600)
- Incorporated WPP simplification and a restriction on coexistence of WPP and slices (JCTVC-I0360 and JCTVC-I0361)
- Incorporated removal of entry point markers, addition of VUI flag tiles_fixed_structure_flag, mandating entry point signalling for each tile and WPP sub-stream, and entry point offsets being relative to end of slice header (JCTVC-I0159, JCTVC-I0233, JCTVC-I0237 and JCTVC-I0357)
- Incorporated a fix for an unhandled LTRP case, a fix to the POC MSB cycle coding, coding of LTRP POC LSB directly as u(v), and no MVP scaling for LTRPs (JCTVC-I0234, JCTVC-I0340, JCTVC-I0422)
- Incorporated HRD buffering for CRA/BLA pictures and sub-picture based CPB operation (JCTVC-I0277 and JCTVC-I0588)
- Incorporated introduction of video parameter set, and extension mechanisms for slice header and slice layer RBSP (JCTVC-I0230 and JCTVC-I0235)
- Incorporated increasing of POC range to 64 bits, a limit of POC difference between the current picture and a long-term reference picture, changes to scaling list syntax, and changed signalling for profile and level (JCTVC-I0045, JCTVC-I0059, and JCTVC-I0499)

- Incorporated a bug fix for the recovery point SEI message and a change to the decoded picture hash SEI message (JCTVC-I0044 and JCTVC-I0218)
- Incorporated a change to the field indication SEI message (JCTVC-I0393)
- Incorporated adaptive loop filter text (JCTVC-I0603)
- Removed implicit weighted prediction (JCTVC-I0589)
- Incorporated simplified merge TMVP refIdx derivation (JCTVC-I0116)
- Incorporated AMVP and merge zero MV candidate list completing (JCTVC-I0314/JCTVC-I0134)
- Incorporated removal of number of combined merge candidate restriction (JCTVC-I0414)
- Removed inter 4x4 partitions
- Incorporated maxNumMergeCand signaling fix (JCTVC-I0256)
- Incorporated no bi-prediction for luma prediction blocks smaller than 8x8 (JCTVC-I0297)
- Incorporated cu_qp_delta parsing to enable CU-level processing (JCTVC-I0219)
- Incorporated simplification for multiple sign bit hiding (JCTVC-I0156)
- Incorporated restricting the range of coeff_abs_level_remaining (JCTVC-I0254)
- Incorporated table removal in contexts assignment of last significant position coding (JCTVC-I0331)
- Incorporated simplified context derivation for significance map (JCTVC-I0296)
- Incorporated context derivation clean-up for significance map (JCTVC-I0373)
- Incorporated simplified coeff_abs_level_remaining binarization (JCTVC-I0487/I0124)
- Incorporated a maximum bound on slices per picture (JCTVC-I0238)
- Incorporated Main Profile coding tool decisions
- Incorporated LM (intra chroma prediction based on luma) mode clean-up (JCTVC-I0148)
- Incorporated LM mode with uniform bit-width multipliers (JCTVC-I0151)
- Incorporated LM mode with uniform bit-width multipliers and reduced look-up table for division approximation (JCTVC-I0166)
- Incorporated LM mode with simplified alpha bit-depth restriction (JCTVC-I0178)
- Incorporated grouping of intra mode bypass bins for NxN intra prediction blocks (JCTVC-I0302)
- Incorporated level 2.1 and modified level 3 and 3.1 (JCTVC-I0472/I0455)

Ed. Notes (D6):

- Incorporated limiting dynamic range when qmatrix is used (JCTVC-H0541)
- Incorporated simplified intra horizontal and vertical modes (JCTVC-H0238)
- Incorporated DC mode as a default mode (JCTVC-H0242)
- Incorporated compatible QP prediction with RC and AQ (JCTVC-H0204)
- Incorporated burst transmission of I_PCM (JCTVC-H0051)
- SAO syntax fix (Ticket #308)
- Unused semantics removal related to reference picture list modification (Ticket #293)
- hPos and vPos table fix in SAO EO (Ticket #300)
- Typo in weighted prediction fix (Ticket #309)
- Incorporated clipping operation in strong deblocking (JCTVC-H0275)
- Incorporated removing sign of SAO offset (JCTVC-H0434)
- Fix QP'Cb and QP'Cr to consider QpBdOffsetC (Ticket #313)
- Fix padding issue of LM mode
- Incorporated deblocking filter simplification (JCTVC-H0473)
- Fix ALF syntax mismatch
- Incorporated ALF with single filter type (JCTVC-H0068)
- Incorporated intra mode coding clean-up and simplification (JCTVC-H0712)
- Fix clipping in DST (Ticket #307)
- Considering pcm_loop_filter_disable_flag in SAO (Ticket #301)
- Fix wrong geometry of sub-pel interpolation filter (Ticket #318)
- Fix max value of num_ref_idx_l0/l1_default_active_minus1 (Ticket #281)
- Fix 16x16 and 32x32 quantization matrices (Ticket #320)
- Incorporated quantization matrix signalling (JCTVC-H0237)
- Incorporated deblocking parameter signalling (JCTVC-H0424/H0398)
- Incorporated chroma mode signalling (JCTVC-H0475/H0326)
- Incorporated 4x4 and 8x8 default quantization matrices (JCTVC-H0461)
- Incorporated lossless mode (JCTVC-H0530)
- Incorporated downsampling of q-matrix (JCTVC-H0230)
- Fix ALF chroma coefficients prediction (Ticket #321)
- Incorporated two stage design ALF with LCU-based syntax (JCTVC-H0274)

- Incorporated SAO with LCU-based syntax (JCTVC-H0273)
- Incorporated change to add a condition for presence of ref_idx_list_curr (JCTVC-H0137 proposal #1)
- Incorporated unification of reference picture list modification processes (JCTVC-H0138)
- Incorporated change to NAL unit header and output flag (part of JCTVC-H0388)
- Incorporated coding treeblock and coding block scanning and address derivation.
- Incorporated changes relating to allowing the bitstream to start with a CRA picture (JCTVC-H0496)
- Incorporated multiple sign bits hiding (JCTVC-H0481)
- Incorporated 8x8 diagonal scan by 4x4 diagonal sub-scans (JCTVC-H0526/H0399)
- Incorporated abs_greater1 and abs_greater2 context reduction (JCTVC-H0130)
- Incorporated 8 bit codeword, change the Rice parameter to 4 (JCTVC-H0498)
- Incorporated high throughput binarization for CABAC (JCTVC-H0554)
- Incorporated sharing sig_coeff_flag cxtCnt=0 at high frequency area (JCTVC-H0095)
- Incorporated unified sig_coeff_flag context selection for 16x16 and 32x32 (JCTVC-H0290)
- Incorporated simplification on sig_coeff_group_flag coding (JCTVC-H0131)
- Incorporated Profiles and Levels (JCTVC-H0738)
- Incorporated CABAC initialization process (JCTVC-H0535)
- Incorporated part_mode context reduction (JCTVC-H0545)
- Incorporated merge index context reduction (JCTVC-H0251)
- Incorporated CABAC bit to bin expantion ratio limit (JCTVC-H0450)
- Incorporated CABAC_init_flag (JCTVC-H0540)
- Incorporated last_sig_coeff_position_prefix for luma context reduction (JCTVC-H0537)
- Incorporated last_sig_coeff_position_prefix for chroma context reduction (JCTVC-H0514)
- Incorporated unified transform and coefficient tree (JCTVC-H0123)
- Fix chroma cbf syntax mismatch (Ticket #295)
- Incorporated decisions relating to picture size and cropping parameters (response to JCTVC-H0485)
- Incorported adoptions on tiles, WPP and entropy slices documented in JCTVC-H0737 (response to JCTVC-H0439, H0463, H0513, H0517, and H0556)
- Incorported decisions on long-term reference picture signalling (response to JCTVC-H0200 and JCTVC-H0531) and a resriction on POC values (response to JCTVC-H0449)
- Merge estimation region syntax (JCTVC-H0082)
- Setting the merge TMVP refidx to 0 for the non-first partition (JCTVC-H0278 / JCTVC-H0199)
- One merge candidate list for all partitions inside a 8x8 CU (JCTVC-H0240 Variant 2 conditioned on log2_parallel_merge_level_minus2 > 0 as described in H0082 section 6)
- Removing non-scaled bi-predictive merging candidates (JCTVC-H0250 /JCTVC-H0164)
- Removing the list empty check and the duplicate check in AMVP for zero motion vector (JCTVC-H0239)
- Removing redundant spatial candidates check in AMVP (JCTVC-H0316 first part)
- Clipping scaled MV to 16 bit and adjust it according to profile/level decisions (JCTVC-H0216 / JCTVC-H0555)
- Fix AMVP non-scaled/scaled candidate WD/HM mismatch by correcting the WD (JCTVC-H0462)
- Motion prediction at entropy slice boundary (JCTVC-H0362)
- Limiting collocated temporal reference to one per picture (JCTVC-H0442)
- Modification of bi-prediction syntax CE9 BP08 (JCTVC-H0111)
- CE9 SP (JCTVC-H0252)
- Incorported adoptions related to reference picture set, SPS syntax and HRD (JCTVC-H0568, H0566, H0567, H0423 and H0412).
- Rewrote SAO and ALF syntax and semantics and part of the processes.
- Incorporated harmonization of number of ALF classes between RA and BA modes (JCTVC-H0409)
- Incorporated harmonization of ALF luma and chroma center coefficient prediction (JCTVC-H0483)
- Incorporated JCTVC-H0174-B with modification of number of bands and coding of the offset band according to JCTVC-H0406 (5b FLC/bypass coding)..
- Imported VUI from AVC with HEVC modifications (based upon JCTVC-F289)
- Imported SEI messages from AVC with HEVC semantic restrictions (based upon JCTVC-E346)
- Incorporated display orientation SEI (VCEG-AR12_r2)
- Incorporated temporal structure SEI (JCTVC-H0423)
- Incorporated decoded picture hash SEI (JCTVC-E490)
- Incorporated field indication SEI and VUI (JCTVC- H0720)
- Incorporated inheriting the QP prediction value at the left edge from the slice header in which the LCU belongs (JCTVC-H0226)
- Fixed missing definition of "cu_qp_delta_enabled_flag" (Ticket #310)
- Incorporated start-code based markers for signalling of tile entry points (JCTVC-F594)

- Made the following clean-up changes:
  - Addressing editing notes and cross references in subclauses 0.2, 0.3, 0.6, 0.7 and Clause 3
  - Various changes to the definitions, including
    - Correcting of the definition of "IDR picture", as marking all reference pictures as "unused for reference" is not anymore immediately after the decoding of the IDR picture due to the reference picture set based picture buffering mechanism
    - Resolving editing notes related to the definitions of "leading picture" and "output order"
    - Resolving editing notes related to the definitions of "reference picture list (X)" (X being 0 or 1)
    - Correcting for the indention of some bullet items in subclause 8.3.3 (Decoding process for generating unavailable reference pictures)
    - Improving definitions of "tile", "tree"
    - Added a definition of "z-scan"
  - Removing mentioning of redundant pictures and changing "primary (coded) picture" to "(coded) picture", auxiliary (coded) picture, and data partitioning
  - Moving of the sentence "The first coded picture in a bitstream shall be an IDR picture or a CRA picture." from the decoder conformance subclause to the bitstream conformance subclause
- Incorporated the ability, at the slice level, to disable loop filtering across slice boundaries (in response to JCTVC-H0391)

Working Draft 5 of High Efficiency Video Coding.

Ed. Notes (WD5):
- Incorporated weighted prediction (JCTVC-F265)
- Removed CAVLC
- Incorporated wavefront parallel processing (JCTVC-F274)
- Incorporated wavefront CABAC flush (JCTVC-F275)
- Incorporated tiles (JCTVC-F335)
- Removed ClipMv (JCTVC-G134)
- Removed merge partition redundancy check (JCTVC-G681)
- Incorporated simplified merge pruning (JCTVC-G1006)
- Incorporated extend scaling factor clipping to 16 (JCTVC-G223)
- Incorporated amvp position dependency removal (JCTVC-G542)
- Incorporated simplified TMVP refidx derivation (JCTVC-G163)
- Incorporated MaxNumMergeCand signalling in slice header (JCTVC-G091)
- Incorporated modified H and center TMVP positions (JCTVC-G082)
- Incorporated intra smoothing for horizontal and vertical directions (G457)
- Incorporated removal of ALF DC offset (JCTVC-G445)
- Incorporated line buffer elimination (JCTVC-G145)
- Incorporated simplified intra mode mapping (JCTVC-G418/G109/G144)
- Incorporated modified intra mode coding (JCTVC-G119)
- Incorporated luma interpolation filter (JCTVC-G778)
- Incorporated chroma interpolation filter (JCTVC-G778)
- Incorporated simplified intra padding (JCTVC-G812)
- Incorporated modified cRiceParam update (JCTVC-G700)
- Incorporated 8bit init values for CABAC (JCTVC-G633)
- Incorporated harmonized pred and part mode binarization (JCTVC-G1042)
- Incorporated significant map context reduction (JCTVC-G1015)
- Incorporated level chroma context reduction (JCTVC-G783)
- Incorporated diagonal sub-block scan for residual coding (JCTVC-G323)
- Removed NSQT remapping and transform reordering (JCTVC-G1038)
- Incorporated modified last_significant_coeff_x/y coding (JCTVC-G201/G704)
- Incorporated shared chroma CBF contexts (JCTVC-G718)
- Incorporated luma intra mode bypass coding (JCTVC-G767)
- Incorporated multi-level significant map (JCTVC-G644)
- Incorporated WD and HM mismatch for LM prediction (JCTVC-G1034)
- Revert the slice boundary padding for adaptive loop filter to WD4
- Incorporated 4x4 BA classification (JCTVC-G609)
- Incorporated virtual boundary processing (JCTVC-G212)
- Incorporated fixed K-table for ALF (JCTVC-G610)
- Incorporated removing 15th merge flag for BA mode in ALF (JCTVC-G216)
- Incorporated prediction of ALF coefficients (JCTVC-G665)

- Incorporated deblocking clean-up (JCTVC-G1035/G620)
- Revert the deblocking decision to HM3 (JCTVC-G088)
- Incorporated support of varying QP in deblocking (JCTVC-G1031)
- Incorporated reducing motion data line buffers (JCTVC-G229)
- Incorporated reference picture set (RPS) (JCTVC-G1002)
- Incorporated reference picture set prediction (JCTVC-G198)
- Incorporated separate decisions for each half (4 lines) of a length 8 block boundaries (JCTVC-G590)
- Incorporated BoG on deblocking fix (JCTVC-G1035)
- Incorporated core transform (JCTVC-G495)
- Incorporated clipping at the output of the first inverse transform (JCTVC-G782)
- Incorporated forbidding level values outside of 16b (JCTVC-G719)
- Incorporated reducing cbf flag signalling redundancy (JCTVC-G444)
- Incorporated changing luma/chroma coefficient interleaving from CU to TU level (JCTVC-G381)
- Incorporated defining MaxIPCMCUSize, MinChromaTrafoSize (JCTVC-G112)
- Incorporated harmonization of implicit TU, AMP and NSQT (JCTVC-G519)
- Incorporated improved weighted prediction (JCTVC-G065)
- Incorporated redundancy removal of explicit weighted prediction syntax (JCTVC-G441)
- Incorporated non-cross-tiles loop filtering for independent tiles (JCTVC-G194)
- Incorporated low latency CABAC initialization for dependent tiles (JCTVC-G197)
- Incorporated AVC-based quantization matrices syntax (JCTVC-G434)
- Incorporated HVS-based quantization matrices (JCTVC-G880)
- Incorporated APS quantization matrices and parameter set extension syntax (JCTVC-G1016)
- Incorporated nal_unit_type value of 14 for APS
- Incorporated SPS syntax for chroma_format_idc from AVC
- Incorporated pure VLC for SAO and ALF (JCTVC-G220)
- Moved slice address and put slice_type and cabac_init_idc into slice and entropy slice header. (JCTVC-G1025)
- Incorporated picture width and height coding using ue(v) rather than u(16) (JCTVC-G325)
- Incorporated ALF and SAO flags in slice header (JCTVC-G566)
- Incorporated marking process for non-TMVP pictures (JCTVC-G398)
- Incorporated max_dec_frame_buffering, num_reorder_frames, and use max_latency_increase (JCTVC-G546)
- Incorporated high level syntax clean up (JCTVC-G507)
- Incorporated chroma QP offset (JCTVC-G509)

Working Draft 4 of High Efficiency Video Coding.

Ed. Notes (WD4):
- Removed inferred merge (JCTVC-F082)
- Incorporated slice header flag to disable 4x4 inter partitions (JCTVC-F744)
- Incorporated modified rounding in MV scaling (JCTVC-F142)
- Removed intermediate amvp spatial candidates redundancy check (JCTVC-F050)
- Incorporated reducing the number of spatial mv scalings to 1 (JCTVC-F088)
- Incorporated spatial merge candidate positions unification (JCTVC-F419)
- Incorporated one reference list check for temporal mvp (JCTVC-F587)
- Incorporated AMVP/merge parsing robustness with simplifications (JCTVC-F470)
- Incorporated unified availability check for intra (JCTVC-F477)
- Incorporated generic interpolation filter (JCTVC-F537)
- Incorporated non-square quadtree transform NSQT (JCTVC-F412)
- Incorporated asymmetric motion partitions AMP (JCTVC-F379)
- Incorporated CBF redundancy reduction (JCTVC-C277)
- Incorporated modified delta QP binarization (JCTVC-F745)
- Incorporated diagonal coefficient scanning in CABAC (JCTVC-F129)
- Incorporated parallel context processing for coefficient levels in CABAC (JCTVC-F130)
- Incorporated context sharing for significant_coeff_flag of 16x16 and 32x32 transforms (JCTVC-F132)
- Incorporated unified scans (JCTVC-F288)
- Incorporated sample adaptive offset (JCTVC-E049)
- Incorporated sample adaptive offset for chroma (JCTVC-F057)
- Incorporated sample adaptive offset offset accuracy (JCTVC-F396)
- Incorporated updated ALF slice padding (JCTVC-D128)
- Incorporated updated ALF slice padding due to ALF filter shape change (JCTVC-F303/F042)
- Incorporated updated ALF slice padding due to unified luma and chroma filter shapes (JCTVC-F157)

- Incorporated ALF filter using subset of pixels (JCTVC-F301)
- Incorporated modified deblocking process for luma (JCTVC-F118)
- Incorporated modified tc_offset in deblocking process (JCTVC-F143)
- Incorporated MDIS and pixel position change of planar (JCTVC-F483)
- Incorporated availability check removal for intra DC filtering (JCTVC-F178)
- Incorporated size-independent intra DC filtering (JCTVC-F252)
- Incorporated modified MDIS table (JCTVC-F126)
- Incorporated simplified intra_FromLuma prediction (JCTVC-F760)
- Incorporated fixed number of MPM (JCTVC-F765)
- Incorporated SAO boundary processing (JCTVC-F232)
- Minor bug in deriving sample positions in SAO process was fixed
- Bug in coding tree syntax table related to the initialization of variable IsCuQpDeltaCoded was fixed
- Incorporated modified last significant coefficient position coding in CABAC (JCTVC-F375)
- Incorporated modified mvd coding in CABAC (JCTVC-F455)
- Incorporated reduced number of contexts in CABAC (JCTVC-F746)
- Incorporated high-level syntax cleanup (JCTVC-F714)
- Incorporated NAL unit type and CDR (CRA) (JCTVC-F462/464)
- Incorporated adaptation parameter set (APS) (JCTVC-F747)

Ed. Notes (WD3):

- Added Residual coding CABAC syntax and semantics
- Added Zig-zag scanning process
- Added CABAC Binarization processes
- Incorporated MV coding (JCTVC-E481)
- Incorporated Compression of reference indices (JCTVC-E059)
- Incorporated Zero merge candidate (JCTVC-E146)
- Incorporated Intra mode coding (JCTVC-E088/E131) (Inserted by TK 31/3/2011 with notes)
- Fixed the CABAC coefficients syntax, semantics and inverse scanning process
- Incorporated CABAC coeffs (JCTVC-E253)
- Moved the EGk binarization from the UEGk subclause in a separate subclause
- Added text representing CABAC entropy coding context initialization
- Added text representing CABAC entropy coding context derivation.
- Mode-dependent 3- scan for intra (JCTVC-D393)
- Incorporated CABAC: Context size reduction (JCTVC-E227/E489)
- Incorporated CABAC: significance map coding simplification (JCTVC-E227/E338/E344/E494)
- Incorporated CABAC: Contexts for MVD (JCTVC-E324)
- Incorporated initial draft of CAVLC text
- CAVLC for 16x16 & 32x32 (JCTVC-E383)
- CAVLC table size reduction (JCTVC-E384)
- CAVLC for RQT (JCTVC-E404)
- CAVLC: counters (JCTVC-E143)
- CAVLC: Intra prediction mode coding in LCEC (JCTVC-D366)
- CAVLC: Inter prediction mode coding in LCEC (JCTVC- D370)
- CAVLC: 4x4 and 8x8 transform coefficient coding in LCEC (JCTVC- D374)
- Block-based ALF (JCTVC-E046/E323)
- ALF parameters to PPS (JCTVC-E045)
- Parallel deblocking (JCTVC-E496/E181/E224)
- Clipping for bi-pred averaging (JCTVC-E242)
- Reference sample padding (JCTVC-E488)
- Transformation processes are replaced by [TBD] mark (meeting note, JCTVC-E243)
- Sub-LCU-level dQP (JCTVC-E051/E220)
- Temporal layer switching and reference list management based on temporal_id (JCTVC-E279/D081/D200)
- Improved text of entropy slice (JCTVC-D070)
- Slice independent deblocking and adaptive loop filtering (JCTVC-D128)
- Fine-granularity slices (JCTVC-E483)
- PCM mode (JCTVC-E057 and JCTVC-E192)
- CAVLC: Inter pred coding (JCTVC-E381)
- CAVLC: Combined coding of inter prediction direction and reference frame index (JCTVC-D141)

- 4x4 DST (JCTVC-E125)
- Planar mode (JCTVC-E321)
- Luma-based chroma intra prediction (JCTVC-E266)
- Modification of DC predictor (JCTVC-E069)
- Bug in mapping table and the corresponding text for mostProbableIntra was fixed. (64x64 uses 3-directions, but the table was specified for 5-directions)
- Non-existing cases of Intra_DC, Intra_Planar and Intra_FromLuma are removed (due to reference sample padding, JCTVC-E488)

Ed. Notes (WD2):

- Incorporated Partial Merging according to JCTVC-D441
    - removed direct mode
    - moved merge to prediction_unit and added candidates
    - added partial merge restrictions
    - inter NxN partitioning only for smallest coding_unit
- Updated transform_tree and transform_coeff syntax
- Added transform_coeff to coding_unit syntax (Fix)
- Incorporated intra NxN partitioning only for smallest coding_unit according to JCTVC-D432
- Incorporated modified temporal motion vector predition according to JCTVC-D164
- Incorporated simplified motion vector prediction according to JCTVC-D231
    - removed median
    - removed pruning process
    - changed the selection manner of left/top predictor
- 8-tap luma interpolation filter according to JCTVC-D344
- 4-tap chroma interpolation filter according to JCTVC-D347
- Improved deblocking filter text according to JCTVC-D395
- IBDI syntax is removed
- Updated syntax and semantics
    - Two tool-enabling flags (adaptive_loop_filter_enabled_flag and cu_qp_delta_enabled_flag) are added in SPS according to software. However, low_delay_coding_enabled_flag is not added – it could be handled by more general reference management scheme. merging_enabled_flag is not added – partial merging (JCTVC-D441) was adopted thus merging cannot be turned off any more. amvp_mode[] is not added since amvp cannot be turned off any more due to absence of median predictor (JCTVC-D231). Note that software has all switches.
    - cu_qp_delta (coding unit layer), syntax and semantics are added. (JCTVC-D258)
    - collocated_from_l0 (slice header), syntax and semantics are added.
- Clean decoding refresh (CDR) (JCTVC-D234).
- Temporal motion vector memory compression (JCTVC-D072)
- Constrained intra prediction (JCTVC-D086)
- Mode-dependent intra smoothing (JCTVC-D282)
- Merging chroma intra prediction process into luma intra prediction process
- Combined reference list (JCTVC-D421)
- Chroma intra prediction mode reordering (JCTVC-D255/D278/D166)
- Adaptive loop filter text is added
- Entropy slice is added (JCTVC-D070)
- High precision bi-directional averaging (JCTVC-D321)
- Reduction of number of intra prediction modes for 64x64 blocks (JCTVC-D100)
- Misc.
    - TPE bits are reduced from 4 to 2
    - Clipping is applied to (temporally) scaled mv – revisit

Ed. Notes (WD1):

- Incorporated the decisions on high-level syntax according to JCTVC-B121
- Incorporated text from JCTVC-B205revision7
- Incorporated text from JCTVC-C319 (as found to be stable)
- Revised coding tree, coding unit and prediction unit syntaxes (coding tree syntax is newly added. needs to be confirmed)
- Initial drafting of decoding process of coding units in intra prediction mode (luma part, JCTVC-B100 and JCTVC-C042)
- Initial drafting of decoding process of coding units in inter prediction mode

- Initial drafting of scaling and transformation process
- Added text, transform 16T and 32T
- Initial drafting of deblocking process
- Improving the text, derivation process for motion vector components and reference indices
- Added text, boundary filtering strength

Open issues:

- Should support for monochrome, 4:2:2 and 4:4:4 (with and w/o separate colour planes) be included from the start? Currently, it has been left in the text as it doesn't seem to affect much text.
- Use of bin string and bit string should be consistent.
- Improve text quality by considering: strict use of terms "unit" and "block" – block = a rectangular 2D array (one component), unit = collective term for specifying information for both luma and chroma. Don’t use ther term ‘unit’ by itself – always use the term ‘unit’ with prefix – coding unit, prediction unit or transform unit.
- Both variables IntraPredMode and IntraPredModeC are used in the syntax table but the actual derivations are specified in the decoding process. Maybe it’s better to move them to the semantics section.
- Binarization of intra_chroma_pred_mode reflecting codeword switching and luma-based chroma intra prediction (JCTVC-E266) is missing.
- Clarification on the use of intra_chroma_pred_mode and IntraPredModeC is needed. The former specifies the syntax item to indicate how to determine the chroma intra prediction mode (IntraPredModeC) as 0 (to Intra_FromLuma), 1 (to Intra_Vertical), 2 (to Intra_Horizontal), 3 (Intra_DC or Intra_Planar) or 4 (re-use luma mode). The latter specifies chroma intra prediction mode, which is actually mapped to the specific prediction process.
- Software-text mismatch of JCTVC-F252: SW applies DC filtering to 64x64 intra block but WD does not.
- There are input/output parameter inconsistencies between function call and actual functions of intra prediction. Function call uses sample position, block, size and chroma index while some functions do not use some parameters.

CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

Foreword

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

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for world-wide standardization. National Bodies that are members of ISO and IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75% of the national bodies casting a vote.

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

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

INTERNATIONAL STANDARD

ITU-T RECOMMENDATION

High efficiency video coding

0 Introduction

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

0.1 Prologue

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

As the costs for both processing power and memory have reduced, network support for coded video data has diversified, and advances in video coding technology have progressed, the need has arisen for an industry standard for compressed video representation with substantially increased coding efficiency and enhanced robustness to network environments. Toward these ends the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG) formed a Joint Collaborative Team on Video Coding (JCT-VC) in 2010 for development of a new Recommendation | International Standard. This Recommendation | International Standard was developed in the JCT-VC.

0.2 Purpose

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

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

0.3 Applications

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

This Recommendation | International Standard is designed to cover a broad range of applications for video content including but not limited to the following:

CATV	Cable TV on optical networks, copper, etc.
DBS	Direct broadcast satellite video services
DSL	Digital subscriber line video services
DTTB	Digital terrestrial television broadcasting
ISM	Interactive storage media (optical disks, etc.)
MMM	Multimedia mailing
MSPN	Multimedia services over packet networks (video streaming, etc.)
RTC	Real-time conversational services (videoconferencing, videophone, etc.)
RVS	Remote video surveillance
SSM	Serial storage media (digital VTR, etc.)

[Ed. (TK): Should we update this list of application areas? Perhaps with the list from the Requirements document? (GJS): Probably a good idea. Which Requirements document?]

0.4 Publication and versions of this Specification

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

This Specification has been jointly developed by ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG). It is published as technically-aligned twin text in both ITU-T and ISO/IEC. As the basis text has been drafted to become both an ITU-T Recommendation and an ISO/IEC International Standard, the term

"Specification" (with capitalization to indicate that it refers to the whole of the text) is used herein when the text refers to itself.

This is the first version of this Specification. Additional versions are anticipated.

0.5 Profiles, tiers and levels

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

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

Considering the practicality of implementing the full syntax of this Specification, however, a limited number of subsets of the syntax are also stipulated by means of "profiles", "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 by this Recommendation | International Standard. Within the bounds imposed by the syntax of a given profile it is still possible to require a very large variation in the performance of encoders and decoders depending upon the values taken by syntax elements in the bitstream such as the specified size of the decoded pictures. In many applications, it is currently neither practical nor economic to implement a decoder capable of dealing with all hypothetical uses of the syntax within a particular profile.

In order to deal with this problem, "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.

0.6 Overview of the design characteristics

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

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

0.7 How to read this Specification

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

It is suggested that the reader starts with clause 1 (Scope) and moves on to clause 3 (Definitions). Clause 6 should be read for the geometrical relationship of the source, input, and output of the decoder. Clause 7 (Syntax and semantics) specifies the order to parse syntax elements from the bitstream. See subclauses 7.1–7.3 for syntactical order and see subclause 7.4 for semantics; 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 Specification, the reader should refer to clauses 2 (Normative references), 4 (Abbreviations), and 5 (Conventions) as needed. Annexes A through E also form an integral part of this Recommendation | International Standard.

Annex A 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 message payloads. Annex E specifies syntax and semantics of the video usability information parameters of the sequence parameter set.

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

1 Scope

This document specifies High efficiency video coding.

2 Normative references

2.1 General

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

2.2 Identical Recommendations | International Standards

– None.

2.3 Paired Recommendations | International Standards equivalent in technical content

– 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*.

[Ed. (GJS): Delete this note. This section is for normative references. We reference AVC here in order to refer to it to specify aspects such as SEI messages that are referred to here rather than being copied as text here. There is no need to include self-references here.]

2.4 Additional references

– ISO 11664-1, *Colorimetry — Part 1: CIE standard colorimetric observers.*

3 Definitions

[Ed. (TW) Needs more work including turning them into 1 sentence each.]

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

3.1 **access unit**: 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*.

NOTE 1 – In addition to containing the coded slice segment NAL units of the coded picture, an access unit may also contain other NAL units not containing slice segments of the coded picture. The decoding of an access unit always results in a decoded picture.

3.2 **AC transform coefficient**: Any *transform coefficient* for which the *frequency index* in one or both dimensions is non-zero.

3.3 **associated non-VCL NAL unit**: A *non-VCL NAL unit* for which a particular *VCL NAL unit* is the *associated VCL NAL unit* of the *non-VCL NAL unit*.

3.4 **associated RAP picture**: The previous *RAP picture* in *decoding order* (if present).

3.5 **associated VCL NAL unit**: For *non-VCL NAL units* with nal_unit_type equal to EOS_NUT, EOB_NUT, FD_NUT, or SUFFIX_SEI_NUT, or in the range of RSV_NVCL45..RSV_NVCL47, or in the range of UNSPEC48..UNSPEC63, the preceding *VCL NAL unit* in *decoding order*; and for *non-VCL NAL units* with nal_unit_type equal to other values, the next *VCL NAL unit* in *decoding order*.

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

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

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

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

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

3.11 **bitstream**: A sequence of bits, in the form of a *NAL unit stream* or a *byte stream*, that forms the representation of *coded pictures* and associated data forming one or more *coded video sequences*.

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

3.13 **broken link**: A location in a *bitstream* at which it is indicated that some subsequent *pictures* in *decoding order* may contain serious visual artefacts due to unspecified operations performed in the generation of the *bitstream*.

3.14 **broken link access (BLA) access unit**: An *access unit* in which the *coded picture* is a *BLA picture*.

3.15 **broken link access (BLA) picture**: A *RAP picture* for which each *slice segment* has nal_unit_type equal to BLA_W_LP, BLA_W_DLP or BLA_N_LP.

NOTE 2 – A BLA picture contains only I slices, and may be the first picture in the bitstream in decoding order, or may appear later in the bitstream. Each BLA picture begins a new coded video sequence, and has the same effect on the decoding process as an IDR picture. However, a BLA picture contains syntax elements that specify a non-empty reference picture set. When a BLA picture has nal_unit_type equal to BLA_W_LP, it may have associated RASL pictures, which are not output by the decoder and may not be decodable, as they may contain references to pictures that are not present in the bitstream. When a BLA picture has nal_unit_type equal to BLA_W_LP, it may also have associated RADL pictures, which are specified to be decoded. When a BLA picture has nal_unit_type equal to BLA_W_DLP, it does not have associated RASL pictures but may have associated RADL pictures, which are specified to be decoded. When a BLA picture has nal_unit_type equal to BLA_N_LP, it does not have any associated leading pictures.

3.16 **buffering period**: The set of *access units* starting with an *access unit* that contains a buffering period SEI message and containing all subsequent *access units* in *decoding order* up to but not including the next *access unit* (when present) that contains a buffering period SEI message.

3.17 **byte**: A sequence of 8 bits, written and read with the most significant bit on the left and the least significant bit on the right. When represented in a sequence of data bits, the most significant bit of a byte is first.

3.18 **byte-aligned**: A position in a *bitstream* is byte-aligned when the position is an integer multiple of 8 bits from the position of the first bit in the *bitstream*, and a bit or *byte* or *syntax element* is said to be byte-aligned when the position at which it appears in a *bitstream* is byte-aligned.

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

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

3.21 **chroma**: An adjective specifying that a sample array or single sample is representing one of the two colour difference signals related to the primary colours. The symbols used for a chroma array or sample are Cb and Cr.

NOTE 3 – The term chroma is used rather than the term chrominance in order to avoid the implication of the use of linear light transfer characteristics that is often associated with the term chrominance.

3.22 **clean random access (CRA) access unit**: An *access unit* in which the *coded picture* is a *CRA picture*.

3.23 **clean random access (CRA) picture**: A *RAP picture* for which each *slice segment* has nal_unit_type equal to CRA_NUT.

NOTE 4 – A CRA picture contains only I slices, 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 is the first picture in the bitstream in decoding order, the CRA picture is the first picture of a coded video sequence in decoding order, and any associated RASL pictures are not output by the decoder and may not be decodable, as they may contain references to pictures that are not present in the bitstream.

3.24 **coded picture**: A *coded representation* of a *picture* containing all *coding tree units* of the *picture*.

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

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

3.27 **coded slice segment NAL unit**: A *NAL unit* that has nal_unit_type in the range of 1 to 14, inclusive, which indicates that the *NAL unit* contains a coded *slice segment*.

3.28 **coded video sequence**: A sequence of *access units* that consists, in decoding order, of a *CRA access unit* that is the first access unit in the bitstream, an *IDR access unit* or a *BLA access unit*, followed by zero or more non-IDR and non-BLA *access units* including all subsequent *access units* up to but not including any subsequent *IDR or BLA access unit*.

3.29 **coding block**: An NxN *block* of samples for some value of N such that the division of a *coding tree block* into *coding blocks* is a *partitioning*.

3.30 **coding tree block**: An NxN *block* of samples for some value of N. The division of one of the arrays that compose a *picture* that has three sample arrays or of the array that compose a *picture* in monochrome format or a *picture* that is coded using three separate colour planes into *coding tree blocks* is a *partitioning*. [Ed. (GJS): Multi-sentence.]

3.31 **coding tree unit**: A *coding tree block* of *luma* samples, two corresponding *coding tree blocks* of *chroma* samples of a *picture* that has three sample arrays, or a *coding tree 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. The division of a *slice* into *coding tree units* is a *partitioning*. [Ed. (YK): Multi-sentence.]

3.32 **coding unit**: 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. The division of a *coding tree unit* into *coding units* is a *partitioning*. [Ed. (YK): Multi-sentence.]

3.33 **column**: An integer number of *coding tree blocks*. *Columns* are delineated from one another by vertical boundaries that extend from the top boundary to the bottom boundary of the *picture* and are ordered consecutively from left to right in the *picture*. The division of each *picture* into *columns* is a *partitioning*. [Ed. (YK): Multi-sentence. (GJS): Also, I believe we also use the term in the ordinary sense of a column of entries in an array. If we need a special term, it should be something like CTB column.]

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

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

3.36 **decoded picture**: A *decoded picture* is derived by decoding a *coded picture*.

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

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

3.39 **decoder under test (DUT)**: A *decoder* that is tested for conformance to this Specification by operating the *hypothetical stream scheduler* to deliver a conforming *bitstream* to the *decoder* and to the *hypothetical reference decoder* and comparing the values and timing or order of the output of the two *decoders*.

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

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

3.42 **decoding unit**: An *access unit* if SubPicCpbFlag is equal to 0 or a subset of an *access unit* otherwise, consisting of one or more *VCL NAL units* in an *access unit* and the *associated non-VCL NAL units*.

3.43 **dependent slice segment**: A *slice segment* for which the values of some *syntax elements* of the *slice segment header* are inferred from the values for the preceding *independent slice segment* in *decoding order*.

3.44 **display process**: A process not specified in this Specification having, as its input, the cropped decoded *pictures* that are the output of the *decoding process*.

3.45 **elementary stream**: A sequence of one or more *bitstreams*.

NOTE 5 – An elementary stream that consists of two or more bitstreams would typically have been formed by splicing together two or more bitstreams (or parts thereof).

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

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

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

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

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

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

3.52 **hypothetical reference decoder (HRD)**: A hypothetical *decoder* model that specifies constraints on the variability of conforming *NAL unit streams* or conforming *byte streams* that an encoding process may produce.

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

3.54 **independent slice segment**: A *slice segment* for which the values of the *syntax elements* of the *slice segment header* are not inferred from the values for a preceding *slice segment*.

3.55 **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.

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

3.57 **instantaneous decoding refresh (IDR) picture**: A *RAP picture* for which each *slice segment* has nal_unit_type equal to IDR_W_DLP or IDR_N_LP.

NOTE 6 – An IDR picture contains only I slices, 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 coded video sequence in decoding order. When an IDR picture has nal_unit_type equal to IDR_W_DLP, it may have associated RADL pictures. When an IDR picture has nal_unit_type equal to IDR_N_LP, it does not have any associated leading pictures. An IDR picture does not have associated RASL pictures.

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

3.59 **inter prediction**: A *prediction* derived in a manner that is dependent on data elements (e.g. sample values or motion vectors) of *reference pictures* other than the current *decoded picture*.

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

3.61 **intra prediction**: A *prediction* derived from only data elements (e.g. sample values) of the same decoded *slice*.

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

3.63 **inverse transform**: A part of the *decoding process* by which a set of *transform coefficients* are converted into spatial-domain values.

3.64 **layer**: A set of *VCL NAL units* that all have a particular value of nuh_reserved_zero_6bits and the *associated non-VCL NAL units*, or one of a set of syntactical structures in a non-branching hierarchical relationship.

NOTE 7 – Depending on the context, either the first layer concept or the second layer concept applies. The first layer concept is also referred to as a scalable layer, wherein a layer may be a spatial scalable layer, a quality scalable layer, a view, etc. A temporal true subset of a scalable layer, or a sub-layer, is not a layer. The second layer concept is also referred to as a coding layer, wherein higher layers contain lower layers, and the coding layers are the coded video sequence, picture, slice, slice segment, and coding tree unit layers.

3.65 **leading picture**: A *picture* that precedes the *associated RAP picture* in *output order*.

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

3.67 **level**: A defined set of constraints on the values that may be taken by the *syntax elements* and variables of this Specification. The same set of levels is defined for all *profiles*, with most aspects of the definition of each level being in common across different *profiles.* Individual implementations may, within specified constraints,

support a different level for each supported *profile*. In a different context, level is the value of a *transform coefficient* prior to *scaling*. [Ed. (YK): Multi-sentence.]

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

3.69 **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*).

3.70 **long-term reference picture**: A *picture* that is marked as "used for long-term reference".

3.71 **long-term reference picture set**: The two reference picture set lists that may contain long-term reference pictures.

3.72 **luma**: An adjective specifying that a sample array or single sample is representing the monochrome signal related to the primary colours. The symbol or subscript used for luma is Y or L.

NOTE 8 – The term luma is used rather than the term luminance in order to avoid the implication of the use of linear light transfer characteristics that is often associated with the term luminance. The symbol L is sometimes used instead of the symbol Y to avoid confusion with the symbol y as used for vertical location.

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

NOTE 9 – In some places where the optional nature of the described behaviour is intended to be emphasized, the phrase "may or may not" is used to provide emphasis.

3.74 **motion vector**: A two-dimensional vector used for *inter prediction* that provides an offset from the coordinates in the *decoded picture* to the coordinates in a *reference picture*.

3.75 **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).

3.76 **network abstraction layer (NAL) unit**: A *syntax structure* containing an indication of the type of data to follow and *bytes* containing that data in the form of an *RBSP* interspersed as necessary with *emulation prevention bytes*.

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

3.78 **non-reference picture**: A *picture* that is marked as "unused for reference".

NOTE 10 – A non-reference picture contains samples that cannot be used for inter prediction in the decoding process of subsequent pictures in decoding order.

3.79 **non-VCL NAL unit**: A *NAL unit* that is not a *VCL NAL unit*.

3.80 **note**: A term that is used to prefix *informative* remarks (used exclusively in an *informative* context).

3.81 **operation point**: A *bitstream* created from another *bitstream* by operation of the *sub-bitstream extraction process*.

3.82 **operation point set**: A set of *operation points* that have the same set of nuh_reserved_zero_6bits values in the corresponding *bitstreams*.

3.83 **output order**: The order in which the *decoded pictures* are output from the *decoded picture buffer* (for the *decoded pictures* that are to be output from the *decoded picture buffer*).

3.84 **parameter**: A *syntax element* of a *video parameter set*, *sequence parameter set* or *picture parameter set*, or the second word of the defined term *quantization parameter*.

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

3.86 **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.

3.87 **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 *slice segment header*.

3.88 **picture order count**: A variable that is associated with each *picture* to be output from the *decoded picture buffer* that indicates the position of the associated *picture* in *output order* relative to the *output order* positions of the other *pictures* to be output from the *decoded picture buffer* in the same *coded video sequence*.

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

3.90 **prediction block**: A rectangular MxN *block* of samples on which the same *prediction* is applied. The division of a *coding block* into *prediction blocks* is a *partitioning*. [Ed. (YK): Multi-sentence.]

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

3.92 **prediction unit**: A *prediction block* of *luma* samples, two corresponding *prediction blocks* of *chroma* samples of a *picture* that has three sample arrays, or a *prediction block* of samples of a monochrome *picture* or a *picture* that is coded using three separate colour planes and *syntax structures* used to predict the *prediction block* samples.

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

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

3.95 **prefix SEI NAL unit**: An *SEI NAL unit* that has nal_unit_type equal to PREFIX_SEI_NUT.

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

3.97 **quadtree**: A *tree* in which a parent node can be split into four child nodes. A child node may become parent node for another split into four child nodes. [Ed. (YK): Multi-sentence.]

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

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

3.100 **random access decodable leading (RADL) access unit**: An *access unit* in which the *coded picture* is a *RADL picture*.

3.101 **random access decodable leading (RADL) picture**: A *coded picture* for which each *slice segment* has nal_unit_type equal to RADL_NUT.

NOTE 11 – All RADL pictures are leading pictures. RADL pictures are not used as reference pictures for the decoding process of trailing pictures of the same associated RAP picture. When present, all RADL pictures precede, in decoding order, all trailing pictures of the same associated RAP picture.

3.102 **random access point (RAP) access unit**: An *access unit* in which the *coded picture* is a *RAP picture*.

3.103 **random access point (RAP) picture**: A coded *picture* for which each *slice segment* has nal_unit_type in the range of 7 to 12, inclusive.

NOTE 12 – A RAP picture contains only I slices, and may be a BLA picture, a CRA picture or an IDR picture. The first picture in the bitstream must be a RAP picture. Provided the necessary parameter sets are available when they need to be activated, the RAP picture and all subsequent non-RASL pictures in decoding order can be correctly decoded without performing the decoding process of any pictures that precede the RAP picture in decoding order. There may be pictures in a bitstream that contain only I slices that are not RAP pictures.

3.104 **random access skipped leading (RASL) access unit:** An *access unit* in which the *coded picture* is a *RASL picture.*

3.105 **random access skipped leading (RASL) picture:** A *coded picture* for which each *slice segment* has nal_unit_type equal to RASL_NUT.

NOTE 13 – All RASL pictures are leading pictures of an associated BLA or CRA picture. When the associated RAP picture is a BLA picture or is the first coded picture in the bitstream, 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 present, all RASL pictures precede, in decoding order, all trailing pictures of the same associated RAP picture.

3.106 **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.

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

3.108 **raw byte sequence payload (RBSP) stop bit:** A bit equal to 1 present within a *raw byte sequence payload (RBSP)* after a *string of data bits*. The location of the end of the *string of data bits* within an *RBSP* can be identified by searching from the end of the *RBSP* for the *RBSP stop bit*, which is the last non-zero bit in the *RBSP*. [Ed. (GJS): Multi-sentence definition.]

3.109 **recovery point**: A point in the *bitstream* at which the recovery of an exact or an approximate representation of the *decoded pictures* represented by the *bitstream* is achieved after a *random access* or *broken link*.

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

3.111 **reference picture**: A *picture* that is a *short-term reference picture* or a *long-term reference picture*.

NOTE 14 – A reference picture contains samples that may be used for inter prediction in the decoding process of subsequent pictures in decoding order.

3.112 **reference picture list**: A list of *reference pictures* that is used for *inter prediction* of a *P* or *B slice.* For the *decoding process* of a *P slice*, there is one reference picture list – *reference picture list 0.* For the *decoding process* of a *B slice*, there are two reference picture lists – *reference picture list 0* and *reference picture list 1*. [Ed. (GJS): Multi-sentence definition.]

3.113 **reference picture list 0**: A *reference picture list* used for *inter prediction* of a *P* or *B slice*. All *inter prediction* used for *P slices* uses reference picture list 0. Reference picture list 0 is one of two *reference picture lists* used for *bi-prediction* for a *B slice*, with the other being *reference picture list 1*. [Ed. (GJS): This definition refers to *bi-prediction* in italics, but there is no definition of that term.] [Ed. (GJS): It may be beneficial to add a definition of "uni-prediction" or "single-list prediction" to refer to inter prediction that is not bi-prediction.] [Ed. (GJS): The definition should avoid giving the impression that all inter prediction in a B slice is bi-prediction.] [Ed. (GJS): Multi-sentence definition.]

3.114 **reference picture list 1**: A *reference picture list* used for *inter prediction* of a *B slice*. Reference picture list 1 is one of two *reference picture lists* used for *bi-prediction* for a *B slice*, with the other being *reference picture list 0*. [Ed. (GJS): This definition refers to *bi-prediction* in italics, but there is no definition of that term.] [Ed. (GJS): It may be beneficial to add a definition of "uni-prediction" or "single-list prediction" to refer to inter prediction that is not bi-prediction.] [Ed. (GJS): The definition should avoid giving the impression that all inter prediction in a B slice is bi-prediction.] [Ed. (GJS): The definition should avoid giving the impression that reference picture list 1 is only used for bi-prediction.] [Ed. (GJS): Multi-sentence definition.]

3.115 **reference picture set**: A set of *reference pictures* associated with a *picture*, consisting of all *reference pictures* that are prior to the associated *picture* in decoding order, that may be used for *inter prediction* of the associated *picture* or any *picture* following the associated *picture* in *decoding order*.

3.116 **reserved**: The term reserved, when used in the clauses specifying some values of a particular *syntax element*, are for future use by ITU-T | ISO/IEC. These values shall not be used in *bitstreams* conforming to this version of this Specification, but may be used in future extensions of this Specification by ITU-T | ISO/IEC. [Ed. (GJS): Multi-sentence definition.]

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

3.118 **row**: An integer number of *coding tree blocks*. *Rows* are delineated from one another by horizontal boundaries that extend from the left boundary to the right boundary of the *picture* and are ordered consecutively from top to bottom in the *picture*. The division of each *picture* into *rows* is *a partitioning*. [Ed. (GJS): Multi-sentence. (GJS): Also, I believe we also use the term in the ordinary sense of a row of entries in an array. If we need a special term, it should be something like CTB column.]

3.119 **sample aspect ratio**: Specifies, for assisting the *display process*, which is not specified in this Specification, the ratio between the intended horizontal distance between the columns and the intended vertical distance between the rows of the *luma* sample array in a *picture*. Sample aspect ratio is expressed as *h*:*v*, where *h* is horizontal width and *v* is vertical height (in arbitrary units of spatial distance). [Ed. (GJS): Multi-sentence.]

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

3.121 **sequence parameter set (SPS)**: A *syntax structure* containing *syntax elements* that apply to zero or more entire *coded video sequences* as determined by the content of a *syntax element* found in the *picture parameter set* referred to by a *syntax element* found in each *slice segment header*.

3.122 **shall**: A term used to express mandatory requirements for conformance to this Specification. 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 output *pictures* to those output from the *decoding process* described herein conforms to the *decoding process* requirements of this Specification. [Ed. (GJS): Multi-sentence definition.]

3.123 **short-term reference picture**: A *picture* that is marked as "used for short-term reference".

3.124 **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.

3.125 **slice**: An integer number of *coding tree units* contained in one *independent slice segment* and all subsequent *dependent slice segments* (if any) that precede the next *independent slice segment* (if any) within the same *access unit*.

3.126 **slice header**: The *slice segment header* of the *independent slice segment* that is a current *slice segment* or is the *independent slice segment* that precedes a current *dependent slice segment*.

3.127 **slice segment**: An integer number of *coding tree units* ordered consecutively in the *tile scan* and contained in a single *NAL unit*; the division of each *picture* into slice segments is a *partitioning*.

3.128 **slice segment header**: A part of a coded *slice segment* containing the data elements pertaining to the first or all *coding tree units* represented in the *slice segment*.

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

3.130 **start code prefix**: A unique sequence of three *bytes* equal to 0x000001 embedded in the *byte stream* as a prefix to each *NAL unit.* The location of a start code prefix can be used by a *decoder* to identify the beginning of a new *NAL unit* and the end of a previous *NAL unit*. Emulation of start code prefixes is prevented within *NAL units* by the inclusion of *emulation prevention bytes*. [Ed. (GJS): Multi-sentence definition – just move the 2nd and 3rd sentences somewhere else.]

3.131 **step-wise temporal sub-layer access (STSA) access unit**: An *access unit* in which the *coded picture* is an *STSA picture*.

3.132 **step-wise temporal sub-layer access (STSA) picture**: A *coded picture* for which each *slice segment* has nal_unit_type equal to STSA_R or STSA_N.

NOTE 15 – 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 sub-layer containing the STSA picture, from the immediately lower sub-layer. STSA pictures must have TemporalId greater than 0.

3.133 **string of data bits (SODB)**: A sequence of some number of bits representing *syntax elements* present within a *raw byte sequence payload* prior to the *raw byte sequence payload stop bit.* Within an SODB, the left-most bit is considered to be the first and most significant bit, and the right-most bit is considered to be the last and least significant bit. [Ed. (YK): Multi-sentence.]

3.134 **sub-bitstream extraction process**: A specified process by which a set of *NAL units* is selected from a *bitstream* based on the associated TemporalId variable and nuh_reserved_zero_6bits syntax element values.

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

3.136 **sub-layer non-reference picture**: A *picture* that contains samples that cannot be used for *inter prediction* in the *decoding process* of subsequent *pictures* of the same and lower *sub-layers* in *decoding order*.

3.137 **sub-layer representation**: A subset of the *bitstream* consisting of *NAL units* of a particular *sub-layer* and the lower *sub-layers*.

3.138 **suffix SEI NAL unit**: An *SEI NAL unit* that has nal_unit_type equal to SUFFIX_SEI_NUT.

3.139 **supplemental enhancement information (SEI) NAL unit**: A *NAL unit* that has nal_unit_type equal to PREFIX_SEI_NUT or SUFFIX_SEI_NUT.

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

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

3.142 **temporal sub-layer access (TSA) access unit:** An *access unit* in which the *coded picture* is a *TSA picture*.

3.143 **temporal sub-layer access (TSA) picture**: A *coded picture* for which each *slice segment* has nal_unit_type equal to TSA_R or TSA_N.

NOTE 16 – A TSA picture and pictures following the TSA picture in decoding do not use pictures with TemporalId equal to or greater than that of the TSA picture for inter prediction reference. A TSA picture enables up-switching, at the TSA picture, to the sub-layer containing the TSA picture or any higher sub-layer, from the immediately lower sub-layer. TSA pictures must have TemporalId greater than 0.

3.144 **temporal sub-layer**: A temporal scalable layer of a temporal scalable *bitstream* consisting of *VCL NAL units* with a particular value of TemporalId and the associated *non-VCL NAL units.* [Ed. (GJS): I don't like having a variable name in a definition.]

3.145 **tier**: [Ed. (YK): Add a definition of "tier", which was mentioned in subclause 0.5.]

3.146 **tile**: An integer number of *coding tree blocks* co-occurring in one *column* and one *row*, ordered consecutively in *coding tree block raster scan* of the *tile*. The division of each *picture* into *tiles* is a *partitioning*. *Tiles* in a *picture* are ordered consecutively in tile *raster scan* of the *picture*. [Ed. (YK): Multi-sentence.]

3.147 **tile scan**: A specific sequential ordering of *coding tree blocks partitioning* a *picture*. The tile scan traverses the *coding tree blocks* in *coding tree block raster scan* within a *tile* and traverses *tiles* in *tile raster scan within a picture*. Although a *slice* contains coding tree blocks that are consecutive in *coding tree block raster scan* of a *tile*, these *coding tree blocks* are not necessarily consecutive in *coding tree block raster scan* of the picture. [Ed. (YK): Multi-sentence.]

3.148 **trailing picture**: A *picture* that follows the *associated RAP picture* in *output order*.

3.149 **transform block**: A rectangular MxN *block* of samples on which the same *transform* is applied. The division of a *coding block* into *transform blocks* is a *partitioning*. [Ed. (YK): Multi-sentence.]

3.150 **transform coefficient**: A scalar quantity, considered to be in a frequency domain, that is associated with a particular one-dimensional or two-dimensional *frequency index* in an *inverse transform* part of the *decoding process*.

3.151 **transform coefficient level**: An integer quantity representing the value associated with a particular two-dimensional frequency index in the *decoding process* prior to *scaling* for computation of a *transform coefficient* value.

3.152 **transform unit**: A *transform block* of *luma* samples of size 8x8, 16x16, or 32x22 or four *transform blocks* of *luma samples* of size 4x4, two corresponding *transform blocks* of *chroma* samples of a *picture* that has three sample arrays, or a *transform block* of *luma* samples of size 8x8, 16x16, or 32x22 or four *transform blocks* of *luma samples* of size 4x4 of a monochrome *picture* or a *picture* that is coded using three separate colour planes and *syntax structures* used to transform the *transform block* samples.

3.153 **tree**: A tree is a finite set of nodes with a unique root node. A terminating node is called a *leaf*.

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

3.155 **unspecified:** The term unspecified, when used in the clauses specifying some values of a particular *syntax element*, indicates that the values have no specified meaning in this Specification and will not have a specified meaning in the future as an integral part of future versions of this Specification. [Ed. (YK): Add a note saying that unspecified values of syntax elements, e.g. nal_unit_type, may be used by external systems specifications that reference this Specification.]

3.156 **video coding layer (VCL) NAL unit**: A collective term for *coded slice segment 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.

3.157 **video parameter set (VPS)**: A *syntax structure* containing *syntax elements* that apply to zero or more entire *coded video sequences* as determined by the content of a *syntax element* found in the *sequence parameter set* referred to by a *syntax element* found in the *picture parameter set* referred to by a *syntax element* found in each *slice segment header*.

3.158 **z-scan order**: A specific sequential ordering of *blocks partitioning* a *picture*. When the *blocks* are of the same size as *coding tree blocks*, the z-scan order is identical to *coding tree block raster scan* of the *picture*. When the *blocks* are of a smaller size than *coding tree blocks*, i.e. *coding tree blocks* are further partitioned into smaller *coding blocks*, the z-scan order traverses from *coding tree block* to *coding tree block* in *coding tree block raster scan* of the *picture*, and inside each *coding tree block*, which may be divided into *quadtrees* hierachically to lower levels, the z-scan order traverses from *quadree* to *quadree* of a particular level in *quadtree*-of-the-particular-level *raster scan* of the *quadtree* of the immediately higher level. [Ed. (YK): Multi-sentence definition.]

4 Abbreviations

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

B Bi-predictive

BLA Broken Link Access

CABAC Context-based Adaptive Binary Arithmetic Coding

CB Coding Block

CBR	Constant Bit Rate
CRA	Clean Random Access
CPB	Coded Picture Buffer
CTB	Coding Tree Block
CTU	Coding Tree Unit
CU	Coding Unit
DPB	Decoded Picture Buffer
DUT	Decoder Under Test
FIFO	First-In, First-Out
GDR	Gradual Decoding Refresh
HRD	Hypothetical Reference Decoder
HSS	Hypothetical Stream Scheduler
I	Intra
IDR	Instantaneous Decoding Refresh
LSB	Least Significant Bit
MSB	Most Significant Bit
NAL	Network Abstraction Layer
P	Predictive
PB	Prediction Block
PPS	Picture Parameter Set
PU	Prediction Unit
RADL	Random Access Decodable Leading (Picture)
RAP	Random Access Point
RASL	Random Access Skipped Leading (Picture)
RBSP	Raw Byte Sequence Payload
SEI	Supplemental Enhancement Information
SODB	String Of Data Bits
SPS	Sequence Parameter Set
STSA	Step-wise Temporal Sub-layer Access
TB	Transform Block
TSA	Temporal Sub-layer Access
TU	Transform Unit
UUID	Universal Unique Identifier
VBR	Variable Bit Rate
VCL	Video Coding Layer
VPS	Video Parameter Set
VUI	Video Usability Information

5 Conventions

5.1 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.

5.2 Arithmetic operators

The following arithmetic operators are defined as follows:

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

5.3 Logical operators

The following logical operators are defined as follows:

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

5.4 Relational operators

The following relational operators are defined as follows:

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

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

5.5 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 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 LSBs as a result of the left shift have a value equal to 0.

5.6 Assignment operators

The following arithmetic operators are defined as follows:

=	Assignment operator.
++	Increment, i.e. x++ is equivalent to $x = x + 1$; when used in an array index, evaluates to the value of the variable prior to the increment operation.
−−	Decrement, i.e. x− − is equivalent to $x = x - 1$; when used in an array index, evaluates to the value of the variable prior to the decrement operation.
+=	Increment by amount specified, i.e. x += 3 is equivalent to x = x + 3, and x += (−3) is equivalent to x = x + (−3).
−=	Decrement by amount specified, i.e. x −= 3 is equivalent to x = x − 3, and x −= (−3) is equivalent to x = x − (−3).

5.7 Range notation

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

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

5.8 Mathematical functions

The following mathematical functions are defined as follows:

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

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

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

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

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

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

Log2(x) the base-2 logarithm of x. (5-7)

Log10(x) the base-10 logarithm of x. (5-8)

$$\text{Min}(x, y) = \begin{cases} x & ; \quad x <= y \\ y & ; \quad x > y \end{cases} \tag{5-9}$$

$$\text{Max}(x, y) = \begin{cases} x & ; \quad x >= y \\ y & ; \quad x < y \end{cases} \tag{5-10}$$

$$\text{Round}(x) = \text{Sign}(x) * \text{Floor}(\text{Abs}(x) + 0.5) \tag{5-11}$$

$$\text{Sign}(x) = \begin{cases} 1 & ; \quad x > 0 \\ 0 & ; \quad x = 0 \\ -1 & ; \quad x < 0 \end{cases} \tag{5-12}$$

$$\text{Sqrt}(x) = \sqrt{x} \tag{5-13}$$

$$\text{Swap}(x, y) = (y, x) \tag{5-14}$$

5.9 Order of operation precedence

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

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

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

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

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

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

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

5.10 Variables, syntax elements, and tables

Syntax elements in the bitstream are represented in **bold** type. Each syntax element is described by its name (all lower case letters with underscore characters), and one descriptor for its method of coded representation. The decoding process

behaves according to the value of the syntax element and to the values of previously decoded syntax elements. When a value of a syntax element is used in the syntax tables or the text, it appears in regular (i.e. not bold) type.

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

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

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

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

Functions that are not syntax functions (including mathematical functions specified in subclause 5.8) 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 s_{yx}. An array construction is denoted by { {...} {...} }, where each inner pair of brackets denote a row of an array in the increasing order of rows. Each inner bracket contains elements of the corresponsing row of an array in the increasing order of columns. [Ed. (GJS): Try to improve the phrasing of this paragraph.]

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

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

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

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

5.11 Text description of logical operations

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

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

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

5.12 Processes

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

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

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

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

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

6.1 Bitstream formats

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

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

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

6.2 Source, decoded, and output picture formats

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

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

The source and decoded pictures are each comprised of one or 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 Annex E.

The variables SubWidthC, and SubHeightC are specified in Table 6-1, depending on the chroma format sampling structure, which is specified through chroma_format_idc and separate_colour_plane_flag. Other values of chroma_format_idc, SubWidthC, and SubHeightC may be specified in the future by ITU-T | ISO/IEC.

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

chroma_format_idc	separate_colour_plane_flag	Chroma format	SubWidthC	SubHeightC
0	0	monochrome	1	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.

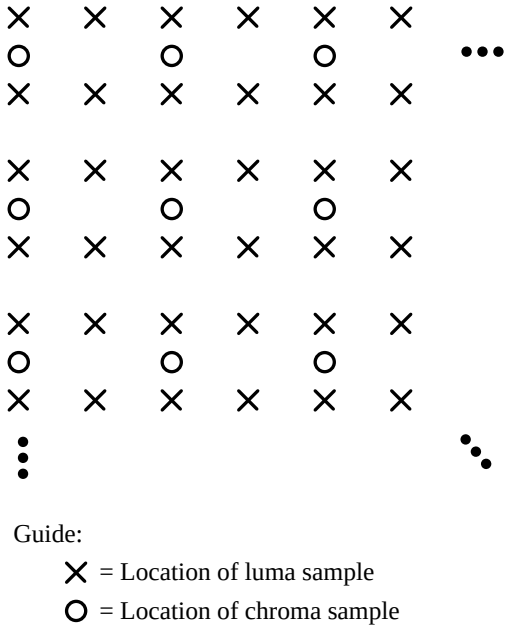
In 4:4:4 sampling, depending on the value of separate_colour_plane_flag, the following applies.

– If separate_colour_plane_flag is equal to 0, each of the two chroma arrays has the same height and width as the luma array.

– Otherwise (separate_colour_plane_flag is equal to 1), the three colour planes are separately processed as monochrome sampled pictures.

The number of bits necessary for the representation of each of the samples in the luma and chroma arrays in a video sequence is in the range of 8 to 14, 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 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 6-1. Alternative chroma sample relative locations may be indicated in video usability information (see Annex E).



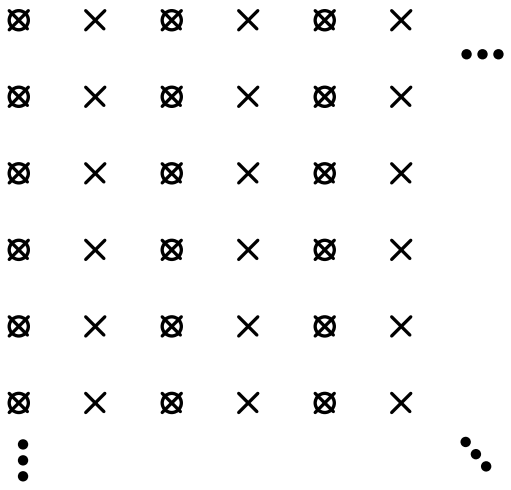
Guide:

× = Location of luma sample

○ = Location of chroma sample

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

When the value of chroma_format_idc is equal to 2, the chroma samples are co-sited with the corresponding luma samples and the nominal locations in a picture are as shown in Figure 6-2.



Guide:

× = Location of luma sample

○ = Location of chroma sample

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

When the value of 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 6-3.

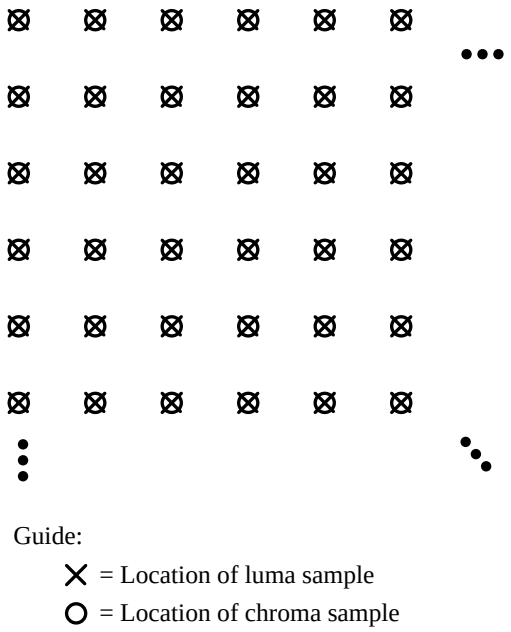


Figure 6-3 – Nominal vertical and horizontal locations of 4:4:4 luma and chroma samples in a picture

6.3 Spatial subdivision of pictures, slices, slice segments, and tiles

This subclause specifies how a picture is partitioned into slices, slice segments, tiles and coding tree blocks. Pictures are divided into slices and tiles. A slice is a sequence of one or more slice segments starting with an independent slice segment and containing all subsequent dependent slice segments (if any) that precede the next independent slice segment (if any) within the same access unit. A slice segment is a sequence of coding tree units. [Ed. (GJS): Clarify w.r.t. coding tree blocks vs. coding tree units vs. luma coding tree blocks vs. Cb or Cr coding tree blocks.] Likewise, a tile is a sequence of coding tree units.

For example, a picture may be divided into two slices as shown in Figure 6-4. In this example, the first slice is composed of an independent slice segment containing 4 coding tree units, a dependent slice segment containing 32 coding tree units, and another dependent slice segment containing 24 coding tree units; and the second slice consists of a single indpendent slice segment containing the remaining 39 coding tree units of the picture.

As another example, a picture may be divided into two tiles separated by a horizontal tile boundary as shown in Figure 6-5. The left side of the figure illustrates a case in which the picture only contains one slice, starting with an independent slice segment and followed by four dependent slice segments. The right side of the figure illustrates an alternative case in which the picture contains two slices in the first tile and one slice in the second tile.

Unlike slices, tiles are always rectangular. A tile always contains an integer number of coding tree units, and may consist of coding tree units contained in more than one slice. Similarly, a slice may consist of coding tree units contained in more than one tile.

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

– All coding tree units in a slice belong to the same tile.

– All coding tree units in a tile belong to the same slice.

NOTE 1 – Within the same picture, there may be both slices that contain multiple tiles and tiles that contain multiple slices.

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

– All coding tree units in a slice segment belong to the same tile.

– All coding tree units in a tile belong to the same slice segment.

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

NOTE 2 – When separate_colour_plane_flag is equal to 0, each coding tree block of a picture is contained in exactly one slice. When separate_colour_plane_flag is equal to 1, each coding tree block of a colour component is contained in exactly one slice (i.e. information for each coding tree block of a picture is present in exactly three slices and these three slices have different values of colour_plane_id).

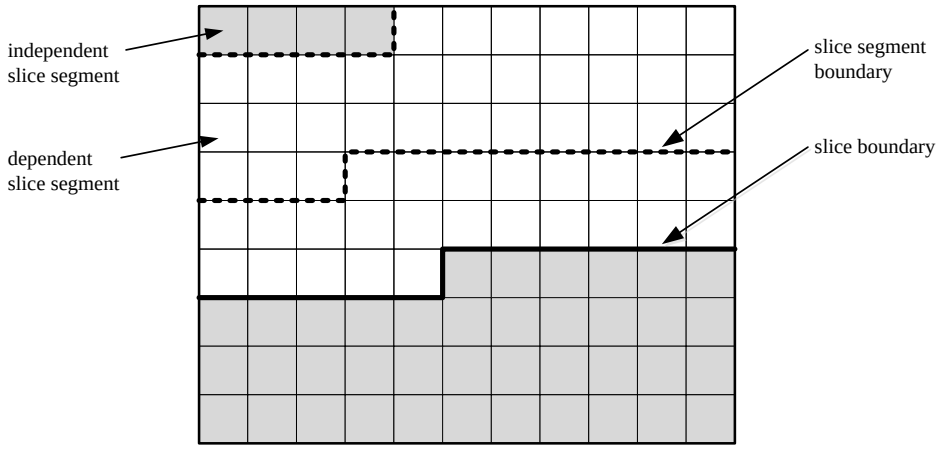


Figure 6-4 – A picture with 11 by 9 luma coding tree blocks that is partitioned into two slices, the first of which is partitioned into three slice segments (informative)

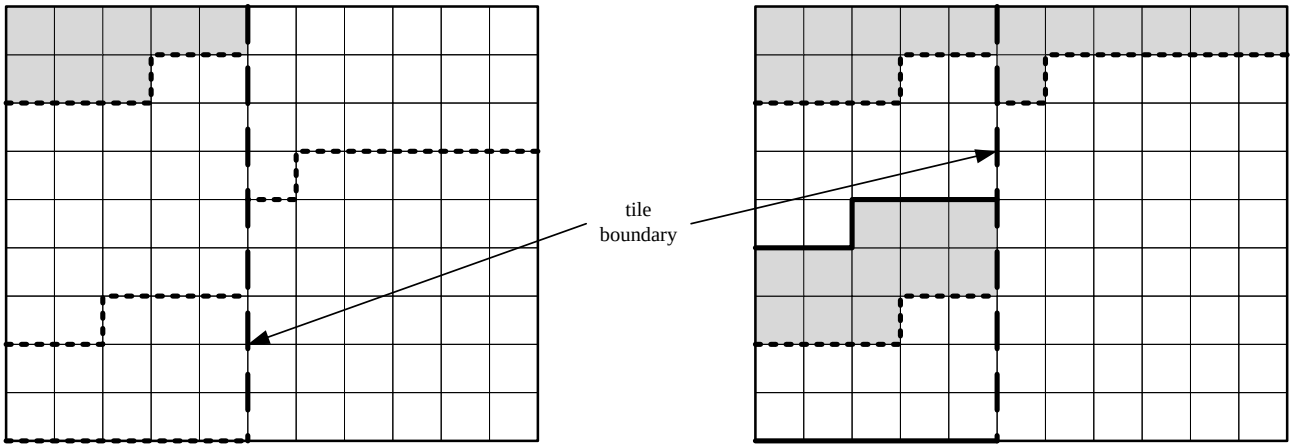


Figure 6-5 – A picture with 11 by 9 luma coding tree blocks that is partitioned into two tiles and one slice (left) or is partitioned into two tiles and three slices (right) (informative)

[Ed. (BB): Consider adding a new section for the following text describing the block and quadtree structures.]

The samples are processed in units of coding tree blocks. The array size for each luma coding tree block in both width and height is CtbSizeY in units of samples. The width and height of the array for each chroma coding tree block are CtbWidthC and CtbHeightC, respectively, in units of samples.

Each coding tree block 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 coding tree block. The quadtree is split until a leaf is reached, which is referred to as the coding block. When the picture width is not an integer number of coding tree block sizes, the coding tree blocks at the right picture boundary are incomplete. When the picture height is not an integer number of coding tree block sizes, the coding tree blocks at the bottom picture 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 encapsulated in a "unit" as follows.

– One prediction block (monochrome picture or separate_colour_plane_flag is equal to 1) or three prediction blocks (luma and chroma) and associated prediction syntax structures units are encapsulated in a prediction unit.

– One transform block (monochrome picture or separate_colour_plane_flag is equal to 1) or three transform blocks (luma and chroma) and associated transform syntax structures units are encapsulated in a transform unit.

– One coding block (monochrome picture or separate_colour_plane_flag is equal to 1) or three coding blocks (luma and chroma), the associated coding syntax structures and the associated prediction and transform units are encapsulated in a coding unit.

– One coding tree block (monochrome picture or separate_colour_plane_flag is equal to 1) or three coding tree blocks (luma and chroma), the associated coding tree syntax structures and the associated coding units are encapsulated in a coding tree unit.

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

– The division of each picture into components.

– The division of each picture into tiles.

– The division of each picture into slices.

– The division of each component into coding tree blocks.

– The division of each slice into slice segments.

– The division of each slice segment into coding tree units.

– The division of each tile into coding tree units.

– The division of each coding tree unit into coding tree blocks.

– The division of each coding tree block into coding blocks.

– The division of each coding tree unit into coding units.

– The division of each coding unit into prediction units.

– 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 prediction blocks.

– The division of each coding block into transform blocks.

– The division of each prediction unit into prediction blocks.

– The division of each transform unit into transform blocks.

6.4 Availability processes

6.4.1 Derivation process for z-scan order 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 (xN, yN) covered by a neighbouring block relative to the top-left luma sample of the current picture.

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

The minimum luma block address in z-scan order minBlockAddrCurr of the current block is derived as follows.

$$minBlockAddrCurr = MinTbAddrZS[\ xCurr >> Log2MinTrafoSize\ ][\ yCurr >> Log2MinTrafoSize\ ] \quad (6\text{-}1)$$

The minimum luma block address in z-scan order minBlockAddrN of the neighbouring block covering the location (xN, yN) is derived as follows.

– If one or more of the following conditions are true, minBlockAddrN is set to −1.

  – xN is less than 0

  - yN is less than 0
  - xN is greater than pic_width_in_luma_samples
  - yN is greater than pic_height_in_luma_samples
- Otherwise (xN and yN are inside picture boundaries),

$$minBlockAddrN = MinTbAddrZS[\ xN >> Log2MinTrafoSize\ ][\ yN >> Log2MinTrafoSize\ ] \quad (6\text{-}2)$$

The neighbouring block availability availableN is derived as follows.

- If one or more of the following conditions are true, availableN is set to FALSE:
  - minBlockAddrN is less than 0,
  - minBlockAddrN is greater than minBlockAddrCurr,
  - the variable SliceAddrRS associated with the slice segment containing the neighbouring block with the minimum luma block address minBlockAddrN differs in value from the variable SliceAddrRS associated with the slice segment containing the current block with the minimum luma block address minBlockAddrCurr.
  - the neighbouring block with the minimum luma block address minBlockAddrN is contained in a different tile than the current block with the minimum luma block address minBlockAddrCurr.
- Otherwise, availableN is set to TRUE.

6.4.2 Derivation process for prediction block availability

Inputs to this process are:

- the luma location (xC, yC) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
- a variable nCbS specifying the size of the current luma coding block,
- the luma location (xP, yP) of the top-left sample of the current luma prediction block relative to the top-left luma sample of the current picture,
- variables nPbW and nPbH specifying the width and the height of the current luma prediction block,
- a variable partIdx specifying the partition index of the current prediction unit within the current coding unit.
- the luma location (xN, yN) covered by a neighbouring prediction block relative to the top-left luma sample of the current picture.

Output of this process is the availability of the neighbouring prediction block covering the location (xN, yN), denoted as availableN is derived as follows.

The variable sameCb specifying whether the current luma prediction block and the neighbouring luma prediction block cover the same luma coding block.

- If all of the following conditions are true, sameCb is set equal to TRUE.
  - xC is less than or equal than xN,
  - yC is less than or equal than yN,
  - (xC + nCbS) is greater than xN,
  - (yC + nCbS) is greater than yN.
- Otherwise, sameCb is set equal to FALSE.

The neighbouring prediction block availability availableN is derived as follows.

- If sameCb is equal to FALSE, the derivation process for z-scan order block availability as specified in subclause 6.4.1 is invoked with (xCurr, yCurr) set equal to (xP, yP) and the luma location (xN, yN) as the input and the output is assigned to availableN.
- Otherwise, if all of the following conditions are true, availableN is set equal to FALSE.
  - (nPbW << 1) is equal to nCbS,
  - (nPbH << 1) is equal to nCbS,

- partIdx is equal to 1,
- (yC + nPbH) is less than or equal to yN,
- (xC + nPbW) is greater than xN.

– Otherwise, availableN is set equal to TRUE.

When availableN is equal to TRUE and CuPredMode[xN][yN] is equal to MODE_INTRA, availableN is set equal to FALSE.

6.5 Scanning processes

6.5.1 Coding tree block raster and tile scanning conversion process

The list colWidth[i] for i ranging from 0 to num_tile_columns_minus1, inclusive, specifying the width of the i-th tile column in units of CTBs, is derived as follows.

```
if( uniform_spacing_flag )
    for( i = 0; i <= num_tile_columns_minus1; i++ )
        colWidth[ i ] = ( ( i + 1 ) * PicWidthInCtbsY ) / ( num_tile_columns_minus1 + 1 ) −
                        ( i * PicWidthInCtbsY ) / ( num_tile_columns_minus1 + 1 )
else {
    colWidth[ num_tile_columns_minus1 ] = PicWidthInCtbsY
    for( i = 0; i < num_tile_columns_minus1; i++ ) {
        colWidth[ i ] = column_width_minus1[ i ] + 1
        colWidth[ num_tile_columns_minus1 ] −= colWidth[ i ]
    }
}
```

The list rowHeight[j] for j ranging from 0 to num_tile_rows_minus1, inclusive, specifying the height of the j-th tile row in units of CTBs, is derived as follows.

```
if( uniform_spacing_flag )
    for( j = 0; j <= num_tile_rows_minus1; j++ )
        rowHeight[ j ]  = ( ( j + 1 ) * PicHeightInCtbsY ) / ( num_tile_rows_minus1 + 1 ) −
                          ( j * PicHeightInCtbsY ) / ( num_tile_rows_minus1 + 1)
else {
    rowHeight[ num_tile_rows_minus1 ] = PicHeightInCtbsY
    for( j = 0; j < num_tile_rows_minus1; j++ ) {
        rowHeight[ j ] = row_height_minus1[ j ] + 1
        rowHeight[ num_tile_rows_minus1 ] −= rowHeight[ j ]
    }
}
```

The list colBd[i] for i ranging from 0 to num_tile_columns_minus1 + 1, inclusive, specifying the location of the i-th column boundary in units of coding tree blocks, is derived as follows.

```
for( colBd[ 0 ] = 0, i = 0; i <= num_tile_columns_minus1; i++ )
        colBd[ i + 1 ] = colBd[ i ] + colWidth[ i ]
```

The list rowBd[j] for j ranging from 0 to num_tile_rows_minus1 + 1, inclusive, specifying the location of the j-th row boundary in units of coding tree blocks, is derived as follows.

```
for( rowBd[ 0 ] = 0, j = 0; j <= num_tile_rows_minus1; j++ )
    rowBd[ j + 1 ] = rowBd[ j ] + rowHeight[ j ]
```

The list CtbAddrRStoTS[ctbAddrRS] for ctbAddrRS ranging from 0 to PicSizeInCtbsY − 1, inclusive, specifying the conversion from a CTB address in CTB raster scan of a picture to a CTB address in tile scan, is derived as follows.

```
 for( ctbAddrRS = 0; ctbAddrRS < PicSizeInCtbsY; ctbAddrRS++ ) {
        tbX = ctbAddrRS % PicWidthInCtbsY
        tbY = ctbAddrRS / PicWidthInCtbsY
        for( i = 0; i <= num_tile_columns_minus1; i++ )
              if( tbX >= colBd[ i ] )
                    tileX = i
```

```
        for( j = 0; j <= num_tile_rows_minus1; j++ )
            if( tbY >= rowBd[ j ] )
                tileY = j
        CtbAddrRStoTS[ ctbAddrRS ] = 0
        for( i = 0; i < tileX; i++ )
            CtbAddrRStoTS[ ctbAddrRS ] += rowHeight[ tileY ] * colWidth[ i ]
        for( j = 0; j < tileY; j++ )
            CtbAddrRStoTS[ ctbAddrRS ] += PicWidthInCtbsY * rowHeight[ j ]
        CtbAddrRStoTS[ ctbAddrRS ] += ( tbY − rowBd[ tileY ] ) * colWidth[ tileX ] + tbX − colBd[ tileX ]
    }
```

The list CtbAddrTStoRS[ctbAddrTS] for ctbAddrTS ranging from 0 to PicSizeInCtbsY − 1, inclusive, specifying the conversion from a CTB address in tile scan to a CTB address in CTB raster scan of a picture, is derived as follows.

```
    for( ctbAddrRS = 0; ctbAddrRS < PicSizeInCtbsY; ctbAddrRS++ )
        CtbAddrTStoRS[ CtbAddrRStoTS[ ctbAddrRS ] ] = ctbAddrRS
```

The list TileId[ctbAddrTS] for ctbAddrTS ranging from 0 to PicSizeInCtbsY − 1, inclusive, specifying the conversion from a CTB address in tile scan to a tile ID, is derived as follows.

```
    for( j = 0, tIdx = 0; j <= num_tile_rows_minus1; j++ )
        for( i = 0; i <= num_tile_columns_minus1; i++, tIdx++ )
            for( y = rowBd[ j ]; y < rowBd[ j + 1 ]; y++ )
                for( x = colBd[ i ]; x < colBd[ i + 1 ]; x++ )
                    TileId[ CtbAddrRStoTS[ y*PicWidthInCtbsY+ x ] ] = tIdx
```

The values of ColumnWidthInLumaSamples[i], specifying the width of the i-th tile column in units of luma samples, are set equal to colWidth[i] << Log2CtbSizeY for i ranging from 0 to num_tile_columns_minus1, inclusive.

The values of RowHeightInLumaSamples[j], specifying the height of the j-th tile row in units of luma samples, are set equal to rowHeight[j] << Log2CtbSizeY for j ranging from 0 to num_tile_rows_minus1, inclusive.

6.5.2 Z-scan order array initialization process

The array MinTbAddrZS with elements MinTbAddrZS[x][y] for x ranging from 0 to (PicWidthInCtbsY << (Log2CtbSizeY − Log2MinTrafoSize)) − 1, inclusive, and y ranging from 0 to (PicHeightInCtbsY << (Log2CtbSizeY − Log2MinTrafoSize)) − 1, specifying the conversion from a location (x, y) in units of minimum blocks to a minimum block address in z-scan order, inclusive is derived as follows.

```
    for( y = 0; y < ( PicHeightInCtbsY << ( Log2CtbSizeY − Log2MinTrafoSize ) ); y++ )
        for( x = 0; x < ( PicWidthInCtbsY << ( Log2CtbSizeY − Log2MinTrafoSize ) ); x++) {
            tbX = ( x << Log2MinTrafoSize ) >> Log2CtbSizeY
            tbY = ( y << Log2MinTrafoSize ) >> Log2CtbSizeY
            ctbAddrRS = PicWidthInCtbsY * tbY + tbX
            MinTbAddrZS[ x ][ y ] = CtbAddrRStoTS[ ctbAddrRS ] << (( Log2CtbSizeY − Log2MinTrafoSize ) * 2)
            for( i = 0, p = 0; i < ( Log2CtbSizeY − Log2MinTrafoSize ); i++ ) {
                m = 1 << i
                p += ( m & x ? m * m : 0 ) + ( m & y ? 2 * m * m : 0 )
            }
            MinTbAddrZS[ x ][ y ] += p
        }
```

6.5.3 Up-right diagonal scan order array initialization process

Inputs to this process is a block size blkSize.

Output of this process is the array diagScan[sPos][sComp]. The array index sPos specify the scan position ranging from 0 to (blkSize * blkSize) − 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 blkSize, the array diagScan is derived as follows.

```
    i = 0
    x = 0
    y = 0
    stopLoop = FALSE
    while( !stopLoop ) {
        while( y >= 0 ) {
```

```
            if( x < blkSize && y < blkSize ) {
                diagScan[ i ][ 0 ] = x
                diagScan[ i ][ 1 ] = y
                i++
            }
            y− −
            x++
        }
        y = x
        x = 0
        if( i >= blkSize * blkSize )
            stopLoop = TRUE
    }
```

6.5.4 Horizontal scan order array initialization process

Inputs to this process is a block size blkSize.

Output of this process is the array horScan[sPos][sComp]. The array index sPos specifies the scan position ranging from 0 to (blkSize * blkSize) − 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 blkSize, the array horScan is derived as follows.

```
i = 0
for( y = 0; y < blkSize; y++ )
    for( x = 0; x < blkSize; x++ ) {
        horScan[ i ][ 0 ] = x
        horScan[ i ][ 1 ] = y
        i++
    }
```

6.5.5 Vertical scan order array initialization process

Inputs to this process is a block size blkSize.

Output of this process is the array verScan[sPos][sComp]. The array index sPos specifies the scan position ranging from 0 to (blkSize * blkSize) − 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 blkSize, the array verScan is derived as follows.

```
i = 0
for( x = 0; x < blkSize; x++ )
    for( y = 0; y < blkSize; y++ ) {
        verScan[ i ][ 0 ] = x
        verScan[ i ][ 1 ] = y
        i++
    }
```

7 Syntax and semantics

7.1 Method of specifying syntax in tabular form

The syntax tables specify a superset of the syntax of all allowed bitstreams. Additional constraints on the syntax 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 pseudo code used to describe the syntax. When **syntax_element** appears, it specifies that a syntax element is parsed from the bitstream and the bitstream pointer is advanced to the next position beyond the syntax element in the bitstream parsing process.

	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(v)
conditioning statement	
/* A group of statements enclosed in curly brackets is a compound statement and is treated functionally as a single statement. */	
{	
statement	
statement	
…	
}	
/* A "while" structure specifies a test of whether a condition is true, and if true, specifies evaluation of a statement (or compound statement) repeatedly until the condition is no longer true */	
while(condition)	
statement	
/* A "do … while" structure specifies evaluation of a statement once, followed by a test of whether a condition is true, and if true, specifies repeated evaluation of the statement until the condition is no longer true */	
do	
statement	
while(condition)	
/* An "if … else" structure specifies a test of whether a condition is true, and if the condition is true, specifies evaluation of a primary statement, otherwise, specifies evaluation of an alternative statement. The "else" part of the structure and the associated alternative statement is omitted if no alternative statement evaluation is needed */	
if(condition)	
primary statement	
else	
alternative statement	
/* A "for" structure specifies evaluation of an initial statement, followed by a test of a condition, and if the condition is true, specifies repeated evaluation of a primary statement followed by a subsequent statement until the condition is no longer true. */	
for(initial statement; condition; subsequent statement)	
primary statement	

7.2 Specification of syntax functions 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 RBSP, the return value of more_rbsp_data( ) is equal to FALSE.
- Otherwise, the RBSP data is searched for the last (least significant, right-most) bit equal to 1 that is present in the RBSP. Given the position of this bit, which is the first bit (rbsp_stop_one_bit) of the rbsp_trailing_bits( ) syntax structure, the following applies.
  - If there is more data in an RBSP before the rbsp_trailing_bits( ) syntax structure, the return value of more_rbsp_data( ) is equal to TRUE.
  - Otherwise, the return value of more_rbsp_data( ) is equal to FALSE.

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

more_rbsp_trailing_data( ) is specified as follows.

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

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 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.

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.

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

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

- ae(v): context-adaptive arithmetic entropy-coded syntax element. The parsing process for this descriptor is specified in subclause 9.2.
- 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).
- se(v): signed integer 0-th order Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.1.
- u(n): unsigned integer using n bits. When n is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function read_bits(n) interpreted as a binary representation of an unsigned integer with most significant bit written first.
- ue(v): unsigned integer 0-th order Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.1.

7.3 Syntax in tabular form

7.3.1 NAL unit syntax

7.3.1.1 General NAL unit syntax

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

7.3.1.2 NAL unit header syntax

nal_unit_header( ) {	**Descriptor**
forbidden_zero_bit	f(1)
nal_unit_type	u(6)
nuh_reserved_zero_6bits	u(6)
nuh_temporal_id_plus1	u(3)
}	

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

7.3.2.1 Video parameter set RBSP syntax

video_parameter_set_rbsp( ) {	**Descriptor**
vps_video_parameter_set_id	u(4)
vps_reserved_three_2bits	u(2)
vps_reserved_zero_6bits	u(6)
vps_max_sub_layers_minus1	u(3)
vps_temporal_id_nesting_flag	u(1)
vps_reserved_0xffff_16bits	u(16)
profile_tier_level(1, vps_max_sub_layers_minus1)	
bit_rate_pic_rate_info(0, vps_max_sub_layers_minus1)	
vps_sub_layer_ordering_info_present_flag	u(1)
for(i = (vps_sub_layer_ordering_info_present_flag ? 0 : vps_max_sub_layers_minus1); i <= vps_max_sub_layers_minus1; i++) {	
vps_max_dec_pic_buffering[i]	ue(v)
vps_max_num_reorder_pics[i]	ue(v)
vps_max_latency_increase[i]	ue(v)
}	
vps_max_nuh_reserved_zero_layer_id	u(6)
vps_num_op_sets_minus1	ue(v)
for(i = 1; i <= vps_num_op_sets_minus1; i++)	
operation_point_set(i)	
vps_num_hrd_parameters	ue(v)
for(i = 0; i < vps_num_hrd_parameters; i++) {	
hrd_op_set_idx[i]	ue(v)
if(i > 0)	
cprms_present_flag[i]	u(1)
hrd_parameters(cprms_present_flag[i], vps_max_sub_layers_minus1)	
}	
vps_extension_flag	u(1)
if(vps_extension_flag)	
while(more_rbsp_data( ))	
vps_extension_data_flag	u(1)
rbsp_trailing_bits( )	
}	

7.3.2.2 Sequence parameter set RBSP syntax

seq_parameter_set_rbsp( ) {	**Descriptor**
sps_video_parameter_set_id	u(4)
sps_max_sub_layers_minus1	u(3)
sps_temporal_id_nesting_flag	u(1)
profile_tier_level(1, sps_max_sub_layers_minus1)	
sps_seq_parameter_set_id	ue(v)
chroma_format_idc	ue(v)
if(chroma_format_idc = = 3)	
separate_colour_plane_flag	u(1)
pic_width_in_luma_samples	ue(v)
pic_height_in_luma_samples	ue(v)
conformance_window_flag	u(1)
if(conformance_window_flag) {	
conf_win_left_offset	ue(v)
conf_win_right_offset	ue(v)
conf_win_top_offset	ue(v)
conf_win_bottom_offset	ue(v)
}	
bit_depth_luma_minus8	ue(v)
bit_depth_chroma_minus8	ue(v)
log2_max_pic_order_cnt_lsb_minus4	ue(v)
sps_sub_layer_ordering_info_present_flag	u(1)
for(i = (sps_sub_layer_ordering_info_present_flag ? 0 : sps_max_sub_layers_minus1); i <= sps_max_sub_layers_minus1; i++) {	
sps_max_dec_pic_buffering[i]	ue(v)
sps_max_num_reorder_pics[i]	ue(v)
sps_max_latency_increase[i]	ue(v)
}	
log2_min_luma_coding_block_size_minus3	ue(v)
log2_diff_max_min_luma_coding_block_size	ue(v)
log2_min_transform_block_size_minus2	ue(v)
log2_diff_max_min_transform_block_size	ue(v)
max_transform_hierarchy_depth_inter	ue(v)
max_transform_hierarchy_depth_intra	ue(v)
scaling_list_enable_flag	u(1)
if(scaling_list_enable_flag) {	
sps_scaling_list_data_present_flag	u(1)
if(sps_scaling_list_data_present_flag)	
scaling_list_data( )	
}	
amp_enabled_flag	u(1)
sample_adaptive_offset_enabled_flag	u(1)
pcm_enabled_flag	u(1)
if(pcm_enabled_flag) {	
pcm_sample_bit_depth_luma_minus1	u(4)
pcm_sample_bit_depth_chroma_minus1	u(4)

log2_min_pcm_luma_coding_block_size_minus3	ue(v)
log2_diff_max_min_pcm_luma_coding_block_size	ue(v)
pcm_loop_filter_disable_flag	u(1)
}	
num_short_term_ref_pic_sets	ue(v)
for(i = 0; i < num_short_term_ref_pic_sets; i++)	
short_term_ref_pic_set(i)	
long_term_ref_pics_present_flag	u(1)
if(long_term_ref_pics_present_flag) {	
num_long_term_ref_pics_sps	ue(v)
for(i = 0; i < num_long_term_ref_pics_sps; i++) {	
lt_ref_pic_poc_lsb_sps[i]	u(v)
used_by_curr_pic_lt_sps_flag[i]	u(1)
}	
}	
sps_temporal_mvp_enable_flag	u(1)
strong_intra_smoothing_enable_flag	u(1)
vui_parameters_present_flag	u(1)
if(vui_parameters_present_flag)	
vui_parameters( )	
sps_extension_flag	u(1)
if(sps_extension_flag)	
while(more_rbsp_data( ))	
sps_extension_data_flag	u(1)
rbsp_trailing_bits( )	
}	

7.3.2.3 Picture parameter set RBSP syntax

pic_parameter_set_rbsp( ) {	**Descriptor**
pps_pic_parameter_set_id	ue(v)
pps_seq_parameter_set_id	ue(v)
dependent_slice_segments_enabled_flag	u(1)
sign_data_hiding_flag	u(1)
cabac_init_present_flag	u(1)
num_ref_idx_l0_default_active_minus1	ue(v)
num_ref_idx_l1_default_active_minus1	ue(v)
init_qp_minus26	se(v)
constrained_intra_pred_flag	u(1)
transform_skip_enabled_flag	u(1)
cu_qp_delta_enabled_flag	u(1)
if (cu_qp_delta_enabled_flag)	
diff_cu_qp_delta_depth	ue(v)
pps_cb_qp_offset	se(v)
pps_cr_qp_offset	se(v)
pps_slice_chroma_qp_offsets_present_flag	u(1)
weighted_pred_flag	u(1)

weighted_bipred_flag	u(1)
output_flag_present_flag	u(1)
transquant_bypass_enable_flag	u(1)
tiles_enabled_flag	u(1)
entropy_coding_sync_enabled_flag	u(1)
if(tiles_enabled_flag) {	
num_tile_columns_minus1	ue(v)
num_tile_rows_minus1	ue(v)
uniform_spacing_flag	u(1)
if(!uniform_spacing_flag) {	
for(i = 0; i < num_tile_columns_minus1; i++)	
column_width_minus1[i]	ue(v)
for(i = 0; i < num_tile_rows_minus1; i++)	
row_height_minus1[i]	ue(v)
}	
loop_filter_across_tiles_enabled_flag	u(1)
}	
loop_filter_across_slices_enabled_flag	u(1)
deblocking_filter_control_present_flag	u(1)
if(deblocking_filter_control_present_flag) {	
deblocking_filter_override_enabled_flag	u(1)
pps_disable_deblocking_filter_flag	u(1)
if(!pps_disable_deblocking_filter_flag) {	
pps_beta_offset_div2	se(v)
pps_tc_offset_div2	se(v)
}	
}	
pps_scaling_list_data_present_flag	u(1)
if(pps_scaling_list_data_present_flag)	
scaling_list_data( )	
lists_modification_present_flag	u(1)
log2_parallel_merge_level_minus2	ue(v)
num_extra_slice_header_bits	u(3)
slice_segment_header_extension_present_flag	u(1)
pps_extension_flag	u(1)
if(pps_extension_flag)	
while(more_rbsp_data( ))	
pps_extension_data_flag	u(1)
rbsp_trailing_bits( )	
}	

7.3.2.4 Supplemental enhancement information RBSP syntax

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

7.3.2.5 Access unit delimiter RBSP syntax

access_unit_delimiter_rbsp( ) {	**Descriptor**
pic_type	u(3)
rbsp_trailing_bits( )	
}	

7.3.2.6 End of sequence RBSP syntax

end_of_seq_rbsp( ) {	**Descriptor**
}	

7.3.2.7 End of bitstream RBSP syntax

end_of_bitstream_rbsp( ) {	**Descriptor**
}	

7.3.2.8 Filler data RBSP syntax

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

7.3.2.9 Slice segment layer RBSP syntax

slice_segment_layer_rbsp( ) {	**Descriptor**
slice_segment_header( )	
slice_segment_data( )	
rbsp_slice_segment_trailing_bits( )	
}	

7.3.2.10 RBSP slice segment trailing bits syntax

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

7.3.2.11 RBSP trailing bits syntax

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

7.3.2.12 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)
}	

7.3.3 Profile, tier and level syntax

profile_tier_level(profilePresentFlag, maxNumSubLayersMinus1) {	**Descriptor**
if(profilePresentFlag) {	
general_profile_space	u(2)
general_tier_flag	u(1)
general_profile_idc	u(5)
for(i = 0; i < 32; i++)	
general_profile_compatibility_flag[i]	u(1)
general_reserved_zero_16bits [Ed. (GJS): Adjust semantics accordingly.]	u(16)
}	
general_level_idc	u(8)
for(i = 0; i < maxNumSubLayersMinus1; i++) {	
if(profilePresentFlag)	
sub_layer_profile_present_flag[i]	u(1)
sub_layer_level_present_flag[i]	u(1)
if(profilePresentFlag && sub_layer_profile_present_flag[i]) {	
sub_layer_profile_space[i]	u(2)
sub_layer_tier_flag[i]	u(1)
sub_layer_profile_idc[i]	u(5)
for(j = 0; j < 32; j++)	
sub_layer_profile_compatibility_flag[i][j]	u(1)
sub_layer_reserved_zero_16bits[i]	u(16)
}	
if(sub_layer_level_present_flag[i])	
sub_layer_level_idc[i]	u(8)
}	
}	

7.3.4 Bit rate and picture rate information syntax

bit_rate_pic_rate_info(TempLevelLow, TempLevelHigh) {	**Descriptor**
for(i = TempLevelLow; i <= TempLevelHigh; i++) {	
bit_rate_info_present_flag[i]	**u(1)**
pic_rate_info_present_flag[i]	**u(1)**
if(bit_rate_info_present_flag[i]) {	
avg_bit_rate[i]	**u(16)**
max_bit_rate [i]	**u(16)**
}	
if(pic_rate_info_present_flag[i]) {	
constant_pic_rate_idc[i]	**u(2)**
avg_pic_rate[i]	**u(16)**
}	
}	
}	

7.3.5 Operation point set syntax

operation_point_set(opsIdx) {	**Descriptor**
for(i = 0; i <= vps_max_nuh_reserved_zero_layer_id; i++)	
layer_id_included_flag[opsIdx][i]	u(1)
}	

7.3.6 Scaling list data syntax

scaling_list_data( ) {	**Descriptor**
for(sizeId = 0; sizeId < 4; sizeId++)	
for(matrixId = 0; matrixId < ((sizeId = = 3) ? 2 : 6); matrixId++) {	
scaling_list_pred_mode_flag[sizeId][matrixId]	u(1)
if(!scaling_list_pred_mode_flag[sizeId][matrixId])	
scaling_list_pred_matrix_id_delta[sizeId][matrixId]	ue(v)
else {	
nextCoef = 8	
coefNum = Min(64, (1 << (4 + (sizeId << 1))))	
if(sizeId > 1) {	
scaling_list_dc_coef_minus8[sizeId − 2][matrixId]	se(v)
nextCoef = scaling_list_dc_coef_minus8[sizeId − 2][matrixId] + 8	
}	
for(i = 0; i < coefNum; i++) {	
scaling_list_delta_coef	se(v)
nextCoef = (nextCoef + scaling_list_delta_coef + 256) % 256	
ScalingList[sizeId][matrixId][i] = nextCoef	
}	
}	
}	
}	

7.3.7 Supplemental enhancement information message syntax

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

7.3.8 Slice segment header syntax

7.3.8.1 General slice segment header syntax

slice_segment_header( ) {	**Descriptor**
first_slice_segment_in_pic_flag	u(1)
if(RapPicFlag)	
no_output_of_prior_pics_flag	u(1)
slice_pic_parameter_set_id	ue(v)
if(!first_slice_segment_in_pic_flag) {	
if(dependent_slice_segments_enabled_flag)	
dependent_slice_segment_flag	u(1)
slice_segment_address	u(v)
}	
if(!dependent_slice_segment_flag) {	
for (i = 0; i < num_extra_slice_header_bits; i++)	
slice_reserved_undetermined_flag[i]	u(1)
slice_type	ue(v)
if(output_flag_present_flag)	
pic_output_flag	u(1)
if(separate_colour_plane_flag = = 1)	
colour_plane_id	u(2)
if(!IdrPicFlag) {	
pic_order_cnt_lsb	u(v)
short_term_ref_pic_set_sps_flag	u(1)
if(!short_term_ref_pic_set_sps_flag)	
short_term_ref_pic_set(num_short_term_ref_pic_sets)	
else	
short_term_ref_pic_set_idx	u(v)
if(long_term_ref_pics_present_flag) {	
if(num_long_term_ref_pics_sps > 0)	
num_long_term_sps	ue(v)
num_long_term_pics	ue(v)
for(i = 0; i < num_long_term_sps + num_long_term_pics; i++) {	
if(i < num_long_term_sps)	
lt_idx_sps[i]	u(v)
else {	
poc_lsb_lt[i]	u(v)
used_by_curr_pic_lt_flag[i]	u(1)
}	
delta_poc_msb_present_flag[i]	u(1)
if(delta_poc_msb_present_flag[i])	
delta_poc_msb_cycle_lt[i]	ue(v)
}	
}	
if(sps_temporal_mvp_enable_flag)	
slice_temporal_mvp_enable_flag	u(1)
}	
if(sample_adaptive_offset_enabled_flag) {	

slice_sao_luma_flag	u(1)
slice_sao_chroma_flag	u(1)
}	
if(slice_type = = P || slice_type = = B) {	
num_ref_idx_active_override_flag	u(1)
if(num_ref_idx_active_override_flag) {	
num_ref_idx_l0_active_minus1	ue(v)
if(slice_type = = B)	
num_ref_idx_l1_active_minus1	ue(v)
}	
if(lists_modification_present_flag && NumPocTotalCurr > 1)	
ref_pic_lists_modification( )	
if(slice_type = = B)	
mvd_l1_zero_flag	u(1)
if(cabac_init_present_flag)	
cabac_init_flag	u(1)
if(slice_temporal_mvp_enable_flag) {	
if(slice_type = = B)	
collocated_from_l0_flag	u(1)
if((collocated_from_l0_flag && num_ref_idx_l0_active_minus1 > 0) || (!collocated_from_l0_flag && num_ref_idx_l1_active_minus1 > 0)) [Ed. (GJS): Does the logic of this condition check make sense when the slice is not a B slice?]	
collocated_ref_idx	ue(v)
}	
if((weighted_pred_flag && slice_type = = P) || (weighted_bipred_flag && slice_type = = B))	
pred_weight_table( )	
five_minus_max_num_merge_cand	ue(v)
}	
slice_qp_delta	se(v)
if(pps_slice_chroma_qp_offsets_present_flag) {	
slice_cb_qp_offset	se(v)
slice_cr_qp_offset	se(v)
}	
if(deblocking_filter_control_present_flag) {	
if(deblocking_filter_override_enabled_flag)	
deblocking_filter_override_flag	u(1)
if(deblocking_filter_override_flag) {	
slice_disable_deblocking_filter_flag	u(1)
if(!slice_disable_deblocking_filter_flag) {	
slice_beta_offset_div2	se(v)
slice_tc_offset_div2	se(v)
}	
}	
}	
if(loop_filter_across_slices_enabled_flag && (slice_sao_luma_flag || slice_sao_chroma_flag || !slice_disable_deblocking_filter_flag))	
slice_loop_filter_across_slices_enabled_flag	u(1)

}	
if(tiles_enabled_flag || entropy_coding_sync_enabled_flag) {	
num_entry_point_offsets	ue(v)
if(num_entry_point_offsets > 0) {	
offset_len_minus1	ue(v)
for(i = 0; i < num_entry_point_offsets; i++)	
entry_point_offset[i]	u(v)
}	
}	
if(slice_segment_header_extension_present_flag) {	
slice_segment_header_extension_length	ue(v)
for(i = 0; i < slice_segment_header_extension_length; i++)	
slice_segment_header_extension_data_byte[i]	u(8)
}	
byte_alignment( )	
}	

7.3.8.2 Short-term reference picture set syntax

[Ed. This appears in the SPS and SH. It should probably get its own level 3 heading.]

short_term_ref_pic_set(idxRps) {	**Descriptor**
if(idxRps != 0)	
inter_ref_pic_set_prediction_flag	u(1)
if(inter_ref_pic_set_prediction_flag) {	
if(idxRps = = num_short_term_ref_pic_sets)	
delta_idx_minus1	ue(v)
delta_rps_sign	u(1)
abs_delta_rps_minus1	ue(v)
for(j = 0; j <= NumDeltaPocs[RIdx]; j++) {	
used_by_curr_pic_flag[j]	u(1)
if(!used_by_curr_pic_flag[j])	
use_delta_flag[j]	u(1)
}	
}	
else {	
num_negative_pics	ue(v)
num_positive_pics	ue(v)
for(i = 0; i < num_negative_pics; i++) {	
delta_poc_s0_minus1[i]	ue(v)
used_by_curr_pic_s0_flag[i]	u(1)
}	
for(i = 0; i < num_positive_pics; i++) {	
delta_poc_s1_minus1[i]	ue(v)
used_by_curr_pic_s1_flag[i]	u(1)
}	
}	
}	

7.3.8.3 Reference picture list modification syntax

ref_pic_lists_modification( ) {	**Descriptor**
ref_pic_list_modification_flag_l0	u(1)
if(ref_pic_list_modification_flag_l0)	
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++)	
list_entry_l0[i]	u(v)
if(slice_type = = B) {	
ref_pic_list_modification_flag_l1	u(1)
if(ref_pic_list_modification_flag_l1)	
for(i = 0; i <= num_ref_idx_l1_active_minus1; i++)	
list_entry_l1[i]	u(v)
}	
}	

7.3.8.4 Weighted prediction parameters syntax

pred_weight_table( ) {	**Descriptor**
luma_log2_weight_denom	ue(v)
if(chroma_format_idc != 0)	
delta_chroma_log2_weight_denom	se(v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++)	
luma_weight_l0_flag[i]	u(1)
if(chroma_format_idc != 0)	
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++)	
chroma_weight_l0_flag[i]	u(1)
for(i = 0; i <= num_ref_idx_l0_active_minus1; 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(slice_type = = B) {	
for(i = 0; i <= num_ref_idx_l1_active_minus1; i++)	
luma_weight_l1_flag[i]	u(1)
if(chroma_format_idc != 0)	
for(i = 0; i <= num_ref_idx_l1_active_minus1; i++)	
chroma_weight_l1_flag[i]	u(1)
for(i = 0; i <= num_ref_idx_l1_active_minus1; i++) {	
if(luma_weight_l1_flag[i]) {	
delta_luma_weight_l1[i]	se(v)
luma_offset_l1[i]	se(v)
}	
if(chroma_weight_l1_flag[i])	
for(j = 0; j < 2; j++) {	
delta_chroma_weight_l1[i][j]	se(v)
delta_chroma_offset_l1[i][j]	se(v)
}	
}	
}	
}	

7.3.9 Slice segment data syntax

7.3.9.1 General slice segment data syntax

slice_segment_data( ) {	**Descriptor**
do {	
coding_tree_unit( )	
end_of_slice_segment_flag	ae(v)
CtbAddrInTS++	
CtbAddrInRS = CtbAddrTStoRS[CtbAddrInTS]	

if(!end_of_slice_segment_flag &&
((tiles_enabled_flag && TileId[CtbAddrInTS] != TileId[CtbAddrInTS − 1]) ||
(entropy_coding_sync_enabled_flag && CtbAddrInTS % PicWidthInCtbsY = = 0))
) {	

end_of_sub_stream_one_bit /* equal to 1 */	ae(v)
byte_alignment( )	
}	
} while(!end_of_slice_segment_flag)	
}	

7.3.9.2 Coding tree unit syntax

coding_tree_unit( ) {	**Descriptor**
xCtb = (CtbAddrInRS % PicWidthInCtbsY) << Log2CtbSizeY	
yCtb = (CtbAddrInRS / PicWidthInCtbsY) << Log2CtbSizeY	
CtbAddrInSliceSeg = CtbAddrInRS − slice_segment_address	
if(slice_sao_luma_flag || slice_sao_chroma_flag)	
sao(xCtb >> Log2CtbSizeY, yCtb >> Log2CtbSizeY)	
coding_quadtree(xCtb, yCtb, Log2CtbSizeY, 0)	
}	

7.3.9.3 Sample adaptive offset syntax

sao(rx, ry){	**Descriptor**
if(rx > 0) {	
leftCtbInSliceSeg = CtbAddrInSliceSeg > 0	
leftCtbInTile = TileId[CtbAddrInTS] = = TileId[CtbAddrRStoTS[CtbAddrInRS − 1]]	
if(leftCtbInSliceSeg && leftCtbInTile)	
sao_merge_left_flag	ae(v)
}	
if(ry > 0 && !sao_merge_left_flag) {	
upCtbInSliceSeg = (CtbAddrInRS − PicWidthInCtbsY) >= slice_segment_address	
upCtbInTile = TileId[CtbAddrInTS] = = TileId[CtbAddrRStoTS[CtbAddrInRS − PicWidthInCtbsY]]	
if(upCtbInSliceSeg && upCtbInTile)	
sao_merge_up_flag	ae(v)
}	
if(!sao_merge_up_flag && !sao_merge_left_flag)	
for(cIdx = 0; cIdx < 3; cIdx++)	
if((slice_sao_luma_flag && cIdx = = 0) || (slice_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[cIdx][rx][ry][i]	ae(v)
sao_band_position[cIdx][rx][ry]	ae(v)
} else {	
if(cIdx = = 0)	
sao_eo_class_luma	ae(v)
if(cIdx = = 1)	
sao_eo_class_chroma	ae(v)
}	
}	
}	
}	

7.3.9.4 Coding quadtree syntax

coding_quadtree(x0, y0, log2CbSize, cqtDepth) {	**Descriptor**
if(x0 + (1 << log2CbSize) <= pic_width_in_luma_samples && y0 + (1 << log2CbSize) <= pic_height_in_luma_samples && log2CbSize > Log2MinCbSizeY)	
split_cu_flag[x0][y0]	ae(v)
if(cu_qp_delta_enabled_flag && log2CbSize >= Log2MinCuQpDeltaSize) {	
IsCuQpDeltaCoded = 0	
CuQpDelta = 0	
}	
if(split_cu_flag[x0][y0]) {	
x1 = x0 + ((1 << log2CbSize) >> 1)	
y1 = y0 + ((1 << log2CbSize) >> 1)	
coding_quadtree(x0, y0, log2CbSize − 1, cqtDepth + 1)	
if(x1 < pic_width_in_luma_samples)	
coding_quadtree(x1, y0, log2CbSize − 1, cqtDepth + 1)	
if(y1 < pic_height_in_luma_samples)	
coding_quadtree(x0, y1, log2CbSize − 1, cqtDepth + 1)	
if(x1 < pic_width_in_luma_samples && y1 < pic_height_in_luma_samples)	
coding_quadtree(x1, y1, log2CbSize − 1, cqtDepth + 1)	
} else	
coding_unit(x0, y0, log2CbSize)	
}	

7.3.9.5 Coding unit syntax

coding_unit(x0, y0, log2CbSize) {	**Descriptor**
if(transquant_bypass_enable_flag)	
cu_transquant_bypass_flag	ae(v)
if(slice_type != I)	
cu_skip_flag[x0][y0]	ae(v)
nCbS = (1 << log2CbSize)	
if(cu_skip_flag[x0][y0])	
prediction_unit(x0, y0, nCbS, nCbS)	
else {	
if(slice_type != I)	
pred_mode_flag	ae(v)
if(CuPredMode[x0][y0] != MODE_INTRA || log2CbSize == Log2MinCbSizeY)	
part_mode	ae(v)
if(CuPredMode[x0][y0] == MODE_INTRA) {	
if(PartMode == PART_2Nx2N && pcm_enabled_flag && log2CbSize >= Log2MinIpcmCbSizeY && log2CbSize <= Log2MaxIpcmCbSizeY)	
pcm_flag[x0][y0]	ae(v)
if(pcm_flag[x0][y0]) {	
while(!byte_aligned( ))	
pcm_alignment_zero_bit	f(1)
pcm_sample(x0, y0, log2CbSize)	
} else {	
pbOffset = (PartMode == PART_NxN) ? (nCbS / 2) : nCbS	
for(j = 0; j < nCbS; j = j + pbOffset)	
for(i = 0; i < nCbS; i = i + pbOffset)	
prev_intra_luma_pred_flag[x0 + i][y0+ j]	ae(v)
for(j = 0; j < nCbS; j = j + pbOffset)	
for(i = 0; i < nCbS; i = i + pbOffset)	
if(prev_intra_luma_pred_flag[x0 + i][y0+ j])	
mpm_idx[x0 + i][y0+ j]	ae(v)
else	
rem_intra_luma_pred_mode[x0 + i][y0+ j]	ae(v)
intra_chroma_pred_mode[x0][y0]	ae(v)
}	
} else {	
if(PartMode == PART_2Nx2N)	
prediction_unit(x0, y0, nCbS, nCbS)	
else if(PartMode == PART_2NxN) {	
prediction_unit(x0, y0, nCbS, nCbS / 2)	
prediction_unit(x0, y0 + (nCbS / 2), nCbS, nCbS / 2)	
} else if(PartMode == PART_Nx2N) {	
prediction_unit(x0, y0, nCbS / 2, nCbS)	
prediction_unit(x0 + (nCbS / 2), y0, nCbS / 2, nCbS)	
} else if(PartMode == PART_2NxnU) {	
prediction_unit(x0, y0, nCbS, nCbS / 4)	
prediction_unit(x0, y0 + (nCbS / 4), nCbS, nCbS *3 / 4)	
} else if(PartMode == PART_2NxnD) {	
prediction_unit(x0, y0, nCbS, nCbS *3 / 4)	

prediction_unit(x0, y0 + (nCbS * 3 / 4), nCbS, nCbS / 4)	
} else if(PartMode = = PART_nLx2N) {	
prediction_unit(x0, y0, nCbS /4, nCbS)	
prediction_unit(x0 + (nCbS / 4), y0, nCbS *3 / 4, nCbS)	
} else if(PartMode = = PART_nRx2N) {	
prediction_unit(x0, y0, nCbS *3 / 4, nCbS)	
prediction_unit(x0 + (nCbS * 3 / 4), y0, nCbS / 4, nCbS)	
} else { /* PART_NxN */	
prediction_unit(x0, y0, nCbS / 2, nCbS / 2)	
prediction_unit(x0 + (nCbS / 2), y0, nCbS / 2, nCbS / 2)	
prediction_unit(x0, y0 + (nCbS / 2), nCbS / 2, nCbS / 2)	
prediction_unit(x0 + (nCbS / 2), y0 + (nCbS / 2), nCbS / 2, nCbS / 2)	
}	
}	
if(!pcm_flag[x0][y0]) {	
if(CuPredMode[x0][y0] != MODE_INTRA && !(PartMode = = PART_2Nx2N && merge_flag[x0][y0]))	
rqt_root_cbf	ae(v)
if(rqt_root_cbf) {	
MaxTrafoDepth = (CuPredMode[x0][y0] = = MODE_INTRA ? max_transform_hierarchy_depth_intra + IntraSplitFlag : max_transform_hierarchy_depth_inter)	
transform_tree(x0, y0, x0, y0, log2CbSize, 0, 0)	
}	
}	
}	
}	

7.3.9.6 Prediction unit syntax

prediction_unit(x0, y0, nPbW, nPbH) {	**Descriptor**
if(cu_skip_flag[x0][y0]) {	
if(MaxNumMergeCand > 1)	
merge_idx[x0][y0]	ae(v)
} else { /* MODE_INTER */	
merge_flag[x0][y0]	ae(v)
if(merge_flag[x0][y0]) {	
if(MaxNumMergeCand > 1)	
merge_idx[x0][y0]	ae(v)
} else {	
if(slice_type = = B)	
inter_pred_idc[x0][y0]	ae(v)
if(inter_pred_idc[x0][y0] != Pred_L1) {	
if(num_ref_idx_l0_active_minus1 > 0)	
ref_idx_l0[x0][y0]	ae(v)
mvd_coding(x0, y0, 0)	
mvp_l0_flag[x0][y0]	ae(v)
}	
if(inter_pred_idc[x0][y0] != Pred_L0) {	
if(num_ref_idx_l1_active_minus1 > 0)	
ref_idx_l1[x0][y0]	ae(v)
if(mvd_l1_zero_flag && inter_pred_idc[x0][y0] = = Pred_BI) {	
MvdL1[x0][y0][0] = 0	
MvdL1[x0][y0][1] = 0	
} else	
mvd_coding(x0, y0, 1)	
mvp_l1_flag[x0][y0]	ae(v)
}	
}	
}	
}	

7.3.9.7 PCM sample syntax

pcm_sample(x0, y0, log2CbSize) {	**Descriptor**
for(i = 0; i < 1 << (log2CbSize << 1); i++)	
pcm_sample_luma[i]	u(v)
for(i = 0; i < (1 << (log2CbSize << 1)) >> 1; i++)	
pcm_sample_chroma[i]	u(v)
}	

7.3.9.8 Transform tree syntax

transform_tree(x0, y0, xBase, yBase, log2TrafoSize, trafoDepth, blkIdx) {	**Descriptor**
if(log2TrafoSize <= Log2MaxTrafoSize && log2TrafoSize > Log2MinTrafoSize && trafoDepth < MaxTrafoDepth && !(IntraSplitFlag && trafoDepth = = 0))	
split_transform_flag[x0][y0][trafoDepth]	ae(v)
if(log2TrafoSize > 2) {	
if(trafoDepth = = 0 || cbf_cb[xBase][yBase][trafoDepth − 1])	
cbf_cb[x0][y0][trafoDepth]	ae(v)
if(trafoDepth = = 0 || cbf_cr[xBase][yBase][trafoDepth − 1])	
cbf_cr[x0][y0][trafoDepth]	ae(v)
}	
if(split_transform_flag[x0][y0][trafoDepth]) {	
x1 = x0 + ((1 << log2TrafoSize) >> 1)	
y1 = y0 + ((1 << log2TrafoSize) >> 1)	
transform_tree(x0, y0, x0, y0, log2TrafoSize − 1, trafoDepth + 1, 0)	
transform_tree(x1, y0, x0, y0, log2TrafoSize − 1, trafoDepth + 1, 1)	
transform_tree(x0, y1, x0, y0, log2TrafoSize − 1, trafoDepth + 1, 2)	
transform_tree(x1, y1, x0, y0, log2TrafoSize − 1, trafoDepth + 1, 3)	
} else {	
if(CuPredMode[x0][y0] = = MODE_INTRA || trafoDepth != 0 || cbf_cb[x0][y0][trafoDepth] || cbf_cr[x0][y0][trafoDepth])	
cbf_luma[x0][y0][trafoDepth]	ae(v)
transform_unit (x0, y0, xBase, yBase, log2TrafoSize, trafoDepth, blkIdx)	
}	
}	

7.3.9.9 Motion vector difference syntax

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

7.3.9.10 Transform unit syntax

transform_unit(x0, y0, xBase, yBase, log2TrafoSize, trafoDepth, blkIdx) {	**Descriptor**
if(cbf_luma[x0][y0][trafoDepth] | | cbf_cb[x0][y0][trafoDepth] | | cbf_cr[x0][y0][trafoDepth]) {	
if(cu_qp_delta_enabled_flag && !IsCuQpDeltaCoded) {	
cu_qp_delta_abs	ae(v)
if(cu_qp_delta_abs)	
cu_qp_delta_sign	ae(v)
}	
if(cbf_luma[x0][y0][trafoDepth])	
residual_coding(x0, y0, log2TrafoSize, 0)	
if(log2TrafoSize > 2) {	
if(cbf_cb[x0][y0][trafoDepth])	
residual_coding(x0, y0, log2TrafoSize − 1, 1)	
if(cbf_cr[x0][y0][trafoDepth])	
residual_coding(x0, y0, log2TrafoSize − 1, 2)	
} else if(blkIdx = = 3) {	
if(cbf_cb[xBase][yBase][trafoDepth])	
residual_coding(xBase, yBase, log2TrafoSize, 1)	
if(cbf_cr[xBase][yBase][trafoDepth])	
residual_coding(xBase, yBase, log2TrafoSize, 2)	
}	
}	
}	

7.3.9.11 Residual coding syntax

residual_coding(x0, y0, log2TrafoSize, cIdx) {	**Descriptor**
if(transform_skip_enabled_flag && !cu_transquant_bypass_flag && (log2TrafoSize = = 2))	
transform_skip_flag[x0][y0][cIdx]	ae(v)
last_significant_coeff_x_prefix	ae(v)
last_significant_coeff_y_prefix	ae(v)
if(last_significant_coeff_x_prefix > 3)	
last_significant_coeff_x_suffix	ae(v)
if(last_significant_coeff_y_prefix > 3)	
last_significant_coeff_y_suffix	ae(v)
lastScanPos = 16	
lastSubBlock = (1 << (log2TrafoSize − 2)) * (1 << (log2TrafoSize − 2)) − 1	
do {	
if(lastScanPos = = 0) {	
lastScanPos = 16	
lastSubBlock− −	
}	
lastScanPos− −	
xS = ScanOrder[log2TrafoSize − 2][scanIdx][lastSubBlock][0]	
yS = ScanOrder[log2TrafoSize − 2][scanIdx][lastSubBlock][1]	
xC = (xS << 2) + ScanOrder[2][scanIdx][lastScanPos][0]	
yC = (yS << 2) + ScanOrder[2][scanIdx][lastScanPos][1]	
} while((xC != LastSignificantCoeffX) || (yC != LastSignificantCoeffY))	
for(i = lastSubBlock; i >= 0; i− −) {	
xS = ScanOrder[log2TrafoSize − 2][scanIdx][i][0]	
yS = ScanOrder[log2TrafoSize − 2][scanIdx][i][1]	
inferSbDcSigCoeffFlag = 0	
if((i < lastSubBlock) && (i > 0)) {	
coded_sub_block_flag[xS][yS]	ae(v)
inferSbDcSigCoeffFlag = 1	
}	
for(n = (i = = lastSubBlock) ? lastScanPos − 1 : 15; n >= 0; n− −) {	
xC = (xS << 2) + ScanOrder[2][scanIdx][n][0]	
yC = (yS << 2) + ScanOrder[2][scanIdx][n][1]	
if(coded_sub_block_flag[xS][yS] && (n > 0 || !inferSbDcSigCoeffFlag)) {	
significant_coeff_flag[xC][yC]	ae(v)
if(significant_coeff_flag[xC][yC])	
inferSbDcSigCoeffFlag = 0	
}	
}	
firstSigScanPos = 16	
lastSigScanPos = −1	
numGreater1Flag = 0	
lastGreater1ScanPos = −1	
for(n = 15; n >= 0; n− −) {	
xC = (xS << 2) + ScanOrder[2][scanIdx][n][0]	
yC = (yS << 2) + ScanOrder[2][scanIdx][n][1]	
if(significant_coeff_flag[xC][yC]) {	

if(numGreater1Flag < 8) {	
coeff_abs_level_greater1_flag[n]	ae(v)
numGreater1Flag++	
if(coeff_abs_level_greater1_flag[n] && lastGreater1ScanPos = = −1)	
lastGreater1ScanPos = n	
}	
if(lastSigScanPos = = −1)	
lastSigScanPos = n	
firstSigScanPos = n	
}	
}	
signHidden = (lastSigScanPos − firstSigScanPos > 3 && !cu_transquant_bypass_flag)	
if(lastGreater1ScanPos != −1)	
coeff_abs_level_greater2_flag[lastGreater1ScanPos]	ae(v)
for(n = 15; n >= 0; n− −) {	
xC = (xS << 2) + ScanOrder[2][scanIdx][n][0]	
yC = (yS << 2) + ScanOrder[2][scanIdx][n][1]	
if(significant_coeff_flag[xC][yC] && (!sign_data_hiding_flag | | !signHidden | | n != firstSigScanPos))	
coeff_sign_flag[n]	ae(v)
}	
numSigCoeff = 0	
sumAbsLevel = 0	
for(n = 15; n >= 0; n− −) {	
xC = (xS << 2) + ScanOrder[2][scanIdx][n][0]	
yC = (yS << 2) + ScanOrder[2][scanIdx][n][1]	
if(significant_coeff_flag[xC][yC]) {	
baseLevel = 1 + coeff_abs_level_greater1_flag[n] + coeff_abs_level_greater2_flag[n]	
if(baseLevel = = ((numSigCoeff < 8) ? ((n = = lastGreater1ScanPos) ? 3 : 2) : 1))	
coeff_abs_level_remaining[n]	ae(v)
TransCoeffLevel[x0][y0][cIdx][xC][yC] = (coeff_abs_level_remaining[n] + baseLevel) * (1 − 2 * coeff_sign_flag[n])	
if(sign_data_hiding_flag && signHidden) {	
sumAbsLevel += (coeff_abs_level_remaining[n] + baseLevel)	
if(n = = firstSigScanPos && ((sumAbsLevel % 2) = = 1))	
TransCoeffLevel[x0][y0][cIdx][xC][yC] = − TransCoeffLevel[x0][y0][cIdx][xC][yC]	
}	
numSigCoeff++	
}	
}	
}	
}	

7.4 Semantics

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

7.4.1 NAL unit semantics

7.4.1.1 General NAL unit semantics

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

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

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

– Otherwise, the RBSP contains the SODB as follows:

1) The first byte of the RBSP contains the (most significant, left-most) eight bits of the SODB; the next byte of the RBSP contains the next eight bits of the SODB, etc., until fewer than eight bits of the SODB remain. [Ed. (GJS): Generally, there are way too many places where the word "shall" is used. If it is not a testable constraint on the content of a conforming bitstream and is not a testable constraint on the output behaviour of the decoder, it should not be expressed using a "shall".]

2) rbsp_trailing_bits( ) are present after the SODB as follows:

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

   ii) The next bit consists of a single rbsp_stop_one_bit equal to 1.

   iii) When the rbsp_stop_one_bit is not the last bit of a byte-aligned byte, one or more rbsp_alignment_zero_bit is present to result in byte alignment.

3) One or more cabac_zero_word 16-bit syntax elements equal to 0x0000 may be present in some RBSPs after the rbsp_trailing_bits( ) at the end of the RBSP.

Syntax structures having these RBSP properties are denoted in the syntax tables using an "_rbsp" suffix. These structures 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 7-1.

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

7.4.1.2 NAL unit header semantics

forbidden_zero_bit shall be equal to 0.

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

NAL units that use nal_unit_type in the range of UNSPEC48..UNSPEC63, inclusive, for which semantics are not specified, shall not affect the decoding process specified in this Specification.

NOTE 1 – NAL unit types in the range of UNSPEC48..UNSPEC63 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 NAL unit types in the range of UNSPEC48..UNSPEC63 for different purposes, particular care must be exercised in the design of encoders that generate NAL units with nal_unit_type in the range of UNSPEC48..UNSPEC63, and in the design of decoders that interpret the content of NAL units with nal_unit_type in the range of UNSPEC48..UNSPEC63.

For purposes other than 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 2 – This requirement allows future definition of compatible extensions to this Specification.

Table 7-1 – 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, 1	TRAIL_N, TRAIL_R	Coded slice segment of a non-TSA, non-STSA trailing picture slice_segment_layer_rbsp( )	VCL
2, 3	TSA_N, TSA_R	Coded slice segment of a TSA picture slice_segment_layer_rbsp( )	VCL
4, 5	STSA_N, STSA_R	Coded slice segment of an STSA picture slice_layer_rbsp( )	VCL
6, 7	RADL_N, RADL_R	Coded slice segment of a RADL picture slice_layer_rbsp( )	VCL
8, 9	RASL_N, RASL_R,	Coded slice segment of a RASL picture slice_layer_rbsp( )	VCL
10, 12, 14	RSV_VCL_N10 RSV_VCL_N12 RSV_VCL_N14	Reserved // reserved non-RAP non-reference VCL NAL unit types	VCL
11, 13, 15	RSV_VCL_R11 RSV_VCL_R13 RSV_VCL_R15	Reserved // reserved non-RAP reference VCL NAL unit types	VCL
16, 17, 18	BLA_W_LP BLA_W_DLP BLA_N_LP	Coded slice segment of a BLA picture slice_segment_layer_rbsp( ) [Ed. (YK): BLA_W_DLP -> BLA_W_RADL?]	VCL
19, 20	IDR_W_DLP IDR_N_LP	Coded slice segment of an IDR picture slice_segment_layer_rbsp( ) [Ed. (YK: IDR_W_DLP -> IDR_W_RADL?]	VCL
21	CRA_NUT	Coded slice segment of a CRA picture slice_segment_layer_rbsp( )	VCL
22, 23	RSV_RAP_VCL22.. RSV_RAP_VCL23	Reserved // reserved RAP VCL NAL unit types [Ed.(YK): Some RAP related restrictions need to be specified for these 2 NUTs]	VCL
24..31	RSV_VCL24.. RSV_VCL31	Reserved // reserved non-RAP VCL NAL unit types [Ed.(YK): Some non-RAP related restrictions may need to be specified for these NUTs]	VCL
32	VPS_NUT	Video parameter set video_parameter_set_rbsp( )	non-VCL
33	SPS_NUT	Sequence parameter set seq_parameter_set_rbsp( )	non-VCL
34	PPS_NUT	Picture parameter set pic_parameter_set_rbsp( )	non-VCL
35	AUD_NUT	Access unit delimiter access_unit_delimiter_rbsp( )	non-VCL
36	EOS_NUT	End of sequence end_of_seq_rbsp( )	non-VCL
37	EOB_NUT	End of bitsteam end_of_bitstream_rbsp( )	non-VCL
38	FD_NUT	Filler data filler_data_rbsp( )	non-VCL

39, 40	PREFIX_SEI_NUT SUFFIX_SEI_NUT	Supplemental enhancement information (SEI) sei_rbsp( )	non-VCL
41..47	RSV_NVCL41.. RSV_NVCL47	Reserved	non-VCL
48..63	UNSPEC48.. UNSPEC63	Unspecified	non-VCL

NOTE 3 – A CRA picture may have associated RASL or RADL pictures present in the bitstream.

NOTE 4 – A BLA picture having nal_unit_type equal to BLA_W_LP may have associated RASL or RADL pictures present in the bitstream. A BLA picture having nal_unit_type equal to BLA_W_DLP does not have associated RASL pictures present in the bitstream, but may have associated RADL pictures in the bitstream. A BLA picture having nal_unit_type equal to BLA_N_LP does not have associated leading pictures present in the bitstream.

NOTE 5 – An 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_DLP does not have associated RASL pictures present in the bitstream, but may have associated RADL pictures in the bitstream.

NOTE 6 – When the value of nal_unit_type is equal to TRAIL_N, TSA_N or STSA_N, the decoded picture is not included in any of RefPicSetStCurrBefore, RefPicSetStCurrAfter and RefPicSetLtCurr of any picture with the same value of TemporalId. A coded picture with nal_unit_type equal to TRAIL_N, TSA_N or STSA_N may be discarded without affecting the decodability of other pictures with the same value of TemporalId.

The variable IdrPicFlag is specified as

$$IdrPicFlag = (nal\_unit\_type == IDR\_W\_DLP \,||\, nal\_unit\_type == IDR\_N\_LP) \quad (7\text{-}1)$$

The variable RapPicFlag is specified as

$$RapPicFlag = (nal\_unit\_type >= 16 \;\&\&\; nal\_unit\_type <= 23) \quad (7\text{-}2)$$

All coded slice segment NAL units of an access unit shall have the same value of nal_unit_type.

Each picture, other than the first picture in the bitstream, is considered to be associated with the previous RAP picture in decoding order.

When a picture is a leading picture, it shall be a RADL or RASL picture.

When a picture is a trailing picture, it shall not be a RADL or RASL picture.

When a picture is a leading picture, it shall precede, in decoding order, all trailing pictures that are associated with the same RAP picture.

No RASL pictures shall be present in the bitstream that are associated with a BLA picture having nal_unit_type equal to BLA_W_DLP or BLA_N_LP.

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

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

NOTE 7 – It is possible to perform random access at the position of a RAP access unit by discarding all access units before the RAP access unit (and to correctly decode the RAP 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 needs to be activated.

Any picture that has PicOutputFlag equal to 1 that precedes a RAP picture in decoding order shall precede the RAP picture in output order and shall precede any RADL picture associated with the RAP picture in output order.

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

Any RASL picture associated with a CRA picture shall follow, in output order, any RAP picture that precedes the CRA picture in decoding order. [Ed. (TK) Why do we need this restriction? (GJS): Just so that each RAP-to-RAP chunk is a roughly-sensible segment of the video content, I suppose. We have other such "unnecessary" constraints, such as the constraint that RASL's need to be leading pictures and the constraints on relative decoding order of RASL, RADL and trailing pictures.]

When sps_temporal_id_nesting_flag is equal to 1 and TemporalId is greater than 0, the nal_unit_type shall be equal to TSA_R, TSA_N, RADL_R, RADL_N, RASL_R, or RASL_N.

nuh_reserved_zero_6bits shall be equal to 0. Other values of nuh_reserved_zero_6bits may be specified in the future by ITU-T | ISO/IEC. For purposes other than as specified in Annex C, decoders shall ignore (i.e. remove from the bitstream and discard) all NAL units with values of nuh_reserved_zero_6bits not equal to 0.

NOTE 8 – It is anticipated that in future scalable or 3D video coding extensions of this specification, this syntax element will be used to identify additional layers that may be present in the coded video sequence, wherein a layer may be, e.g. a spatial scalable layer, a quality scalable layer, a texture view or a depth view.

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 specified as

$$TemporalId = nuh\_temporal\_id\_plus1 - 1 \quad (7\text{-}3)$$

If nal_unit_type is in the range of 16 to 23 (coded slice segment of a RAP picture), inclusive, TemporalId shall be equal to 0; otherwise, when nal_unit_type is equal to TSA_R, TSA_N, STSA_R, or STSA_N, TemporalId shall not be equal to 0.

The value of TemporalId shall be the same for all VCL NAL units of an access unit. The value of TemporalId of an access unit is the value of the TemporalId of the VCL NAL units of the access unit.

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

– If nal_unit_type is equal to VPS_NUT, SPS_NUT, EOS_NUT, or EOB_NUT, TemporalId shall be equal to 0.

– Otherwise, if nal_unit_type is equal to AUD_NUT or FD_NUT, TemporalId shall be equal to the TemporalId of the access unit containing the non-VCL NAL unit.

– Otherwise, when nal_unit_type is equal to PREFIX_SEI_NUT or SUFFIX_SEI_NUT, TemporalId shall be greater than or equal to the TemporalId of the access unit containing the NAL unit.

NOTE 9 – 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 access units to which the non-VCL NAL unit applies. When nal_unit_type is equal to VPS_NUT or SPS_NUT, TemporalId must be equal to 0, as a sequence parameter set applies at least to one RAP access unit. When nal_unit_type is equal to AUD_NUT or FD_NUT, TemporalId must be equal to the TemporalId of the access unit containing the non-VCL NAL unit, as access unit delimiter or filler data only applies to the containing access unit. When nal_unit_type is equal to PPS_NUT, TemporalId may be less than, equal to, or greater than the TemporalId of the containing access unit, as a picture parameter set may be repeated in access units not referring to the picture parameter set (for error resilience purposes), and all picture parameter sets may be included in the beginning of a bitstream, wherein the first coded picture has TemporalId equal to 0. When nal_unit_type is equal to PREFIX_SEI_NUT or SUFFIX_SEI_NUT, TemporalId may be greater than or equal to the TemporalId of the containing access unit, as an SEI NAL unit may contain information, e.g. in a buffering period SEI message or a picture timing SEI message that applies to a bitstream subset that includes access units for which the TemporalId values are greater than the TemporalId of the access unit containing the SEI NAL unit.

7.4.1.3 Encapsulation of an SODB within an RBSP (informative)

This subclause 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 larger 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 is 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.

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

– no byte-aligned start code prefix is emulated within the NAL unit,

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

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

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

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

7.4.1.4.1 Order of video, sequence, and picture parameter set RBSPs and their activation

This subclause specifies the activation process of video, sequence and picture parameter sets.

NOTE 1 – The video, sequence, and picture parameter set mechanism decouples the transmission of infrequently changing information from the transmission of coded block data. Video, sequence, and picture parameter sets may, in some applications, be conveyed "out-of-band".

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

When a picture parameter set RBSP (with a particular value of pps_pic_parameter_set_id) is not active and it is referred to by a coded slice segment NAL unit (using a value of slice_pic_parameter_set_id equal to the pps_pic_parameter_set_id value), it is activated. This picture parameter set RBSP is called the active picture parameter set RBSP until it is deactivated by the activation of another picture parameter set RBSP. A picture parameter set RBSP, with that particular value of pps_pic_parameter_set_id, shall be available to the decoding process prior to its activation, included in at least one access unit with TemporalId less than or equal to the TemporalId of the picture parameter set NAL unit, unless the picture parameter set is provided through external means.

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

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

When a sequence parameter set RBSP (with a particular value of sps_seq_parameter_set_id) is not already active and it is referred to by activation of a picture parameter set RBSP (using a value of pps_seq_parameter_set_id equal to the sps_seq_parameter_set_id value) or is referred to by an SEI NAL unit containing an active parameter sets SEI message (using that value of sps_seq_parameter_set_id), it is activated. [Ed. (GJS): Check syntax element name usage.] This sequence parameter set RBSP is called the active sequence parameter set RBSP until it is deactivated by the activation of another sequence parameter set RBSP. A sequence parameter set RBSP, with that particular value of sps_seq_parameter_set_id shall be available to the decoding process prior to its activation, included in at least one access unit with TemporalId equal to 0, unless the sequence parameter set is provided through external means. An activated sequence parameter set RBSP shall remain active for the entire coded video sequence.

NOTE 2 – Because a CRA access unit that is the first access unit in the bitstream, an IDR access unit, or a BLA access unit begins a new coded video sequence and an activated sequence parameter set RBSP must remain active for the entire coded video sequence, a sequence parameter set RBSP can only be activated by an active parameter sets SEI message when the active

parameter sets SEI message is part of a CRA access unit that is the first access unit in the bitstream, an IDR access unit, or a BLA access unit.

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

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

When a video parameter set RBSP (with a particular value of vps_video_parameter_set_id) is not already active and it is referred to by activation of a sequence parameter set RBSP (using a value of sps_video_parameter_set_id equal to the vps_video_parameter_set_id value) or is referred to by an SEI NAL unit containing an active parameter sets SEI message (using that value of sps_seq_parameter_set_id), it is activated. [Ed. (GJS): Check syntax element name usage.] This video parameter set RBSP is called the active video parameter set RBSP until it is deactivated by the activation of another video parameter set RBSP. A video parameter set RBSP, with that particular value of vps_video_parameter_set_id shall be available to the decoding process prior to its activation, included in at least one access unit with TemporalId equal to 0, unless the video parameter set is provided through external means. An activated video parameter set RBSP shall remain active for the entire coded video sequence.

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

NOTE 3 – If video parameter set RBSP, sequence parameter set RBSP, or picture parameter set RBSP are conveyed within the bitstream, these constraints impose an order constraint on the NAL units that contain the video parameter set RBSP, sequence parameter set RBSP, or picture parameter set RBSP, respectively. Otherwise (video parameter set RBSP, sequence parameter set RBSP, or picture parameter set RBSP are conveyed by other means not specified in this Specification), they must be available to the decoding process in a timely fashion such that these constraints are obeyed.

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

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

7.4.1.4.2 Order of access units and association to coded video sequences

A bitstream conforming to this Specification consists of one or more coded video sequences.

A coded video sequence consists of one or more access units. The order of NAL units and coded pictures and their association to access units is described in subclause 7.4.1.4.3.

If a coded video sequence is the first coded video sequence in the bitstream, the first access unit of the coded video sequence is a RAP access unit, which may be an IDR, BLA or CRA access unit; otherwise, the first access unit of the coded video sequence is an IDR or BLA access unit. All access units in a coded video sequence that are not the first access unit in the coded video sequence are non-IDR and non-BLA access units.

It is a requirement of bitstream conformance that, when present, the next access unit after an access unit that contains an end of sequence NAL unit shall be an IDR or BLA access unit.

It is a requirement of bitstream conformance that, when present, the next access unit after an access unit that contains an end of bitstream NAL unit shall be a RAP access unit, which may be an IDR, BLA or CRA access unit.

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

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

An access unit consists of one coded picture and zero or more non-VCL NAL units. The association of VCL NAL units to coded pictures is described in subclause 7.4.1.4.4.

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

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

– access unit delimiter NAL unit (when present),

– video parameter set NAL unit (when present),

– sequence parameter set NAL unit (when present),

– picture parameter set NAL unit (when present),

– Prefix SEI NAL unit (when present),

– NAL units with nal_unit_type in the range of RSV_NVCL41..RSV_NVCL44 (when present),

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

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

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

– When any prefix SEI NAL units are present, they shall not follow the last VCL NAL unit of the access unit. [Ed. (GJS): Can the currently-specified prefix SEI NAL units (that have a whole-picture scope) be placed between the VCL NAL units of the picture?]

– When a prefix SEI NAL unit having nuh_reserved_zero_6bits equal to 0 and containing a buffering period SEI message is present, the prefix SEI NAL unit shall also contain an active parameter sets SEI message, and the active parameter sets SEI message and the buffering period SEI message shall be the first and the second SEI message payloads, respectively, of the first SEI NAL unit in the access unit. [Ed. (YK): This restriction should only apply to SEI NAL units with nuh_reserved_zero_6bits equal to 0. Since nuh_reserved_zero_6bits is required to be equal to 0 in this version of this Specification, now we don't have to mention nuh_reserved_zero_6bits equal to 0, but it may be safer if we do say it now, though it is not needed. Also, if the parsing dependency of picture timing SEI message and decoding unit information SEI message can be resolved, we don't need to have this restriction at all.]

– NAL units having nal_unit_type equal to FD_NUT, SUFFIX_SEI_NUT, in the range of RSV_NVCL45..RSV_NVCL47, or in the range of UNSPEC48..UNSPEC63 shall not precede the first VCL NAL unit of the coded picture.

– When an end of sequence NAL unit is present, it shall be the last NAL unit in the access unit other than an end of bitstream NAL unit (if present).

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

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

The structure of access units not containing any NAL units with nal_unit_type equal to FD_NUT, VPS_NUT, SPS_NUT, PPS_NUT, or in the ranges of RSV_RAP_VCL22..RSV_RAP_VCL23, RSV_VCL24..RSV_VCL31, RSV_NVCL41..RSV_NVCL47, or UNSPEC48..UNSPEC63 is shown in Figure 7-1.

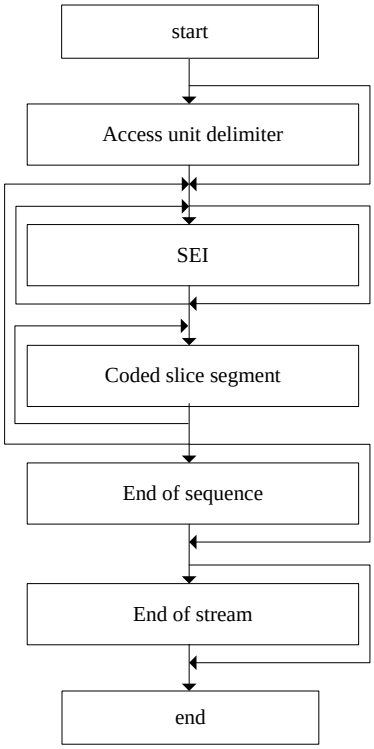


Figure 7-1 – Structure of an access unit not containing any NAL units with nal_unit_type equal to FD_NUT, SUFFIX_SEI_NUT, VPS_NUT, SPS_NUT, PPS_NUT, or in the ranges of RSV_RAP_VCL22.. RSV_RAP_VCL23, RSV_VCL24..RSV_VCL31, RSV_NVCL41..RSV_NVCL47, or UNSPEC48..UNSPEC63

7.4.1.4.4 Order of VCL NAL units and association to coded pictures

This subclause specifies the order of VCL NAL units and association to coded pictures.

Each VCL NAL unit is part of a coded picture.

The order of the VCL NAL units within a coded picture is constrained as follows [Ed (YK): May need to improve to support the case when separate_colour_plane_flag = = 1.]:

– The first VCL NAL unit of the coded picture shall have first_slice_segment_in_pic_flag equal to 1.

– Let sliceSegAddrA and sliceSegAddrB be the slice_segment_address values of any two coded slice segment NAL units A and B within the same coded picture. When one or more of the following conditions are true, coded slice segment NAL unit A shall precede the coded slice segment NAL unit B.

  – TileId[CtbAddrRStoTS[sliceSegAddrA]] is less than TileId[CtbAddrRStoTS[sliceSegAddrB]].

  – TileId[CtbAddrRStoTS[sliceSegAddrA]] is equal to TileId[CtbAddrRStoTS[sliceSegAddrB]] and CtbAddrRStoTS[sliceSegAddrA] is less than CtbAddrRStoTS[sliceSegAddrB].

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

7.4.2.1 Video parameter set RBSP semantics

NOTE 1 – Encoders conforming to this version of this Specification are required to include video parameter set NAL units in the bitstream, as specified in subclause 7.4.1.4.1. However, the video parameter set RBSP contains information that is not necessary for operation of the decoding process of this version of this Specification. Decoders conforming to this version of this Specification may ignore (remove from the bitstream and discard) the content of all video parameter set NAL units.

Any two instances of the syntax structure hrd_parameters( ) included in a video parameter set RBSP shall not have the same content.

Any two instances of the syntax structure operation_point_set( ) included in a video parameter set RBSP shall not have the same content.

vps_video_parameter_set_id identifies the video parameter set for reference by other syntax elements.

vps_reserved_three_2bits shall be equal to 3 in bitstreams conforming to this version of this Specification. Other values for vps_reserved_three_2bits are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore the value of vps_reserved_three_2bits.

vps_reserved_zero_6bits shall be equal to 0 in bitstreams conforming to this version of this Specification. Other values for vps_reserved_zero_6bits are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore the value of vps_reserved_zero_6bits.

NOTE 2 – It is anticipated that in future scalable or 3D video coding extensions of this Specification, this field specifies the maximum number of layers that may be present in the coded video sequence, wherein a layer may e.g. be a spatial scalable layer, a quality scalable layer, a texture view or a depth view.

vps_max_sub_layers_minus1 plus 1 specifies the maximum number of temporal sub-layers that may be present in the bitstream. The value of vps_max_sub_layers_minus1 shall be in the range of 0 to 6, inclusive.

vps_temporal_id_nesting_flag, when vps_max_sub_layers_minus1 is greater than 0, specifies whether inter prediction is additionally restricted for coded video sequences referring to the video parameter set. When vps_max_sub_layers_minus1 is equal to 0, vps_temporal_id_nesting_flag shall be equal to 1.

NOTE 3 – The syntax element vps_temporal_id_nesting_flag is used to indicate that temporal sub-layer up-switching, i.e. switching from decoding of up to any TemporalId tIdN to decoding up to any TemporalId tIdM that is greater than tIdN, is always possible.

vps_reserved_0xffff_16bits shall be equal to 0xFFFF in bitstreams conforming to this version of this Specification. Other values for vps_reserved_0xffff_16bits are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore the value of vps_reserved_0xffff_16bits.

NOTE 4 – It is anticipated that in future scalable or 3D video coding extensions of this Specification, this field specifies the byte offset of the next set of fixed-length coded information in the video parameter set NAL unit, starting from the beginning of the NAL unit. Video parameter set information for non-base layer or view starts from a byte-aligned position of the video parameter set NAL unit, with fixed-length coded information that is essential for session negotiation and/or capability exchange. The byte offset would then help to locate and access those essential information in the video parameter set NAL unit without the need of entropy decoding, which may not be possible for some network entities that may desire to access only the information in the video parameter set that is essential for session negotiation and/or capability exchange.

vps_sub_layer_ordering_info_present_flag equal to 1 specifies that vps_max_dec_pic_buffering[i], vps_max_num_reorder_pics[i], and vps_max_latency_increase[i] are present for vps_max_sub_layers_minus1 + 1 sub-layers, vps_sub_layer_ordering_info_present_flag equal to 0 specifies that the values of vps_max_dec_pic_buffering[vps_max_sub_layers_minus1], vps_max_num_reorder_pics[vps_max_sub_layers_minus1], and vps_max_latency_increase[vps_max_sub_layers_minus1] apply to all sub-layers.

vps_max_dec_pic_buffering[i] specifies the required size of the decoded picture buffer in units of picture storage buffers when HighestTid is equal to i. The value of vps_max_dec_pic_buffering[i] shall be in the range of 0 to MaxDpbSize (as specified in subclause A.4), inclusive. When i is greater than 0, vps_max_dec_pic_buffering[i] shall be greater than or equal to vps_max_dec_pic_buffering[i − 1]. When vps_max_dec_pic_buffering[i] is not present for i in the range of 0 to vps_max_sub_layers_minus1 − 1 due to vps_sub_layer_ordering_info_present_flag being equal to 0, it is inferred to be equal to vps_max_dec_pic_buffering[vps_max_sub_layers_minus1].

vps_max_num_reorder_pics[i] indicates the maximum allowed number of pictures preceding any picture in decoding order and succeeding that picture in output order when HighestTid is equal to i. The value of vps_max_num_reorder_pics[i] shall be in the range of 0 to vps_max_dec_pic_buffering[i], inclusive. When i is greater than 0, vps_max_num_reorder_pics[i] shall be greater than or equal to vps_max_num_reorder_pics[i − 1]. When vps_max_num_reorder_pics[i] is not present for i in the range of 0 to vps_max_sub_layers_minus1 − 1 due to vps_sub_layer_ordering_info_present_flag being equal to 0, it is inferred to be equal to vps_max_num_reorder_pics[vps_max_sub_layers_minus1].

vps_max_latency_increase[i] not equal to 0 is used to compute the value of MaxLatencyPictures[i] as specified by setting MaxLatencyPictures[i] to vps_max_num_reorder_pics[i] + vps_max_latency_increase[i]. When vps_max_latency_increase[i] is not equal to 0, the value of MaxLatencyPictures[i] specifies the maximum number of pictures that can precede any picture in the coded video sequence in output order and follow that picture in decoding order when HighestTid is equal to i. When vps_max_latency_increase[i] is equal to 0, no corresponding limit is expressed. The value of vps_max_latency_increase[i] shall be in the range of 0 to $2^{32} - 2$, inclusive. When vps_max_latency_increase[i] is not present for i in the range of 0 to vps_max_sub_layers_minus1 − 1 due to vps_sub_layer_ordering_info_present_flag being equal to 0, it is inferred to be equal to vps_max_latency_increase[vps_max_sub_layers_minus1].

vps_max_nuh_reserved_zero_layer_id specifies the maximum allowed value of nuh_reserved_zero_6bits of all NAL units in the coded video sequence.

vps_num_op_sets_minus1 plus 1 specifies the number of operation point sets that are specified by the video parameter set. In bitstreams conforming to this version of this Specification, the value of vps_num_op_sets_minus1 shall be equal to 0. Although the value of vps_num_op_sets_minus1 is required to be equal to 0 in this version of this Specification, decoders shall allow other values of vps_num_op_sets_minus1 in the range of 0 to 1023, inclusive, to appear in the syntax.

Each operation point is identified by a set OpLayerIdSet, which includes and only includes the set of nuh_reserved_zero_6bits values of all NAL units included in the operation point, and a variable OpTid, which is equal to the highest TemporalId of NAL units included in the operation point. The bitstream subset associated with the operation point identified by OpLayerIdSet and OpTid refers to the output of the sub-bitstream extraction process as specified in subclause 10.1 with the bitstream, OpTid and OpLayerIdSet as inputs. The OpLayerIdSet and OpTid that identify an operation point are also referred to the OpLayerIdSet and OpTid of the operation point, respectively.

Each operation point set consists of all operation points for which the included NAL units have the same set OpLayerIdSet and different values of OpTid. An operation point is thus also referred to as being identified by the particular value of OpLayerIdSet and is considered to have OplayerIdSet equal to the particular value. The OpLayerIdSet that identifies an operation point set is also referred to as the OpLayerIdSet of the operation point set.

The 0-th operation point set specified by the video parameter set is identified by the OpLayerIdSet that includes the value 0 only. The i-th operation point set, with i greater than 0, specified by the video parameter set is specified by the syntax structure operation_point_set(i).

vps_num_hrd_parameters specifies the number of hrd_parameters( ) syntax structures present in the video parameter set RBSP. In bitstreams conforming to this version of this Specification, the value of vps_num_hrd_parameters shall be less than or equal to 1. Although the value of vps_num_hrd_parameters is required to be less than or equal to 1 in this version of this Specification, decoders shall allow other values of vps_num_hrd_parameters in the range of 0 to 1024, inclusive to appear in the syntax.

hrd_op_set_idx[i] specifies the index, in the list of operation point sets specified by the video parameter set, of the operation point set to which the i-th hrd_parameters( ) syntax structure in the video parameter set applies. In bitstreams conforming to this version of this Specification, the value of hrd_op_set_idx[i] shall be equal to 0. Although the value of hrd_op_set_idx[i] is required to be less than or equal to 1 in this version of this Specification, decoders shall allow other values of hrd_op_set_idx[i] in the range of 0 to 1023 to appear in the syntax.

cprms_present_flag[i] equal to 1 specifies that the HRD parameters that are common for all sub-layers are present in the i-th hrd_parameters( ) syntax structure in the video parameter set. cprms_present_flag[i] equal to 0 specifies that the HRD parameters that are common for all sub-layers are not present in the i-th hrd_parameters( ) syntax structure in the video parameter set and are derived to be the same as the (i − 1)-th hrd_parameters( ) syntax structure in the video parameter set. cprms_present_flag[0] is inferred to be equal to 1.

vps_extension_flag equal to 0 specifies that no vps_extension_data_flag syntax elements are present in the video parameter set RBSP syntax structure. vps_extension_flag shall be equal to 0 in bitstreams conforming to this version of this Specification. The value of 1 for vps_extension_flag is reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore all data that follow the value 1 for vps_extension_flag in a video parameter set NAL unit.

vps_extension_data_flag may have any value. It shall not affect the conformance to profiles specified in this version of this Specification.

7.4.2.2 Sequence parameter set RBSP semantics

sps_video_parameter_set_id identifies the vps_video_parameter_set_id of the active video parameter set.

sps_max_sub_layers_minus1 plus 1 specifies the maximum number of temporal sub-layers that may be present in each coded video sequence referring to the sequence parameter set. The value of sps_max_sub_layers_minus1 shall be in the range of 0 to 6, inclusive.

sps_temporal_id_nesting_flag, when sps_max_sub_layers_minus1 is greater than 0, specifies whether inter prediction is additionally restricted for coded video sequences referring to the sequence parameter set. When vps_temporal_id_nesting_flag is equal to 1, sps_temporal_id_nesting_flag shall be equal to 1. When sps_max_sub_layers_minus1 is equal to 0, sps_temporal_id_nesting_flag shall be equal to 1.

NOTE 1 – The syntax element sps_temporal_id_nesting_flag is used to indicate that temporal up-switching, i.e. switching from decoding up to any TemporalId tIdN to decoding up to any TemporalId tIdM that is greater than tIdN, is always possible in the coded video sequence.

sps_seq_parameter_set_id provides an identifier for the sequence parameter set for reference by other syntax elements. The value of sps_seq_parameter_set_id shall be in the range of 0 to 15, inclusive.

chroma_format_idc specifies the chroma sampling relative to the luma sampling as specified in subclause 6.2. The value of chroma_format_idc shall be in the range of 0 to 3, inclusive.

separate_colour_plane_flag equal to 1 specifies that the three colour components of the 4:4:4 chroma format are coded separately. separate_colour_plane_flag equal to 0 specifies that the colour components are not coded separately. When separate_colour_plane_flag is not present, it is inferred to be equal to 0. When 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) that each use the monochrome coding syntax. In this case, each colour plane is associated with a specific colour_plane_id value.

NOTE 2 – There is no dependency in decoding processes between the colour planes having different colour_plane_id values. For example, the decoding process of a monochrome picture with one value of colour_plane_id does not use any data from monochrome pictures having different values of colour_plane_id for inter prediction.

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

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

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

pic_width_in_luma_samples specifies the width of each decoded picture in units of luma samples. pic_width_in_luma_samples shall not be equal to 0 and shall be an integer multiple of MinCbSizeY.

pic_height_in_luma_samples specifies the height of each decoded picture in units of luma samples. pic_height_in_luma_samples shall not be equal to 0 and shall be an integer multiple of MinCbSizeY.

conformance_window_flag equal to 1 indicates that the conformance cropping window offset parameters follow next in the sequence parameter set. conformance_window_flag equal to 0 indicates that the conformance cropping window offset parameters are not present.

conf_win_left_offset, **conf_win_right_offset**, **conf_win_top_offset**, and **conf_win_bottom_offset** specify the samples of the pictures in the coded video sequence that are output from the decoding process, in terms of a rectangular region specified in picture coordinates for output. When conformance_window_flag is equal to 0, the values of conf_win_left_offset, conf_win_right_offset, conf_win_top_offset, and 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 * conf_win_left_offset to pic_width_in_luma_samples − (SubWidthC * conf_win_right_offset + 1) and vertical picture coordinates from SubHeightC * conf_win_top_offset to pic_height_in_luma_samples − (SubHeightC * conf_win_bottom_offset + 1), inclusive. It is a requirement of bitstream conformance that the value of conf_win_left_offset shall be in the range of 0 to (pic_width_in_luma_samples / SubWidthC) − (conf_win_right_offset + 1), inclusive; and that the value of conf_win_top_offset shall be in the range of 0 to (pic_height_in_luma_samples / SubHeightC) − (conf_win_bottom_offset + 1), inclusive.

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.

bit_depth_luma_minus8 + 8 specifies the bit depth of the samples of the luma array and the value of the luma quantization parameter range offset $QpBdOffset_Y$, as specified by

$$BitDepth_Y = 8 + bit\_depth\_luma\_minus8 \quad (7\text{-}4)$$
$$QpBdOffset_Y = 6 * bit\_depth\_luma\_minus8 \quad (7\text{-}5)$$

bit_depth_luma_minus8 shall be in the range of 0 to 6, inclusive.

bit_depth_chroma_minus8 + 8 specifies the bit depth of the samples of the chroma arrays and the value of the chroma quantization parameter range offset $QpBdOffset_C$, as specified by

$$BitDepth_C = 8 + bit\_depth\_chroma\_minus8 \quad (7\text{-}6)$$
$$QpBdOffset_C = 6 * bit\_depth\_chroma\_minus8 \quad (7\text{-}7)$$

bit_depth_chroma_minus8 shall be in the range of 0 to 6, inclusive.

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:

$$MaxPicOrderCntLsb = 2^{(log2\_max\_pic\_order\_cnt\_lsb\_minus4 + 4)} \quad (7\text{-}8)$$

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

sps_sub_layer_ordering_info_present_flag equal to 1 specifies that sps_max_dec_pic_buffering[i], sps_max_num_reorder_pics[i], and sps_max_latency_increase[i] are present for sps_max_sub_layers_minus1 + 1 sub-layers, sps_sub_layer_ordering_info_present_flag equal to 0 specifies that the values of sps_max_dec_pic_buffering[sps_max_sub_layers_minus1], sps_max_num_reorder_pics[sps_max_sub_layers_minus1], and sps_max_latency_increase[sps_max_sub_layers_minus1] apply to all sub-layers.

sps_max_dec_pic_buffering[i] specifies the maximum required size of the decoded picture buffer in units of picture storage buffers when HighestTid is equal to i. The value of sps_max_dec_pic_buffering[i] shall be in the range of 0 to MaxDpbSize (as specified in subclause A.4), inclusive. When i is greater than 0, sps_max_dec_pic_buffering[i] shall be greater than or equal to sps_max_dec_pic_buffering[i − 1]. The value of sps_max_dec_pic_buffering[i] shall be less than or equal to vps_max_dec_pic_buffering[i] for each value of i. When sps_max_dec_pic_buffering[i] is not present for i in the range of 0 to sps_max_sub_layers_minus1 − 1 due to sps_sub_layer_ordering_info_present_flag being equal to 0, it is inferred to be equal to sps_max_dec_pic_buffering[sps_max_sub_layers_minus1].

sps_max_num_reorder_pics[i] indicates the maximum allowed number of pictures preceding any picture in decoding order and succeeding that picture in output order when HighestTid is equal to i. The value of sps_max_num_reorder_pics[i] shall be in the range of 0 to sps_max_dec_pic_buffering[i], inclusive. When i is greater than 0, sps_max_num_reorder_pics[i] shall be greater than or equal to sps_max_num_reorder_pics[i − 1]. The value of sps_max_num_reorder_pics[i] shall be less than or equal to vps_max_num_reorder_pics[i] for each value of i. When sps_max_num_reorder_pics[i] is not present for i in the range of 0 to sps_max_sub_layers_minus1 − 1 due to sps_sub_layer_ordering_info_present_flag being equal to 0, it is inferred to be equal to sps_max_num_reorder_pics[sps_max_sub_layers_minus1].

sps_max_latency_increase[i] not equal to 0 is used to compute the value of MaxLatencyPictures[i] as specified by setting MaxLatencyPictures[i] equal to sps_max_num_reorder_pics[i] + sps_max_latency_increase[i]. When sps_max_latency_increase[i] is not equal to 0, the value of MaxLatencyPictures[i] specifies the maximum number of pictures that can precede any picture in the coded video sequence in output order and follow that picture in decoding order when HighestTid is equal to i. When sps_max_latency_increase[i] is equal to 0, no corresponding limit is expressed. The value of sps_max_latency_increase[i] shall be in the range of 0 to $2^{32} − 2$, inclusive. The value of sps_max_latency_increase[i] shall be less than or equal to vps_max_latency_increase[i] for each value of i. When sps_max_latency_increase[i] is not present for i in the range of 0 to sps_max_sub_layers_minus1 − 1 due to sps_sub_layer_ordering_info_present_flag being equal to 0, it is inferred to be equal to sps_max_latency_increase[sps_max_sub_layers_minus1].

log2_min_luma_coding_block_size_minus3 specifies the minimum size of a luma coding block.

log2_diff_max_min_luma_coding_block_size specifies the difference between the maximum and minimum luma coding block size.

The variables Log2MinCbSizeY, Log2CtbSizeY, MinCbSizeY, CtbSizeY, PicWidthInMinCbsY, PicWidthInCtbsY, PicHeightInMinCbsY, PicHeightInCtbsY, PicSizeInMinCbsY, PicSizeInCtbsY, and PicSizeInSamplesY are set as follows.

Log2MinCbSizeY = log2_min_luma_coding_block_size_minus3 + 3 (7-9)
Log2CtbSizeY = Log2MinCbSizeY + log2_diff_max_min_luma_coding_block_size (7-10)
MinCbSizeY = 1 << Log2MinCbSizeY (7-11)
CtbSizeY = 1 << Log2CtbSizeY (7-12)
PicWidthInMinCbsY = pic_width_in_luma_samples / MinCbSizeY (7-13)
PicWidthInCtbsY = Ceil(pic_width_in_luma_samples ÷ CtbSizeY) (7-14)
PicHeightInMinCbsY = pic_height_in_luma_samples / MinCbSizeY (7-15)
PicHeightInCtbsY = Ceil(pic_height_in_luma_samples ÷ CtbSizeY) (7-16)
PicSizeInMinCbsY = PicWidthInMinCbsY * PicHeightInMinCbsY (7-17)
PicSizeInCtbsY = PicWidthInCtbsY * PicHeightInCtbsY (7-18)
PicSizeInSamplesY = pic_width_in_luma_samples * pic_height_in_luma_samples (7-19)
PicWidthInSamplesC = pic_width_in_luma_samples / SubWidthC (7-20)
PicHeightInSamplesC = pic_height_in_luma_samples / SubHeightC (7-21)

[Ed. (GJS): Variable names MinCbSizeY and CtbSizeY violate the editorial convention by being substrings of other variable names. (BB): Consider rename of Log2MinCbSizeY, Log2CtbSizeY and log2TrafoSize to MinCbLog2SizeY, CtbLog2SizeY, and trafoLog2Size (and other similar names, if any).]

The variables CtbWidthC and CtbHeightC, which specify the width and height, respectively, of the array for each chroma coding tree block, are derived as follows.

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

– Otherwise, CtbWidthC and CtbHeightC are derived as

$$CtbWidthC = CtbSizeY / SubWidthC \quad (7\text{-}22)$$
$$CtbHeightC = CtbSizeY / SubHeightC \quad (7\text{-}23)$$

log2_min_transform_block_size_minus2 specifies the minimum transform block size.

The variable Log2MinTrafoSize is set equal to log2_min_transform_block_size_minus2 + 2. The bitstream shall not contain data that result in Log2MinTrafoSize greater than or equal to Log2MinCbSizeY.

log2_diff_max_min_transform_block_size specifies the difference between the maximum and minimum transform block size.

The variable Log2MaxTrafoSize is set equal to log2_min_transform_block_size_minus2 + 2 + log2_diff_max_min_transform_block_size.

The bitstream shall not contain data that result in Log2MaxTrafoSize greater than Min(Log2CtbSizeY, 5).

The array ScanOrder[log2BlockSize][scanIdx][sPos][sComp] specifies the mapping of the scan position sPos, ranging from 0 to ((1 << log2BlockSize) * (1 << log2BlockSize)) − 1, inclusive, to horizontal and vertical components of the scan-order matrix. The array index scanIdx equal to 0 specifies an up-right diagonal scan order, scanIdx equal to 1 specifies a horizontal scan order, and scanIdx equal to 2 specifies a vertical scan order. The array index sComp equal to 0 specifies the horizontal component and the array index sComp equal to 1 specifies the vertical component. The array ScanOrder is derived as follows.

For the variable log2BlockSize ranging from Min(2, Log2MinTrafoSize − 2) to 3, inclusive, the scanning order array ScanOrder is derived as follows.

– The up-right diagonal scan order array initialization process as specified in subclause 6.5.3 is invoked with 1 << log2BlockSize as input and the output is assigned to ScanOrder[log2BlockSize][0].

– The horizontal scan order array initialization process as specified in subclause 6.5.4 is invoked with 1 << log2BlockSize as input and the output is assigned to ScanOrder[log2BlockSize][1].

– The vertical scan order array initialization process as specified in subclause 6.5.5 is invoked with 1 << log2BlockSize as input and the output is assigned to ScanOrder[log2BlockSize][2].

max_transform_hierarchy_depth_inter specifies the maximum hierarchy depth for transform units of coding units coded in inter prediction mode. The value of max_transform_hierarchy_depth_inter shall be in the range of 0 to Log2CtbSizeY − Log2MinTrafoSize, inclusive.

max_transform_hierarchy_depth_intra specifies the maximum hierarchy depth for transform blocks of coding blocks coded in intra prediction mode. The value of max_transform_hierarchy_depth_intra shall be in the range of 0 to Log2CtbSizeY − Log2MinTrafoSize, inclusive.

scaling_list_enable_flag equal to 1 specifies that a scaling list is used for the scaling process for transform coefficients. scaling_list_enable_flag equal to 0 specifies that scaling list is not used for the scaling process for transform coefficients.

sps_scaling_list_data_present_flag equal to 1 specifies that scaling list data are present in the sequence parameter set. sps_scaling_list_data_present_flag equal to 0 specifies that scaling list data are not present in the sequence parameter set. When not present, the value of sps_scaling_list_data_present_flag is inferred to be equal to 0. When scaling_list_enable_flag is equal to 1 and sps_scaling_list_data_present_flag is equal to 0, the default scaling list data is used to derive the array ScalingFactor as described in the scaling list data semantics specified in subclause 7.4.6.

amp_enabled_flag equal to 1 specifies that asymmetric motion partitions, i.e. PartMode equal to PART_2NxnU, PART_2NxnD, PART_nLx2N, or PART_nRx2N, may be used in coding tree blocks; amp_enabled_flag equal to 0 specifies that asymmetric motion partitions cannot be used in coding tree blocks.

sample_adaptive_offset_enabled_flag equal to 1 specifies that the sample adaptive offset process is applied to the reconstruced picture after the deblocking filter process. sample_adaptive_offset_enabled_flag equal to 0 specifies that the sample adaptive offset process is not applied to the reconstruced picture after the deblocking filter process.

pcm_enabled_flag equal to 0 specifies that PCM data shall not be present in the coded video sequence.

NOTE 4 – When Log2MinCbSizeY is equal to 6, PCM data is not present in the coded video sequence even if pcm_enabled_flag is equal to 1. The maximum size of coding block with pcm_enabled_flag equal to 1 is restricted to be less than or equal to Min(Log2CtbSizeY, 5). Encoders are encouraged to use an appropriate combination of log2_min_luma_coding_block_size_minus3, log2_min_pcm_luma_coding_block_size_minus3, and log2_diff_max_min_pcm_luma_coding_block_size values when sending PCM data in the coded video sequence.

pcm_sample_bit_depth_luma_minus1 + 1 specifies the number of bits used to represent each of PCM sample values of luma component. The value of pcm_sample_bit_depth_luma_minus1 + 1 shall be less than or equal to the value of $BitDepth_Y$.

$$PCMBitDepth_Y = 1 + pcm\_sample\_bit\_depth\_luma\_minus1 \quad (7\text{-}24)$$

pcm_sample_bit_depth_chroma_minus1 + 1 specifies the number of bits used to represent each of PCM sample values of chroma components. The value of pcm_sample_bit_depth_chroma_minus1 + 1 shall be less than or equal to the value of $BitDepth_C$.

$$PCMBitDepth_C = 1 + pcm\_sample\_bit\_depth\_chroma\_minus1 \quad (7\text{-}25)$$

log2_min_pcm_luma_coding_block_size_minus3 + 3 specifies the minimum size of coding blocks with pcm_flag equal to 1.

The variable Log2MinIpcmCbSizeY is set equal to log2_min_pcm_luma_coding_block_size_minus3 + 3. The variable Log2MinIpcmCbSizeY shall be in the range of Log2MinCbSizeY to Min(Log2CtbSizeY, 5), inclusive.

log2_diff_max_min_pcm_luma_coding_block_size specifies the difference between the maximum and minimum size of coding blocks with pcm_flag equal to 1.

The variable Log2MaxIpcmCbSizeY is set equal to log2_min_pcm_luma_coding_block_size_minus3 + 3 + log2_diff_max_min_pcm_luma_coding_block_size. The variable Log2MaxIpcmCbSizeY shall be equal or less than Min(Log2CtbSizeY, 5).

pcm_loop_filter_disable_flag specifies whether the loop filter process is disabled on reconstructed samples in a coding unit with pcm_flag equal to 1. If the pcm_loop_filter_disable_flag value is equal to 1, deblocking filter and sample adaptive offset filter processes on the reconstructed samples in a coding unit with pcm_flag equal to 1 are disabled; otherwise if the pcm_loop_filter_disable_flag value is equal to 0, deblocking filter and sample adaptive offset filter processes on the reconstructed samples in a coding unit with pcm_flag equal to 1 are not disabled. When pcm_loop_filter_disable_flag is not present, it is inferred to be equal to 0.

[Ed. (WJ): select one expression – enabled_flag or disable_flag]

num_short_term_ref_pic_sets specifies the number of short-term reference picture sets that are specified in the sequence parameter set. The value of num_short_term_ref_pic_sets shall be in the range of 0 to 64, inclusive.

NOTE 5 – A decoder must allocate space for a total number of num_short_term_ref_pic_sets + 1 short-term reference picture sets since a coded video sequence may contain a short-term reference picture set explicitly signalled in the slice headers of a current picture. A short-term reference picture set directly signalled in the slice header will always have an index equal to num_short_term_ref_pic_sets in the list of short-term reference picture sets.

long_term_ref_pics_present_flag equal to 0 specifies that no long-term reference picture is used for inter prediction of any coded picture in the coded video sequence. long_term_ref_pics_present_flag equal to 1 specifies that long-term reference pictures may be used for inter prediction of one or more coded pictures in the coded video sequence.

num_long_term_ref_pics_sps specifies the number of candidate long-term reference pictures that are specified in the sequence parameter set. The value of num_long_term_ref_pics_sps shall be in the range of 0 to 32, inclusive.

lt_ref_pic_poc_lsb_sps[i] specifies the picture order count modulo MaxPicOrderCntLsb of the i-th candidate long-term reference picture specified in the sequence parameter set. The number of bits used to represent lt_ref_pic_poc_lsb_sps[i] is equal to log2_max_pic_order_cnt_lsb_minus4 + 4.

used_by_curr_pic_lt_sps_flag[i] equal to 0 specifies that the i-th candidate long-term reference picture specified in the sequence parameter set is not used for reference by a picture that includes in its reference picture set the i-th candidate long-term reference picture specified in the sequence parameter set.

sps_temporal_mvp_enable_flag equal to 1 specifies that slice_temporal_mvp_enable_flag is present in the slice headers of pictures with IdrPicFlag equal to 0 in the coded video sequence. sps_temporal_mvp_enable_flag equal to 0 specifies that slice_temporal_mvp_enable_flag is not present in slice headers and that temporal motion vector predictors is not used in the coded video sequence.

strong_intra_smoothing_enable_flag equal to 1 specifies that bi-linear interpolation is conditionally used in the filtering process in the coded video sequence as specified in subclause 8.4.4.2.3. strong_intra_smoothing_enable_flag equal to 0 specifies that that the bi-linear interpolation is not used in the coded video sequence.

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

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

sps_extension_data_flag may have any value. Its value does not affect decoder conformance to profiles specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore (remove from the bitstream and discard) all sps_extension_data_flag syntax elements.

7.4.2.3 Picture parameter set RBSP semantics

pps_pic_parameter_set_id identifies the picture parameter set for reference by other syntax elements. The value of pps_pic_parameter_set_id shall be in the range of 0 to 63, inclusive.

pps_seq_parameter_set_id specifies the value of sps_seq_parameter_set_id for the active sequence parameter set. The value of pps_seq_parameter_set_id shall be in the range of 0 to 15, inclusive.

dependent_slice_segments_enabled_flag equal to 1 specifies the presence of the syntax element dependent_slice_segment_flag in the slice segment headers for coded pictures referring to the picture parameter set. dependent_slice_segments_enabled_flag equal to 0 specifies the absence of the syntax element dependent_slice_segment_flag in the slice segment headers for coded pictures referring to the picture parameter set.

sign_data_hiding_flag equal to 0 specifies that sign bit hiding is disabled. sign_data_hiding_flag equal to 1 specifies that sign bit hiding is enabled.

cabac_init_present_flag equal to 1 specifies that cabac_init_flag is present in slice headers referring to the picture parameter set. cabac_init_present_flag equal to 0 specifies that cabac_init_flag is not present in slice headers referring to the picture parameter set.

num_ref_idx_l0_default_active_minus1 specifies the inferred value of num_ref_idx_l0_active_minus1 for P and B slices with num_ref_idx_active_override_flag equal to 0. The value of num_ref_idx_l0_default_active_minus1 shall be in the range of 0 to 15, inclusive.

num_ref_idx_l1_default_active_minus1 specifies the inferred value of num_ref_idx_l1_active_minus1 for B slices with num_ref_idx_active_override_flag equal to 0. The value of num_ref_idx_l1_default_active_minus1 shall be in the range of 0 to 15, inclusive.

init_qp_minus26 specifies the initial value minus 26 of $SliceQP_Y$ for each slice. The initial value is modified at the slice segment layer when a non-zero value of slice_qp_delta is decoded, and is modified further when a non-zero value of cu_qp_delta_abs is decoded at the coding unit layer. The value of init_qp_minus26 shall be in the range of $-(26 + QpBdOffset_Y)$ to +25, inclusive.

constrained_intra_pred_flag equal to 0 specifies that intra prediction allows usage of residual data and decoded samples of neighbouring coding blocks coded using either intra or inter prediction modes. constrained_intra_pred_flag equal to 1 specifies constrained intra prediction, in which case intra prediction only uses residual data and decoded samples from neighboring coding blocks coded using intra prediction modes.

transform_skip_enabled_flag equal to 1 specifies that transform_skip_flag may be present in the residual coding syntax. transform_skip_enabled_flag equal to 0 specifies that transform_skip_flag is not present in the residual coding syntax.

cu_qp_delta_enabled_flag equal to 1 specifies that the diff_cu_qp_delta_depth syntax element is present in the picture parameter set and that cu_qp_delta_abs can be present in transform unit syntax. cu_qp_delta_enabled_flag equal to 0 specifies that the diff_cu_qp_delta_depth syntax element is not present in the picture parameter set and that cu_qp_delta_abs is not present in transform unit syntax.

diff_cu_qp_delta_depth specifies the granularity for QP_Y values within a picture. The value of diff_cu_qp_delta_depth shall be in the range of 0 to log2_diff_max_min_luma_coding_block_size, inclusive.

The variable Log2MinCuQpDeltaSize, specifying the minimum luma coding block size of coding units that convey cu_qp_delta_abs and cu_qp_delta_sign, is derived as follows.

$$Log2MinCuQpDeltaSize = Log2CtbSizeY - diff\_cu\_qp\_delta\_depth \quad (7\text{-}26)$$

pps_cb_qp_offset and **pps_cr_qp_offset** specify 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.

pps_slice_chroma_qp_offsets_present_flag equal to 1 indicates that the slice_cb_qp_offset and slice_cr_qp_offset syntax elements are present in the associated slice headers. pps_slice_chroma_qp_offsets_present_flag equal to 0 indicates that these syntax elements are not present in the associated slice headers.

weighted_pred_flag equal to 0 specifies that weighted prediction is not applied to P slices. weighted_pred_flag equal to 1 specifies that weighted prediction is applied to P slices.

weighted_bipred_flag equal to 0 specifies that the default weighted prediction is applied to B slices. weighted_bipred_flag equal to 1 specifies that weighted prediction is applied to B slices.

output_flag_present_flag equal to 1 indicates that the pic_output_flag syntax element is present in the associated slice headers. output_flag_present_flag equal to 0 indicates that the pic_output_flag syntax element is not present in the associated slice headers.

transquant_bypass_enable_flag equal to 1 specifies that cu_transquant_bypass_flag is present. transquant_bypass_enable_flag equal to 0 specifies that cu_transquant_bypass_flag is not present.

tiles_enabled_flag equal to 1 specifies that there is more than one tile in each picture referring to the picture parameter set. tiles_enabled_flag equal to 0 specifies that there is only one tile in each picture referring to the picture parameter set.

entropy_coding_sync_enabled_flag equal to 1 specifies that a specific synchronization process for context variables is invoked before decoding the first coding tree block of a row of coding tree blocks in each tile in each picture referring to the picture parameter set, and a specific memorization process for context variables is invoked after decoding two coding tree blocks of a row of coding tree blocks in each tile in each picture referring to the picture parameter set. 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 first coding tree block of a row of coding tree blocks in each tile in each picture referring to the picture parameter set, and no specific memorization process for context variables is required to be invoked after decoding two coding tree blocks of a row of coding tree blocks in each tile in each picture referring to the picture parameter set. [Ed. (GJS): Consider using a different word than "memorization".]

It's a requirement of bitstream conformance that the values of tiles_enabled_flag and entropy_coding_sync_enabled_flag shall be the same, respectively, for all picture parameter sets that are activated within a coded video sequence.

When entropy_coding_sync_enabled_flag is equal to 1 and the first coding tree block in a slice is not the first coding tree block of a row of coding tree blocks in a tile, it is a requirement of bitstream conformance that the last coding tree block in the slice shall belong to the same row of coding tree blocks as the first coding tree block in the slice.

When entropy_coding_sync_enabled_flag is equal to 1 and the first coding tree block in a slice segment is not the first coding tree block of a row of coding tree blocks in a tile, it is a requirement of bitstream conformance that the last coding tree block in the slice segment shall belong to the same row of coding tree blocks as the first coding tree block in the slice segment.

num_tile_columns_minus1 plus 1 specifies the number of tile columns partitioning the picture. num_tile_columns_minus1 shall be in the range of 0 to PicWidthInCtbsY − 1, inclusive. When not present, the value of num_tile_columns_minus1 is inferred to be equal to 0.

num_tile_rows_minus1 plus 1 specifies the number of tile rows partitioning the picture. num_tile_rows_minus1 shall be in the range of 0 to PicHeightInCtbsY − 1, inclusive. When not present, the value of num_tile_rows_minus1 is inferred to be equal to 0.

When tiles_enable_flag is equal to 1 and num_tile_columns_minus1 is equal to 0, num_tile_rows_minus1 shall not be equal to 0.

uniform_spacing_flag equal to 1 specifies that column boundaries and likewise row boundaries are distributed uniformly across the picture. uniform_spacing_flag equal to 0 specifies that column boundaries and likewise row boundaries are not distributed uniformly across the picture but signalled explicitly using the syntax elements column_width_minus1[i] and row_height_minus1[i].

column_width_minus1[i] plus 1 specifies the width of the i-th tile column in units of coding tree blocks.

row_height_minus1[i] plus 1 specifies the height of the i-th tile row in units of coding tree blocks.

The following variables are derived by invoking the coding tree block raster and tile scanning conversion process as specified in subclause 6.5.1.

– The list CtbAddrRStoTS[ctbAddrRS] for ctbAddrRS ranging from 0 to PicSizeInCtbsY − 1, inclusive, specifying the conversion from a CTB address in CTB raster scan of a picture to a CTB address in tile scan,

– the list CtbAddrTStoRS[ctbAddrTS] for ctbAddrTS ranging from 0 to PicSizeInCtbsY − 1, inclusive, specifying the conversion from a CTB address in tile scan to a CTB address in CTB raster scan of a picture,

– the list TileId[ctbAddrTS] for ctbAddrTS ranging from 0 to PicSizeInCtbsY − 1, inclusive, specifying the conversion from a CTB address in tile scan to a tile ID,

– the list ColumnWidthInLumaSamples[i] for i ranging from 0 to num_tile_columns_minus1, inclusive, specifying the width of the i-th tile column in units of luma samples,

– the list RowHeightInLumaSamples[j] for j ranging from 0 to num_tile_rows_minus1, inclusive, specifying the height of the j-th tile row in units of luma samples.

The values of ColumnWidthInLumaSamples[i] for i ranging from 0 to num_tile_columns_minus1, inclusive, and RowHeightInLumaSamples[j] for j ranging from 0 to num_tile_rows_minus1, inclusive, shall all be greater than 0.

The array MinTbAddrZS with elements MinTbAddrZS[x][y] for x ranging from 0 to (PicWidthInCtbsY << (Log2CtbSizeY − Log2MinTrafoSize)) − 1, inclusive, and y ranging from 0 to (PicHeightInCtbsY << (Log2CtbSizeY − Log2MinTrafoSize)) − 1, inclusive, specifying the conversion from a location (x, y) in units of minimum transform blocks to a transform block address in z-scan order, is derived by invoking the z-scan order array initialization process as specified in subclause 6.5.2.

loop_filter_across_tiles_enabled_flag equal to 1 specifies that in-loop filtering operations are performed across tile boundaries. loop_filter_across_tiles_enabled_flag equal to 0 specifies that in-loop filtering operations are not performed across tile boundaries. The in-loop filtering operations include the deblocking filter and sample adaptive offset filter operations. When not present, the value of loop_filter_across_tiles_enabled_flag is inferred to be equal to 1.

loop_filter_across_slices_enabled_flag equal to 1 specifies that the slice_loop_filter_across_slices_enabled_flag determines whether in-loop filtering operations are performed across left and upper boundaries of the current slice; otherwise, the in-loop filtering operations are not applied across left and upper boundaries of the current slice. The in-loop filtering operations include the deblocking filter and sample adaptive offset filter.

deblocking_filter_control_present_flag equal to 1 specifies the presence of deblocking filter control syntax elements in the picture parameter set and in the slice headers for pictures referring to the picture parameter set. deblocking_filter_control_present_flag equal to 0 specifies the absence of deblocking filter control syntax elements in the picture parameter set and in the slice headers for pictures referring to the picture parameter set.

deblocking_filter_override_enabled_flag equal to 1 specifies the presence of deblocking_filter_overriding_flag in the slice headers for pictures referring to the picture parameter set. deblocking_filter_override_enabled_flag equal to 0 specifies the absence of deblocking_filter_overriding_flag in the slice headers for pictures referring to the picture parameter set. When not present, the value of deblocking_filter_override_enabled_flag is inferred to be equal to 0.

pps_disable_deblocking_filter_flag equal to 1 specifies that the operation of deblocking filter shall not be applied for pictures referring to the picture parameter set when deblocking_filter_override_enabled_flag is equal to 0. pps_disable_deblocking_filter_flag equal to 0 specifies that the operation of the deblocking filter shall be applied for pictures referring to the picture parameter set when deblocking_filter_override_enabled_flag is equal to 0. When not present, the value of pps_disable_deblocking_filter_flag is inferred to be equal to 0.

pps_beta_offset_div2 and **pps_tc_offset_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied for pictures referring to the picture parameter set unless the default deblocking parameter offsets are overriden by the deblocking parameter offsets present in the slice headers for pictures referring to the picture parameter set. The values of pps_beta_offset_div2 and pps_tc_offset_div2 shall both be in the range of −6 to 6, inclusive. When not present, the value of pps_beta_offset_div2 and pps_tc_offset_div2 are inferred to be equal to 0.

pps_scaling_list_data_present_flag equal to 1 specifies that parameters are present in the picture parameter set to modify the scaling lists specified in the active sequence parameter set. pps_scaling_list_data_present_flag equal to 0 specifies that the scaling lists used for the pictures referring to the picture parameter set is inferred to be equal to those specified by the active sequence parameter set. When scaling_list_enable_flag is equal to 0, the value of pps_scaling_list_data_present_flag shall be equal to 0. When scaling_list_enable_flag is equal to 1 and pps_scaling_list_data_present_flag is equal to 0, the default scaling list data is used to derive the array ScalingFactor as described in the scaling list data semantics 7.4.6.

lists_modification_present_flag equal to 1 specifies that the syntax structure ref_pic_lists_modification( ) is present in the slice segment header. lists_modification_present_flag equal to 0 specifies that the syntax structure ref_pic_lists_modification( ) is not present in the slice segment header.

log2_parallel_merge_level_minus2 specifies the parallel processing level of merge/skip mode. The value of log2_parallel_merge_level_minus2 shall be in the range of 0 to log2_min_luma_coding_block_size_minus3 + 1 + log2_diff_max_min_luma_coding_block_size, inclusive.

num_extra_slice_header_bits equal to 0 specifies that no extra slice header bits are present in the slice header RBSP for coded pictures referring to the picture parameter set. num_extra_slice_header_bits shall be equal to 0 in bitstreams

conforming to this version of this Specification. Other values for num_extra_slice_header_bits are reserved for future use by ITU-T | ISO/IEC. However, decoders shall allow num_extra_slice_header_bits to have any value.

slice_segment_header_extension_present_flag equal to 0 specifies that no slice segment header extension syntax elements are present in the slice segment headers for coded pictures referring to the picture parameter set. slice_segment_header_extension_present_flag shall be equal to 0 in bitstreams conforming to this version of this Specification. The value of 1 for slice_segment_header_extension_present_flag is reserved for future use by ITU-T | ISO/IEC.

pps_extension_flag equal to 0 specifies that no pps_extension_data_flag syntax elements are present in the picture parameter set RBSP syntax structure. pps_extension_flag shall be equal to 0 in bitstreams conforming to this version of this Specification. The value of 1 for pps_extension_flag is reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore all data that follow the value 1 for pps_extension_flag in a picture parameter set NAL unit.

pps_extension_data_flag may have any value. Its value does not affect decoder conformance to profiles specified in this version of this Specification.

7.4.2.4 Supplemental enhancement information RBSP semantics

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

7.4.2.5 Access unit delimiter RBSP semantics

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

pic_type indicates that the slice_type values for all slices of the coded picture are members of the set listed in Table 7-2 for the given value of pic_type.

Table 7-2 – Interpretation of pic_type

pic_type	slice_type values that may be present in the coded picture
0	I
1	P, I
2	B, P, I

7.4.2.6 End of sequence RBSP semantics

When the next subsequent NAL unit in decoding order is not an end of bitstream NAL unit, the end of sequence RBSP specifies that the next subsequent access unit in the bitstream in decoding order (if any) is an IDR or BLA access unit. The syntax content of the SODB and RBSP for the end of sequence RBSP are empty.

7.4.2.7 End of bitstream RBSP semantics

The end of bitstream RBSP indicates that no additional NAL units are present in the bitstream that are subsequent to the end of bitstream RBSP in decoding order. The syntax content of the SODB and RBSP for the end of bitstream RBSP are empty.

NOTE – When an elementary stream contains more than one bitstream, the last NAL unit of the last access unit of a bitstream must contain an end of bitstream NAL unit and the first access unit of the subsequent bitstream must be a RAP access unit. This RAP access unit may be a CRA, BLA, or IDR access unit.

7.4.2.8 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.

ff_byte is a byte equal to 0xFF.

7.4.2.9 Slice segment layer RBSP semantics

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

7.4.2.10 RBSP slice segment trailing bits semantics

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

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

Let BinCountsInNALunits be the number of times that the parsing process function DecodeBin( ), specified in subclause 9.2.3.2, is invoked to decode the contents of all VCL NAL units of a coded picture.

Let the variable RawMinCuBits be derived as

$$RawMinCuBits = MinCbSizeY * MinCbSizeY * (BitDepthY + BitDepthC / 2) \quad (7\text{-}27)$$

The value of BinCountsInNALunits shall be less than or equal to (32 ÷ 3) * NumBytesInVclNALunits + (RawMinCuBits * PicSizeInMinCbsY) ÷ 32.

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

7.4.2.11 RBSP trailing bits semantics

rbsp_stop_one_bit shall be equal to 1.

rbsp_alignment_zero_bit shall be equal to 0.

7.4.2.12 Byte alignment semantics

alignment_bit_equal_to_one shall be equal to 1.

alignment_bit_equal_to_zero shall be equal to 0.

7.4.3 Profile, tier and level semantics

general_profile_space specifies the context for the interpretation of general_profile_idc and general_profile_combatibility_flag[i] for all values of i in the range of 0 to 31, inclusive. The value of general_profile_space shall be equal to 0 in bitstreams conforming to this version of this Specification. Other values for general_profile_space are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore the coded video sequence if general_profile_space is not equal to 0. [Ed. (GJS): Ignore the CVS or ignore the VPS/SPS? Think about this.]

general_tier_flag specifies the tier context for the interpretation of general_level_idc as specified in Annex A.

general_profile_idc, when general_profile_space is equal to 0, indicates a profile to which the coded video sequence 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.

[Ed. (DS): We might prefer not to use the general_profile_idc value zero, or reserve it to mean "no profile signalled, bitstream is unconstrained"; this gives us one spare bit in the profile compatibility flags array.]

general_profile_compatibility_flag[i] equal to 1, when general_profile_space is equal to 0, indicates that the coded video sequence conforms to the profile indicated by general_profile_idc equal to i as specified in Annex A. When general_profile_space is equal to 0, general_profile_compatibility_flag[general_profile_idc] shall be equal to 1. The value of general_profile_compatibility_flag[i] shall be equal to 0 for any value of i that is not specified as an allowed value of general_profile_idc in Annex A.

general_reserved_zero_16bits shall be equal to 0 in bitstreams conforming to this version of this Specification. Other values for general_reserved_zero_16bits are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore the value of general_reserved_zero_16bits.

general_level_idc indicates a level to which the coded video sequence conforms as specified in Annex A.

NOTE 1 – A greater value of general_level_idc indicates a higher level. The maximum level signalled in the video parameter set for a coded video sequence may be higher than the level signalled in the sequence parameter set for the same coded video sequence.

NOTE 2 – For bitstreams compatible with more than one profile, the value of general_profile_idc should be set to the "best viewed as" profile.

NOTE 3 – The general_reserved_zero_16bits can be used to indicate the respect of further constraints on the bitstream (e.g. that a selected tool, permitted by the profiles signalled, is nonetheless not used). These flags are ideally profile independent, but unlike general_level_idc, it is allowed to be contextual on general_profile_idc.

sub_layer_profile_present_flag[i] equal to 1, when profilePresentFlag is equal to 1, specifies that profile information is present in the profile_tier_level( ) syntax structure for the representation of the sub-layer with TemporalId equal to i. sub_layer_profile_present_flag[i] equal to 0 specifies that profile information is not present in the profile_tier_level( )

syntax structure for the representations of the sub-layer with TemporalId equal to i. When not present, the value of sub_layer_profile_present_flag[i] is inferred to be equal to 0.

sub_layer_level_present_flag[i] equal to 1 specifies that level information is present in the profile_tier_level( ) syntax structure for the representation of the sub-layer with TemporalId equal to i. sub_layer_level_present_flag[i] equal to 0 specifies that level information is not present in the profile_tier_level( ) syntax structure for the representation of the sub-layer with TemporalId equal to i.

sub_layer_profile_space[i], **sub_layer_tier_flag**[i], **sub_layer_profile_idc**[i], **sub_layer_profile_compatibility_flag**[i][j], **sub_layer_reserved_zero_16bits**[i], and **sub_layer_level_idc**[i] have the same semantics as general_profile_space, general_tier_flag, general_profile_idc, general_profile_compatibility_flag[j], general_reserved_zero_16bits, and general_level_idc, respectively, but apply to the representation of the sub-layer with TemporalId equal to i.

When not present, the value of sub_layer_tier_flag[i] is inferred to be equal to 0.

NOTE 4 – It is possible that sub_layer_tier_flag[i] is not present and sub_layer_level_idc[i] is present. In this case, a default value of sub_layer_tier_flag[i] is needed for interpretation of sub_layer_level_idc[i].

7.4.4 Bit rate and picture rate information semantics

bit_rate_info_present_flag[i] equal to 1 specifies that the bit rate information for the i-th sub-layer is present. bit_rate_info_present_flag[i] equal to 0 specifies that the bit rate information for the i-th sub-layer is not present.

pic_rate_info_present_flag[i] equal to 1 specifies that picture rate information for the i-th sub-layer is present. pic_rate_info_present_flag[i] equal to 0 specifies that picture rate information for the i-th sub-layer is not present.

avg_bit_rate[i] indicates the average bit rate of the representation of the i-th sub-layer. The average bit rate for the representation of the i-th sub-layer in bits per second is given by BitRateBPS(avg_bit_rate[i]) with the function BitRateBPS( ) being specified by

$$BitRateBPS(x) = (x \& (2^{14} - 1)) * 10^{(2 + (x >> 14))} \quad (7\text{-}28)$$

The average bit rate is derived according to the access unit removal time specified in Annex C of this Specification. In the following, bTotal is the number of bits in all NAL units of the representation of the i-th sub-layer, t_1 is the removal time (in seconds) of the first access unit to which the video parameter set applies, and t_2 is the removal time (in seconds) of the last access unit (in decoding order) to which the video parameter set applies.

With x specifying the value of avg_bit_rate[i], the following applies:

– If t_1 is not equal to t_2, the following condition shall be true:

$$(x \& (2^{14} - 1)) == Round(bTotal \div ((t_2 - t_1) * 10^{(2 + (x >> 14))})) \quad (7\text{-}29)$$

– Otherwise (t_1 is equal to t_2), the following condition shall be true:

$$(x \& (2^{14} - 1)) == 0 \quad (7\text{-}30)$$

max_bit_rate_layer[i] indicates an upper bound for the bit rate of the representation of the i-th sub-layer in any one-second time window of access unit removal time as specified in Annex C. The upper bound for the bit rate in bits per second is given by BitRateBPS(max_bit_rate_layer[i]). The bit rate values are derived according to the access unit removal time specified in Annex C of this Specification. In the following, t_1 is any point in time (in seconds), t_2 is set equal to $t_1 + 1 \div 100$, and bTotal is the number of bits in all NAL units of access units with a removal time greater than or equal to t_1 and less than t_2. With x specifying the value of max_bit_rate_layer[i], the following condition shall be obeyed for all values of t_1:

$$(x \& (2^{14} - 1)) >= bTotal \div ((t_2 - t_1) * 10^{(2 + (x >> 14))}) \quad (7\text{-}31)$$

constant_pic_rate_idc[i] indicates whether the picture rate of the representation of the i-th sub-layer is constant. In the following, a temporal segment tSeg is any set of two or more consecutive access units, in decoding order, of the representation of the i-th sub-layer, fTotal(tSeg) is the number of pictures in the temporal segment tSeg, t_1(tSeg) is the removal time (in seconds) of the first access unit (in decoding order) of the temporal segment tSeg, t_2(tSeg) is the removal time (in seconds) of the last access unit (in decoding order) of the temporal segment tSeg, and avgFR(tSeg) is the average frame rate in the temporal segment tSeg, which is given by:

$$avgFR(tSeg) == Round(fTotal(tSeg) * 256 \div (t_2(tSeg) - t_1(tSeg))) \quad (7\text{-}32)$$

If the representation of the i-th sub-layer only contains one access unit or the value of avgFR(tSeg) is constant over all temporal segments of the representation of the i-th sub-layer, the picture rate is constant; otherwise, the picture rate is not constant.

constant_pic_rate_idc[i] equal to 0 indicates that the picture rate of the representation of the i-th sub-layer is not constant. constant_pic_rate_idc[i] equal to 1 indicates that the picture rate of the representation of the i-th sub-layer is constant. constant_pic_rate_idc[i] equal to 2 indicates that the picture rate of the representation of the i-th sub-layer may or may not be constant. [Ed. (GJS): If equal to 1, is the bitstream non-conforming if the picture rate happens to turn out to be constant?] [Ed. (GJS): If there are only two pictures, is the picture rate constant?], The value of constant_pic_rate_idc[i] shall be in the range of 0 to 2, inclusive.

avg_pic_rate[i] indicates the average picture rate, in units of picture per 256 seconds, of representation of the i-th sub-layer. With fTotal being the number of pictures in the representation of the i-th sub-layer, t_1 being the removal time (in seconds) of the first access unit to which the video parameter set applies, and t_2 being the removal time (in seconds) of the last access unit (in decoding order) to which the video parameter set applies, the following applies:

– If t_1 is not equal to t_2, the following condition shall be true:

$$avg\_pic\_rate[i] == Round(fTotal * 256 \div (t_2 - t_1)) \quad (7\text{-}33)$$

– Otherwise (t_1 is equal to t_2), the following condition shall be true:

$$avg\_pic\_rate[i] == 0 \quad (7\text{-}34)$$

7.4.5 Operation point layer set semantics

The operation_point_set(opsIdx) syntax structure specifies the OpLayerIdSet of the opsIdx-th operation point set specified by the video parameter set.

layer_id_included_flag[opsIdx][i] equal to 0 specifies that the value of nuh_reserved_zero_6bits equal to i is not included in the OpLayerIdSet of the opsIdx-th operation point set specified by the video parameter set. layer_id_included_flag[opsIdx][i] equal to 1 specifies that the value of nuh_reserved_zero_6bits equal to i is included in the OpLayerIdSet of the opsIdx-th operation point set specified by the video parameter set. The sum of all layer_id_included_flag[opsIdx][i] values for i from 0 to vps_max_nuh_reserved_zero_layer_id, inclusive, shall be in the range of 1 to vps_reserved_zero_6bits + 1, inclusive.

The variable numLayerIdsMinus1[opsIdx] and the variables layerId[opsIdx][j], for j in the range of 0 to numLayerIdsMinus1[opsIdx], inclusive, are derived as follows.

```
j = 0
for( i = 0; i <= vps_max_nuh_reserved_zero_layer_id; i++ )
    if( layer_id_included_flag[ opsIdx ][ i ] )                    (7-35)
        layerId[ opsIdx ][ j++ ] = i
numLayerIdsMinus1[ opsIdx ] = j − 1
```

The values of numLayerIdsMinus1[0] and layerId[0][0] are defined as follows:

$$numLayerIdsMinus1[0] = 0 \quad (7\text{-}36)$$

$$layerId[0][0] = 0 \quad (7\text{-}37)$$

Any two sets layerId[opsIdx1] and layerId[opsIdx2], where opsIdx1 is not equal to opsIdx2, shall not include the same set of nuh_reserved_zero_6bits values.

The layerIdSet of the opsIdx-th operation point set specified in the video parameter set is set to the set of nuh_reserved_zero_6bits values equal to layerId[opsIdx][i], for i in the range of 0 to numLayerIdsMinus1[opsIdx], inclusive.

7.4.6 Scaling list data semantics

scaling_list_pred_mode_flag[sizeId][matrixId] equal to 0 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_matrix_id_delta[sizeId][matrixId]. scaling_list_pred_mode_flag[sizeId][matrixId] equal to 1 specifies that the values of the scaling list are explicitly signalled.

scaling_list_pred_matrix_id_delta[sizeId][matrixId] specifies the reference scaling list used to derive ScalingList[sizeId][matrixId] as follows.

– If scaling_list_pred_matrix_id_delta is equal to 0, the scaling list is inferred from the default scaling list ScalingList[sizeId][matrixId][i] as specified in Table 7-5 and Table 7-6 for i = 0..Min(64, (1 << (4 + (sizeId << 1)))).

– Otherwise, the scaling list is inferred from the reference scaling list as follows.

refMatrixId = matrixId − scaling_list_pred_matrix_id_delta[sizeId][matrixId] (7-38)

ScalingList[sizeId][matrixId][i] = ScalingList[sizeId][refMatrixId][i]
with i =0 .. Min(64, (1 << (4 + (sizeId << 1)))) (7-39)

The value of scaling_list_pred_matrix_id_delta[sizeId][matrixId] shall be in the range of 0 to matrixId, inclusive.

Table 7-3 – Specification of sizeId

Size of quantization matrix	sizeId
4x4	0
8x8	1
16x16	2
32x32	3

Table 7-4 – Specification of matrixId according to sizeId, prediction mode and colour component

sizeId	CuPredMode	cIdx (colour component)	matrixId
0, 1, 2	MODE_INTRA	0 (Y)	0
0, 1, 2	MODE_INTRA	1 (Cb)	1
0, 1, 2	MODE_INTRA	2 (Cr)	2
0, 1, 2	MODE_INTER	0 (Y)	3
0, 1, 2	MODE_INTER	1 (Cb)	4
0, 1, 2	MODE_INTER	2 (Cr)	5
3	MODE_INTRA	0 (Y)	0
3	MODE_INTER	0 (Y)	1

scaling_list_dc_coef_minus8[sizeId − 2][matrixId] plus 8 specifies the DC value of the scaling list for 16x16 size when sizeId is equal to 2 and specifies the DC value of the scaling list for 32x32 size when sizeId is equal to 3. The value of scaling_list_dc_coef_minus8[sizeId − 2][matrixId] shall be in the range of −7 to 247, inclusive. When scaling_list_dc_coef_minus8 is not present, it is inferred to be equal to 8.

scaling_list_delta_coef specifies the difference between the current matrix coefficient ScalingList[sizeId][matrixId][i] and the previous matrix coefficient ScalingList[sizeId][matrixId][i − 1], when scaling_list_pred_mode_flag[sizeId][matrixId] is equal to 1. The value of scaling_list_delta_coef shall be in the range of −128 to 127, inclusive. The value of ScalingList[sizeId][matrixId][i] shall be greater than 0.

Table 7-5 – Specification of default values of ScalingList[0][matrixId][i] with i = 0..15

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ScalingList[0][0..2][i]	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
ScalingList[0][3..5][i]	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16

Table 7-6 – Specification of default values of ScalingList[1..3][matrixId][i] with i = 0..63

i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
ScalingList[1..2][0..2][i] ScalingList[3][0][i]	16	16	16	16	16	16	16	16	16	16	17	16	17	16	17	18
ScalingList[1..2][3..5][i] ScalingList[3][1][i]	16	16	16	16	16	16	16	16	16	16	17	17	17	17	17	18
i − 16	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
ScalingList[1..2][0..2][i] ScalingList[3][0][i]	17	18	18	17	18	21	19	20	21	20	19	21	24	22	22	24
ScalingList[1..2][3..5][i] ScalingList[3][1][i]	18	18	18	18	18	20	20	20	20	20	20	20	24	24	24	24
i − 32	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
ScalingList[1..2][0..2][i] ScalingList[3][0][i]	24	22	22	24	25	25	27	30	27	25	25	29	31	35	35	31
ScalingList[1..2][3..5][i] ScalingList[3][1][i]	24	24	24	24	25	25	25	25	25	25	25	28	28	28	28	28
i − 48	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
ScalingList[1..2][0..2][i] ScalingList[3][0][i]	29	36	41	44	41	36	47	54	54	47	65	70	65	88	88	115
ScalingList[1..2][3..5][i] ScalingList[3][1][i]	28	33	33	33	33	33	41	41	41	41	54	54	54	71	71	91

The four-dimensional array ScalingFactor[sizeId][matrixId][x][y], with x, y = 0..(1 << (2 + sizeId)) − 1, specifies the array of scaling factors according to the variables sizeId specified in Table 7-3 and matrixId specified in Table 7-4.

The elements of the quantization matrix of size 4x4, ScalingFactor[0][matrixId][ ][ ], are derived as follows:

ScalingFactor[0][matrixId][x][y] = ScalingList[0][matrixId][i] (7-40)
with i = 0..15 and matrixId = 0..5

where x=ScanOrder[2][0][i][0] and y=ScanOrder[2][0][i][1]

The elements of the quantization matrix of size 8x8, ScalingFactor[1][matrixId][ ][ ], are derived as follows:

ScalingFactor[1][matrixId][x][y] = ScalingList[1][matrixId][i] (7-41)
with i = 0..63 and matrixId = 0..5

where x=ScanOrder[3][0][i][0] and y=ScanOrder[3][0][i][1]

The elements of the quantization matrix of size 16x16, ScalingFactor[2][matrixId][ ][ ], are derived as follows:

ScalingFactor[2][matrixId][x *2 + k][y * 2 + j] = ScalingList[2][matrixId][i] (7-42)
with i = 0..63, j = 0..1, k = 0..1 and matrixId = 0..5

where x=ScanOrder[3][0][i][0] and y=ScanOrder[3][0][i][1]

ScalingFactor[2][matrixId][0][0] = scaling_list_dc_coef_minus8[0][matrixId] + 8 (7-43)
with matrixId = 0..5

The elements of the quantization matrix of size 32x32, ScalingFactor[3][matrixId][ ][ ], are derived as follows:

ScalingFactor[3][matrixId][x * 4 + k][y * 4 + j] = ScalingList[3][matrixId][i] (7-44)
with i = 0..63, j = 0..3, k = 0..3 and matrixId = 0..1

where x=ScanOrder[3][0][i][0] and y=ScanOrder[3][0][i][1]

ScalingFactor[3][matrixId][0][0] = scaling_list_dc_coef_minus8[1][matrixId] + 8 (7-45)
with matrixId = 0..1

7.4.7 Supplemental enhancement information message semantics

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

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

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

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

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

7.4.8 Slice segment header semantics

7.4.8.1 General slice segment header semantics

When present, the value of the slice segment header syntax elements slice_pic_parameter_set_id, pic_output_flag, no_output_of_prior_pics_flag, pic_order_cnt_lsb, short_term_ref_pic_set_sps_flag, short_term_ref_pic_set_idx, num_long_term_sps, num_long_term_pics, and slice_temporal_mvp_enable_flag shall be the same in all slice segment headers of a coded picture. When present, the value of the slice segment header syntax elements lt_idx_sps[i], poc_lsb_lt[i], used_by_curr_pic_lt_flag[i], delta_poc_msb_present_flag[i], and delta_poc_msb_cycle_lt[i] shall be the same in all slice segment headers of a coded picture for each possible value of i.

first_slice_segment_in_pic_flag equal to 1 specifies that the slice segment is the first slice segment of the picture in decoding order. first_slice_segment_in_pic_flag equal to 0 specifies that the slice segment is not the first slice segment of the picture in decoding order.

no_output_of_prior_pics_flag specifies how the previously-decoded pictures in the decoded picture buffer are treated after decoding of an IDR or a BLA picture. See Annex C. When the current picture is a CRA picture, or the current picture is an IDR or BLA picture that is the first picture in the bitstream, the value of no_output_of_prior_pics_flag has no effect on the decoding process. When the current picture is an IDR or BLA picture that is not the first picture in the bitstream and the value of pic_width_in_luma_samples or pic_height_in_luma_samples or sps_max_dec_pic_buffering[HighestTid] derived from the active sequence parameter set is different from the value of pic_width_in_luma_samples or pic_height_in_luma_samples or sps_max_dec_pic_buffering[HighestTid] derived from the sequence parameter set active for the preceding picture, no_output_of_prior_pics_flag equal to 1 may (but should not) be inferred by the decoder, regardless of the actual value of no_output_of_prior_pics_flag. [Ed. (GJS): Check why this had not been moved previously.] [Ed. (GJS): Use a variable here.]

slice_pic_parameter_set_id specifies the value of pps_pic_parameter_set for the picture parameter set in use. The value of slice_pic_parameter_set_id shall be in the range of 0 to 63, inclusive.

dependent_slice_segment_flag equal to 1 specifies that the value of each slice segment header syntax element that is not present is inferred to be equal to the value of the corresponding slice segment header syntax element in the slice header. When not present, the value of dependent_slice_segment_flag is inferred to be equal to 0.

The variable SliceAddrRS is derived as follows.

– If dependent_slice_segment_flag is equal to 0, SliceAddrRS is set equal to slice_segment_address.

– Otherwise, SliceAddrRS is set equal to SliceAddrRS of the preceding slice segment containing the coding tree block for which the coding tree block address is ctbAddrTStoRS[ctbAddrRStoTS[slice_segment_address] − 1].

slice_segment_address specifies the address of the first coding tree block in the slice segment, in coding tree block raster scan of a picture. The length of the slice_segment_address syntax element is Ceil(Log2(PicSizeInCtbsY)) bits. The value of slice_segment_address shall be in the range of 1 to PicSizeInCtbsY − 1, inclusive and the value of slice_segment_address shall not be equal to the value of slice_segment_address of any other coded slice segment NAL unit of the same coded picture. When slice_segment_address is not present, it is inferred to be equal to 0.

The variable CtbAddrInRS, specifying a coding tree block address in coding tree block raster scan of a picture, is set equal to slice_segment_address. The variable CtbAddrInTS, specifying a coding tree block address in tile scan, is set equal to CtbAddrRStoTS[CtbAddrInRS]. The variable CuQpDelta, 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.

slice_reserved_undetermined_flag[i] has semantics and values that are reserved for future specification by ITU-T | ISO/IEC. Decoders shall ignore the presence and value of slice_reserved_undetermined_flag[i].

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

Table 7-7 – Name association to slice_type

slice_type	Name of slice_type
0	B (B slice)
1	P (P slice)
2	I (I slice)

When nal_unit_type has a value in the range of 16 to 23, inclusive (RAP picture), slice_type shall be equal to 2.

When sps_max_dec_pic_buffering[TemporalId] is equal to 0, slice_type shall be equal to 2.

pic_output_flag affects the decoded picture output and removal processes as specified in Annex C. When pic_output_flag is not present, it is inferred to be equal to 1.

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

NOTE 1 – There is no dependency between the decoding processes of pictures having different values of colour_plane_id.

pic_order_cnt_lsb specifies the picture order count modulo MaxPicOrderCntLsb for the current picture. The length of the pic_order_cnt_lsb syntax element is log2_max_pic_order_cnt_lsb_minus4 + 4 bits. The value of the pic_order_cnt_lsb shall be in the range of 0 to MaxPicOrderCntLsb − 1, inclusive. When pic_order_cnt_lsb is not present, pic_order_cnt_lsb is inferred to be equal to 0, except as specified in subclause 8.3.3.1. [Ed. (GJS): This syntax element name violates the convention against having a syntax element name that is a sub-string of another syntax element name.]

short_term_ref_pic_set_sps_flag equal to 1 specifies that the short-term reference picture set of the current picture is created using syntax elements in the active sequence parameter set. short_term_ref_pic_set_sps_flag equal to 0 specifies that the short-term reference picture set of the current picture is created using syntax elements in the short_term_ref_pic_set( ) syntax structure in the slice header.

short_term_ref_pic_set_idx specifies the index to the list of the short-term reference picture sets specified in the active sequence parameter set that is used for creation of the reference picture set of the current picture. The syntax element short_term_ref_pic_set_idx is represented by Ceil(Log2(num_short_term_ref_pic_sets)) bits. The value of short_term_ref_pic_set_idx shall be in the range of 0 to num_short_term_ref_pic_sets − 1, inclusive.

The variable StRpsIdx is derived as follows:

– If short_term_ref_pic_set_sps_flag is equal to 1, StRpsIdx is set equal to short_term_ref_pic_set_idx.

– Otherwise, StRpsIdx is set equal to num_short_term_ref_pic_sets.

num_long_term_sps specifies the number of candidate long-term reference pictures specified in the active sequence parameter set that are included in the long-term reference picture set of the current picture. The value of num_long_term_sps shall be in the range of 0 to Min(num_long_term_ref_pics_sps, sps_max_dec_pic_buffering[sps_max_sub_layers_minus1] −NumNegativePics[StRpsIdx] − NumPositivePics[StRpsIdx]), inclusive. When not present, the value of num_long_term_sps is inferred to be equal to 0.

num_long_term_pics specifies the number of long-term reference pictures specified in the slice header, which are included in the long-term reference picture set of the current picture. The value of num_long_term_pics shall be in the range of 0 to sps_max_dec_pic_buffering[sps_max_sub_layers_minus1] − NumNegativePics[StRpsIdx] − NumPositivePics[StRpsIdx] − num_long_term_sps, inclusive. When not present, the value of num_long_term_pics is inferred to be equal to 0.

lt_idx_sps[i] specifies an index into the list of candidate long-term reference pictures specified in the active sequence parameter set for identification of the picture that is included in the long-term reference pictur set of the current picture. The value of lt_idx_sps[i] shall be in the range of 0 to num_long_term_ref_pics_sps − 1, inclusive. The number of bits used to represent lt_idx_sps[i] is equal to Ceil(Log2(num_long_term_ref_pics_sps)).

poc_lsb_lt[i] specifies the value of the picture order count modulo MaxPicOrderCntLsb of the i-th long-term reference picture that is included in the long-term reference picture set of the current picture. The length of the poc_lsb_lt[i] syntax element is log2_max_pic_order_cnt_lsb_minus4 + 4 bits.

used_by_curr_pic_lt_flag[i] equal to 0 specifies that the i-th long-term reference picture included in the long-term reference picture set of the current picture is not used for reference by the current picture.

The variables PocLsbLt[i] and UsedByCurrPicLt[i] are derived as follows:

– If i is less than num_long_term_sps, PocLsbLt[i] is set equal to lt_ref_pic_poc_lsb_sps[lt_idx_sps[i]] and UsedByCurrPicLt[i] is set equal to used_by_curr_pic_lt_sps_flag[lt_idx_sps[i]].

– Otherwise, PocLsbLt[i] is set equal to poc_lsb_lt[i] and UsedByCurrPicLt[i] is set equal to used_by_curr_pic_lt_flag[i].

delta_poc_msb_present_flag[i] equal to 1 specifies that delta_poc_msb_cycle_lt[i] is present. delta_poc_msb_present_flag[i] equal to 0 specifies that delta_poc_msb_cycle_lt[i] is not present. delta_poc_msb_present_flag[i] shall be equal to 1 when there is more than one reference picture in the decoded picture buffer with picture order count modulo MaxPicOrderCntLsb equal to PocLsbLt[i].

delta_poc_msb_cycle_lt[i] is used to determine the value of the most significant bits of the picture order count value of the i-th long-term reference picture that is included in the long-term reference picture set of the current picture. When delta_poc_msb_cycle_lt[i] is not present, it is inferred to be equal to 0.

The variable DeltaPocMSBCycleLt[i] is derived as follows:

```
if( i  = =  0  | |  i = =  num_long_term_sps )
    DeltaPocMSBCycleLt[ i ] = delta_poc_msb_cycle_lt[ i ]
else                                                                    (7-46)
    DeltaPocMSBCycleLt[ i ] = delta_poc_msb_cycle_lt[ i ] + DeltaPocMSBCycleLt[ i − 1 ]
```

slice_temporal_mvp_enable_flag specifies whether temporal motion vector predictors can be used for inter prediction. If slice_temporal_mvp_enable_flag is equal to 0, temporal motion vector predictors shall not be used in decoding of the current picture. If slice_temporal_mvp_enable_flag is equal to 1, temporal motion vector predictors may be used in decoding of the current picture. When not present, the value of slice_temporal_mvp_enable_flag is inferred to be equal to 0. [Ed. (GJS): Check the use of "shall" in this paragraph. Is that a bitstream constraint or a decoding process specification? Is it duplicating a prescription expressed elsewhere?]

When both slice_temporal_mvp_enable_flag and TemporalId are equal to 0, the decoding process of all coded pictures that follow the current picture in decoding order shall not use temporal motion vectors from any picture that precedes the current picture in decoding order. [Ed. (GJS): Check the use of "shall" in this paragraph.]

slice_sao_luma_flag equal to 1 specifies that SAO is enabled for the luma component in the current slice; slice_sao_luma_flag equal to 0 specifies that SAO is disabled for the luma component in the current slice. When slice_sao_luma_flag is not present, it is inferred to be equal to 0.

slice_sao_chroma_flag equal to 1 specifies that SAO is enabled for the chroma component in the current slice; slice_sao_chroma_flag equal to 0 specifies that SAO is disabled for the chroma component in the current slice. When slice_sao_chroma_flag is not present, it is inferred to be equal to 0.

num_ref_idx_active_override_flag equal to 1 specifies that the syntax element num_ref_idx_l0_active_minus1 is present for P and B slices and that the syntax element num_ref_idx_l1_active_minus1 is present for B slices. num_ref_idx_active_override_flag equal to 0 specifies that the syntax elements num_ref_idx_l0_active_minus1 and num_ref_idx_l1_active_minus1 are not present.

num_ref_idx_l0_active_minus1 specifies the maximum reference index for reference picture list 0 that may be used to decode the slice. num_ref_idx_l0_active_minus1 shall be in the range of 0 to 15, inclusive. When the current slice is a P or B slice and num_ref_idx_l0_active_minus1 is not present, num_ref_idx_l0_active_minus1 is inferred to be equal to num_ref_idx_l0_default_active_minus1.

num_ref_idx_l1_active_minus1 specifies the maximum reference index for reference picture list 1 that shall be used to decode the slice. num_ref_idx_l1_active_minus1 shall be in the range of 0 to 15, inclusive. When the current slice is a B slice and num_ref_idx_l1_active_minus1 is not present, num_ref_idx_l1_active_minus1 is inferred to be equal to num_ref_idx_l1_default_active_minus1.

mvd_l1_zero_flag equal to 1 indicates that the mvd_coding(x0, y0, 1) syntax structure is not parsed and MvdL1[x0][y0][compIdx] is set equal to 0 for compIdx = 0..1. mvd_l1_zero_flag equal to 0 indicates that the mvd_coding(x0, y0, 1) syntax structure is parsed.

cabac_init_flag specifies the method for determining the initialization table used in the initialization process for context variables. When cabac_init_flag is not present, it is inferred to be 0.

collocated_from_l0_flag equal to 1 specifies the picture that contains the collocated partition is derived from list 0, otherwise the picture is derived from list 1.When collocated_from_l0_flag is not present, it is inferred to be equal to 1.

collocated_ref_idx specifies the reference index of the picture that contains the collocated partition. When the current slice is a P slice, collocated_ref_idx refers to a picture in list 0. When the current slice is a B slice, collocated_ref_idx refers to a picture in list 0 if collocated_from_l0 is 1, otherwise it refers to a picture in list 1. collocated_ref_idx shall always refer to a valid list entry, and the resulting picture shall be the same for all slices of a coded picture. When collocated_ref_idx is not present, it is inferred to be equal to 0.

five_minus_max_num_merge_cand specifies the maximum number of merging MVP candidates supported in the slice subtracted from 5. The maximum number of merging MVP candidates, MaxNumMergeCand is derived as

$$MaxNumMergeCand = 5 - five\_minus\_max\_num\_merge\_cand \quad (7\text{-}47)$$

The value of five_minus_max_num_merge_cand shall be limited such that MaxNumMergeCand is in the range of 1 to 5, inclusive.

slice_qp_delta specifies the initial value of QP_Y to be used for the coding blocks in the slice until modified by the value of CuQpDelta in the coding unit layer. The initial value of the QP_Y quantization parameter for the slice, $SliceQP_Y$, is derived as

$$SliceQP_Y = 26 + init\_qp\_minus26 + slice\_qp\_delta \quad (7\text{-}48)$$

The value of slice_qp_delta shall be limited such that $SliceQP_Y$ is in the range of $-QpBdOffset_Y$ to +51, inclusive.

slice_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 slice_cb_qp_offset shall be in the range of −12 to +12, inclusive. When slice_cb_qp_offset is not present, it is inferred to be equal to 0. The value of pps_cb_qp_offset + slice_cb_qp_offset shall be in the range of −12 to +12, inclusive.

slice_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 slice_cr_qp_offset shall be in the range of −12 to +12, inclusive. When slice_cr_qp_offset is not present, it is inferred to be equal to 0. The value of pps_cr_qp_offset + slice_cr_qp_offset shall be in the range of −12 to +12, inclusive.

deblocking_filter_override_flag equal to 0 specifies that deblocking parameters from the active picture parameter set are used for deblocking the current slice. deblocking_filter_override_flag equal to 1 specifies that deblocking parameters from the slice header are used for deblocking the current slice. When not present, the value of deblocking_filter_override_flag is inferred to be equal to 0.

slice_disable_deblocking_filter_flag equal to 1 specifies that the operation of the deblocking filter is not applied for the current slice. slice_disable_deblocking_filter_flag equal to 0 specifies that the operation of the deblocking filter is applied for the current slice. When slice_disable_deblocking_filter_flag is not present, it is inferred to be equal to pps_disable_deblocking_filter_flag.

slice_beta_offset_div2 and **slice_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) for the current slice. The values of slice_beta_offset_div2 and slice_tc_offset_div2 shall be in the range of −6 to 6, inclusive. When not present, the values of slice_beta_offset_div2 and slice_tc_offset_div2 are inferred to be equal to pps_beta_offset_div2 and pps_tc_offset_div2, respectively.

slice_loop_filter_across_slices_enabled_flag equal to 1 specifies that in-loop filtering operations are performed across the left and upper boundaries of the current slice; otherwise, the in-loop operations are not applied across left and upper boundaries of the current slice. The in-loop filtering operations include the deblocking filter and sample adaptive offset filter. When slice_loop_filter_across_slices_enabled_flag is not present, it is inferred to be equal to loop_filter_across_slices_enabled_flag.

num_entry_point_offsets specifies the number of entry_point_offset[i] syntax elements in the slice header. When not present, the value of num_entry_point_offsets is inferred to be equal to 0.

The value of num_entry_point_offsets shall be constrained as follows:

– If tiles_enabled_flag is equal to 0 and entropy_coding_sync_enabled_flag is equal to 1, the value of num_entry_point_offsets shall be in the range of 0 to PicHeightInCtbsY − 1, inclusive.

– Otherwise, if tiles_enabled_flag is equal to 1 and entropy_coding_sync_enabled_flag is equal to 0, the value of num_entry_point_offsets shall be in the range of 0 to (num_tile_columns_minus1 + 1) * (num_tile_rows_minus1 + 1) − 1, inclusive.

– Otherwise, when tiles_enabled_flag is equal to 1 and entropy_coding_sync_enabled_flag is equal to 1, the value of num_entry_point_offsets shall be in the range of 0 to (num_tile_columns_minus1 + 1) * PicHeightInCtbsY − 1, inclusive.

offset_len_minus1 plus 1 specifies the length, in bits, of the entry_point_offset[i] syntax elements. The value of offset_len_minus1 shall be in the range of 0 to 31, inclusive.

entry_point_offset[i] specifies the i-th entry point offset in bytes, and is represented by offset_len_minus1 plus 1 bits. The slice segment data that follows the slice segment header consists of num_entry_point_offsets + 1 subsets, with subset index values ranging from 0 to num_entry_point_offsets, inclusive. The first byte of the slice segment data is considered byte 0. When present, emulation prevention bytes that appear in the slice segment data portion of the coded slice segment NAL unit are counted as part of the slice segment data for purposes of subset identification. Subset 0 consists of bytes 0 to entry_point_offset[0] − 1, inclusive, of the coded slice segment data, subset k, with k in the range of 1 to num_entry_point_offsets − 1, inclusive, consists of bytes firstByte[k] to lastByte[k], inclusive, of the coded slice segment data with firstByte[k] and lastByte[k] defined as:

$$\text{firstByte}[k] = \sum_n(\text{entry\_point\_offset}[n - 1]) \text{ with } n = 1..k \quad (7\text{-}49)$$

$$\text{lastByte}[k] = \text{firstByte}[k] - 1 + \text{entry\_point\_offset}[k] \quad (7\text{-}50)$$

The last subset (with subset index equal to num_entry_point_offsets) consists of the remaining bytes of the coded slice segment data.

When tiles_enabled_flag is equal to 1 and entropy_coding_sync_enabled_flag is equal to 0, each subset shall consist of all coded bits of all coding tree units in the slice segment that are within the same tile. In this case, the number of subsets (i.e. the value of num_entry_point_offsets + 1) shall be equal to the number of tiles that contain coding tree units that are in the coded slice segment.

NOTE 2 – When tiles_enabled_flag is equal to 1 and entropy_coding_sync_enabled_flag is equal to 0, each slice must include either a subset of the coding tree units of one tile (in which case the syntax element entry_point_offset[i] is not present) or must include all coding tree units of an integer number of complete tiles.

When tiles_enabled_flag is equal to 0 and entropy_coding_sync_enabled_flag is equal to 1, each subset k with k in the range of 0 to num_entry_point_offsets, inclusive, shall consist of all coded bits of all coding tree units in the slice semgnet that are within the same row of coding tree units in the picture. In this case, the number of subsets (i.e. the value of num_entry_point_offsets + 1) shall be equal to the number of rows of coding tree units of the picture that contain coding tree units that are in the coded slice segment.

NOTE 3 – The last subset (i.e. subset k for k equal to num_entry_point_offsets) may or may not contain all coding tree units that are within a row of coding tree units in the picture.

When tiles_enabled_flag is equal to 1 and entropy_coding_sync_enabled_flag is equal to 1, each subset k with k in the range of 0 to num_entry_point_offsets − 1, inclusive, shall consist of all coded bits of all coding tree units in the slice segment that are within the same row of coding tree units in a tile. In this case, the number of subsets (i.e. the value of num_entry_point_offsets + 1) shall be equal to the number of rows of coding tree units of tiles that contain coding tree units that are in the coded slice segment.

[Ed. (RS): Search/fix uses of "coding tree blocks" vs. "coding tree units" and "coding blocks" vs. "coding units".]

slice_segment_header_extension_length specifies the length of the slice segment header extension data in bytes, not including the bits used for signalling slice_segment_header_extension_length itself. The value of slice_segment_header_extension_length shall be in the range of 0 to 256, inclusive.

slice_segment_header_extension_data_byte may have any value. Decoders shall ignore the value of slice_segment_header_extension_data_byte. Its value does not affect decoder conformance to profiles specified in this version of this Specification.

7.4.8.2 Short-term reference picture set semantics

A short-term reference picture set may be present in a sequence parameter set or in a slice header. When a short-term reference picture set is present in a slice header, the content of the short-term reference picture set syntax structure shall be the same in all slice headers of a picture and the value of idxRps shall be equal to the syntax element num_short_term_ref_pic_sets from the active sequence parameter set. In the following, when the short_term_ref_pic_set(idxRps) syntax structure is included in the sequence parameter set, the term "the current picture" applies to all pictures in the coded video sequence with short_term_ref_pic_set_idx equal to idxRps.

inter_ref_pic_set_prediction_flag equal to 1 specifies that the reference picture set of the current picture is predicted using another reference picture set in the active sequence parameter set. When inter_ref_pic_set_prediction_flag is not present, it is inferred to be equal to 0.

delta_idx_minus1 plus 1 specifies the difference between the index of the reference picture set of the current picture and the index of the reference picture set used for inter reference picture set prediction. The value of delta_idx_minus1 shall be in the range of 0 to idxRps − 1, inclusive. When delta_idx_minus1 is not present, it is inferred to be equal to 0.

The variable RIdx is derived as follows.

RIdx = idxRps − (delta_idx_minus1 + 1) (7-51)

delta_rps_sign and **abs_delta_rps_minus1** together specify the value of the variable DeltaRPS as follows

DeltaRPS = (1 − 2 * delta_rps_sign) * (abs_delta_rps_minus1 + 1) (7-52)

The variable DeltaRPS represents the value to be added to picture order count difference values of the reference picture set used for inter reference picture set prediction to obtain the picture order count difference values of the current reference picture set.

used_by_curr_pic_flag[j] equal to 0 specifies that the j-th picture is not used for reference by the current picture.

use_delta_flag[j] equal to 1 specifies that the j-th picture is included in the reference picture set of the current picture. use_delta_flag[j] equal to 0 specifies that the j-th picture is not included in the reference picture set of the current picture. When use_delta_flag[j] is not present, its value is inferred to be equal to 1.

When inter_ref_pic_set_prediction_flag is equal to 1, the variables DeltaPocS0[idxRps][i], UsedByCurrPicS0[idxRps][i], and NumNegativePics[idxRps] are derived as follows.

```
i = 0
for( j = NumPositivePics[ RIdx ] − 1; j >= 0; j− − ) {
    dPoc = DeltaPocS1[ RIdx ][ j ] + DeltaRPS
    if( dPoc < 0 && use_delta_flag[ NumNegativePics[ RIdx ] + j ] ) {
        DeltaPocS0[ idxRps ][ i ] = dPoc
        UsedByCurrPicS0[ idxRps ][ i++ ] = used_by_curr_pic_flag[ NumNegativePics[ RIdx ] + j ]
    }
}
if( DeltaRPS < 0 && use_delta_flag[ NumDeltaPocs[ RIdx ] ] ) {
    DeltaPocS0[ idxRps ][ i ] = DeltaRPS
    UsedByCurrPicS0[ idxRps ][ i++ ] = used_by_curr_pic_flag[ NumDeltaPocs[ RIdx ] ]          (7-53)
}
for( j = 0; j < NumNegativePics[ RIdx ]; j++ ) {
    dPoc = DeltaPocS0[ RIdx ][ j ] + DeltaRPS
    if( dPoc < 0 && use_delta_flag[ j ] ) {
        DeltaPocS0[ idxRps ][ i ] = dPoc
        UsedByCurrPicS0[ idxRps ][ i++ ] = used_by_curr_pic_flag[ j ]
    }
}
NumNegativePics[ idxRps ] = i
```

When inter_ref_pic_set_prediction_flag is equal to 1, the variables DeltaPocS1[idxRps][i], UsedByCurrPicS1[idxRps][i], and NumPositivePics[idxRps] are derived as follows.

```
i = 0
for( j = NumNegativePics[ RIdx ] − 1; j >= 0; j− − ) {
    dPoc = DeltaPocS0[ RIdx ][ j ] + DeltaRPS
    if( dPoc > 0 && use_delta_flag[ j ] ) {
        DeltaPocS1[ idxRps ][ i ] = dPoc
        UsedByCurrPicS1[ idxRps ][ i++ ] = used_by_curr_pic_flag[ j ]
    }
}
if( DeltaRPS > 0 && use_delta_flag[ NumDeltaPocs[ RIdx ] ] ) {
    DeltaPocS1[ idxRps ][ i ] = DeltaRPS
    UsedByCurrPicS1[ idxRps ][ i++ ] = used_by_curr_pic_flag[ NumDeltaPocs[ RIdx ] ]          (7-54)
}
for( j = 0; j < NumPositivePics[ RIdx ]; j++) {
    dPoc = DeltaPocS1[ RIdx ][ j ] + DeltaRPS
    if( dPoc > 0 && use_delta_flag[ NumNegativePics[ RIdx ] + j ] ) {
        DeltaPocS1[ idxRps ][ i ] = dPoc
        UsedByCurrPicS1[ idxRps ][ i++ ] = used_by_curr_pic_flag[ NumNegativePics[ RIdx ] + j ]
    }
}
NumPositivePics[ idxRps ] = i
```

num_negative_pics specifies the number of the following delta_poc_s0_minus1[i] and used_by_curr_pic_s0_flag[i] syntax elements. The value of num_negative_pics shall be in the range of 0 to sps_max_dec_pic_buffering[sps_max_sub_layers_minus1], inclusive.

num_positive_pics specifies the number of the following delta_poc_s1_minus1[i] and used_by_curr_pic_s1_flag1[i] syntax elements. The value of num_positive_pics shall be in the range of 0 to sps_max_dec_pic_buffering[sps_max_sub_layers_minus1] − num_negative_pics, inclusive.

delta_poc_s0_minus1[i] plus 1 specifies an absolute difference between two picture order count values for the i-th reference picture that has picture order count less than that of the current picture. The value of delta_poc_s0_minus1[i] shall be in the range of 0 to $2^{15} - 1$, inclusive.

used_by_curr_pic_s0_flag[i] equal to 0 specifies that the i-th reference picture that has picture order count less than that of the current picture is not used for reference by the current picture.

delta_poc_s1_minus1[i] plus 1 specifies an absolute difference between two picture order count values for the i-th reference picture that has picture order count greater than that of the current picture. The value of delta_poc_s1_minus1[i] shall be in the range of 0 to $2^{15} - 1$, inclusive.

used_by_curr_pic_s1_flag[i] equal to 0 specifies that the i-th reference picture that has picture order count greater than that of the current picture is not used for reference by the current picture.

When inter_ref_pic_set_prediction_flag is equal to 0, the variables NumNegativePics[idxRps], NumPositivePics[idxRps], UsedByCurrPicS0[idxRps][i], UsedByCurrPicS1[idxRps][i], DeltaPocS0[idxRps][i], and DeltaPocS1[idxRps][i] are derived as follows.

NumNegativePics[idxRps] = num_negative_pics (7-55)
NumPositivePics[idxRps] = num_positive_pics (7-56)
UsedByCurrPicS0[idxRps][i] = used_by_curr_pic_s0_flag[i] (7-57)
UsedByCurrPicS1[idxRps][i] = used_by_curr_pic_s1_flag[i] (7-58)

– If the variable i is equal to 0, the following applies.

DeltaPocS0[idxRps][i] = − (delta_poc_s0_minus1[i] + 1) (7-59)
DeltaPocS1[idxRps][i] = delta_poc_s1_minus1[i] + 1 (7-60)

– Otherwise, the following applies.

DeltaPocS0[idxRps][i] = DeltaPocS0[idxRps][i − 1] − (delta_poc_s0_minus1[i] + 1) (7-61)
DeltaPocS1[idxRps][i] = DeltaPocS1[idxRps][i − 1] + (delta_poc_s1_minus1[i] + 1) (7-62)

The variable NumDeltaPocs[idxRps] is derived as follows.

NumDeltaPocs[idxRps] = NumNegativePics[idxRps] + NumPositivePics[idxRps] (7-63)

7.4.8.3 Reference picture list modification semantics

ref_pic_list_modification_flag_l0 equal to 1 indicates that reference picture list 0 is specified explicitly by a list of list_entry_l0[i] values. ref_pic_list_modification_flag_l0 equal to 0 indicates that reference picture list 0 is determined implicitly. When ref_pic_list_modification_flag_l0 is not present in the slice header, it is inferred to be equal to 0.

list_entry_l0[i] specifies the index of the reference picture in RefPicListTemp0 to be placed at the current position of reference picture list 0. The length of the list_entry_l0[i] syntax element is Ceil(Log2(NumPocTotalCurr)) bits. The value of list_entry_l0[i] shall be in the range of 0 to NumPocTotalCurr − 1, inclusive. When the syntax element list_entry_l0[i] is not present in the slice header, it is inferred to be equal to 0. [Ed. (YK/GJS): OK to remove that last sentence?]

The variable NumPocTotalCurr is derived as follows.

```
NumPocTotalCurr = 0;
for( i = 0; i < NumNegativePics[ StRpsIdx ]; i++)
    if(UsedByCurrPicS0[ StRpsIdx ][ i ] = = 1)
        NumPocTotalCurr++
for( i = 0; i < NumPositivePics[ StRpsIdx ]; i++)                    (7-64)
    if(UsedByCurrPicS1[ StRpsIdx ][ i ] = = 1)
        NumPocTotalCurr++
for( i = 0; i < num_long_term_sps + num_long_term_pics; i++ )
    if( UsedByCurrPicLt[ i ] = = 1)
        NumPocTotalCurr++
```

ref_pic_list_modification_flag_l1 equal to 1 indicates that reference picture list 1 is specified explicitly by a list of list_entry_l1[i] values. ref_pic_list_modification_flag_l1 equal to 0 indicates that reference picture list 1 is determined implicitly. When ref_pic_list_modification_flag_l1 is not present in the slice header, it is inferred to be equal to 0.

list_entry_l1[i] specifies the index of the reference picture in RefPicListTemp1 to be placed at the current position of reference picture list 1. The length of the list_entry_l1[i] syntax element is Ceil(Log2(NumPocTotalCurr)) bits. The value of list_entry_l1[i] shall be in the range of 0 to NumPocTotalCurr − 1, inclusive. When the syntax element list_entry_l1[i] is not present in the slice header, it is inferred to be equal to 0. [Ed. (YK/GJS): OK to remove that last sentence?]

7.4.8.4 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.

The variable ChromaLog2WeightDenom is specified by luma_log2_weight_denom + delta_chroma_log2_weight_denom and it shall be in the range of 0 to 7, inclusive.

luma_weight_l0_flag[i] equal to 1 specifies that weighting factors for the luma component of list 0 prediction using RefPicList0[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 RefPicList0[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 RefPicList0[i].

The variable LumaWeightL0[i] is specified by (1 << luma_log2_weight_denom) + delta_luma_weight_l0[i]. When luma_weight_l0_flag[i] is equal to 1, the value of delta_luma_weight_l0[i] shall be in the range of −128 to 127, inclusive. When luma_weight_l0_flag[i] is equal to 0, LumaWeightL0[i] is inferred to be equal to $2^{\text{luma\_log2\_weight\_denom}}$.

luma_offset_l0[i **]** is the additive offset applied to the luma prediction value for list 0 prediction using RefPicList0[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 as 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 RefPicList0[i] with j equal to 0 for Cb and j equal to 1 for Cr.

The variable ChromaWeightL0[i][j] is specified by (1 << ChromaLog2WeightDenom) + delta_chroma_weight_l0[i][j]. When chroma_weight_l0_flag[i] is equal to 1, the value of delta_chroma_weight_l0[i][j] shall be in the range of −128 to 127, inclusive. When chroma_weight_l0_flag[i] is equal to 0**,** ChromaWeightL0[i][j] is inferred to be equal to $2^{\text{ChromaLog2WeightDenom}}$.

delta_chroma_offset_l0[i **][** j **]** is the difference of the additive offset applied to the chroma prediction values for list 0 prediction using RefPicList0[i] with j equal to 0 for Cb and j equal to 1 for Cr.

The variable ChromaOffsetL0[i][j] is specified as follows:

```
ChromaOffsetL0[ i ][ j ] = Clip3( −128, 127, ( delta_chroma_offset_l0[ i ][ j ] −          (7-65)
                ( ( 128 * ChromaWeightL0[ i ][ j ] ) >> ChromaLog2WeightDenom ) + 128 ) )
```

The value of delta_chroma_offset_l0[i][j] shall be in the range of −512 to 511, inclusive. When chroma_weight_l0_flag[i] is equal to 0**,** ChromaOffsetL0[i][j] is inferred to be equal to 0.

luma_weight_l1_flag[i]**, 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 luma_weight_l0_flag[i], 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, list 0, and List0 replaced by l1, list 1, and List1, respectively.

The variable sumWeightL0Flags is specified as the sum of luma_weight_l0_flag[i] + 2 * chroma_weight_l0_flag[i], for i = 0..num_ref_idx_l0_active_minus1.

When slice_type is equal to B, the variable sumWeightL1Flags is specified as the sum of luma_weight_l1_flag[i] + 2 * chroma_weight_l1_flag[i], for i = 0..num_ref_idx_l1_active_minus1.

It is a requirement of bitstream conformance that, when slice_type is equal to P, sumWeightL0Flags shall be less than or equal to 24, and when slice_type is equal to B, the sum of sumWeightL0Flags and sumWeightL1Flags shall be less than or equal to 24.

7.4.9 Slice segment data semantics

7.4.9.1 General slice segment data semantics

end_of_slice_segment_flag equal to 0 specifies that another coding tree unit is following in the slice. end_of_slice_segment_flag equal to 1 specifies the end of the slice segment, i.e. that no further coding tree unit follows in the slice segment.

end_of_sub_stream_one_bit shall be equal to 1.

7.4.9.2 Coding tree unit semantics

The coding tree unit is the root node of the coding quadtree structure.

7.4.9.3 Sample adaptive offset semantics

sao_merge_left_flag equal to 1 specifies that the syntax elements sao_type_idx_luma, sao_type_idx_chroma, sao_band_position, sao_eo_class_luma, sao_eo_class_chroma, sao_offset_abs and sao_offset_sign are derived from the corresponding syntax elements of the left coding tree block; equal to 0 specifies that these syntax elements are not derived from the corresponding syntax elements of the left coding tree block. 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 are derived from the corresponding syntax elements of the above coding tree block; equal to 0 specifies that these syntax elements are not derived from the corresponding syntax elements of the above coding tree block. When sao_merge_up_flag is not present, it is inferred to be equal to 0.

sao_type_idx_luma indicates the offset type for the luma component. The array SaoTypeIdx[cIdx][rx][ry] indicates the offset type as specified in Table 7-8 of current coding tree block at position rx and 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 inferred 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 indicates 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 inferred 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 7-8 – 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] indicates the offset value of i-th category of current coding tree block at position rx and 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 set 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 set equal to sao_offset_abs[cIdx][rx][ry − 1][i].

– Otherwise, sao_offset_abs[cIdx][rx][ry][i] is set equal to 0.

The variable bitDepth is derived as follows.

– If cIdx is equal to 0, bitDepth is set equal to $BitDepth_Y$.

– Otherwise (cIdx is equal to1 or 2), bitDepth is set equal to $BitDepth_C$.

sao_offset_sign[cIdx][rx][ry][i] specifies the sign of sao_offset[cIdx][rx][ry][i] when SaoTypeIdx[cIdx][rx][ry] is equal to 1.

When sao_offset_sign[cIdx][rx][ry][i] is not present, it is inferred as follows.

– If sao_merge_left_flag is equal to 1, sao_offset_sign[cIdx][rx][ry][i] is set equal to sao_offset_sign[cIdx][rx − 1][ry][i]

– Otherwise, if sao_merge_up_flag is equal to 1, sao_offset_sign[cIdx][rx][ry][i] is set equal to sao_offset_sign[cIdx][rx][ry − 1][i].

– Otherwise, sao_offset_sign[cIdx][rx][ry][i] is set equal 0.

The variable offsetSign is derived as follows.

– If SaoTypeIdx[cIdx][rx][ry] is equal to 2 and i is greater than 1,offsetSign is set equal to −1.

– Otherwise, if SaoTypeIdx[cIdx][rx][ry] is equal to 2 and i is less than 2, offsetSign is set equal to 1.

– Otherwise, if sao_offset_sign[cIdx][rx][ry][i] is equal to 0, offsetSign is set to equal to 1.

– Otherwise, offsetSign is set to equal to −1.

The array SaoOffsetVal is derived as follows.

$$SaoOffsetVal[cIdx][rx][ry][0] = 0 \quad (7\text{-}66)$$

$$SaoOffsetVal[cIdx][rx][ry][i + 1] = offsetSign * sao\_offset\_abs[cIdx][rx][ry][i] << (bitDepth - Min(bitDepth, 10)) \quad (7\text{-}67)$$

sao_band_position[cIdx][rx][ry] indicates 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 set 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 set equal to sao_band_position[cIdx][rx][ry − 1].

– Otherwise, sao_band_position[cIdx][rx][ry] is set equal to 0.

sao_eo_class_luma indicates the edge offset class for the luma component. The array SaoEoClass[cIdx][rx][ry] indicates the offset type as specified in Table 7-9 of current coding tree block at position rx and 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 inferred 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 indicates 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 inferred 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 7-9 – Specification of the SAO edge offset class

SaoEoClass[cIdx][rx][ry]	SAO edge offset class (informative)
0	1D 0-degree edge offset
1	1D 90-degree edge offset
2	1D 135-degree edge offset
3	1D 45-degree edge offset

7.4.9.4 Coding quadtree semantics

split_cu_flag[x0][y0] specifies whether a coding unit is split into coding units with half horizontal and vertical size. 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 split_cu_flag[x0][y0] is not present, the following applies:

– If log2CbSize is greater than Log2MinCbSizeY, the value of split_cu_flag[x0][y0] is inferred to be equal to 1.

– Otherwise (log2CbSize is equal to Log2MinCbSizeY), the value of split_cu_flag[x0][y0] is inferred to be equal to 0.

The array CtDepth[x][y] specifies the coding tree depth for a luma coding block covering location (x, y). When split_cu_flag[x0][y0] is equal to 0, CtDepth[x][y] is inferred to be equal to cqtDepth for x = x0..x0 + nCbS − 1 and y = y0..y0 + nCbS − 1.

7.4.9.5 Coding unit semantics

cu_transquant_bypass_flag equal to 1 specifies that the scaling and transform process as specified in subclause 8.6 and the in-loop filter process as specified in subclause 8.7 are bypassed. When cu_transquant_bypass_flag is not present, it is inferred to be equal to 0.

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 the merging candidate index 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. The variable CuPredMode[x][y] is derived as follows for x = x0..x0 + nCbS − 1 and y = y0..y0 + nCbS − 1.

– If pred_mode_flag is equal to 0,
    – CuPredMode[x][y] is set to MODE_INTER.
– Otherwise (pred_mode_flag is equal to 1),
    – CuPredMode[x][y] is set to MODE_INTRA.

When pred_mode_flag is not present, the variable CuPredMode[x][y] is derived as follows for x = x0..x0 + nCbS − 1 and y = y0..y0 + nCbS − 1.

– If slice_type is equal to I,
    – CuPredMode[x][y] is inferred to be equal to MODE_INTRA
– Otherwise (slice_type is equal to P or B), if cu_skip_flag[x0][y0] is equal to 1,
    – CuPredMode[x][y] is inferred to be equal to MODE_SKIP

part_mode specifies partitioning mode of the current coding unit. The semantics of part_mode depend on CuPredMode[x0][y0]. The variables PartMode and IntraSplitFlag are derived from the value of part_mode as defined in Table 7-10.

The value of part_mode is restricted as follows.

– If CuPredMode[x0][y0] is equal to MODE_INTRA, part_mode shall be equal to 0 or 1.
– Otherwise (CuPredMode[x0][y0] is equal to MODE_INTER), the following applies
    – If log2CbSize is greater than Log2MinCbSizeY and amp_enabled_flag is equal to 1, part_mode shall be in the range of 0 to 2, inclusive and in the range of 4 to 7, inclusive.
    – Otherwise, if log2CbSize is greater than Log2MinCbSizeY and amp_enabled_flag is equal to 0, part_mode shall be in the range of 0 to 2, inclusive.
    – Otherwise, if log2CbSize is equal to 3, the value of part_mode shall be in the range of 0 to 2, inclusive.
    – Otherwise (log2CbSize is greater than 3), the value of part_mode shall be in the range of 0 to 3, inclusive.

When part_mode is not present, the variables PartMode and IntraSplitFlag are derived as follows.

– PartMode is inferred to be equal to PART_2Nx2N,
– IntraSplitFlag is inferred to be equal to 0.

pcm_flag[x0][y0] equal to 1 specifies that pcm_sample( ) syntax is present and transform_tree( ) syntax is not present in the coding unit including the luma coding block at location (x0, y0). pcm_flag[x0][y0] equal to 0 specifies that pcm_sample( ) syntax is not present. When pcm_flag[x0][y0] is not present, it is inferred to be equal to 0.

The value of pcm_flag[x0 + i][y0 + j] with i = 1..nCbS − 1, j = 1..nCbS − 1 is inferred to be equal to pcm_flag[x0][y0].

pcm_alignment_zero_bit is a bit equal to 0.

Table 7-10 – Name association to prediction mode and partitioning type

CuPredMode[x0][y0]	part_mode	IntraSplitFlag	PartMode
MODE_INTRA	0	0	PART_2Nx2N
	1	1	PART_NxN
MODE_INTER	0	0	PART_2Nx2N
	1	0	PART_2NxN
	2	0	PART_Nx2N
	3	0	PART_NxN
	4	0	PART_2NxnU
	5	0	PART_2NxnD
	6	0	PART_nLx2N
	7	0	PART_nRx2N

prev_intra_luma_pred_flag[x0 + i][y0 + j], **mpm_idx[** x0 + i **][** y0 + j **]** and **rem_intra_luma_pred_mode**[x0 + i][y0 + j] specify the intra prediction mode for luma samples. The array indices x0 + i, y0 + j specify the location (x0 + i, y0 + j) of the top-left luma sample of the considered prediction block relative to the top-left luma sample of the picture. When prev_intra_luma_pred_flag[x0 + i][y0 + j] is equal to 1, the intra prediction mode is inferred from a neighbouring intra-predicted prediction unit according to subclause 8.4.2.

intra_chroma_pred_mode[x0][y0] specifies the intra prediction mode for chroma samples. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered prediction block relative to the top-left luma sample of the picture.

rqt_root_cbf equal to 1 specifies that transform_tree( ) syntax is present for the current coding unit. rqt_root_cbf equal to 0 specifies that transform_tree( ) syntax is not present for the current coding unit.

When rqt_root_cbf is not present, its value is inferred to be equal to 1.

7.4.9.6 Prediction unit semantics

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 prediction 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.

merge_flag[x0][y0] specifies whether the inter prediction parameters for the current prediction 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 prediction block relative to the top-left luma sample of the picture.

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 prediction block relative to the top-left luma sample of the picture.

When merge_idx[x0][y0] is not present, it is inferred to be equal to 0.

inter_pred_idc[x0][y0] specifies whether list0, list1 or bi-prediction is used for the current prediction unit according to Table 7-11. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered prediction block relative to the top-left luma sample of the picture.

Table 7-11 – Name association to inter prediction mode

inter_pred_idc	Name of inter_pred_idc	
	(nPbW + nPbH) != 12	(nPbW + nPbH) = = 12
0	Pred_L0	Pred_L0
1	Pred_L1	Pred_L1
2	Pred_BI	na

When inter_pred_idc[x0][y0] is not present, it is inferred to be equal to Pred_L0.

ref_idx_l0[x0][y0] specifies the list 0 reference picture index for the current prediction unit. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered prediction block relative to the top-left luma sample of the picture.

When ref_idx_l0[x0][y0] is not present it is inferred to be equal to 0.

ref_idx_l1[x0][y0] has the same semantics as ref_idx_l0, with l0 and list 0 replaced by l1 and list 1, respectively.

7.4.9.7 PCM sample semantics

pcm_sample_luma[i] represents a coded luma sample value in the raster scan within the coding unit. The number of bits used to represent each of these samples is $PCMBitDepth_Y$.

pcm_sample_chroma[i] represents a coded chroma sample value in the raster scan within the coding unit. The first half of the values represent coded Cb samples and the remaining half of the values represent coded Cr samples. The number of bits used to represent each of these samples is $PCMBitDepth_C$.

7.4.9.8 Transform tree semantics

split_transform_flag[x0][y0][trafoDepth] specifies whether a block is split into four blocks with half horizontal and half vertical size for the purpose of transform coding. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered block relative to the top-left luma sample of the picture. The array index trafoDepth specifies the current subdivision level of a coding block into blocks for the purpose of transform coding. trafoDepth is equal to 0 for blocks that correspond to coding blocks.

The variable interSplitFlag is derived as follows.

– If max_transform_hierarchy_depth_inter is equal to 0 and CuPredMode[x0][y0] is equal to MODE_INTER and PartMode is not equal to PART_2Nx2N and trafoDepth is equal to 0, interSplitFlag is set to 1.

– Otherwise, interSplitFlag is set to 0.

When split_transform_flag[x0][y0][trafoDepth] is not present, it is inferred as follows:

– If one or more of the following conditions are true, the value of split_transform_flag[x0][y0][trafoDepth] is inferred to be equal to 1.

  – log2TrafoSize is greater than Log2MaxTrafoSize

  – IntraSplitFlag is equal to 1 and trafoDepth is equal to 0

  – interSplitFlag is equal to 1

– Otherwise, the value of split_transform_flag[x0][y0][trafoDepth] is inferred to be equal to 0.

cbf_luma[x0][y0][trafoDepth] 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. The array index trafoDepth specifies the current subdivision level of a coding block into blocks for the purpose of transform coding. trafoDepth is equal to 0 for blocks that correspond to coding blocks.

When cbf_luma[x0][y0][trafoDepth] is not present, it is inferred to be equal to 1.

cbf_cb[x0][y0][trafoDepth] 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 unit. The array index trafoDepth specifies the current subdivision level of a coding block into blocks for the purpose of transform coding. trafoDepth is equal to 0 for blocks that correspond to coding blocks.

When cbf_cb[x0][y0][trafoDepth] is not present, the value of cbf_cb[x0][y0][trafoDepth] is inferred as follows.

– If trafoDepth is greater than 0 and log2TrafoSize is equal to 2, cbf_cb[x0][y0][trafoDepth] is inferred to be equal to cbf_cb[xBase][yBase][trafoDepth − 1]

– Otherwise, cbf_cb[x0][y0][trafoDepth] is inferred to be equal to 0.

cbf_cr[x0][y0][trafoDepth] 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 unit. The array index trafoDepth specifies the current subdivision level of a coding block into blocks for the purpose of transform coding. trafoDepth is equal to 0 for blocks that correspond to coding blocks.

When cbf_cr[x0][y0][trafoDepth] is not present, the value of cbf_cr[x0][y0][trafoDepth] is inferred as follows.

– If trafoDepth is greater than 0 and log2TrafoSize is equal to 2, cbf_cr[x0][y0][trafoDepth] is inferred to be equal to cbf_cr[xBase][yBase][trafoDepth − 1]

– Otherwise, cbf_cr[x0][y0][trafoDepth] is inferred to be equal to 0.

7.4.9.9 Motion vector difference semantics

abs_mvd_greater0_flag[compIdx **]** specifies whether the absolute value of a motion vector component difference is greater than 0. The horizontal motion vector component difference is assigned compIdx = 0 and the vertical motion vector component is assigned compIdx = 1.

abs_mvd_greater1_flag[compIdx **]** specifies whether the absolute value of a motion vector component difference is greater than 1. The horizontal motion vector component difference is assigned compIdx = 0 and the vertical motion vector component is assigned compIdx = 1.

When abs_mvd_greater1_flag[compIdx] is not present, it is inferred to be equal to 0.

abs_mvd_minus2[compIdx **]** is the absolute value of a motion vector component difference minus 2. The horizontal motion vector component difference is assigned compIdx = 0 and the vertical motion vector component is assigned compIdx = 1.

When abs_mvd_minus2[compIdx] is not present, it is inferred as follows.

– If abs_mvd_greater1_flag[compIdx] is equal to 0, abs_mvd_minus2[compIdx] is inferred to be equal to −1.

– Otherwise (abs_mvd_greater1_flag[compIdx] is equal to 1), abs_mvd_minus2[compIdx] is inferred to be equal to 0.

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.

The horizontal motion vector component difference is assigned compIdx = 0 and the vertical motion vector component is assigned compIdx = 1.

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.

$$\begin{aligned} lMvd[compIdx] = {} & abs\_mvd\_greater0\_flag[compIdx] * \\ & (abs\_mvd\_minus2[compIdx] + 2) * \\ & (1 - 2 * mvd\_sign\_flag[compIdx]) \end{aligned} \tag{7-68}$$

The variable MvdLX[x0][y0][compIdx] with X being either 0 or 1, specifies the difference between a list X vector component to be used and its prediction. The value of MvdLX[x0][y0][compIdx] shall be in the range of -2^{15} to $2^{15} - 1$. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered prediction 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.

7.4.9.10 Transform unit semantics

The transform coefficient levels are parsed into 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 luma, equal to 1 for Cb, and equal to 2 for Cr. The array indices xC, 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 subclause 7.3.9.11 residual coding syntax, it is inferred to be equal to 0.

cu_qp_delta_abs specifies the absolute value of the difference between a luma quantization parameter for the coding unit containing cu_qp_delta_abs and its prediction.

cu_qp_delta_sign specifies the sign of a CuQpDelta as follows.

– If cu_qp_delta_sign is equal to 0, the corresponding CuQpDelta has a positive value.

– Otherwise (cu_qp_delta_sign is equal to 1), the corresponding CuQpDelta has a negative value.

When cu_qp_delta_sign is not present, it is inferred to be equal to 0.

When cu_qp_delta_abs is present, the variables IsCuQpDeltaCoded and CuQpDelta are derived as follows.

IsCuQpDeltaCoded = 1 (7-69)

CuQpDelta = cu_qp_delta_abs * (1 − 2 * cu_qp_delta_sign) (7-70)

The decoded value of CuQpDelta shall be in the range of −(26+ $QpBdOffset_Y$ / 2) to +(25+ $QpBdOffset_Y$ / 2), inclusive.

7.4.9.11 Residual coding semantics

For intra prediction, different scanning orders are used. The variable scanIdx specifies which scan order is used where scanIdx equal to 0 specifies an up-right diagonal scan order, scanIdx equal to 1 specifies a horizontal scan order, and scanIdx equal to 2 specifies a vertical scan order. The value of scanIdx is derived as follows.

– If CuPredMode[x0][y0] is equal to MODE_INTRA and one or more of the following conditions are true,

  – log2TrafoSize is equal to 2

  – log2TrafoSize is equal to 3 and cIdx is equal to 0

  predModeIntra is derived as follows.

  – If cIdx is equal to 0, predModeIntra is set equal to IntraPredModeY[x0][y0].

  – Otherwise, predModeIntra is set equal to IntraPredModeC.

  scanIdx is derived as follows.

  – If predModeIntra is in the range from 6 to 14, inclusive, scanIdx is set equal to 2.

  – Otherwise if predModeIntra is in the range from 22 to 30, inclusive, scanIdx is set equal to 1.

  – Otherwise, scanIdx is set equal to 0.

– Otherwise, scanIdx is 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 luma, equal to 1 for Cb, and equal to 2 for Cr. transform_skip_flag[x0][y0][cIdx] equal to 1, specifies that no transform will be applied to the current transform block. When transform_skip_flag[x0][y0][cIdx] is not present, it is inferred to be equal to 0.

last_significant_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_significant_coeff_x_prefix shall be in the range from 0 to (log2TrafoSize << 1) − 1, inclusive.

last_significant_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_significant_coeff_y_prefix shall be in the range from 0 to (log2TrafoSize << 1) − 1, inclusive.

last_significant_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_significant_coeff_x_suffix shall be in the range from 0 to (1 << ((last_significant_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_significant_coeff_x_suffix is not present, the following applies.

LastSignificantCoeffX = last_significant_coeff_x_prefix (7-71)

– Otherwise (last_significant_coeff_x_suffix is present), the following applies.

$$LastSignificantCoeffX = (1 << ((last\_significant\_coeff\_x\_prefix >> 1) - 1)) * (2 + (last\_significant\_coeff\_x\_prefix \& 1)) + last\_significant\_coeff\_x\_suffix \quad (7\text{-}72)$$

last_significant_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_significant_coeff_y_suffix shall be in the range from 0 to (1 << ((last_significant_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_significant_coeff_y_suffix is not present, the following applies.

$$LastSignificantCoeffY = last\_significant\_coeff\_y\_prefix \quad (7\text{-}73)$$

– Otherwise (last_significant_coeff_y_suffix is present), the following applies.

$$LastSignificantCoeffY = (1 << ((last\_significant\_coeff\_y\_prefix >> 1) - 1)) * (2 + (last\_significant\_coeff\_y\_prefix \& 1)) + last\_significant\_coeff\_y\_suffix \quad (7\text{-}74)$$

When scanIdx is equal to 2, the coordinates are swapped as follows.

$$(LastSignificantCoeffX, LastSignificantCoeffY) = Swap(LastSignificantCoeffX, LastSignificantCoeffY) \quad (7\text{-}75)$$

coded_sub_block_flag[xS][yS] specifies the following for the sub-block at location (xS, yS) within the current transform block, where a sub-block is a (4x4) array of 16 transform coefficient levels.

– If coded_sub_block_flag[xS][yS] is equal to 0, the 16 transform coefficient levels of the sub-block at location (xS, yS) are inferred to be equal to 0;

– Otherwise (coded_sub_block_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 significant_coeff_flag syntax elements is present for the sub-block at location (xS, yS) .

  – Otherwise, at least one of the 16 transform coefficient levels of the sub-block at location (xS, yS) has a non zero value.

When coded_sub_block_flag[xS][yS] is not present, it is inferred as follows.

– If one or more of the following conditions are true, coded_sub_block_flag[xS][yS] is inferred to be equal to 1.

  – (xS, yS) is equal to (0, 0)

  – (xS, yS) is equal to (LastSignificantCoeffX >> 2 , LastSignificantCoeffY >> 2)

– Otherwise, coded_sub_block_flag[xS][yS] is inferred to be equal to 0.

significant_coeff_flag[xC **][** yC **]** specifies for the transform coefficient position (xC, yC) within the current transform block whether the corresponding transform coefficient level at location (xC, yC) is non-zero as follows.

– If significant_coeff_flag[xC][yC] is equal to 0, the transform coefficient level at location (xC, yC) is set equal to 0.

– Otherwise (significant_coeff_flag[xC][yC] is equal to 1), the transform coefficient level at location (xC, yC) has a non-zero value.

When significant_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, significant_coeff_flag[xC][yC] is inferred to be equal to 1.

  – (xC & 3, yC & 3) is equal to (0, 0)

  – inferSbDcSigCoeffFlag is equal to 1

  – coded_sub_block_flag[xS][yS] is equal to 1

– Otherwise, significant_coeff_flag[xC][yC] is inferred to be equal to 0.

coeff_abs_level_greater1_flag[n **]** specifies for the scanning position n whether there are transform coefficient levels greater than 1.

When coeff_abs_level_greater1_flag[n] is not present, it is inferred to be equal to 0.

coeff_abs_level_greater2_flag[n **]** specifies for the scanning position n whether there are transform coefficient levels greater than 2.

When coeff_abs_level_greater2_flag[n] is not present, it is inferred to be equal to 0.

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.

coeff_abs_level_remaining[n **]** is the remaining absolute value of a transform coefficient level that is coded with Golomb-Rice code at the scanning position n. The value of coeff_abs_level_remaining[n] is constrained that the corresponding value of TransCoeffLevel[x0][y0][cIdx][xC][yC] is in the range from −32768 to 32767, inclusive. When coeff_abs_level_remaining [n] is not present, it is inferred as 0.

8 Decoding process

8.1 General decoding process

The input of this process is a bitstream and the output is a list of decoded pictures.

The set TargetDecLayerIdSet, which specifies the set of nuh_reserved_zero_6bits values of NAL units to be decoded, is specified as follows:

– If some external means not specified in this Specification is available to set TargetDecLayerIdSet, TargetDecLayerIdSet is set by the external means.

– Otherwise if the decoding process is invoked in a bitstream conformance test as specified in subclause C.1, TargetDecLayerIdSet is set as specified in subclause C.1.

– Otherwise, TargetDecLayerIdSet contains only one nuh_reserved_zero_6bits vlaue that is equal to 0.

The variable HighestTid, which identifies the highest temporal sub-layer to be decoded, is specified as follows:

– If some external means not specified in this Specification is available to set HighestTid, HighestTid is set by the external means.

– Otherwise if the decoding process is invoked in a bitstream conformance test as specified in subclause C.1, HighestTid is set as specified in subclause C.1.

– Otherwise, HighestTid is set to sps_max_sub_layers_minus1.

The sub-bitstream extraction process as specified in subclause 10.1 is applied with the bitstream, HighestTid and TargetDecLayerIdSet as inputs and the output is assigned to a bitstream referred to as BitstreamToDecode.

The following applies to each coded picture (referred to as the current picture,and denoted by the variable CurrPic) in BitstreamToDecode.

Depending on the value of chroma_format_idc, the number of sample arrays of the current picture is as follows:

– If chroma_format_idc is equal to 0, the current picture consists of 1 sample array S_L.

– Otherwise (chroma_format_idc is not equal to 0), the current picture consists of 3 sample arrays S_L, S_{Cb}, S_{Cr}.

The decoding process for the current picture takes as input the syntax elements and upper-case variables from clause 7. When interpreting the semantics of each syntax element in each NAL unit, the term "the bitstream" (or part thereof, e.g. a coded video sequence of the bitstream) refers to BitstreamToDecode (or part thereof).

The decoding process is specified such that all decoders will produce numerically identical cropped output pictures. Any decoding process that produces identical cropped 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.

When the current picture is a BLA picture that has nal_unit_type equal to BLA_W_LP or is a CRA picture, the following applies:

– If some external means not specified in this Specification is available to set the variable UseAltCpbParamsFlag to a value, UseAltCpbParamsFlag is set to the value provided by the external means.

– Otherwise, the value of UseAltCpbParamsFlag is set to 0.

When the current picture is a CRA picture, the following applies:

– If some external means not specified in this Specification is available to set the variable HandleCraAsBlaFlag to a value, HandleCraAsBlaFlag is set to the value provided by the external means.

– Otherwise, the value of HandleCraAsBlaFlag is set to 0.

When the current picture is a CRA picture and HandleCraAsBlaFlag is equal to 1, the value of no_output_of_prior_pics_flag is set to 1 and the following applies during the parsing and decoding processes for each coded slice segment NAL unit: [Ed. (GJS): Do not set a syntax element equal to a value. You can only do that with variables. Create a variable called something like NoOutputOfPriorPicsFlag and use that instead.]

– If UseAltCpbParamsFlag is equal to 0, the value of nal_unit_type is set to BLA_W_LP. [Ed. (GJS): Do not set a syntax element equal to a value. You can only do that with variables. Create a variable called something like BlaPicActionFlag and use that instead.]

– Otherwise, the value of nal_unit_type is set to BLA_W_DLP. [Ed. (GJS): Do not set a syntax element equal to a value. Create a variable called something like BlaPicActionFlag and use that instead.]

Each picture referred to in this clause is a complete coded picture.

Depending on the value of separate_colour_plane_flag, the decoding process is structured as follows:

– If separate_colour_plane_flag is equal to 0, the decoding process is invoked a single time with the current picture being the output.

– Otherwise (separate_colour_plane_flag is equal to 1), the decoding process is invoked three times. Inputs to the decoding process are all NAL units of the coded picture with identical value of colour_plane_id. The decoding process of NAL units with a particular value of colour_plane_id is specified as if only a coded video sequence with monochrome colour format with that particular value of colour_plane_id would be present in the bitstream. The output of each of the three decoding processes is assigned to the 3 sample arrays of the current picture with the NAL units with colour_plane_id equal to 0 being assigned to S_L, the NAL units with colour_plane_id equal to 1 being assigned to S_{Cb}, and the NAL units with colour_plane_id equal to 2 being assigned to S_{Cr}.

NOTE – The variable ChromaArrayType is derived as 0 when separate_colour_plane_flag is equal to 1 and chroma_format_idc is equal to 3. In the decoding process, the value of this variable is evaluated resulting in operations identical to that of monochrome pictures with chroma_format_idc being equal to 0.

The decoding process operates as follows for the current picture CurrPic:

1. The decoding of NAL units is specified in subclause 8.2.
2. The processes in subclause 8.3 specify decoding processes using syntax elements in the slice segment layer and above:
   - Variables and functions relating to picture order count are derived in subclause 8.3.1 (which only needs to be invoked for the first slice segment of a picture).
   - The decoding process for reference picture set in subclause 8.3.2 is invoked, wherein reference pictures may be marked as "unused for reference" or "used for long-term reference" (which only needs to be invoked for the first slice segment of a picture).
   - When the current picture is a BLA picture or is a CRA picture that is the first picture in the bitstream, the decoding process for generating unavailable reference pictures specified in subclause 8.3.3 is invoked (which only needs to be invoked for the first slice segment of a picture).
   - PicOutputFlag is set as follows:
     - If the current picture is a RASL picture and the previous RAP picture in decoding order is a BLA picture or is a CRA picture that is the first coded picture in the bitstream, PicOutputFlag is set equal to 0.
     - Otherwise, PicOutputFlag is set equal to pic_output_flag.

– At the beginning of the decoding process for each P or B slice, the decoding process for reference picture lists construction specified in subclause 8.3.4 is invoked for derivation of reference picture list 0 (RefPicList0), and when decoding a B slice, reference picture list 1 (RefPicList1).

– After all slices of the current picture have been decoded, the decoded picture is marked as "used for short-term reference".

3. The processes in subclauses 8.4, 8.5, 8.6, and 8.7 specify decoding processes using syntax elements in the coding tree unit layer and above.

8.2 NAL unit decoding process

Inputs to this process are NAL units.

Outputs of this process are the RBSP syntax structures encapsulated within the NAL units.

The decoding process for each NAL unit extracts the RBSP syntax structure from the NAL unit and then operates the decoding processes specified for the RBSP syntax structure in the NAL unit as follows.

Subclause 8.3 describes the decoding process for VCL NAL units.

NAL units with nal_unit_type equal to VPS_NUT, SPS_NUT, and PPS_NUT contain video parameter sets, sequence parameter sets, and picture parameter sets, respectively. Sequence parameter sets are used in the decoding processes of other NAL units as determined by reference to a sequence parameter set within the picture parameter sets or active parameter sets SEI messages. Picture parameter sets are used in the decoding processes of other NAL units as determined by reference to a picture parameter set within the slice segment headers.

8.3 Slice decoding process

8.3.1 Decoding process for picture order count

Output of this process is PicOrderCntVal, the picture order count of the current picture.

Picture order counts are used to identify pictures, for deriving motion parameters in merge mode and motion vector prediction, to represent picture order differences between pictures for motion vector derivation, and for decoder conformance checking (see subclause C.5).

Each coded picture is associated with one picture order count, denoted as PicOrderCntVal.

When none of the following conditions is true: [Ed. (GJS): Logical structure of sentence seems strange.]

– The current picture is an IDR

– The current picture is a BLA picture

– The current picture is a CRA picture and is the first coded picture in the bitstream

the variables prevPicOrderCntLsb and prevPicOrderCntMsb are derived as follows. Let prevTid0Pic be the previous reference picture in decoding order that has TemporalId equal to 0. The variable prevPicOrderCntLsb is set equal to pic_order_cnt_lsb of prevTid0Pic, and the variable prevPicOrderCntMsb is set equal to PicOrderCntMsb of prevTid0Pic.

The variable PicOrderCntMsb of the current picture is derived as follows.

– If the current picture is an IDR or a BLA picture, or if the first coded picture in the bitstream is a CRA picture and the current picture is the first coded picture in the bitstream, PicOrderCntMsb is set equal to 0.

– Otherwise, PicOrderCntMsb is derived as specified by the following pseudo-code:

```
if( ( pic_order_cnt_lsb <  prevPicOrderCntLsb )  &&
        ( ( prevPicOrderCntLsb − pic_order_cnt_lsb )  >=  ( MaxPicOrderCntLsb / 2 ) ) )
    PicOrderCntMsb = prevPicOrderCntMsb + MaxPicOrderCntLsb                    (8-1)
else if( (pic_order_cnt_lsb  >  prevPicOrderCntLsb )  &&
        ( (pic_order_cnt_lsb − prevPicOrderCntLsb )  >  ( MaxPicOrderCntLsb / 2 ) ) )
    PicOrderCntMsb = prevPicOrderCntMsb − MaxPicOrderCntLsb
else
    PicOrderCntMsb = prevPicOrderCntMsb
```

PicOrderCntVal is derived as

```
PicOrderCntVal = PicOrderCntMsb + pic_order_cnt_lsb                            (8-2)
```

NOTE 1 – All IDR pictures will have PicOrderCntVal equal to 0 since pic_order_cnt_lsb is inferred to be 0 for IDR pictures and prevPicOrderCntLsb and prevPicOrderCntMsb are both set equal to 0.

The value of PicOrderCntVal shall be in the range of -2^{31} to $2^{31} - 1$, inclusive. In one coded video sequence, the PicOrderCntVal values for any two coded pictures shall not be the same.

The function PicOrderCnt(picX) is specified as follows:

PicOrderCnt(picX) = PicOrderCntVal of the picture picX (8-3)

The function DiffPicOrderCnt(picA, picB) is specified as follows:

DiffPicOrderCnt(picA, picB) = PicOrderCnt(picA) − PicOrderCnt(picB) (8-4)

The bitstream shall not contain data that result in values of DiffPicOrderCnt(picA, picB) used in the decoding process that are not in the range of -2^{15} to $2^{15} - 1$, inclusive. [Ed. (GJS/YKW): Include the limit in all places where such a POC difference is used in the decoding process (i.e. TMVP).]

NOTE 2 – Let X be the current picture and Y and Z be two other pictures in the same sequence, Y and Z are considered to be in the same output order direction from X when both DiffPicOrderCnt(X, Y) and DiffPicOrderCnt(X, Z) are positive or both are negative.

NOTE 3 – Many encoders assign PicOrderCntVal proportional to the sampling time of the corresponding picture relative to the sampling time of the previous IDR or BLA picture.

8.3.2 Decoding process for reference picture set

This process is invoked once per picture, after decoding of a slice header but prior to the decoding of any coding unit and prior to the decoding process for reference picture list construction for the slice as specified in subclause 8.3.3. 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".

NOTE 1 – The reference picture set is an absolute description of the reference pictures used in the decoding process of the current and future coded pictures. The reference picture set signalling is explicit in the sense that all reference pictures included in the reference picture set are listed explicitly and there is no default reference picture set construction process in the decoder that depends on the status of the DPB. [Ed. (GJS): Search for line spacing exactly 9.95 pt and change to "single".]

A picture can be marked as "unused for reference", "used for short-term reference", or "used for long-term reference", but only one among these three. 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).

When the current picture is the first picture in the bitstream, the DPB is initialized to be an empty set of pictures.

When the current picture is an IDR picture or a BLA picture, all reference pictures currently in the DPB (if any) are marked as "unused for reference".

Short-term reference pictures are identified by their PicOrderCntVal values. Long-term reference pictures are identified either by their PicOrderCntVal values or their pic_order_cnt_lsb values.

Five lists of picture order count values are constructed to derive the reference picture set; PocStCurrBefore, PocStCurrAfter, PocStFoll, PocLtCurr, and PocLtFoll with NumPocStCurrBefore, NumPocStCurrAfter, NumPocStFoll, NumPocLtCurr, and NumPocLtFoll number of elements, respectively.

– If the current picture is an IDR picture, PocStCurrBefore, PocStCurrAfter, PocStFoll, PocLtCurr, and PocLtFoll are all set to empty, and NumPocStCurrBefore, NumPocStCurrAfter, NumPocStFoll, NumPocLtCurr, and NumPocLtFoll are all set to 0.

– Otherwise, the following applies for derivation of the five lists of picture order count values and the numbers of entries.

```
for( i = 0, j = 0, k = 0; i < NumNegativePics[ StRpsIdx ] ; i++ )
    if( UsedByCurrPicS0[ StRpsIdx ][ i ] )
        PocStCurrBefore[ j++ ] = PicOrderCntVal + DeltaPocS0[ StRpsIdx ][ i ]
    else
        PocStFoll[ k++ ] = PicOrderCntVal + DeltaPocS0[ StRpsIdx ][ i ]
NumPocStCurrBefore = j

for( i = 0, j = 0; i < NumPositivePics[ StRpsIdx ]; i++ )
    if( UsedByCurrPicS1[ StRpsIdx ][ i ] )
        PocStCurrAfter[ j++ ] = PicOrderCntVal + DeltaPocS1[ StRpsIdx ][ i ]
```

```
        else
            PocStFoll[ k++ ] = PicOrderCntVal + DeltaPocS1[ StRpsIdx ][ i ]
    NumPocStCurrAfter = j
    NumPocStFoll = k                                                                    (8-5)
    for( i = 0, j = 0, k = 0; i < num_long_term_sps + num_long_term_pics; i++ ) {
        pocLt = PocLsbLt[ i ]
        if( delta_poc_msb_present_flag[ i ] )
            pocLt += PicOrderCntVal − DeltaPocMSBCycleLt[ i ] * MaxPicOrderCntLsb − pic_order_cnt_lsb
        if( UsedByCurrPicLt[ i ] ) {
            PocLtCurr[ j ] = pocLt
            CurrDeltaPocMsbPresentFlag[ j++ ] = delta_poc_msb_present_flag[ i ]
        } else {
            PocLtFoll[ k ] = pocLt
            FollDeltaPocMsbPresentFlag[ k++ ] = delta_poc_msb_present_flag[ i ]
        }
    }
    NumPocLtCurr = j
    NumPocLtFoll = k
```

where PicOrderCntVal is the picture order count of the current picture as specified in subclause 8.3.1.

NOTE 2 – A value of StRpsIdx in the range from 0 to num_short_term_ref_pic_sets − 1, inclusive, indicates that a short-term reference picture set from the active sequence parameter set is being used, where StRpsIdx is the index of the short-term reference picture set to the list of short-term reference picture sets in the order in which they are signalled in the sequence parameter set. StRpsIdx equal to num_short_term_ref_pic_sets indicates that a short-term reference picture set explicitly signalled in the slice header is being used.

For each i in the range of 0 to NumPocLtCurr − 1, inclusive, when CurrDeltaPocMsbPresentFlag[i] is equal to 1, it is a requirement of bitstream conformance that the following conditions apply:

– There shall be no j in the range of 0 to NumPocStCurrBefore − 1, inclusive, for which PocLtCurr[i] is equal to PocStCurrBefore[j].

– There shall be no j in the range of 0 to NumPocStCurrAfter − 1, inclusive, for which PocLtCurr[i] is equal to PocStCurrAfter[j].

– There shall be no j in the range of 0 to NumPocStFoll − 1, inclusive, for which PocLtCurr[i] is equal to PocStFoll[j].

– There shall be no j in the range of 0 to NumPocLtCurr − 1, inclusive, where j is not equal to i, for which PocLtCurr[i] is equal to PocLtCurr[j].

For each i in the range of 0 to NumPocLtFoll − 1, inclusive, when FollDeltaPocMsbPresentFlag[i] is equal to 1, it is a requirement of bitstream conformance that the following conditions apply:

– There shall be no j in the range of 0 to NumPocStCurrBefore − 1, inclusive, for which PocLtFoll[i] is equal to PocStCurrBefore[j].

– There shall be no j in the range of 0 to NumPocStCurrAfter − 1, inclusive, for which PocLtFoll[i] is equal to PocStCurrAfter[j].

– There shall be no j in the range of 0 to NumPocStFoll − 1, inclusive, for which PocLtFoll[i] is equal to PocStFoll[j].

– There shall be no j in the range of 0 to NumPocLtFoll − 1, inclusive, where j is not equal to i, for which PocLtFoll[i] is equal to PocLtFoll[j].

– There shall be no j in the range of 0 to NumPocLtCurr − 1, inclusive, for which PocLtFoll[i] is equal to PocLtCurr[j].

For each i in the range of 0 to NumPocLtCurr − 1, inclusive, when CurrDeltaPocMsbPresentFlag[i] is equal to 0, it is a requirement of bitstream conformance that the following conditions apply:

– There shall be no j in the range of 0 to NumPocStCurrBefore − 1, inclusive, for which PocLtCurr[i] is equal to (PocStCurrBefore[j] & (MaxPicOrderCntLsb − 1)).

– There shall be no j in the range of 0 to NumPocStCurrAfter − 1, inclusive, for which PocLtCurr[i] is equal to (PocStCurrAfter[j] & (MaxPicOrderCntLsb − 1)).

– There shall be no j in the range of 0 to NumPocStFoll − 1, inclusive, for which PocLtCurr[i] is equal to (PocStFoll[j] & (MaxPicOrderCntLsb − 1)).

– There shall be no j in the range of 0 to NumPocLtCurr − 1, inclusive, where j is not equal to i, for which PocLtCurr[i] is equal to (PocLtCurr[j] & (MaxPicOrderCntLsb − 1)).

For each i in the range of 0 to NumPocLtFoll − 1, inclusive, when FollDeltaPocMsbPresentFlag[i] is equal to 0, it is a requirement of bitstream conformance that the following conditions apply:

– There shall be no j in the range of 0 to NumPocStCurrBefore − 1, inclusive, for which PocLtFoll[i] is equal to (PocStCurrBefore[j] & (MaxPicOrderCntLsb − 1)).

– There shall be no j in the range of 0 to NumPocStCurrAfter − 1, inclusive, for which PocLtFoll[i] is equal to (PocStCurrAfter[j] & (MaxPicOrderCntLsb − 1)).

– There shall be no j in the range of 0 to NumPocStFoll − 1, inclusive, for which PocLtFoll[i] is equal to (PocStFoll[j] & (MaxPicOrderCntLsb − 1)).

– There shall be no j in the range of 0 to NumPocLtFoll − 1, inclusive, where j is not equal to i, for which PocLtFoll[i] is equal to (PocLtFoll[j] & (MaxPicOrderCntLsb − 1)).

– There shall be no j in the range of 0 to NumPocLtCurr − 1, inclusive, for which PocLtFoll[i] is equal to (PocLtCurr[j] & (MaxPicOrderCntLsb − 1)).

The reference picture set consists of five lists of reference pictures; RefPicSetStCurrBefore, RefPicSetStCurrAfter, RefPicSetStFoll, RefPicSetLtCurr and RefPicSetLtFoll. The variable NumPocTotalCurr is derived in 7.4.8.3, reference picture list modification semantics.

It is a requirement of bitstream conformance that the following applies to the value of NumPocTotalCurr:

– If the current picture is a BLA or CRA picture, the value of NumPocTotalCurr shall be equal to 0.

– Otherwise, when the current picture contains a P or B slice, the value of NumPocTotalCurr shall not be equal to 0.

NOTE 3 – RefPicSetStCurrBefore, RefPicSetStCurrAfter and RefPicSetLtCurr contains all reference pictures that may be used in inter prediction of the current picture and that may be used in inter prediction of one or more of the pictures following the current picture in decoding order. RefPicSetStFoll and RefPicSetLtFoll consists of all reference pictures that are *not* used in inter prediction of the current picture but may be used in inter prediction of one or more of the pictures following the current picture in decoding order.

The derivation process for the reference picture set and picture marking are performed according to the following ordered steps, where DPB refers to the decoded picture buffer as described in Annex C:

1. The following applies:

```
for( i = 0; i < NumPocLtCurr; i++ )
    if( !CurrDeltaPocMsbPresentFlag[ i ] )
        if( there is a long-term reference picture picX in the DPB [Ed. (JB): Should be made more precise.
(GJS): Seems roughly OK to me.]
                with pic_order_cnt_lsb equal to PocLtCurr[ i ] )
            RefPicSetLtCurr[ i ] = picX
        else if( there is a short-term reference picture picY in the DPB
                with pic_order_cnt_lsb equal to PocLtCurr[ i ] )
            RefPicSetLtCurr[ i ] = picY
        else
            RefPicSetLtCurr[ i ] = "no reference picture"
    else
        if( there is a long-term reference picture picX in the DPB
                with PicOrderCntVal equal to PocLtCurr[ i ] )
            RefPicSetLtCurr[ i ] = picX
        else if( there is a short-term reference picture picY in the DPB
                with PicOrderCntVal equal to PocLtCurr[ i ] )
            RefPicSetLtCurr[ i ] = picY
        else
            RefPicSetLtCurr[ i ] = "no reference picture"
                                                                                    (8-6)
for( i = 0; i < NumPocLtFoll; i++ )
    if( !FollDeltaPocMsbPresentFlag[ i ] )
        if( there is a long-term reference picture picX in the DPB
                with pic_order_cnt_lsb equal to PocLtFoll[ i ] )
```

```
                RefPicSetLtFoll[ i ] = picX
            else if( there is a short-term reference picture picY in the DPB
                    with pic_order_cnt_lsb equal to PocLtFoll[ i ] )
                RefPicSetLtFoll[ i ] = picY
            else
                RefPicSetLtFoll[ i ] = "no reference picture"
        else
            if( there is a long-term reference picture picX in the DPB
                    with PicOrderCntVal to PocLtFoll[ i ] )
                RefPicSetLtFoll[ i ] = picX
            else if( there is a short-term reference picture picY in the DPB
                    with PicOrderCntVal equal to PocLtFoll[ i ] )
                RefPicSetLtFoll[ i ] = picY
            else
                RefPicSetLtFoll[ i ] = "no reference picture"
```

2. All reference pictures included in RefPicSetLtCurr and RefPicSetLtFoll are marked as "used for long-term reference".

3. The following applies:

```
for( i = 0; i < NumPocStCurrBefore; i++ )
    if( there is a short-term reference picture picX in the DPB
            with PicOrderCntVal equal to PocStCurrBefore[ i ])
        RefPicSetStCurrBefore[ i ] = picX
    else
        RefPicSetStCurrBefore[ i ] = "no reference picture"

for( i = 0; i < NumPocStCurrAfter; i++ )
    if( there is a short-term reference picture picX in the DPB
            with PicOrderCntVal equal to PocStCurrAfter[ i ])
        RefPicSetStCurrAfter[ i ] = picX
    else
        RefPicSetStCurrAfter[ i ] = "no reference picture"                    (8-7)

for( i = 0; i < NumPocStFoll; i++ )
    if( there is a short-term reference picture picX in the DPB
            with PicOrderCntVal equal to PocStFoll[ i ])
        RefPicSetStFoll[ i ] = picX
    else
        RefPicSetStFoll[ i ] = "no reference picture"
```

4. All reference pictures in the decoded picture buffer that are not included in RefPicSetLtCurr, RefPicSetLtFoll, RefPicSetStCurrBefore, RefPicSetStCurrAfter or RefPicSetStFoll are marked as "unused for reference".

NOTE 4 – There may be one or more reference pictures that are included in the reference picture set but not present in the decoded picture buffer. Entries in RefPicSetStFoll or RefPicSetLtFoll that are equal to "no reference picture" should be ignored. An unintentional picture loss should be inferred for each entry in RefPicSetStCurrBefore, RefPicSetStCurrAfter and RefPicSetLtCurr that is equal to "no reference picture".

It is a requirement of bitstream conformance that the reference picture set is restricted as follows:

– There shall be no entry in RefPicSetStCurrBefore, RefPicSetStCurrAfter or RefPicSetLtCurr for which one or more of the following are true.
  – The entry is equal to "no reference picture".
  – The entry is a picture that has nal_unit_type equal to TRAIL_N, TSA_N, STSA_N, RADL_N, RASL_N, RSV_VCL_N10, RSV_VCL_N12, or RSV_VCL_N14 and TemporalId equal to that of the current picture.
  – The entry is a picture that has TemporalId greater than that of the current picture.
– There shall no entry in RefPicSetLtCurr or RefPicSetLtFoll for which the difference between the picture order count value of the current picture and the picture order count value of the entry is greater than or equal to 2^{24}.
– When the current picture is a TSA picture, there shall be no picture included in the reference picture set with TemporalId greater than or equal to the TemporalId of the current picture.

– When the current picture is an STSA picture, there shall be no picture included in RefPicSetStCurrBefore, RefPicSetStCurrAfter or RefPicSetLtCurr that has TemporalId 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, there shall be no picture that has TemporalId equal to that of the current picture included in RefPicSetStCurrBefore, RefPicSetStCurrAfter or RefPicSetLtCurr that preceded the STSA picture in decoding order.

– When the current picture is a CRA picture, there shall be no picture included in the reference picture set that precedes, in decoding order, any preceding RAP picture in decoding order (when present).

– When the current picture is a trailing picture, there shall be no picture in RefPicSetStCurrBefore, RefPicSetStCurrAfter or RefPicSetLtCurr that was generated by the decoding process for generating unavailable reference pictures as specified in subclause 8.3.3.

– When the current picture is a trailing picture, there shall be no picture in the reference picture set that precedes the associated RAP picture in output order or decoding order.

– When the current picture is a RADL picture, there shall be no picture included in RefPicSetStCurrBefore, RefPicSetStCurrAfter or RefPicSetLtCurr that is any of the following types of pictures.

  – A RASL picture.

  – A picture that was generated by the decoding process for generating unavailable reference pictures as specified in subclause 8.3.3.

  – A picture that precedes the associated RAP picture in decoding order.

– When the sps_temporal_id_nesting_flag is equal to 1, the following applies. Let tIdA be the value of TemporalId of the current picture picA. Any picture picB with TemporalId equal to tIdB that is less than or equal to tIdA shall not be included in RefPicSetStCurrBefore, RefPicSetStCurrAfter and RefPicSetLtCurr of picA when there exists a picture picC with TemporalId equal to tIdC that is less than tIdB, which follows the picture picB in decoding order and precedes the picture picA in decoding order.

NOTE 5 – A picture cannot be included in more than one of the five reference picture set lists.

8.3.3 Decoding process for generating unavailable reference pictures

8.3.3.1 General decoding process for generating unavailable reference pictures

[Ed. (GJS): Scan the table of contents to look for any cases where headings do not appear there, as this is almost certainly due to having some style other than the ordinary heading style used on the headings of the missing sections.]

This process is invoked once per coded picture when the current picture is a BLA picture or is a CRA picture that is the first picture in the bitstream.

NOTE – This process is primarily specified only for the specification of syntax constraints for RASL pictures. The entire specification of the decoding process for RASL pictures associated with a CRA picture at the beginning of the bitstream or for RASL pictures associated with a BLA picture is only included herein for purposes of specifying constraints on the allowed syntax content of such RASL pictures. During the decoding process, any RASL pictures associated with a CRA picture at the beginning of the bitstream or any RASL pictures associated with a BLA picture may be ignored (removed from the bitstream and discarded), as these pictures are not specified for output and have no effect on the decoding process of any other pictures that are specified for output. However, in HRD operations as specified in Annex C, RASL access units may need to be taken into consideration in derivation of CPB arrival and removal times.

When this process is invoked, the following applies.

– For each RefPicSetStFoll[i], with i in the range of 0 to NumPocStFoll − 1, inclusive, that is equal to "no reference picture", a picture is generated as specified in subclause 8.3.3.2, and the following applies.

  – The value of PicOrderCntVal for the generated picture is set to PocStFoll[i].

  – The value of PicOutputFlag for the generated picture is set to 0.

  – The generated picture is marked as "used for short-term reference".

  – RefPicSetStFoll[i] is set to be the generated reference picture.

– For each RefPicSetLtFoll[i], with i in the range of 0 to NumPocLtFoll − 1, inclusive, that is equal to "no reference picture", a picture is generated as specified in subclause 8.3.3.2, and the following applies.

  – The value of PicOrderCntVal for the generated picture is set to PocLtFoll[i].

– The value of pic_order_cnt_lsb for the generated picture is inferred to be equal to (PocLtFoll[i] & (MaxPicOrderCntLsb – 1))

– The value of PicOutputFlag for the generated picture is set to 0.

– The generated picture is marked as "used for long-term reference".

– RefPicSetLtFoll[i] is set to the generated reference picture.

8.3.3.2 Generation of one unavailable picture

When this process is invoked, an unavailable picture is generated as follows:

– The value of each element in the sample array S_L for the picture is set to 1 << ($BitDepth_Y$ − 1).

– The value of each element in the sample arrays S_{Cb} and S_{Cr} for the picture is set to 1 << ($BitDepth_C$ − 1).

– The prediction mode CuPredMode[x][y] is set to MODE_INTRA for x = 0..pic_width_in_luma_samples − 1, y = 0..pic_height_in_luma_samples − 1.

8.3.4 Decoding process for reference picture lists construction

This process is invoked at the beginning of the decoding process for each P or B slice.

Reference pictures are addressed through reference indices as specified in subclause 8.5.3.2.1. A reference index is an index into a reference picture list. When decoding a P slice, there is a single reference picture list RefPicList0. When decoding a B slice, there is a second independent reference picture list RefPicList1 in addition to RefPicList0.

At the beginning of the decoding process for each slice, the reference picture list RefPicList0, and for B slices RefPicList1, are derived as follows.

The variable NumRpsCurrTempList0 is set equal to Max(num_ref_idx_l0_active_minus1 + 1, NumPocTotalCurr) and the list RefPicListTemp0 is constructed as follows:

```
rIdx = 0
while( rIdx < NumRpsCurrTempList0 ) {
    for( i = 0; i < NumPocStCurrBefore && rIdx < NumRpsCurrTempList0; rIdx++, i++ )
        RefPicListTemp0[ rIdx ] = RefPicSetStCurrBefore[ i ]
    for( i = 0;  i < NumPocStCurrAfter && rIdx < NumRpsCurrTempList0; rIdx++, i++ )          (8-8)
        RefPicListTemp0[ rIdx ] = RefPicSetStCurrAfter[ i ]
    for( i = 0; i < NumPocLtCurr && rIdx < NumRpsCurrTempList0; rIdx++, i++ )
        RefPicListTemp0[ rIdx ] = RefPicSetLtCurr[ i ]
}
```

The list RefPicList0 is constructed as follows:

```
for( rIdx = 0; rIdx <= num_ref_idx_l0_active_minus1; rIdx++)                              (8-9)
    RefPicList0[ rIdx ] = ref_pic_list_modification_flag_l0 ? RefPicListTemp0[ list_entry_l0[ rIdx ] ] :
                                                              RefPicListTemp0[ rIdx ]
```

When the slice is a B slice, the variable NumRpsCurrTempList1 is set equal to Max(num_ref_idx_l1_active_minus1 + 1, NumPocTotalCurr) and the list RefPicListTemp1 is constructed as follows:

```
rIdx = 0
while( rIdx < NumRpsCurrTempList1 ) {
    for( i = 0; i < NumPocStCurrAfter && rIdx < NumRpsCurrTempList1; rIdx++, i++ )
        RefPicListTemp1[ rIdx ] = RefPicSetStCurrAfter[ i ]
    for( i = 0;  i < NumPocStCurrBefore && rIdx < NumRpsCurrTempList1; rIdx++, i++ )         (8-10)
        RefPicListTemp1[ rIdx ] = RefPicSetStCurrBefore[ i ]
    for( i = 0; i < NumPocLtCurr && rIdx < NumRpsCurrTempList1; rIdx++, i++ )
        RefPicListTemp1[ rIdx ] = RefPicSetLtCurr[ i ]
}
```

When the slice is a B slice, the list RefPicList1 is constructed as follows:

```
for( rIdx = 0; rIdx <= num_ref_idx_l1_active_minus1; rIdx++)                              (8-11)
    RefPicList1[ rIdx ] = ref_pic_list_modification_flag_l1 ? RefPicListTemp1[ list_entry_l1[ rIdx ] ] :
                                                              RefPicListTemp1[ rIdx ]
```

8.4 Decoding process for coding units coded in intra prediction mode

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

Inputs to this process are:

– a luma location (xC, yC) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable log2CbSize specifying the size of the current luma coding block.

Output of this process is:

– a modified reconstructed picture before deblocking filtering.

The derivation process for quantization parameters as specified in subclause 8.6.1 is invoked with the luma location (xC, yC) as input.

A variable nS is set equal to (1 << log2CbSize).

Depending on pcm_flag[xC][yC] and IntraSplitFlag, the decoding process for luma samples is specified as follows.

– If pcm_flag[xC][yC] is equal to 1, the reconstucted picture is modified as follows:

$$S_L[xC + i][yC + j] = pcm\_sample\_luma[(nS * j) + i] << (BitDepth_Y - PCMBitDepth_Y), \text{ with } i, j = 0..nS-1 \quad (8\text{-}12)$$

– Otherwise (pcm_flag[xC][yC] is equal to 0), if IntraSplitFlag is equal to 0, the following ordered steps apply:

1. The derivation process for the intra prediction mode as specified in subclause 8.4.2 is invoked with the luma location (xC, yC) as the input.

2. The general decoding process for intra blocks as specified in subclause 8.4.4.1 is invoked with the luma location (xC, yC), the variable log2TrafoSize set equal to log2CbSize, the variable trafoDepth set equal to 0, the variable predModeIntra set equal to IntraPredModeY[xC][yC] and the variable cIdx set equal to 0 as the inputs and the output is a modified reconstructed picture before deblocking filtering.

– Otherwise (pcm_flag[xC][yC] is equal to 0 and IntraSplitFlag is equal to 1), for the variable blkIdx proceeding over the values 0..3, the following ordered steps apply:

1. The variable xBS is set equal to xC + (nS >> 1) * (blkIdx % 2).

2. The variable yBS is set equal to yC + (nS >> 1) * (blkIdx / 2).

3. The derivation process for the intra prediction mode as specified in subclause 8.4.2 is invoked with the luma location (xBS, yBS) as the input.

4. The general decoding process for intra blocks as specified in subclause 8.4.4.1 is invoked with the luma location (xBS, yBS), the variable log2TrafoSize set equal to log2CbSize − 1, the variable trafoDepth set equal to 1, the variable predModeIntra set equal to IntraPredModeY[xBS][yBS] and the variable cIdx set equal to 0 as the inputs and the output is a modified reconstructed picture before deblocking filtering.

Depending on pcm_flag[xC][yC], the decoding process for chroma samples is specified as follows:

– If pcm_flag[xC][yC] is equal to 1, the reconstucted picture is modified as follows:

$$S_{Cb}[xC/2 + i][yC/2 + j] = pcm\_sample\_chroma[(nS/2 * j) + i] << (BitDepth_C - PCMBitDepth_C) \text{ with } i, j = 0..nS/2-1 \quad (8\text{-}13)$$

$$S_{Cr}[xC/2 + i][yC/2 + j] = pcm\_sample\_chroma[(nS/2 * (j + nS/2)) + i] << (BitDepth_C - PCMBitDepth_C) \text{ with } i, j = 0..nS/2-1 \quad (8\text{-}14)$$

– Otherwise (pcm_flag[xC][yC] is equal to 0), the following ordered steps apply:

1. The derivation process for the chroma intra prediction mode as specified in 8.4.3 is invoked with the luma location (xC, yC) as input and the output is the variable IntraPredModeC.

2. The general decoding process for intra blocks as specified in subclause 8.4.4.1 is invoked with the chroma location (xC/2, yC/2), the variable log2TrafoSize set equal to log2CbSize − 1, the variable trafoDepth set equal to 0, the variable predModeIntra set equal to IntraPredModeC, and the variable cIdx set equal to 1 as the inputs and the output is a modified reconstructed picture before deblocking filtering.

3. The general decoding process for intra blocks as specified in subclause 8.4.4.1 is invoked with the chroma location (xC/2, yC/2), the variable log2TrafoSize set equal to log2CbSize − 1, the variable trafoDepth set equal to 0, the variable predModeIntra set equal to IntraPredModeC, and the variable cIdx set equal to 2 as the inputs and the output is a modified reconstructed picture before deblocking filtering.

8.4.2 Derivation process for luma intra prediction mode

Inputs to this process is a luma location (xB, yB) specifying the top-left luma sample of the current block relative to the top-left luma sample of the current picture.

Table 8-1 specifies the value for the intra prediction mode and the associated names.

Table 8-1 – Specification of intra prediction mode and associated names

Intra prediction mode	Associated names
0	Intra_Planar
1	Intra_DC
Otherwise (2..34)	Intra_Angular

IntraPredModeY[xB][yB] labelled 0..34 represents directions of predictions as illustrated in Figure 8-1.

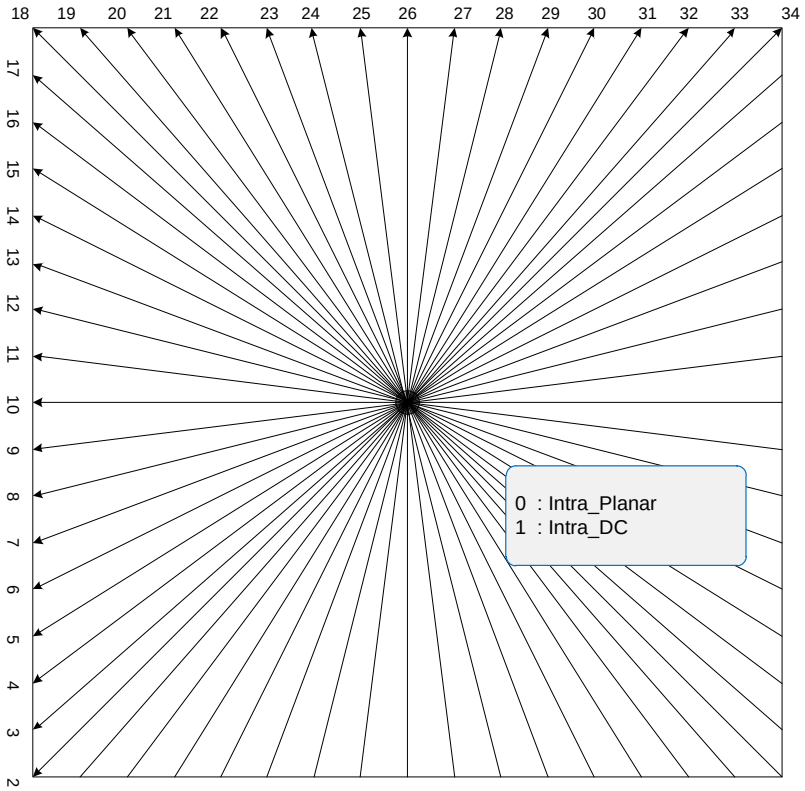


Figure 8-1 – Intra prediction mode directions (informative)

IntraPredModeY[xB][yB] is derived as the following ordered steps. [Ed. (WJ): proponent suggests to move this part to the syntax since the other syntax elements utilize IntraPredModeY. But it seems too complex to move all the following process to the syntax table. Maybe it's better to move this part to the semantics section or simply avoid the use of IntraPredModeY to parse the syntax item]

1. The neighbouring locations (xBA, yBA) and (xBB, yBB) are set equal to (xB−1, yB) and (xB, yB−1), respectively.

2. For N being either replaced A or B, the variables candIntraPredModeN are derived as follows.
   - The availability derivation process for a block in z-scan order as specified in subclause 6.4.1 is invoked with the location (xCurr, yCurr) set equal to (xB, yB) and the neighbouring location (xN, yN) set equal to (xBN, yBN) as the input and the output is assigned to availableN.
   - The candidate intra prediction mode candIntraPredModeN is derived as follows.
     - If availableN is equal to FALSE, candIntraPredModeN is set equal to Intra_DC.
     - Otherwise, if CuPredMode[xBN][yBN] is not equal to MODE_INTRA or pcm_flag[xBN][yBN] is equal to 1, candIntraPredModeN is set equal to Intra_DC,
     - Otherwise, if N is equal to B and yB−1 is less than ((yB >> Log2CtbSizeY) << Log2CtbSizeY), candIntraPredModeB is set equal to Intra_DC.
     - Otherwise, candIntraPredModeN is set equal to IntraPredModeY[xBN][yBN].
3. The candModeList[x] with x = 0..2 is derived as follows:
   - If candIntraPredModeB is equal to candIntraPredModeA, the following applies:
     - If candIntraPredModeA is less than 2 (either Intra_Planar or Intra_DC), candModeList[x] with x = 0..2 is derived as:

       candModeList[0] = Intra_Planar (8-15)
       candModeList[1] = Intra_DC (8-16)
       candModeList[2] = Intra_Angular (26) (8-17)

     - Otherwise, candModeList[x] with x = 0..2 is derived as:

       candModeList[0] = candIntraPredModeA (8-18)
       candModeList[1] = 2 + ((candIntraPredModeA + 29) % 32) (8-19)
       candModeList[2] = 2 + ((candIntraPredModeA − 2 + 1) % 32) (8-20)

   - Otherwise (candIntraPredModeB is not equal to candIntraPredModeA), the following applies:
     - candModeList[0] and candModeList[1] are derived as follows:

       candModeList[0] = candIntraPredModeA (8-21)
       candModeList[1] = candIntraPredModeB (8-22)

     - If none of candModeList[0] and candModeList[1] is equal to Intra_Planar, candModeList[2] is set equal to Intra_Planar,
     - Otherwise, if none of candModeList[0] and candModeList[1] is equal to Intra_DC, candModeList[2] is set equal to Intra_DC,
     - Otherwise, candModeList[2] is set equal to Intra_Angular (26).
4. IntraPredModeY[xB][yB] is derived by applying the following procedure:
   - If prev_intra_luma_pred_flag[xB][yB] is equal to 1, the IntraPredModeY[xB][yB] is set equal to candModeList[mpm_idx].
   - Otherwise IntraPredModeY[xB][yB] is derived by applying the following ordered steps:
     1) The array candModeList[x], x = 0..2 is modified as the following ordered steps:
        i. When candModeList[0] is greater than candModeList[1], both values are swapped as follows.

           (candModeList[0], candModeList[1]) = Swap(candModeList[0], candModeList[1]) (8-23)

        ii. When candModeList[0] is greater than candModeList[2], both values are swapped as follows.

           (candModeList[0], candModeList[2]) = Swap(candModeList[0], candModeList[2]) (8-24)

        iii. When candModeList[1] is greater than candModeList[2], both values are swapped as follows.

           (candModeList[1], candModeList[2]) = Swap(candModeList[1], candModeList[2]) (8-25)

2) IntraPredModeY[xB][yB] is derived as the following ordered steps:

i. IntraPredModeY[xB][yB] = rem_intra_luma_pred_mode[xB][yB]

ii. For i equal to 0 to 2, inclusive, when IntraPredModeY[xB][yB] is greater than or equal to candModeList[i], the value of IntraPredModeY[xB][yB] is incremented by one

8.4.3 Derivation process for chroma intra prediction mode

[Ed.: (WJ) this subclause may be moved to the semantics of intra_chroma_pred_mode syntax]

Input to this process is a luma location (xB, yB) specifying the top-left luma sample of the current block relative to the top-left luma sample of the current picture.

Output of this process is the variable IntraPredModeC.

The chroma intra prediction mode IntraPredModeC is derived using intra_chroma_pred_mode[xB][yB] and IntraPredModeY[xB][yB] as specified in Table 8-2.

Table 8-2 – Specification of IntraPredModeC

intra_chroma_pred_mode[xB][yB]	**IntraPredModeY[xB][yB]**				
	0	**26**	**10**	**1**	**X (0 <= X <= 34)**
0	34	0	0	0	0
1	26	34	26	26	26
2	10	10	34	10	10
3	1	1	1	34	1
4	0	26	10	1	X

8.4.4 Decoding process for intra blocks

8.4.4.1 General decoding process for intra blocks

Inputs to this process are:

– a sample location (xB0, yB0) specifying the top-left sample of the current block relative to the top-left sample of the current picture,

– a variable log2TrafoSize specifying the size of the current transform block,

– a variable trafoDepth specifying the hierarchy depth of the current block relative to the coding unit,

– a variable predModeIntra specifying the intra prediction mode,

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

Output of this process is:

– a modified reconstructed picture before deblocking filtering.

The variable splitFlag is derived as follows:

– If cIdx is equal to 0, splitFlag is set equal to split_transform_flag[xB0][yB0][trafoDepth].

– Otherwise, if all of the following conditions are true, splitFlag is set equal to 1.

  – cIdx is greater than 0

  – split_transform_flag[xB0 << 1][yB0 << 1][trafoDepth] is equal to 1

  – log2TrafoSize is greater than 2

– Otherwise, splitFlag is set equal to 0.

Depending on splitFlag, the following applies:

– If splitFlag is equal to 1, the following ordered steps apply:

1. The variables xB1 and yB1 are derived as follows.
   - The variable xB1 is set equal to xB0 + ((1 << log2TrafoSize) >> 1).
   - The variable yB1 is set equal to yB0 + ((1 << log2TrafoSize) >> 1).
2. The general decoding process for intra blocks as specified in this subclause is invoked with the location (xB0, yB0), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode predModeIntra, and the variable cIdx as the inputs and the output is a modified reconstructed picture before deblocking filtering.
3. The general decoding process for intra blocks as specified in this subclause is invoked with the location (xB1, yB0), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode predModeIntra, and the variable cIdx as the inputs and the output is a modified reconstructed picture before deblocking filtering.
4. The general decoding process for intra blocks as specified in this subclause is invoked with the location (xB0, yB1), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode predModeIntra, and the variable cIdx as the inputs and the output is a modified reconstructed picture before deblocking filtering.
5. The general decoding process for intra blocks as specified in this subclause is invoked with the location (xB1, yB1), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode predModeIntra, and the variable cIdx as the inputs and the output is a modified reconstructed picture before deblocking filtering.

– Otherwise (splitFlag is equal to 0), the following ordered steps apply:

1. The variable nT is set equal to 1 << log2TrafoSize.
2. The general intra sample prediction process as specified in subclause 8.4.4.2.1 is invoked with the location (xB0, yB0), the intra prediction mode predModeIntra, the transform block size nT and the variable cIdx as the inputs and the output is a (nT)x(nT) array predSamples.
3. The scaling and transformation process as specified in subclause 8.6.2 is invoked with the location (xB0, yB0), the variable trafoDepth, the variable cIdx, and the transform size trafoSize set equal to nT as the inputs and the output is a (nT)x(nT) array resSamples.
4. The picture reconstruction process prior to in-loop filtering for a colour component as specified in subclause 8.6.5 is invoked with the transform block location (xB0, yB0), the transform block size nT, the variable cIdx, the (nT)x(nT) array predSamples, and the (nT)x(nT) array resSamples as the inputs.

8.4.4.2 Intra sample prediction

8.4.4.2.1 General intra sample prediction

Inputs to this process are:

– a sample location (xB, yB) specifying the top-left sample of the current block relative to the top-left sample of the current picture,

– a variable predModeIntra specifying the intra prediction mode,

– a variable nT specifying the transform block size,

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

Output of this process is:

– the predicted samples predSamples[x][y], with x, y = 0..nT − 1.

The nT*4+1 neighbouring samples p[x][y] that are constructed samples prior to the deblocking filter process, with x = −1, y = −1..nT*2−1 and x = 0..nT*2 − 1, y = −1, are derived as follows.

– The luma location (xBN, yBN) is specified by

$$xBN = xB + x \tag{8-26}$$

$$yBN = yB + y \tag{8-27}$$

– The availability derivation process for a block in z-scan order as specified in subclause 6.4.1 is invoked with the location (xCurr, yCurr) set equal to (xB, yB) and the neighbouring location (xN, yN) set equal to (xBN, yBN) as the input and the output is assigned to availableN.

– Each sample p[x][y] is derived as follows

  – If one or more of the following conditions are true, the sample p[x][y] is marked as "not available for intra prediction"

    – the variable availableN is equal to FALSE

    – CuPredMode[xBN][yBN] is not equal to MODE_INTRA and constrained_intra_pred_flag is equal to 1

  – Otherwise, the sample p[x][y] is marked as "available for intra prediction" and the sample at the location (xBN, yBN) is assigned to p[x][y].

When at least one sample p[x][y] with x = −1, y = −1..nT*2−1 and x = 0..nT*2−1, y = −1 is marked as "not available for intra prediction", the reference sample substitution process for intra sample prediction in subclause 8.4.4.2.2 is invoked with the samples p[x][y] with x = −1, y = −1..nT*2−1 and x = 0..nT*2−1, y = −1 and nT as input and the modified samples p[x][y] with x = −1, y = −1..nT*2−1 and x = 0..nT*2−1, y = −1 as output.

Depending on predModeIntra, the following ordered steps apply:

1. When cIdx is equal to 0, filtering process of neighbouring samples specified in 8.4.4.2.3 is invoked with the sample array p and the transform block size nT as the inputs and the output is reassigned to the sample array p.

2. Intra sample prediction process according to predModeIntra applies as follows:

   – If predModeIntra is equal to Intra_Planar(0), the corresponding intra prediction mode specified in subclause 8.4.4.2.4 is invoked with the sample array p and the transform block size nT as the inputs and the output are the predicted sample array predSamples.

   – Otherwise, if predModeIntra is equal to Intra_DC(1), the corresponding intra prediction mode specified in subclause 8.4.4.2.5 is invoked with the sample array p, the transform block size nT and the colour component index cIdx as the inputs and the output are the predicted sample array predSamples.

   – Otherwise, if predModeIntra is equal to Intra_Angular(2..34), the corresponding intra prediction mode specified in subclause 8.4.4.2.6 is invoked with the intra prediction mode predModeIntra, the sample array p, the transform block size nT and the colour component index cIdx as the inputs and the output are the predicted sample array predSamples.

8.4.4.2.2 Reference sample substitution process for intra sample prediction

Inputs to this process are:

– reference samples p[x][y] with x = −1, y = −1..nT*2−1 and x = 0..nT*2−1, y = −1 for intra sample prediction,

– a transform block size nT.

Outputs of this process are:

– the modified reference samples p[x][y] with x = −1, y = −1..nT*2−1 and x = 0..nT*2−1, y = −1 for intra sample prediction. [Ed. (GJS): Global check/replace for incorrect use of ordinary hyphens ("-") versus non-breaking hyphens ("-") versus minus signs ("−") versus en-dashes ("–").]

The values of the samples p[x][y] with x = −1, y = −1..nT*2−1 and x = 0..nT*2−1, y = −1 are modified as follows:

– If all samples p[x][y] with x = −1, y = −1..nT*2−1 and x = 0..nT*2−1, y = −1 are marked as "not available for intra prediction," the value (1 << ($BitDepth_Y$ − 1)) is substituted for the values of all samples p[x][y].

– Otherwise (at least one but not all samples p[x][y] are marked as "not available for intra prediction"), the following ordered steps are performed:

  1. If p[−1][nT*2−1] is marked as "not available for intra prediction", searching sequentially starting from x = −1, y = nT*2−1 to x = −1, y = −1, then from x = 0, y = −1 to x = nT*2−1, y = −1. As soon as a sample p[x][y] marked as "available for intra prediction" is found, the search is terminated and the value of p[x][y] is assigned to p[−1][nT*2−1].

  2. For x = −1, y = nT*2−2...−1, if p[x][y] is marked as "not available for intra prediction", the value of p[x][y+1] is substituted for the value of p[x][y].

  3. For x = 0..nT*2−1, y = −1, if p[x][y] is marked as "not available for intra prediction", the value of p[x−1][y] is substituted for the value of p[x][y].

All samples p[x][y] with x = −1, y = −1..nT*2−1 and x = 0..nT*2−1, y = −1 are marked as "available for intra prediction".

8.4.4.2.3 Filtering process of neighbouring samples

Inputs to this process are:

– neighbouring samples p[x][y], with x = −1, y = −1..nT*2−1 and x = 0..nT*2 − 1, y = −1,

– a variable nT specifying the transform block size.

Output of this process are:

– filtered samples pF[x][y], with x = −1, y = −1..nT*2−1 and x = 0..nT*2 − 1, y = −1.

The variable filterFlag is derived as follows.

– If one or more of the following conditions are true, filterFlag is set equal to 0

  – predModeIntra is equal to Intra_DC

  – nT is equal 4

– Otherwise, the following applies.

  – The variable minDistVerHor is set equal to Min(Abs(predModeIntra − 26), Abs(predModeIntra − 10)).

  – The variable intraHorVerDistThres[nT] is specified in Table 8-3.

  – The variable filterFlag is derived as follows.

    – If minDistVerHor is larger than intraHorVerDistThres[nT], filterFlag is set equal to 1,

    – Otherwise, filterFlag is set equal to 0.

Table 8-3 – Specification of intraHorVerDistThres[nT] for various transform block sizes

	nT = 8	**nT = 16**	**nT = 32**
intraHorVerDistThres[nT]	7	1	0

When filterFlag is equal to 1, the following applies.

– The variable biIntFlag is derived as follows.

  – If all of the following conditions are true, biIntFlag is set equal to 1

    – strong_intra_smoothing_enable_flag is equal to 1

    – nT is equal to 32

    – Abs(p[−1][−1] + p[nT*2−1][−1] – 2*p[nT−1][−1]) < (1 << (BitDepthY − 5))

    – Abs(p[−1][−1] + p[−1][nT*2−1] – 2*p[−1][nT−1]) < (1 << (BitDepthY − 5))

  – Otherwise, biIntFlag is set equal to 0.

– The filtering is performed as follows.

  – If biIntFlag is equal to 1, the filtered sample values pF[x][y] with x = −1, y = −1..63 and x = 0..63, y = −1 are derived as follows.

$$pF[-1][63] = p[-1][63] \quad (8\text{-}28)$$

$$pF[63][-1] = p[63][-1] \quad (8\text{-}29)$$

$$pF[-1][y] = p[-1][-1] + ((y + 1)*(p[-1][63] - p[-1][-1]) + 32) >> 6 \text{ for } y = 0..62 \quad (8\text{-}30)$$

$$pF[-1][-1] = p[-1][-1] \quad (8\text{-}31)$$

$$pF[x][-1] = p[-1][-1] + ((x + 1)*(p[63][-1] - p[-1][-1]) + 32) >> 6 \text{ for } x = 0..62 \quad (8\text{-}32)$$

– Otherwise (biIntFlag is equal to 0), the filtered sample values pF[x][y] with x = −1, y = −1..nT*2−1 and x = 0..nT*2 − 1, y = −1 are derived as follows.

pF[−1][nT*2−1] = p[−1][nT*2−1] (8-33)

pF[nT*2−1][−1] = p[nT*2−1][−1] (8-34)

pF[−1][y] = (p[−1][y+1] + 2*p[−1][y] + p[−1][y−1] + 2) >> 2 for y = nT*2−2..0 (8-35)

pF[−1][−1] = (p[−1][0] + 2*p[−1][−1] + p[0][−1] + 2) >> 2 (8-36)

pF[x][−1] = (p[x−1][−1] + 2*p[x][−1] + p[x+1][−1] + 2) >> 2 for x = 0..nT*2−2 (8-37)

8.4.4.2.4 Specification of Intra_Planar (0) prediction mode

Inputs to this process are:

– neighbouring samples p[x][y], with x = −1, y = −1..nT*2−1 and x = 0..nT*2 − 1, y = −1,

– a variable nT specifying the transform block size.

Output of this process are:

– predicted samples predSamples[x][y], with x, y = 0..nT−1.

The values of the prediction samples predSamples[x][y], with x, y = 0..nT−1, are derived by

predSamples[x][y] = ((nT − 1 − x) * p[−1][y] + (x + 1) * p[nT][−1] +
(nT − 1 − y) * p[x][−1] + (y + 1) * p[−1][nT] + nT) >> (Log2(nT) + 1) (8-38)

8.4.4.2.5 Specification of Intra_DC (1) prediction mode

Inputs to this process are:

– neighbouring samples p[x][y], with x = −1, y = −1..nT*2−1 and x = 0..nT*2 − 1, y = −1,

– a variable nT specifying the transform block size,

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

Output of this process are:

– predicted samples predSamples[x][y], with x, y = 0..nT−1.

The values of the prediction samples predSamples[x][y], with x, y = 0..nT−1, are derived as the following ordered steps:

1. A variable dcVal is derived as:

$$\text{dcVal} = \left(\sum_{x'=0}^{nT-1} p[x'][-1] + \sum_{y'=0}^{nT-1} p[-1][y'] + nT \right) >> (k+1) \quad (8\text{-}39)$$

where k=Log2(nT)

2. Depending on the colour component index cIdx, the following applies.

– If cIdx is equal to 0 and nT is less than 32, the following applies.

predSamples[0][0] = (p[−1][0] + 2*dcVal + p[0][−1] + 2) >> 2 (8-40)
predSamples[x][0] = (p[x][−1] + 3*dcVal + 2) >> 2, with x = 1..nT−1 (8-41)
predSamples[0][y] = (p[−1][y] + 3*dcVal + 2) >> 2, with y = 1..nT−1 (8-42)
predSamples[x][y] = dcVal, with x, y = 1..nT−1 (8-43)

– Otherwise, the prediction samples predSamples[x][y] are derived as

predSamples[x][y] = dcVal, with x, y = 0..nT−1 (8-44)

8.4.4.2.6 Specification of Intra_Angular (2..34) prediction mode

Inputs to this process are:

– intra prediction mode predModeIntra,

– neighbouring samples p[x][y], with x = −1, y = −1..nT*2−1 and x = 0..nT*2 − 1, y = −1,

– a variable nT specifying the transform block size,

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

Output of this process are:

– predicted samples predSamples[x][y], with x, y = 0..nT−1.

Figure 8-2 illustrates the total 33 intra angles and Table 8-4 specifies the mapping table between predModeIntra and the angle parameter intraPredAngle.

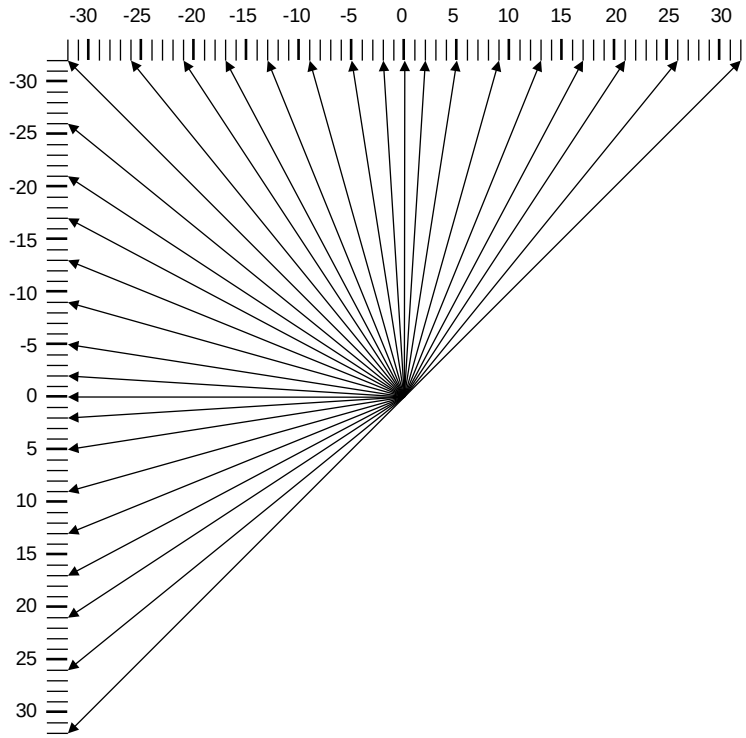


Figure 8-2 – Intra prediction angle definition (informative)

Table 8-4 – Specification of intraPredAngle

predModeIntra	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
intraPredAngle	-	32	26	21	17	13	9	5	2	0	−2	−5	−9	−13	−17	−21	−26
predModeIntra	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**
intraPredAngle	−32	−26	−21	−17	−13	−9	−5	−2	0	2	5	9	13	17	21	26	32

Table 8-5 further specifies the mapping table between predModeIntra and the inverse angle parameter invAngle.

Table 8-5 – Specification of invAngle

predModeIntra	11	12	13	14	15	16	17	18
invAngle	−4096	−1638	−910	−630	−482	−390	−315	−256
predModeIntra	19	20	21	22	23	24	25	26
invAngle	−315	−390	−482	−630	−910	−1638	−4096	-

The values of the prediction samples predSamples[x][y], with x, y = 0..nT−1 are derived as follows.

– If predModeIntra is equal or greater than 18, the following ordered steps apply.

1. The reference sample array ref[x], with x= −nT..2*nT is specified as follows.

ref[x] = p[−1+x][−1], with x = 0..nT (8-45)

– If intraPredAngle is less than 0, the main reference sample array is extended as follows.

– When (nT*intraPredAngle) >>5 is less than −1,

ref[x] = p[−1][−1+((x*invAngle+128)>>8)], with x=(nT*intraPredAngle) >>5..−1 (8-46)

– Otherwise,

ref[x] = p[−1+x][−1], with x=nT+1..2*nT (8-47)

2. The values of the prediction samples predSamples[x][y], with x, y = 0..nT−1 are derived as follows.

a. The index variable iIdx and the multiplication factor iFact are derived by

iIdx = ((y + 1)*intraPredAngle) >> 5 (8-48)

iFact = ((y + 1)*intraPredAngle) & 31 (8-49)

b. 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 by

predSamples[x][y] = ((32 − iFact)*ref[x+iIdx+1] + iFact*ref[x+iIdx+2] + 16) >> 5 (8-50)

– Otherwise, the value of the prediction samples predSamples[x][y] is derived by

predSamples[x][y] = ref[x+iIdx+1] (8-51)

c. When predModeIntra is equal to 26 (vertical), cIdx is equal to 0 and nT is less than 32, the following filtering applies with x = 0, y = 0..nT−1.

predSamples[x][y] = $Clip1_Y$(p[x][−1] + ((p[−1][y] − p[−1][−1]) >> 1)) (8-52)

– Otherwise (predModeIntra is less than 18), the following ordered steps apply.

1. The reference sample array ref[x], with x= −nT..2*nT is specified as follows.

ref[x] = p[−1][−1+x], with x = 0..nT (8-53)

– If intraPredAngle is less than 0, the main reference sample array is extended as follows.

– When (nT*intraPredAngle) >>5 is less than −1,

ref[x] = p[−1+((x*invAngle+128)>>8)][−1], with x=(nT*intraPredAngle) >>5..−1 (8-54)

– Otherwise,

ref[x] = p[−1][−1+x], with x=nT+1..2*nT (8-55)

2. The values of the prediction samples predSamples[x][y], with x, y = 0..nT−1 are derived as follows.

   a. The index variable iIdx and the multiplication factor iFact are derived by

   $$iIdx = ((x + 1)*intraPredAngle) >> 5 \quad (8\text{-}56)$$

   $$iFact = ((x + 1)*intraPredAngle) \,\&\, 31 \quad (8\text{-}57)$$

   b. 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 by

   $$predSamples[x][y] = ((32 - iFact)*ref[y+iIdx+1] + iFact*ref[y+iIdx+2] + 16) >> 5 \quad (8\text{-}58)$$

   – Otherwise, the value of the prediction samples predSamples[x][y] is derived by

   $$predSamples[x][y] = ref[y+iIdx+1] \quad (8\text{-}59)$$

   c. When predModeIntra is equal to 10 (horizontal), cIdx is equal to 0 and nT is less than 32, the following filtering applies with x = 0..nT−1, y = 0.

   $$predSamples[x][y] = Clip1_Y(p[-1][y] + ((p[x][-1] - p[-1][-1]) >> 1)) \quad (8\text{-}60)$$

8.5 Decoding process for coding units coded in inter prediction mode

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

[Ed. (GJS): Create similar subclauses for other places where there is "orphaned" text that is at the parent level of a subclause that contains subordinate subclauses.]

Inputs to this process are:

– a luma location (xC, yC) specifying the top-left sample of the current luma coding block relative to the top left luma sample of the current picture,

– a variable log2CbSize specifying the size of the current coding block.

Output of this process is a modified reconstructed picture before deblocking filtering.

The derivation process for quantization parameters as specified in subclause 8.6.1 is invoked with the luma location (xC, yC) as input.

The variable nCS_L is set equal to 1 << log2CbSize and the variable nCS_C is set equal to (1 << log2CbSize) >> 1.

Decoding process for coding units coded in inter prediction mode consists of following ordered steps:

1. The inter prediction process as specified in subclause 8.5.2 is invoked with the luma location (xC, yC), and the luma coding block size log2CbSize as the inputs and the outputs are three arrays $predSamples_L$, $predSamples_{Cb}$, $predSamples_{Cr}$.

2. The decoding process for the residual signal of coding units coded in inter prediction mode specified in subclause 8.5.4 is invoked with the luma location (xC, yC), luma coding block size log2CbSize as the inputs and the outputs are three arrays $resSamples_L$, $resSamples_{Cb}$, $resSamples_{Cr}$.

3. The reconstructed samples of the current coding unit are derived as follows.

   – The picture reconstruction process prior to in-loop filtering for a colour component as specified in subclause 8.6.5 is invoked with the luma coding block location (xC, yC), the variable nS set equal to nCS_L, the variable cIdx set equal to 0, the $(nCS_L)x(nCS_L)$ array predSamples set equal to $predSamples_L$, and the $(nCS_L)x(nCS_L)$ array resSamples set equal to $resSamples_L$ as the inputs.

   – The picture reconstruction process prior to in-loop filtering for a colour component as specified in subclause 8.6.5 is invoked with the chroma coding block location (xC/2, yC/2), the variable nS set equal to nCS_C, the variable cIdx set equal to 1, the $(nCS_C)x(nCS_C)$ array predSamples set equal to $predSamples_{Cb}$, and the $(nCS_C)x(nCS_C)$ array resSamples set equal to $resSamples_{Cb}$ as the inputs.

   – The picture reconstruction process prior to in-loop filtering for a colour component as specified in subclause 8.6.5 is invoked with the chroma coding block location (xC/2, yC/2), the variable nS set equal to

nCS_C, the variable cIdx set equal to 2, the (nCS_C)x(nCS_C) array predSamples set equal to $predSamples_{Cr}$, and the (nCS_C)x(nCS_C) array resSamples set equal to $resSamples_{Cr}$ as the inputs.

8.5.2 Inter prediction process

This process is invoked when decoding coding unit whose CuPredMode[xC][yC] is not equal to MODE_INTRA.

Inputs to this process are:

– a luma location (xC, yC) specifying the top-left sample of the current luma coding block relative to the top left luma sample of the current picture,

– a variable log2CbSize specifying the size of the current luma coding block,

Outputs of this process are:

– a (nCS_L)x(nCS_L) array $predSamples_L$ of luma prediction samples, where nCS_L is derived as specified below,

– a (nCS_C)x(nCS_C) array $predSamples_{Cb}$ of chroma prediction samples for the component Cb, where nCS_C is derived as specified below,

– a (nCS_C)x(nCS_C) array $predSamples_{Cr}$ of chroma prediction samples for the component Cr, where nCS_C is derived as specified below.

The variable nCS_L is set equal to 1 << log2CbSize and the variable nCS_C is set equal to (1 << log2CbSize) >> 1.

The variable $nCS1_L$ is set equal to nCS_L >> 1.

Depending on PartMode, the following applies:

– If PartMode is equal to PART_2Nx2N, the following ordered steps apply:

1. The decoding process for prediction units in inter prediction mode as specified in subclause 8.5.3 is invoked with the luma location (xC, yC), the luma location (xB, yB) set equal to (0, 0), the size of the luma coding block nCS_L, the width of the luma prediction block nPbW set equal to nCS_L, the height of the luma prediction block nPbH set equal to nCS_L and a partition index partIdx set equal to 0 as inputs, and the outputs are a (nCS_L)x(nCS_L) array $predSamples_L$ and two (nCS_C)x(nCS_C) arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

– Otherwise, if PartMode is equal to PART_2NxN, the following ordered steps apply:

1. The decoding process for prediction units in inter prediction mode as specified in subclause 8.5.3 is invoked with the luma location (xC, yC), the luma location (xB, yB) set equal to (0, 0), the size of the luma coding block nCS_L, the width of the luma prediction block nPbW set equal to nCS_L, the height of the luma prediction block nPbH set equal to $nCS1_L$ and a partition index partIdx set equal to 0 as inputs, and the outputs are a (nCS_L)x(nCS_L) array $predSamples_L$ and two (nCS_C)x(nCS_C) arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

2. The decoding process for prediction units in inter prediction mode as specified in subclause 8.5.3 is invoked with the luma location (xC, yC), the luma location (xB, yB) set equal to (0, $nCS1_L$), the size of the luma coding block nCS_L, the width of the luma prediction block nPbW set equal to nCS_L, the height of the luma prediction block nPbH set equal to $nCS1_L$ and a partition index partIdx set equal to 1 as inputs, and the outputs are the modfied (nCS_L)x(nCS_L) array $predSamples_L$ and the two modified (nCS_C)x(nCS_C) arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

– Otherwise, if PartMode is equal to PART_Nx2N, the following ordered steps apply:

1. The decoding process for prediction units in inter prediction mode as specified in subclause 8.5.3 is invoked with the luma location (xC, yC), the luma location (xB, yB) set equal to (0, 0), the size of the luma coding block nCS_L, the width of the luma prediction block nPbW set equal to $nCS1_L$, the height of the luma prediction block nPbH set equal to nCS_L and a partition index partIdx set equal to 0 as inputs, and the outputs are a (nCS_L)x(nCS_L) array $predSamples_L$ and two (nCS_C)x(nCS_C) arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

2. The decoding process for prediction units in inter prediction mode as specified in subclause 8.5.3 is invoked with the luma location (xC, yC), the luma location (xB, yB) set equal to ($nCS1_L$, 0), the size of the luma coding block nCS_L, the width of the luma prediction block nPbW set equal to $nCS1_L$, the height of the luma prediction block nPbH set equal to nCS_L and a partition index partIdx set equal to 1 as inputs, and the outputs are the modified (nCS_L)x(nCS_L) array $predSamples_L$ and the two modified (nCS_C)x(nCS_C) arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

– Otherwise, if PartMode is equal to PART_2NxnU, the following ordered steps apply:

1. The decoding process for prediction units in inter prediction mode as specified in subclause 8.5.3 is invoked with the luma location (xC, yC), the luma location (xB, yB) set equal to (0, 0), the size of the luma coding

block nCS_L, the width of the luma prediction block nPbW set equal to nCS_L, the height of the luma prediction block nPbH set equal to $nCS1_L >> 1$ and a partition index partIdx set equal to 0 as inputs, and the outputs are a $(nCS_L)x(nCS_L)$ array $predSamples_L$ and two $(nCS_C)x(nCS_C)$ arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

2. The decoding process for prediction units in inter prediction mode as specified in subclause 8.5.3 is invoked with the luma location (xC, yC), the luma location (xB, yB) set equal to (0, $nCS1_L >> 1$), the size of the luma coding block nCS_L, the width of the luma prediction block nPbW set equal to nCS_L, the height of the luma prediction block nPbH set equal to $nCS1_L + (nCS1_L >> 1)$ and a partition index partIdx set equal to 1 as inputs, and the outputs are the modified $(nCS_L)x(nCS_L)$ array $predSamples_L$ and the two modified $(nCS_C)x(nCS_C)$ arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

– Otherwise, if PartMode is equal to PART_2NxnD, the following ordered steps apply:

1. The decoding process for prediction units in inter prediction mode as specified in subclause 8.5.3 is invoked with the luma location (xC, yC), the luma location (xB, yB) set equal to (0, 0), the size of the luma coding block nCS_L, the width of the luma prediction block nPbW set equal to nCS_L, the height of the luma prediction block nPbH set equal to $nCS1_L + (nCS1_L >> 1)$ and a partition index partIdx set equal to 0 as inputs, and the outputs are a $(nCS_L)x(nCS_L)$ array $predSamples_L$ and two $(nCS_C)x(nCS_C)$ arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

2. The decoding process for prediction units in inter prediction mode as specified in subclause 8.5.3 is invoked with the luma location (xC, yC), the luma location (xB, yB) set equal to (0, $nCS1_L + (nCS1_L >> 1)$), the size of the luma coding block nCS_L, the width of the luma prediction block nPbW set equal to nCS_L, the height of the luma prediction block nPbH set equal to $nCS1_L >> 1$ and a partition index partIdx set equal to 1 as inputs, and the outputs are the modified $(nCS_L)x(nCS_L)$ array $predSamples_L$ and the two modified $(nCS_C)x(nCS_C)$ arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

– Otherwise, if PartMode is equal to PART_nLx2N, the following ordered steps apply:

1. The decoding process for prediction units in inter prediction mode as specified in subclause 8.5.3 is invoked with the luma location (xC, yC), the luma location (xB, yB) set equal to (0, 0), the size of the luma coding block nCS_L, the width of the luma prediction block nPbW set equal to $nCS1_L >> 1$, the height of the luma prediction block nPbH set equal to nCS_L and a partition index partIdx set equal to 0 as inputs, and the outputs are a $(nCS_L)x(nCS_L)$ array $predSamples_L$ and two $(nCS_C)x(nCS_C)$ arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

2. The decoding process for prediction units in inter prediction mode as specified in subclause 8.5.3 is invoked with the luma location (xC, yC), the luma location (xB, yB) set equal to ($nCS1_L >> 1$, 0), the size of the luma coding block nCS_L, the width of the luma prediction block nPbW set equal to $nCS1_L + (nCS1_L >> 1)$, the height of the luma prediction block nPbH set equal to nCS_L and a partition index partIdx set equal to 1 as inputs, and the outputs are the modified $(nCS_L)x(nCS_L)$ array $predSamples_L$ and the two modified $(nCS_C)x(nCS_C)$ arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

– Otherwise, if PartMode is equal to PART_nRx2N, the following ordered steps apply:

1. The decoding process for prediction units in inter prediction mode as specified in subclause 8.5.3 is invoked with the luma location (xC, yC), the luma location (xB, yB) set equal to (0, 0), the size of the luma coding block nCS_L, the width of the luma prediction block nPbW set equal to $nCS1_L + (nCS1_L >> 1)$, the height of the luma prediction block nPbH set equal to nCS_L and a partition index partIdx set equal to 0 as inputs, and the outputs are a $(nCS_L)x(nCS_L)$ array $predSamples_L$ and two $(nCS_C)x(nCS_C)$ arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

2. The decoding process for prediction units in inter prediction mode as specified in subclause 8.5.3 is invoked with the luma location (xC, yC), the luma location (xB, yB) set equal to ($nCS1_L + (nCS1_L >> 1)$, 0), the size of the luma coding block nCS_L, the width of the luma prediction block nPbW set equal to $nCS1_L >> 1$, the height of the luma prediction block nPbH set equal to nCS_L and a partition index partIdx set equal to 1 as inputs, and the outputs are the modified $(nCS_L)x(nCS_L)$ array $predSamples_L$ and the two modified $(nCS_C)x(nCS_C)$ arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

– Otherwise, if PartMode is equal to PART_NxN, the following ordered steps apply:

1. The decoding process for prediction units in inter prediction mode as specified in subclause 8.5.3 is invoked with the luma location (xC, yC), the luma location (xB, yB) set equal to (0, 0), the size of the luma coding block nCS_L, the width of the luma prediction block nPbW set equal to $nCS1_L$, the height of the luma prediction block nPbH set equal to $nCS1_L$ and a partition index partIdx set equal to 0 as inputs, and the outputs are a $(nCS_L)x(nCS_L)$ array $predSamples_L$ and two $(nCS_C)x(nCS_C)$ arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

2. The decoding process for prediction units in inter prediction mode as specified in subclause 8.5.3 is invoked with the luma location (xC, yC), the luma location (xB, yB) set equal to ($nCS1_L$, 0), the size of the luma coding block nCS_L, the width of the luma prediction block nPbW set equal to $nCS1_L$, the height of the luma

prediction block nPbH set equal to $nCS1_L$ and a partition index partIdx set equal to 1 as inputs, and the outputs are the modified $(nCS_L)x(nCS_L)$ array $predSamples_L$ and the two modified $(nCS_C)x(nCS_C)$ arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

3. The decoding process for prediction units in inter prediction mode as specified in subclause 8.5.3 is invoked with the luma location (xC, yC), the luma location (xB, yB) set equal to (0, $nCS1_L$), the size of the luma coding block nCS_L, the width of the luma prediction block nPbW set equal to $nCS1_L$, the height of the luma prediction block nPbH set equal to $nCS1_L$ and a partition index partIdx set equal to 2 as inputs, and the outputs are the modified $(nCS_L)x(nCS_L)$ array $predSamples_L$ and the two modified $(nCS_C)x(nCS_C)$ arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

4. The decoding process for prediction units in inter prediction mode as specified in subclause 8.5.3 is invoked with the luma location (xC, yC), the luma location (xB, yB) set equal to ($nCS1_L$, $nCS1_L$), the size of the luma coding block nCS_L, the width of the luma prediction block nPbW set equal to $nCS1_L$, the height of the luma prediction block nPbH set equal to $nCS1_L$ and a partition index partIdx set equal to 3 as inputs, and the outputs are the modified $(nCS_L)x(nCS_L)$ array $predSamples_L$ and the two modified $(nCS_C)x(nCS_C)$ arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

8.5.3 Decoding process for prediction units in inter prediction mode

Inputs to this process are:

– a luma location (xC, yC) specifying the top-left sample of the current luma coding block relative to the top left luma sample of the current picture,

– a luma location (xB, yB) specifying the top-left sample of the current luma prediction block relative to the top left sample of the current luma coding block,

– a variable nCS specifying the size of the current luma coding block,

– a variable nPbW specifying the width of the current luma prediction block,

– a variable nPbH specifying the width of the current luma prediction block,

– a variable partIdx specifying the index of the current prediction unit within the current coding unit.

Outputs of this process are:

– a $(nCS_L)x(nCS_L)$ array $predSamples_L$ of luma prediction samples, where nCS_L is derived as specified below,

– a $(nCS_C)x(nCS_C)$ array $predSamples_{Cb}$ of chroma prediction samples for the component Cb, where nCS_C is derived as specified below,

– a $(nCS_C)x(nCS_C)$ array $predSamples_{Cr}$ of chroma prediction samples for the component Cr, where nCS_C is derived as specified below.

The variable nCS_L is set equal to nCS and the variable nCS_C is set equal to nCS >> 1.

The decoding process for prediction units in inter prediction mode consists of the following ordered steps:

1. The derivation process for motion vector components and reference indices as specified in subclause 8.5.3.1 is invoked with the luma coding block location (xC, yC), the luma prediction block location (xB, yB), the luma coding block size block nCS, the luma prediction block width and height, nPbW and nPbH, and the prediction unit index partIdx specifying as inputs and the luma motion vectors mvL0 and mvL1, the chroma motion vectors mvCL0 and mvCL1, the reference indices refIdxL0 and refIdxL1, and the prediction list utilization flags predFlagL0 and predFlagL1 as outputs.

2. The decoding process for inter sample prediction as specified in subclause 8.5.3.2 is invoked with the luma coding block location (xC, yC), the luma prediction block location (xB, yB), the luma coding block size block nCS, the luma prediction block width height, nPbW and nPbH, the luma motion vectors mvL0 and mvL1, the chroma motion vectors mvCL0 and mvCL1, the reference indices refIdxL0 and refIdxL1, and the prediction list utilization flags predFlagL0 and predFlagL1 as inputs and the inter prediction samples (predSamples) which are a $(nCS_L)x(nCS_L)$ array $predSamples_L$ of prediction luma samples and two $(nCS_C)x(nCS_C)$ arrays $predSamples_{Cr}$, and $predSamples_{Cr}$ of prediction chroma samples, one for each of the chroma components Cb and Cr as outputs.

For use in derivation processes of variables invoked later in the decoding process, the following assignments are made for x = xB..xB + nPbW − 1, y = yB..yB + nPbH− 1:

MvL0[x][y] = mvL0 (8-61)
MvL1[x][y] = mvL1 (8-62)

RefIdxL0[x][y] = refIdxL0 (8-63)
RefIdxL1[x][y] = refIdxL1 (8-64)

PredFlagL0[x][y] = predFlagL0 (8-65)
PredFlagL1[x][y] = predFlagL1 (8-66)

8.5.3.1 Derivation process for motion vector components and reference indices

Inputs to this process are

– a luma location (xC, yC) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a luma location (xB, yB) of the top-left sample of the current luma prediction block relative to the top-left sample of the current luma coding block,

– a variable nCS specifying the size of the current luma coding block,

– variables specifying the width and the height of the luma prediction block, nPbW and nPbH,

– a variable partIdx specifying the index of the current prediction unit within the current coding unit.

Outputs of this process are

– luma motion vectors mvL0 and mvL1

– chroma motion vectors mvCL0 and mvCL1,

– reference indices refIdxL0 and refIdxL1,

– prediction list utilization flags predFlagL0 and predFlagL1.

Let (xP, yP) specify the top-left sample location of the current luma prediction block relative to the top-left luma sample of the current picture where xP = xC + xB and yP = yC + yB.

Let the variable currPic and ListX be the current picture and RefPicListX (with X being 0 or 1) of the current picture, respectively.

The function LongTermRefPic(nPic, nPb, refIdx, LX), with X being either 0 or 1, is defined as follows. If the picture with index refIdx from reference picture list LX of the slice containing prediction block nPb in the picture nPic was marked as "used for long term reference" at the time when nPic was the current picture, LongTermRefPic(nPic, nPb, refIdx, LX) is equal to 1; otherwise LongTermRefPic(nPic, nPb, refIdx, LX) is equal to 0.

For the derivation of the variables mvL0 and mvL1, refIdxL0 and refIdxL1 as well as predFlagL0 and predFlagL1, the following applies.

– If CuPredMode[xC][yC] is equal to MODE_SKIP, the derivation process for luma motion vectors for merge mode as specified in subclause 8.5.3.1.1 is invoked with the luma location (xC, yC), the luma location (xP, yP), variables nCS, nPbW, nPbH and the partition index partIdx as inputs and the output being the luma motion vectors mvL0, mvL1, the reference indices refIdxL0, refIdxL1, and the prediction list utilization flags predFlagL0 and predFlagL1.

– Otherwise, if CuPredMode[xC][yC] is equal to MODE_INTER and merge_flag[xP][yP] is equal to 1, the derivation process for luma motion vectors for merge mode as specified in subclause 8.5.3.1.1 is invoked with the luma location (xC, yC), luma location (xP, yP), variables nCS, nPbW and nPbH and the partition index partIdx as inputs and the outputs being the luma motion vectors mvL0 and mvL1, the reference indices refIdxL0 and refIdxL1, the prediction utilization flags predFlagL0 and predFlagL1.

– Otherwise, for X being replaced by either 0 or 1 in the variables predFlagLX, mvLX, refIdxLX and in Pred_LX and in the syntax elements ref_idx_lX and MvdLX, the following applies.

1. The variables refIdxLX and predFlagLX are derived as follows.

   - If inter_pred_idc[xP][yP] is equal to Pred_LX or Pred_BI,

     refIdxLX = ref_idx_lX[xP][yP] (8-67)
     predFlagLX = 1 (8-68)

   - Otherwise, the variables refIdxLX and predFlagLX are specified by

refIdxLX = −1 (8-69)
predFlagLX = 0 (8-70)

2. The variable mvdLX is derived as follows.

mvdLX[0] = MvdLX[xP][yP][0] (8-71)
mvdLX[1] = MvdLX[xP][yP][1] (8-72)

3. When predFlagLX is equal to 1, the derivation process for luma motion vector prediction in subclause 8.5.3.1.5 is invoked with the luma coding block location (xC, yC), the coding block size nCS, the luma prediction block location (xP, yP), variables nPbW and nPbH, refIdxLX, and the partition index partIdx as the inputs and the output being mvpLX.

4. When predFlagLX is equal to 1, the luma motion vector mvLX is derived as

uLX[0] = (mvpLX[0] + mvdLX[0] + 2^{16}) % 2^{16} (8-73)
mvLX[0] = (uLX[0] >= 2^{15}) ? (uLX[0] − 2^{16}) : uLX[0] (8-74)
uLX[1] = (mvpLX[1] + mvdLX[1] + 2^{16}) % 2^{16} (8-75)
mvLX[1] = (uLX[1] >= 2^{15}) ? (uLX[1] − 2^{16}) : uLX[1] (8-76)

NOTE – The resulting values of mvLX[0] and mvLX[1] as specified above will always be in the range of -2^{15} to $2^{15} - 1$.

When ChromaArrayType is not equal to 0 and predFlagLX (with X being either 0 or 1) is equal to 1, the derivation process for chroma motion vectors in subclause 8.5.3.1.9 is invoked with mvLX and refIdxLX as inputs and the output being mvCLX.

8.5.3.1.1 Derivation process for luma motion vectors for merge mode

This process is only invoked when CuPredMode[xC][yC] is equal to MODE_SKIP or CuPredMode[xC][yC] is equal to MODE_INTER and merge_flag [xP][yP] is equal to 1, where (xP, yP) specify the top-left sample of the current luma prediction block relative to the top-left luma sample of the current picture.

Inputs of this process are

– a luma location (xC, yC) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a luma location (xP, yP) of the top-left sample of the current luma prediction block relative to the top-left luma sample of the current picture,

– a variable nCS specifying the size of the current luma coding block,

– variables specifying the width and the height of the luma prediction block, nPbW and nPbH,

– a variable partIdx specifying the index of the current prediction unit within the current coding unit.

Outputs of this process are

– the luma motion vectors mvL0 and mvL1,

– the reference indices refIdxL0 and refIdxL1,

– the prediction list utilization flags predFlagL0 and predFlagL1.

The variables singleMCLFlag is derived as follows.

– If log2_parallel_merge_level_minus2 is greater than 0 and nCS is equal to 8, singleMCLFlag is set to 1.

– Otherwise, singleMCLFlag is set to 0.

When singleMCLFlag is equal to 1, xP is set equal to xC, yP is set equal to yC, and both nPbW and nPbH are set equal to nCS.

NOTE – When singleMCLFlag is equal to 1, all the prediction units of the current coding unit share a single merge candidate list, which is identical to the merge candidate list of the 2Nx2N prediction unit.

The motion vectors mvL0 and mvL1, the reference indices refIdxL0 and refIdxL1, and the prediction utilization flags predFlagL0 and predFlagL1 are derived as specified by the following ordered steps:

1. The derivation process for merging candidates from neighboring prediction unit partitions in subclause 8.5.3.1.2 is invoked with the luma coding block location (xC, yC), the coding block size nCS, the luma prediction block location (xP, yP), the variable singleMCLFlag, the width and the height of the luma prediction block nPbW

and nPbH and the partition index partIdx as inputs and the output being the availability flags availableFlagA_0, availableFlagA_1, availableFlagB_0, availableFlagB_1, availableFlagB_2, the reference indices refIdxLXA_0, refIdxLXA_1, refIdxLXB_0, refIdxLXB_1, refIdxLXB_2, the prediction list utilization flags predFlagLXA_0, predFlagLXA_1, predFlagLXB_0, predFlagLXB_1, predFlagLXB_2 and the motion vectors mvLXA_0, mvLXA_1, mvLXB_0, mvLXB_1, mvLXB_2 (with X being 0 or 1, respectively).

2. The reference indices for the temporal merging candidate, refIdxLXCol (with X being 0 or 1, respectively), are set equal to 0.

3. The derivation process for temporal luma motion vector prediction in subclause 8.5.3.1.7 is invoked with luma location (xP, yP), the width and the height of the luma prediction block nPbW and nPbH, and refIdxLXCol as the inputs and the output being the availability flags availableFlagLXCol and the temporal motion vectors mvLXCol (with X being 0 or 1, respectively). The variables availableFlagCol and predFlagLXCol (with X being 0 or 1, respectively) are derived as specified below.

   availableFlagCol = availableFlagL0Col | | availableFlagL1Col (8-77)

   predFlagLXCol = availableFlagLXCol (8-78)

4. The merging candidate list, mergeCandList, is constructed as follows.

   1. A_1, if availableFlagA_1 is equal to 1
   2. B_1, if availableFlagB_1 is equal to 1
   3. B_0, if availableFlagB_0 is equal to 1
   4. A_0, if availableFlagA_0 is equal to 1
   5. B_2, if availableFlagB_2 is equal to 1
   6. Col, if availableFlagCol is equal to 1

5. The variable numMergeCand and numOrigMergeCand are set to the number of merging candidates in the mergeCandList.

6. When slice_type is equal to B, the derivation process for combined bi-predictive merging candidates specified in subclause 8.5.3.1.3 is invoked with 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 being in mergeCandList, numMergeCand and numOrigMergeCand given as input and the output is assigned to mergeCandList, numMergeCand, the reference indices refIdxL0combCand$_k$ and refIdxL1combCand$_k$, the prediction list utilization flags predFlagL0combCand$_k$ and predFlagL1combCand$_k$ and the motion vectors mvL0combCand$_k$ and mvL1combCand$_k$ of every new candidate combCand$_k$ being added in mergeCandList. The number of candidates being added numCombMergeCand is set equal to (numMergeCand − numOrigMergeCand). When numCombMergeCand is greater than 0, k ranges from 0 to numCombMergeCand − 1, inclusive.

7. The derivation process for zero motion vector merging candidates specified in subclause 8.5.3.1.4 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 being in mergeCandList and the NumMergeCand as the inputs and the output is assigned to mergeCandList, numMergeCand, the reference indices refIdxL0zeroCand$_m$ and refIdxL1zeroCand$_m$, the prediction list utilization flags predFlagL0zeroCand$_m$ and predFlagL1zeroCand$_m$, the motion vectors mvL0zeroCand$_m$ and mvL1zeroCand$_m$ of every new candidate zeroCand$_m$ being added in mergeCandList. The number of candidates being added numZeroMergeCand is set equal to (numMergeCand − numOrigMergeCand − numCombMergeCand). When numZeroMergeCand is greater than 0, m ranges from 0 to numZeroMergeCand − 1, inclusive.

8. The following assignments are made with N being the candidate at position merge_idx[xP][yP] in the merging candidate list mergeCandList (N = mergeCandList[merge_idx[xP][yP]]) and X being replaced by 0 or 1:

   mvLX[0] = mvLXN[0] (8-79)

   mvLX[1] = mvLXN[1] (8-80)

   refIdxLX = refIdxLXN (8-81)

   predFlagLX = predFlagLXN (8-82)

9. When predFlagL0 is equal to 1 and predFlagL1 is equal to 1, and (nPbW + nPbH) is equal to 12, the following applies.

$$refIdxL1 = -1 \quad (8\text{-}83)$$

$$predFlagL1 = 0 \quad (8\text{-}84)$$

8.5.3.1.2 Derivation process for spatial merging candidates

Inputs to this process are

- a luma location (xC, yC) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
- a variable nCS specifying the size of the current luma coding block,
- a luma location (xP, yP) specifying the top-left sample of the current luma prediction block relative to the top-left luma sample of the current picture,
- a variable singleMCLFlag,
- variables specifying the width and the height of the luma prediction block, nPbW and nPbH,
- a variable partIdx specifying the index of the current prediction unit within the current coding unit.

Outputs of this process are (with X being 0 or 1, respectively)

- the availability flags $availableFlagA_0$, $availableFlagA_1$, $availableFlagB_0$, $availableFlagB_1$, $availableFlagB_2$ of the neighbouring prediction units,
- the reference indices $refIdxLXA_0$, $refIdxLXA_1$, $refIdxLXB_0$, $refIdxLXB_1$, $refIdxLXB_2$ of the neighbouring prediction units,
- the prediction list utilization flags $predFlagLXA_0$, $predFlagLXA_1$, $predFlagLXB_0$, $predFlagLXB_1$, $predFlagLXB_2$ of the neighbouring prediction units,
- the motion vectors $mvLXA_0$, $mvLXA_1$, $mvLXB_0$, $mvLXB_1$, $mvLXB_2$ of the neighbouring prediction units.

For the derivation of $availableFlagA_1$, $refIdxLXA_1$, $predFlagLXA_1$, and $mvLXA_1$ the following applies.

- The luma location (xA_1, yA_1) inside the neighbouring luma coding block is set equal to (xP − 1, yP + nPbH − 1).
- The availability derivation process for a prediction block as specified in subclause 6.4.2 is invoked with the luma location (xC, yC), the current luma coding block size nCbS set equal to nCS, the luma location (xP, yP), the width and the height of the luma prediction block nPbW and nPbH, the luma location (xA_1, yA_1) and the partition index partIdx as inputs and the output is assigned to the prediction block availability flag $availableA_1$.
- When one or more of the following conditions are true, $availableA_1$ is set equal to FALSE
  - (xP >> (log2_parallel_merge_level_minus2 + 2)) is equal to (xA_1 >> (log2_parallel_merge_level_minus2 + 2)) and (yP >> (log2_parallel_merge_level_minus2 + 2)) is equal to (yA_1 >> (log2_parallel_merge_level_minus2 + 2)).
  - singleMCLFlag is equal to 0 and PartMode of the current prediction unit is PART_Nx2N or PART_nLx2N or PART_nRx2N and partIdx is equal to 1
- The variables $availableFlagA_1$, $refIdxLXA_1$, $predFlagLXA_1$, and $mvLXA_1$ are derived as follows.
  - If $availableA_1$ is equal to FALSE, $availableFlagA_1$ is set equal to 0, both components $mvLXA_1$ are set equal to 0, $refIdxLXA_1$ is set equal to −1 and $predFlagLXA_1$ is set equal to 0 (with X being 0 or 1, respectively).
  - Otherwise, $availableFlagA_1$ is set equal to 1 and the following assignments are made.

$$mvLXA_1 = MvLX[\ xA_1\ ][\ yA_1\ ] \quad (8\text{-}85)$$
$$refIdxLXA_1 = RefIdxLX[\ xA_1\ ][\ yA_1\ ] \quad (8\text{-}86)$$
$$predFlagLXA_1 = PredFlagLX[\ xA_1\ ][\ yA_1\ ] \quad (8\text{-}87)$$

For the derivation of $availableFlagB_1$, $refIdxLXB_1$, $predFlagLXB_1$, and $mvLXB_1$ the following applies.

- The luma location (xB_1, yB_1) inside the neighbouring luma coding block is set equal to (xP + nPbW − 1, yP − 1).
- The availability derivation process for a prediction block as specified in subclause 6.4.2 is invoked with the luma location (xC, yC), the current luma coding block size nCbS set equal to nCS, the luma location (xP, yP), the width and the height of the luma prediction block nPbW and nPbH, the luma location (xB_1, yB_1) and the partition index partIdx as inputs and the output is assigned to the prediction block availability flag $availableB_1$.

– When one or more of the following conditions are true, $availableB_1$ is set equal to FALSE

  – (xP >> (log2_parallel_merge_level_minus2 + 2)) is equal to (xB_1 >> (log2_parallel_merge_level_minus2 + 2)) and (yP >> (log2_parallel_merge_level_minus2 + 2)) is equal to (yB_1 >> (log2_parallel_merge_level_minus2 + 2)).

  – singleMCLFlag is equal to 0 and PartMode of the current prediction unit is PART_2NxN or PART_2NxnU or PART_2NxnD and partIdx is equal to 1

– The variables $availableFlagB_1$, $refIdxLXB_1$, $predFlagLXB_1$, and $mvLXB_1$ are derived as follows.

  – If one or more of the following conditions are true, $availableFlagB_1$ is set equal to 0, both components $mvLXB_1$ are set equal to 0, $refIdxLXB_1$ is set equal to −1 and $predFlagLXB_1$ is set equal to 0 (with X being 0 or 1, respectively).

    – $availableB_1$ is equal to FALSE

    – $availableA_1$ is equal to TRUE and the prediction units covering luma location (xA_1, yA_1) and luma location (xB_1, yB_1) have the same motion vectors and the same reference indices

  – Otherwise, $availableFlagB_1$ is set equal to 1 and the following assignments are made.

$$mvLXB_1 = MvLX[\ xB_1\ ][\ yB_1\ ] \quad (8\text{-}88)$$
$$refIdxLXB_1 = RefIdxLX[\ xB_1\ ][\ yB_1\ ] \quad (8\text{-}89)$$
$$predFlagLXB_1 = PredFlagLX[\ xB_1\ ][\ yB_1\ ] \quad (8\text{-}90)$$

For the derivation of $availableFlagB_0$, $refIdxLXB_0$, $predFlagLXB_0$, and $mvLXB_0$ the following applies.

– The luma location (xB_0, yB_0) inside the neighbouring luma coding block is set equal to (xP + nPbW, yP − 1).

– The availability derivation process for a prediction block as specified in subclause 6.4.2 is invoked with the luma location (xC, yC), the current luma coding block size nCbS set equal to nCS, the luma location (xP, yP), the width and the height of the luma prediction block nPbW and nPbH, the luma location (xB_0, yB_0) and the partition index partIdx as inputs and the output is assigned to the prediction block availability flag $availableB_0$.

– When (xP >> (log2_parallel_merge_level_minus2 + 2)) is equal to (xB_0 >> (log2_parallel_merge_level_minus2 + 2)) and (yP >> (log2_parallel_merge_level_minus2 + 2)) is equal to (yB_0 >> (log2_parallel_merge_level_minus2 + 2)), $availableB_0$ is set equal to FALSE

– The variables $availableFlagB_0$, $refIdxLXB_0$, $predFlagLXB_0$, and $mvLXB_0$ are derived as follows.

  – If one or more of the following conditions are true, $availableFlagB_0$ is set equal to 0, both components $mvLXB_0$ are set equal to 0, $refIdxLXB_0$ is set equal to −1 and $predFlagLXB_0$ is set equal to 0 (with X being 0 or 1, respectively).

    – $availableB_0$ is equal to FALSE

    – $availableB_1$ is equal to TRUE and the prediction units covering luma location (xB_1, yB_1) and luma location (xB_0, yB_0) have the same motion vectors and the same reference indices

  – Otherwise, $availableFlagB_0$ is set equal to 1 and the following assignments are made.

$$mvLXB_0 = MvLX[\ xB_0\ ][\ yB_0\ ] \quad (8\text{-}91)$$
$$refIdxLXB_0 = RefIdxLX[\ xB_0\ ][\ yB_0\ ] \quad (8\text{-}92)$$
$$predFlagLXB_0 = PredFlagLX[\ xB_0\ ][\ yB_0\ ] \quad (8\text{-}93)$$

For the derivation of $availableFlagA_0$, $refIdxLXA_0$, $predFlagLXA_0$, and $mvLXA_0$ the following applies.

– The luma location (xA_0, yA_0) inside the neighbouring luma coding block is set equal to (xP − 1, yP + nPbH).

– The availability derivation process for a prediction block as specified in subclause 6.4.2 is invoked with the luma location (xC, yC), the current luma coding block size nCbS set equal to nCS, the luma location (xP, yP), the width and the height of the luma prediction block nPbW and nPbH, the luma location (xA_0, yA_0) and the partition index partIdx as inputs and the output is assigned to the prediction block availability flag $availableA_0$.

– When (xP >> (log2_parallel_merge_level_minus2 + 2)) is equal to (xA_0 >> (log2_parallel_merge_level_minus2 + 2)) and (yP >> (log2_parallel_merge_level_minus2 + 2)) is equal to (yA_0 >> (log2_parallel_merge_level_minus2 + 2)), $availableA_0$ is set equal to FALSE

– The variables $availableFlagA_0$, $refIdxLXA_0$, $predFlagLXA_0$, and $mvLXA_0$ are derived as follows.

– If one or more of the following conditions are true, $availableFlagA_0$ is set equal to 0, both components $mvLXA_0$ are set equal to 0, $refIdxLXA_0$ is set equal to −1 and $predFlagLXA_0$ is set equal to 0 (with X being 0 or 1, respectively).
  – $availableA_0$ is equal to FALSE
  – $availableA_1$ is equal to TRUE and the prediction units covering luma location (xA_1, yA_1) and luma location (xA_0, yA_0) have the same motion vectors and the same reference indices
– Otherwise, $availableFlagA_0$ is set equal to 1 and the following assignments are made.

$$mvLXA_0 = MvLX[\ xA_0\ ][\ yA_0\ ] \quad (8\text{-}94)$$
$$refIdxLXA_0 = RefIdxLX[\ xA_0\ ][\ yA_0\ ] \quad (8\text{-}95)$$
$$predFlagLXA_0 = PredFlagLX[\ xA_0\ ][\ yA_0\ ] \quad (8\text{-}96)$$

For the derivation of $availableFlagB_2$, $refIdxLXB_2$, $predFlagLXB_2$, and $mvLXB_2$ the following applies.

– The luma location (xB_2, yB_2) inside the neighbouring luma coding block is set equal to (xP − 1, yP − 1).
– The availability derivation process for a prediction block as specified in subclause 6.4.2 is invoked with the luma location (xC, yC), the current luma coding block size nCbS set equal to nCS, the luma location (xP, yP), the width and the height of the luma prediction block nPbW and nPbH, the luma location (xB_2, yB_2) and the partition index partIdx as inputs and the output is assigned to the prediction block availability flag $availableB_2$.
– When (xP >> (log2_parallel_merge_level_minus2 + 2)) is equal to (xB_2 >> (log2_parallel_merge_level_minus2 + 2)) and (yP >> (log2_parallel_merge_level_minus2 + 2)) is equal to (yB_2 >> (log2_parallel_merge_level_minus2 + 2)), $availableB_2$ is set equal to FALSE
– The variables $availableFlagB_2$, $refIdxLXB_2$, $predFlagLXB_2$, and $mvLXB_2$ are derived as follows.
  – If one or more of the following conditions are true, $availableFlagB_2$ is set equal to 0, both components $mvLXB_2$ are set equal to 0, $refIdxLXB_2$ is set equal to −1 and $predFlagLXB_2$ is set equal to 0 (with X being 0 or 1, respectively).
    – $availableB_2$ is equal to FALSE
    – $availableA_1$ is equal to TRUE and prediction units covering luma location (xA_1, yA_1) and luma location (xB_2, yB_2) have the same motion vectors and the same reference indices
    – $availableB_1$ is equal to TRUE and the prediction units covering luma location (xB_1, yB_1) and luma location (xB_2, yB_2) have the same motion vectors and the same reference indices
    – $availableFlagA_0$ + $availableFlagA_1$ + $availableFlagB_0$ + $availableFlagB_1$ is equal to 4.
  – Otherwise, $availableFlagB_2$ is set equal to 1 and the following assignments are made.

$$mvLXB_2 = MvLX[\ xB_2\ ][\ yB_2\ ] \quad (8\text{-}97)$$
$$refIdxLXB_2 = RefIdxLX[\ xB_2\ ][\ yB_2\ ] \quad (8\text{-}98)$$
$$predFlagLXB_2 = PredFlagLX[\ xB_2\ ][\ yB_2\ ] \quad (8\text{-}99)$$

8.5.3.1.3 Derivation process for combined bi-predictive merging candidates

Inputs of this process are

– a merging candidate list mergeCandList,
– reference indices refIdxL0N and refIdxL1N of every candidate N being in mergeCandList,
– prediction list utilization flags predFlagL0N and predFlagL1N of every candidate N being in mergeCandList,
– motion vectors mvL0N and mvL1N of every candidate N being in mergeCandList,
– the number of elements numMergeCand within mergeCandList,
– the number of elements numOrigMergeCand within the mergeCandList after the spatial and temporal merge candidate derivation process,

Outputs of this process are

– the merging candidate list mergeCandList,
– the number of elements numMergeCand within mergeCandList.

– reference indices refIdxL0combCand$_k$ and refIdxL1combCand$_k$ of every new candidate combCand$_k$ being added in mergeCandList during the invokation of this process,

– prediction list utilization flags predFlagL0combCand$_k$ and predFlagL1combCand$_k$ of every new candidate combCand$_k$ being added in mergeCandList during the invokation of this process,

– motion vectors mvL0combCand$_k$ and mvL1combCand$_k$ of every new candidate combCand$_k$ being added in mergeCandList during the invokation of this process,

When numOrigMergeCand is greater than 1 and less than MaxNumMergeCand, the variable numInputMergeCand is set to numMergeCand, the variable combIdx is set to 0, the variable combStop is set to FALSE and the following steps are repeated until combStop is equal to TRUE.

1. The variables l0CandIdx and l1CandIdx are derived using combIdx as specified in Table 8-6.
2. The following assignments are made with l0Cand being the candidate at position l0CandIdx and l1Cand being the candidate at position l1CandIdx in the merging candidate list mergeCandList (l0Cand = mergeCandList[l0CandIdx], l1Cand = mergeCandList[l1CandIdx]).
3. When all of the following conditions are true,
   – predFlagL0l0Cand = = 1,
   – predFlagL1l1Cand = = 1,
   – DiffPicOrderCnt(RefPicList0[refIdxL0l0Cand], RefPicList1[refIdxL1l1Cand]) != 0 || mvL0l0Cand != mvL1l1Cand,

   the candidate combCand$_k$ with k equal to (numMergeCand − numInputMergeCand) is added at the end of mergeCandList (mergeCandList[numMergeCand] = combCand$_k$) and the reference indices, the prediction list utilization flags and the motion vectors of combCand$_k$ are derived as follows and numMergeCand is incremented by 1.

   refIdxL0combCand$_k$ = refIdxL0l0Cand (8-100)

   refIdxL1combCand$_k$ = refIdxL1l1Cand (8-101)

   predFlagL0combCand$_k$ = 1 (8-102)

   predFlagL1combCand$_k$ = 1 (8-103)

   mvL0combCand$_k$[0] = mvL0l0Cand[0] (8-104)

   mvL0combCand$_k$[1] = mvL0l0Cand[1] (8-105)

   mvL1combCand$_k$[0] = mvL1l1Cand[0] (8-106)

   mvL1combCand$_k$[1] = mvL1l1Cand[1] (8-107)

   numMergeCand = numMergeCand + 1 (8-108)
4. The variable combIdx is incremented by 1.
5. When combIdx is equal to (numOrigMergeCand * (numOrigMergeCand − 1)) or numMergeCand is equal to MaxNumMergeCand, combStop is set to TRUE.

Table 8-6 – Specification of l0CandIdx and l1CandIdx

combIdx	0	1	2	3	4	5	6	7	8	9	10	11
l0CandIdx	0	1	0	2	1	2	0	3	1	3	2	3
l1CandIdx	1	0	2	0	2	1	3	0	3	1	3	2

8.5.3.1.4 Derivation process for zero motion vector merging candidates

Inputs of this process are

– a merging candidate list mergeCandList,

– reference indices refIdxL0N and refIdxL1N of every candidate N being in mergeCandList,

– prediction list utilization flags predFlagL0N and predFlagL1N of every candidate N being in mergeCandList,

– motion vectors mvL0N and mvL1N of every candidate N being in mergeCandList,

– the number of elements numMergeCand within mergeCandList,

Outputs of this process are

– the merging candidate list mergeCandList,

– the number of elements numMergeCand within mergeCandList.

– reference indices $refIdxL0zeroCand_m$ and $refIdxL10zeroCand_m$ of every new candidate $zeroCand_m$ being added in mergeCandList during the invokation of this process,

– prediction list utilization flags $predFlagL0zeroCand_m$ and $predFlagL10zeroCand_m$ of every new candidate $zeroCand_m$ being added in mergeCandList during the invokation of this process,

– motion vectors $mvL0zeroCand_m$ and $mvL10zeroCand_m$ of every new candidate $zeroCand_m$ being added in mergeCandList during the invokation of this process,

The variable numRefIdx is derived as follows.

– If slice_type is equal to P, numRefIdx is set to num_ref_idx_l0_active_minus1 + 1

– Otherwise (slice_type is equal to B), numRefIdx is set to Min(num_ref_idx_l0_active_minus1 + 1, num_ref_idx_l1_active_minus1 + 1)

When numMergeCand is less than MaxNumMergeCand, the variable numInputMergeCand is set to numMergeCand, the variable zeroIdx is set to 0, and the following steps are repeated until numMergeCand 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 slice_type is equal to P, the candidate $zeroCand_m$ with m equal to (numMergeCand − numInputMergeCand) is added at the end of mergeCandList (mergeCandList[numMergeCand] = $zeroCand_m$) and the reference indices, the prediction list utilization flags and the motion vectors of $zeroCand_m$ are derived as follows and numMergeCand is incremented by 1.

   $$refIdxL0zeroCand_m = (zeroIdx < numRefIdx) ? zeroIdx : 0 \quad (8\text{-}109)$$

   $$refIdxL1zeroCand_m = -1 \quad (8\text{-}110)$$

   $$predFlagL0zeroCand_m = 1 \quad (8\text{-}111)$$

   $$predFlagL1zeroCand_m = 0 \quad (8\text{-}112)$$

   $$mvL0zeroCand_m[\ 0\ ] = 0 \quad (8\text{-}113)$$

   $$mvL0zeroCand_m[\ 1\ ] = 0 \quad (8\text{-}114)$$

   $$mvL1zeroCand_m[\ 0\ ] = 0 \quad (8\text{-}115)$$

   $$mvL1zeroCand_m[\ 1\ ] = 0 \quad (8\text{-}116)$$

   $$numMergeCand = numMergeCand + 1 \quad (8\text{-}117)$$

   – Otherwise (slice_type is equal to B), the candidate $zeroCand_m$ with m equal to (numMergeCand − numInputMergeCand) is added at the end of mergeCandList (mergeCandList[numMergeCand] = $zeroCand_m$) and the reference indices, the prediction list utilization flags and the motion vectors of $zeroCand_m$ are derived as follows and numMergeCand is incremented by 1.

   $$refIdxL0zeroCand_m = (zeroIdx < numRefIdx) ? zeroIdx : 0 \quad (8\text{-}118)$$

   $$refIdxL1zeroCand_m = (zeroIdx < numRefIdx) ? zeroIdx : 0 \quad (8\text{-}119)$$

   $$predFlagL0zeroCand_m = 1 \quad (8\text{-}120)$$

   $$predFlagL1zeroCand_m = 1 \quad (8\text{-}121)$$

   $$mvL0zeroCand_m[\ 0\ ] = 0 \quad (8\text{-}122)$$

   $$mvL0zeroCand_m[\ 1\ ] = 0 \quad (8\text{-}123)$$

$mvL1zeroCand_m[0] = 0$ (8-124)

$mvL1zeroCand_m[1] = 0$ (8-125)

numMergeCand = numMergeCand + 1 (8-126)

2. The variable zeroIdx is incremented by 1.

8.5.3.1.5 Derivation process for luma motion vector prediction

Inputs to this process are

– a luma location (xC, yC) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable nCS specifying the size of the current luma coding block,

– a luma location (xP, yP) specifying the top-left sample of the current luma prediction block relative to the top-left luma sample of the current picture,

– variables specifying the width and the height of the luma prediction block, nPbW and nPbH,

– the reference index of the current prediction unit partition refIdxLX (with X being 0 or 1),

– a variable partIdx specifying the index of the current prediction unit within the current coding unit.

Output of this process is

– the prediction mvpLX of the motion vector mvLX (with X being 0 or 1).

The motion vector predictor mvpLX is derived in the following ordered steps.

1. The derivation process for motion vector predictor candidates from neighboring prediction unit partitions in subclause 8.5.3.1.6 is invoked with the luma coding block location (xC, yC), the coding block size nCS, the luma prediction block location (xP, yP), the width and the height of the luma prediction block nPbW and nPbH, refIdxLX (with X being 0 or 1, respectively), and the partition index partIdx as inputs and the availability flags availableFlagLXN and the motion vectors mvLXN with N being replaced by A, B as the output.
2. If both availableFlagLXA and availableFlagLXB are equal to 1 and mvLXA is not equal to mvLXB, availableFlagLXCol is set equal to 0, otherwise, the derivation process for temporal luma motion vector prediction in subclause 8.5.3.1.7 is invoked with luma location (xP, yP), the width and the height of the luma prediction block nPbW and nPbH, and refIdxLX (with X being 0 or 1, respectively) as the inputs and with the output being the availability flag availableFlagLXCol and the temporal motion vector predictor mvLXCol.
3. The motion vector predictor candidate list, mvpListLX, is constructed as follows.
    1. mvLXA, if availableFlagLXA is equal to 1
    2. mvLXB, if availableFlagLXB is equal to 1
    3. mvLXCol, if availableFlagLXCol is equal to 1
4. The motion vector predictor list is modifed as follows.
    – When mvLXA and mvLXB have the same value, mvLXB is removed from the list and the variable numMVPCandLX is set to the number of elements within the mvpListLX.
    – When numMVPCandLX is less than 2, the following applies repeatedly until numMVPCandLX is equal to 2.

        mvpListLX[numMVPCandLX][0] = 0 (8-127)

        mvpListLX[numMVPCandLX][1] = 0 (8-128)

        numMVPCandLX = numMVPCandLX + 1 (8-129)

    – When numMVPCandLX is greater than 2, all motion vector predictor candidates mvpListLX[idx] with idx greater than 1 are removed from the list.
5. The motion vector of mvpListLX[mvp_lX_flag[xP][yP]] is assigned to mvpLX.

8.5.3.1.6 Derivation process for motion vector predictor candidates

Inputs to this process are

– a luma location (xC, yC) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable nCS specifying the size of the current luma coding block,

– a luma location (xP, yP) specifying the top-left sample of the current luma prediction block relative to the top-left luma sample of the current picture,

– variables specifying the width and the height of the luma prediction block, nPbW and nPbH,

– the reference index of the current prediction unit partition refIdxLX (with X being 0 or 1),

– a variable partIdx specifying the index of the current prediction unit within the current coding unit.

Outputs of this process are (with N being replaced by A, or B)

– the motion vectors mvLXN of the neighbouring prediction units,

– the availability flags availableFlagLXN of the neighbouring prediction units.

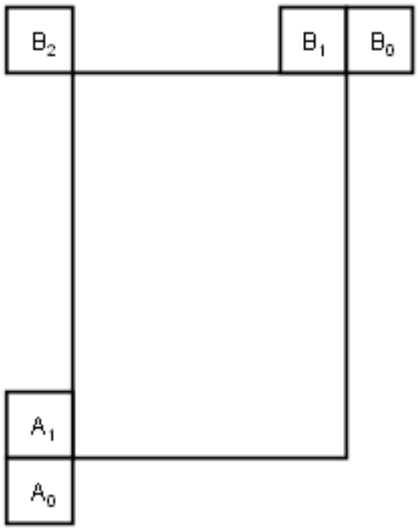


Figure 8-3 – Spatial motion vector neighbours (informative)

The variable currPb specifies the current luma prediction block at luma location (xP, yP) and the variable currPic specifies the current picture.

The variable isScaledFlagLX with X being 0 or 1 is set equal to 0.

The motion vector mvLXA and the availability flag availableFlagLXA are derived in the following ordered steps:

1. The sample location (xA_0, yA_0) is set equal to (xP − 1, yP + nPbH) and the sample location (xA_1, yA_1) is set equal to (xA_0, yA_0 − 1).
2. The availability flag availableFlagLXA is set equal to 0 and the both components of mvLXA are set equal to 0.
3. The availability derivation process for a prediction block as specified in subclause 6.4.2 is invoked with the luma location (xC, yC), the current luma coding block size nCbS set equal to nCS, the luma location (xP, yP), the width and the height of the luma prediction block nPbW and nPbH, the luma location (xN, yN) set equal to (xA_0, yA_0) and the partition index partIdx as inputs and the output is assigned to the prediction block availability flag availableA_0.
4. The availability derivation process for a prediction block as specified in subclause 6.4.2 is invoked with the luma location (xC, yC), the current luma coding block size nCbS set equal to nCS, the luma location (xP, yP), the width and the height of the luma prediction block nPbW and nPbH, the luma location (xN, yN) set equal to (xA_1, yA_1) and the partition index partIdx as inputs and the output is assigned to the prediction block availability flag availableA_1.
5. When availableA_0 or availableA_1 is equal to TRUE, the variable isScaledFlagLX is set equal to 1.

6. The following applies for (xA_k, yA_k) from (xA_0, yA_0) to (xA_1, yA_1).

   – When availableA_k is equal to TRUE , CuPredMode[xA_k][yA_k] is not equal to MODE_INTRA and availableFlagLXA is equal to 0, the following applies.

   – If, PredFlagLX[xA_k][yA_k] is equal to 1 and the reference index refIdxLX[xA_k][yA_k] is equal to the reference index of the current prediction unit refIdxLX, availableFlagLXA is set equal to 1 and the following assignments are made.

   $$mvLXA = MvLX[xA_k][yA_k] \quad (8\text{-}130)$$
   $$refIdxA = RefIdxLX[xA_k][yA_k] \quad (8\text{-}131)$$

   – Otherwise, if PredFlagLY[xA_k][yA_k] (with Y = !X) is equal to 1 and DiffPicOrderCnt(RefPicListY[refIdxLY[xA_k][yA_k]], RefPicListX[refIdxLX]) is equal to 0, availableFlagLXA is set equal to 1 and the following assignments are made.

   $$mvLXA = MvLY[xA_k][yA_k] \quad (8\text{-}132)$$
   $$refIdxA = RefIdxLY[xA_k][yA_k] \quad (8\text{-}133)$$

7. When availableFlagLXA is equal to 0, the following applies for (xA_k, yA_k) from (xA_0, yA_0) to (xA_1, yA_1) or until availableFlagLXA is equal to 1.

   – When availableA_k is equal to TRUE , CuPredMode[xA_k][yA_k] is not equal to MODE_INTRA and availableFlagLXA is equal to 0, the following applies.

   – If PredFlagLX[xA_k][yA_k] is equal to 1 and LongTermRefPic(currPic, currPb, refIdxLX, RefPicListX) is equal to LongTermRefPic(currPic, currPb, RefIdxLX[xA_k][yA_k], RefPicListX), availableFlagLXA is set equal to 1 and the following assignments are made.

   $$mvLXA = MvLX[xA_k][yA_k] \quad (8\text{-}134)$$
   $$refIdxA = RefIdxLX[xA_k][yA_k] \quad (8\text{-}135)$$
   $$refPicListA = RefPicListX \quad (8\text{-}136)$$

   – Otherwise, if PredFlagLY[xA_k][yA_k] (with Y = !X) is equal to 1 and LongTermRefPic(currPic, currPb, refIdxLX, RefPicListX) is equal to LongTermRefPic(currPic, currPb, RefIdxLY[xA_k][yA_k], RefPicListY), availableFlagLXA is set equal to 1 and the following assignments are made.

   $$mvLXA = MvLY[xA_k][yA_k] \quad (8\text{-}137)$$
   $$refIdxA = RefIdxLY[xA_k][yA_k] \quad (8\text{-}138)$$
   $$refPicListA = RefPicListY \quad (8\text{-}139)$$

   – When availableFlagLXA is equal to 1, and both refPicListA[refIdxA] and RefPicListX[refIdxLX] are short-term reference pictures, mvLXA is derived as specified below.

   $$tx = (16384 + (Abs(td) >> 1)) / td \quad (8\text{-}140)$$

   $$distScaleFactor = Clip3(-4096, 4095, (tb * tx + 32) >> 6) \quad (8\text{-}141)$$

   $$mvLXA = Clip3(-32768, 32767, Sign(distScaleFactor * mvLXA) * ((Abs(distScaleFactor * mvLXA) + 127) >> 8)) \quad (8\text{-}142)$$

   where td and tb are derived as

   $$td = Clip3(-128, 127, DiffPicOrderCnt(currPic, refPicListA[refIdxA])) \quad (8\text{-}143)$$

   $$tb = Clip3(-128, 127, DiffPicOrderCnt(currPic, RefPicListX[refIdxLX])) \quad (8\text{-}144)$$

The motion vector mvLXB and the availability flag availableFlagLXB are derived in the following ordered steps:

1. Let a set of three sample location (xB_k, yB_k), with k = 0,1,2, specifies sample locations with xB_0 = xP + nPbW, xB_1 = xB_0− 1, xB_2 = xP − 1 and yB_k = yP − 1. The set of sample locations (xB_k, yB_k) represent the sample locations immediately to the upper side of the above partition boundary and its extended line.
2. The availability flag availableFlagLXB is set equal to 0 and the both components of mvLXB are set equal to 0.
3. Tthe following applies for (xB_k, yB_k) from (xB_0, yB_0) to (xB_2, yB_2).

   – The availability derivation process for a prediction block as specified in subclause 6.4.2 is invoked with the luma location (xC, yC), the current luma coding block size nCbS set equal to nCS, the luma location (xP, yP), the width and the height of the luma prediction block nPbW and nPbH, the luma location

(xN, yN) set equal to (xB_k, yB_k) and the partition index partIdx as inputs and the output is assigned to the prediction block availability flag $availableB_k$.

– When $availableB_k$ is equal to TRUE and availableFlagLXB is equal to 0, the following applies.

  – If PredFlagLX[xB_k][yB_k] is equal to 1, and the reference index refIdxLX[xB_k][yB_k] is equal to the reference index of the current prediction unit refIdxLX, availableFlagLXB is set equal to 1 and the following assignments are made.

$$mvLXB = MvLX[\ xB_k\ ][\ yB_k\ ] \quad (8\text{-}145)$$
$$refIdxB = RefIdxLX[\ xB_k\ ][\ yB_k\ ] \quad (8\text{-}146)$$

  – Otherwise, if PredFlagLY[xB_k][yB_k] (with Y = !X) is equal to 1 and DiffPicOrderCnt(RefPicListY[refIdxLY[xB_k][yB_k]], RefPicListX[refIdxLX]) is equal to 0, availableFlagLXB is set equal to 1 and the following assignments are made.

$$mvLXB = MvLY[\ xB_k\ ][\ yB_k\ ] \quad (8\text{-}147)$$
$$refIdxB = RefIdxLY[\ xB_k\ ][\ yB_k\ ] \quad (8\text{-}148)$$

4. When isScaledFlagLX is equal to 0 and availableFlagLXB is equal to 1, availableFlagLXA is set equal to 1 and the following assignments are made.

$$mvLXA = mvLXB \quad (8\text{-}149)$$
$$refIdxA = refIdxLXB \quad (8\text{-}150)$$

5. When isScaledFlagLX is equal to 0, availableFlagLXB is set equal to 0 and the following applies for (xB_k, yB_k) from (xB_0, yB_0) to (xB_2, yB_2) or until availableFlagLXB is equal to 1.

  – The availability derivation process for a prediction block as specified in subclause 6.4.2 is invoked with the luma location (xC, yC), the current luma coding block size nCbS set equal to nCS, the luma location (xP, yP), the width and the height of the luma prediction block nPbW and nPbH, the luma location (xN, yN) set equal to (xB_k, yB_k) and the partition index partIdx as inputs and the output is assigned to the prediction block availability flag $availableB_k$.

  – When $availableB_k$ is equal to TRUE and availableFlagLXB is equal to 0, the following applies.

    – If PredFlagLX[xB_k][yB_k] is equal to 1 and LongTermRefPic(currPic, currPb, refIdxLX, RefPicListX) is equal to LongTermRefPic(currPic, currPb, RefIdxLX[xB_k][yB_k], RefPicListX), availableFlagLXB is set equal to 1 and the following assignments are made.

$$mvLXB = MvLX[\ xB_k\ ][\ yB_k\ ] \quad (8\text{-}151)$$
$$refIdxB = RefIdxLX[\ xB_k\ ][\ yB_k\ ] \quad (8\text{-}152)$$
$$refPicListB = RefPicListX \quad (8\text{-}153)$$

    – Otherwise, if PredFlagLY[xB_k][yB_k] (with Y = !X) is equal to 1 and LongTermRefPic(currPic, currPb, refIdxLX, RefPicListX) is equal to LongTermRefPic(currPic, currPb, RefIdxLY[xB_k][yB_k], RefPicListY), availableFlagLXB is set equal to 1 and the following assignments are made.

$$mvLXB = MvLY[\ xB_k\ ][\ yB_k\ ] \quad (8\text{-}154)$$
$$refIdxB = RefIdxLY[\ xB_k\ ][\ yB_k\ ] \quad (8\text{-}155)$$
$$refPicListB = RefPicListY \quad (8\text{-}156)$$

  – When availableFlagLXB is equal to 1 and DiffPicOrderCnt(refPicListB[refIdxB], RefPicListX[refIdxLX]) is not equal to 0 and both refPicListB[refIdxB] and RefPicListX[refIdxLX] are short-term reference pictures, mvLXB is derived as specified below.

$$tx = (16384 + (Abs(td) >> 1)) / td \quad (8\text{-}157)$$

$$distScaleFactor = Clip3(-4096, 4095, (tb * tx + 32) >> 6) \quad (8\text{-}158)$$

$$mvLXB = Clip3(-32768, 32767, Sign(distScaleFactor * mvLXB) * ((Abs(distScaleFactor * mvLXB) + 127) >> 8)) \quad (8\text{-}159)$$

where td and tb are derived as

$$td = Clip3(-128, 127, DiffPicOrderCnt(currPic, refPicListB[refIdxB])) \quad (8\text{-}160)$$

$$tb = Clip3(-128, 127, DiffPicOrderCnt(currPic, RefPicListX[refIdxLX])) \quad (8\text{-}161)$$

8.5.3.1.7 Derivation process for temporal luma motion vector prediction

Inputs to this process are

– a luma location (xP, yP) specifying the top-left sample of the current luma prediction block relative to the top-left luma sample of the current picture,

– variables specifying the width and the height of the luma prediction block, nPbW and nPbH,

– a reference index refIdxLX (with X being 0 or 1).

Outputs of this process are

– the motion vector prediction mvLXCol,

– the availability flag availableFlagLXCol.

The variable currPb specifies the current luma prediction block at luma location (xP, yP).

The variables mvLXCol and availableFlagLXCol are derived as follows.

– If slice_temporal_mvp_enable_flag is equal to 0, both components of mvLXCol are set equal to 0 and availableFlagLXCol is set equal to 0.

– Otherwise, the following ordered steps apply.

1. Depending on the values of slice_type, collocated_from_l0_flag, and collocated_ref_idx, the variable colPic, specifying the picture that contains the collocated partition, is derived as follows.

   – If slice_type is equal to B and collocated_from_l0_flag is equal to 0, the variable colPic specifies the picture that contains the collocated partition as specified by RefPicList1[collocated_ref_idx].

   – Otherwise (slice_type is equal to B and collocated_from_l0_flag is equal to 1 or slice_type is equal to P), the variable colPic specifies the picture that contains the collocated partition as specified by RefPicList0[collocated_ref_idx].

2. The bottom right collocated motion vector is derived as follows

   $$xPRb = xP + nPbW \quad (8\text{-}162)$$

   $$yPRb = yP + nPbH \quad (8\text{-}163)$$

   – If (yP >> Log2CtbSizeY) is equal to (yPRb >> Log2CtbSizeY), and xPRb is less than pic_width_in_luma_samples, the following applies.

     – The variable colPb specifies the luma prediction block covering the modified location given by ((xPRb >> 4) << 4, (yPRb >> 4) << 4) inside the collocated picture specified by colPic.

     – The luma location (xPCol, yPCol) is set equal to the top-left sample of the of the collocated luma prediction block specified by colPb relative to the top-left luma sample of the collocated picture specified by colPic.

     – The derivation process for collocated motion vectors as specified in subclause 8.5.3.1.8 is invoked with currPb, colPic, colPb, (xPCol, yPCol), and refIdxLX as inputs and the output being assigned to mvLXCol and availableFlagLXCol.

   – Otherwise, both components of mvLXCol are set equal to 0 and availableFlagLXCol is set equal to 0.

3. When availableFlagLXCol is equal to 0, the central collocated motion vector is derived as follows.

   $$xPCtr = xP + (nPbW >> 1) \quad (8\text{-}164)$$

   $$yPCtr = yP + (nPbH >> 1) \quad (8\text{-}165)$$

   – The variable colPb specifies the luma prediction block covering the modified location given by ((xPCtr >> 4) << 4, (yPCtr >> 4) << 4) inside the colPic.

   – The luma location (xPCol, yPCol) is set equal to the top-left sample of the of the collocated luma prediction block specified by colPb relative to the top-left luma sample of the collocated picture specified by colPic.

   – The derivation process for collocated motion vectors as specified in subclause 8.5.3.1.8 is invoked with currPb, colPic, colPb, (xPCol, yPCol), and refIdxLX as inputs and the output being assigned to mvLXCol and availableFlagLXCol.

8.5.3.1.8 Derivation process for collocated motion vectors

Inputs to this process are

– currPb specifying the current prediction block,

– colPic specifying the collocated picture,

– colPb specifying the collocated prediction block inside the collocated picture specified by colPic,

– a luma location (xPCol, yPCol) specifying the top-left sample of the collocated luma prediction block specified by colPb relative to the top-left luma sample of the collocated picture specified by colPic,

– a reference index refIdxLX (with X being 0 or 1).

Outputs of this process are

– the motion vector prediction mvLXCol,

– the availability flag availableFlagLXCol.

The variable currPic specifies the current picture.

The arrays predFlagLXCol[x][y], mvLXCol[x][y] and refIdxLXCol[x][y] are set equal to the corresponding arrays of the collocated picture specified by colPic, PredFlagLX[x][y], MvLX[x][y] and RefIdxLX[x][y], respectively with X being the value of X this process is invoked for.

The variables mvLXCol and availableFlagLXCol are derived as follows.

– If colPb is coded in an intra 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 predFlagL0Col[xPCol][yPCol] is equal to 0, mvCol, refIdxCol, and listCol are set equal to mvL1Col[xPCol][yPCol], refIdxL1Col[xPCol][yPCol], and L1, respectively.

  – Otherwise if predFlagL0Col[xPCol][yPCol] is equal to 1 and predFlagL1Col[xPCol][yPCol] is equal to 0, mvCol, refIdxCol, and listCol are set equal to mvL0Col[xPCol][yPCol], refIdxL0Col[xPCol][yPCol], and L0, respectively.

  – Otherwise (predFlagL0Col[xPCol][yPCol] is equal to 1 and predFlagL1Col[xPCol][yPCol] is equal to 1), the following assignments are made.

    – If DiffPicOrderCnt(currPic, pic) is less than or equal to 0 for every picture pic in every reference picture list of the current slice, mvCol, refIdxCol, and listCol are set equal to mvLXCol[xPCol][yPCol], refIdxLXCol[xPCol][yPCol] and LX, respectively with X being the value of X this process is invoked for.

    – Otherwise, mvCol, refIdxCol and listCol are set equal to mvLNCol[xPCol][yPCol], refIdxLNCol[xPCol][yPCol] and LN, respectively with N being the value of collocated_from_l0_flag.

  and mvLXCol and availableFlagLXCol are derived as follows.

  – If LongTermRefPic(currPic, currPb, refIdxLX, LX) is not equal to LongTermRefPic(colPic, colPb, 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, refPicListCol[refIdxCol] is set to be the picture with reference index refIdxCol in the reference picture list listCol of the slice containing prediction block currPb in the picture colPic, and the following applies.

$$colPocDiff = DiffPicOrderCnt(colPic, refPicListCol[refIdxCol]) \quad (8\text{-}166)$$

$$currPocDiff = DiffPicOrderCnt(currPic, RefPicListX[refIdxLX]) \quad (8\text{-}167)$$

    – If RefPicListX[refIdxLX] is a long-term reference picture, or colPocDiff is equal to currPocDiff, mvLXCol is derived as:

$$mvLXCol = mvCol \quad (8\text{-}168)$$

    – Otherwise, mvLXCol is derived as scaled version of the motion vector mvCol as specified below.

$$tx = (16384 + (Abs(td) >>1)) / td \quad (8\text{-}169)$$

distScaleFactor = Clip3(−4096, 4095, (tb * tx + 32) >> 6) (8-170)

mvLXCol = Clip3(−32768, 32767, Sign(distScaleFactor * mvCol) *
((Abs(distScaleFactor * mvCol) + 127) >> 8)) (8-171)

where td and tb are derived as

td = Clip3(−128, 127, colPocDiff) (8-172)

tb = Clip3(−128, 127, currPocDiff) (8-173)

8.5.3.1.9 Derivation process for chroma motion vectors

[Ed.: (WJ) 4:2:0 assumption yet]

Inputs to this process are a luma motion vector mvLX and a reference index refIdLX.

Output of this process is a chroma motion vector mvCLX.

A chroma motion vector is derived from the corresponding luma motion vector.

For the derivation of the chroma motion vector mvCLX, the following applies.

mvCLX[0] = mvLX[0] (8-174)

mvCLX[1] = mvLX[1] (8-175)

8.5.3.2 Decoding process for inter prediction samples

Inputs to this process are:

– a luma location (xC, yC) specifying the top-left sample of the current luma coding block relative to the top left luma sample of the current picture,

– a luma location (xB, yB) specifying the top-left sample of the current luma prediction block relative to the top-left sample of the current luma coding block,

– a variable nCS specifying the size of the current luma coding block,

– variables specifying the width and the height of the luma prediction block, nPbW and nPbH,

– luma motion vectors mvL0 and mvL1,

– chroma motion vectors mvCL0 and mvCL1,

– reference indices refIdxL0 and refIdxL1,

– prediction list utilization flags, predFlagL0 and predFlagL1.

Outputs of this process are:

– a (nCS_L)x(nCS_L) array $predSamples_L$ of luma prediction samples, where nCS_L is derived as specified below,

– a (nCS_C)x(nCS_C) array $preSamples_{Cb}$ of chroma prediction samples for the component Cb, where nCS_C is derived as specified below,

– a (nCS_C)x(nCS_C) array $predSamples_{Cr}$ of chroma residual samples for the component Cr, where nCS_C is derived as specified below.

The variable nCS_L is set equal to nCS and the variable nCS_C is set equal to nCS >> 1. [Ed: (WJ) revisit for supporting other chroma formats]

Let $predSamplesL0_L$ and $predSamplesL1_L$ be (nPbW)x(nPbH) arrays of predicted luma sample values and $predSampleL0_{Cb}$, $predSampleL1_{Cb}$, $predSampleL0_{Cr}$, and $predSampleL1_{Cr}$ be (nPbW/2)x(nPbH/2) arrays of predicted chroma sample values.

For LX being replaced by either L0 or L1 in the variables predFlagLX, RefPicListX, refIdxLX, refPicLX, and predPartLX, the following is specified.

When predFlagLX is equal to 1, the following applies.

– The reference picture consisting of an ordered two-dimensional array $refPicLX_L$ of luma samples and two ordered two-dimensional arrays $refPicLX_{Cb}$ and $refPicLX_{Cr}$ of chroma samples is derived by invoking the process specified in subclause 8.5.3.2.1 with refIdxLX as input.

– The arrays predSamplesLX$_L$, predSamplesLX$_{Cb}$, and predSamplesLX$_{Cr}$ are derived by invoking the fractional sample interpolation process specified in subclause 8.5.3.2.2 with the luma locations (xC, yC), (xB, yB), the width and the height of the current luma prediction block nPbW, nPbH, the motion vectors mvLX, mvCLX, and the reference arrays with refPicLX$_L$, refPicLX$_{Cb}$ and refPicLX$_{Cr}$ given as input.

The array predSample$_L$ of the prediction samples of luma component is derived by invoking the weighted sample prediction process specified in subclause 8.5.3.2.3 with the luma location (xB, yB), the width and the height of the current luma prediction block nPbW, nPbH, and the sample arrays predSamplesL0$_L$ and predSamplesL1$_L$ as well as predFlagL0, predFlagL1, refIdxL0, refIdxL1 and cIdx equal to 0 given as input.

The array predSample$_{Cb}$ of the prediction samples of component Cb is derived by invoking the weighted sample prediction process specified in subclause 8.5.3.2.3 with the chroma location (xB/2, yB/2), the width and the height of the current chroma prediction block nPbW$_{Cb}$ set equal to nPbW/2, nPbH$_{Cb}$ set equal to nPbH/2, and the sample arrays predSamplesL0$_{Cb}$ and predSamplesL1$_{Cb}$ as well as predFlagL0, predFlagL1, refIdxL0, refIdxL1, and cIdx equal to 1 given as input.

The array predSample$_{Cr}$ of the prediction samples of component Cr is derived by invoking the weighted sample prediction process specified in subclause 8.5.3.2.3 with the chroma location (xB/2, yB/2), the width and the height of the current chroma prediction block nPbW$_{Cr}$ set equal to nPbW/2, nPbH$_{Cr}$ set equal to nPbH/2, and the sample arrays predSamplesL0$_{Cr}$ and predSamplesL1$_{Cr}$ as well as predFlagL0, predFlagL1, refIdxL0, refIdxL1, and cIdx equal to 2 given as input.

8.5.3.2.1 Reference picture selection process

Input to this process is a reference index refIdxLX.

Output of this process is a reference picture consisting of a two-dimensional array of luma samples refPicLX$_L$ and two two-dimensional arrays of chroma samples refPicLX$_{Cb}$ and refPicLX$_{Cr}$.

The output reference picture RefPicListX[refIdxLX] consists of a (pic_width_in_luma_samples)x(pic_height_in_luma_samples) array of luma samples refPicLX$_L$ and two (PicWidthInSamplesC)x(PicHeightInSamplesC) arrays of chroma samples refPicLX$_{Cb}$ and refPicLX$_{Cr}$.

The reference picture sample arrays refPicLX$_L$, refPicLX$_{Cb}$, and refPicLX$_{Cr}$ correspond to decoded sample arrays S_L, S_{Cb}, S_{Cr} derived in subclause 8.7 for a previously-decoded picture.

8.5.3.2.2 Fractional sample interpolation process

Inputs to this process are:

– a luma location (xC, yC) specifying the top-left sample of the current luma coding block relative to the top left luma sample of the current picture,

– a luma location (xB, yB) specifying the top-left sample of the current luma prediction block relative to the top left sample of the current luma coding block,

– the width and height of the prediction block, nPbW and nPbH, in luma-sample units,

– a luma motion vector mvLX given in quarter-luma-sample units,

– a chroma motion vector mvCLX given in eighth-chroma-sample units,

– the selected reference picture sample arrays refPicLX$_L$, refPicLX$_{Cb}$, and refPicLX$_{Cr}$.

Outputs of this process are:

– a (nPbW)x(nPbH) array predSampleLX$_L$ of prediction luma sample values,

– two (nPbW/2)x(nPbH/2) arrays predSampleLX$_{Cb}$, and predSampleLX$_{Cr}$ of prediction chroma sample values.

The location (xP, yP) given in full-sample units of the upper-left luma samples of the current prediction block relative to the upper-left luma sample location of the given reference sample arrays is derived by

$$xP = xC + xB \quad (8\text{-}176)$$
$$yP = yC + yB \quad (8\text{-}177)$$

Let (xInt$_L$, yInt$_L$) be a luma location given in full-sample units and (xFrac$_L$, yFrac$_L$) be an offset given in quarter-sample units. These variables are used only inside this subclause for specifying fractional-sample locations inside the reference sample arrays refPicLX$_L$, refPicLX$_{Cb}$, and refPicLX$_{Cr}$.

For each luma sample location ($x_L = 0..nPbW-1$, $y_L = 0..nPbH-1$) inside the prediction luma sample array predSampleLX$_L$, the corresponding prediction luma sample value predSampleLX$_L$[x_L, y_L] is derived as follows:

– The variables $xInt_L$, $yInt_L$, $xFrac_L$, and $yFrac_L$ are derived by

$$xInt_L = xP + (mvLX[0] >> 2) + x_L \quad (8\text{-}178)$$
$$yInt_L = yP + (mvLX[1] >> 2) + y_L \quad (8\text{-}179)$$

$$xFrac_L = mvLX[0] \& 3 \quad (8\text{-}180)$$
$$yFrac_L = mvLX[1] \& 3 \quad (8\text{-}181)$$

– The prediction luma sample value $predSampleLX_L[x_L, y_L]$ is derived by invoking the process specified in subclause 8.5.3.2.2.1 with ($xInt_L$, $yInt_L$), ($xFrac_L$, $yFrac_L$) and $refPicLX_L$ given as input.

Let ($xInt_C$, $yInt_C$) be a chroma location given in full-sample units and ($xFrac_C$, $yFrac_C$) be an offset given in one-eighth sample units. These variables are used only inside this subclause for specifying general fractional-sample locations inside the reference sample arrays $refPicLX_{Cb}$ and $refPicLX_{Cr}$.

For each chroma sample location ($x_C = 0..nPbW/2-1$, $y_C = 0..nPbH/2-1$) inside the prediction chroma sample arrays $predSampleLX_{Cb}$ and $predSampleLX_{Cr}$, the corresponding prediction chroma sample values $predSampleLX_{Cb}[x_C, y_C]$ and $predSampleLX_{Cr}[x_C, y_C]$ are derived asfollows:

– The variables $xInt_C$, $yInt_C$, $xFrac_C$, and $yFrac_C$ are derived by

$$xInt_C = (xP / 2) + (mvCLX[0] >> 3) + x_C \quad (8\text{-}182)$$
$$yInt_C = (yP / 2) + (mvCLX[1] >> 3) + y_C \quad (8\text{-}183)$$

$$xFrac_C = mvLX[0] \& 7 \quad (8\text{-}184)$$
$$yFrac_C = mvLX[1] \& 7 \quad (8\text{-}185)$$

– The prediction sample value $predSampleLX_{Cb}[x_C, y_C]$ is derived by invoking the process specified in subclause 8.5.3.2.2.2 with ($xInt_C$, $yInt_C$), ($xFrac_C$, $yFrac_C$) and $refPicLX_{Cb}$ given as input.

– The prediction sample value $predSampleLX_{Cr}[x_C, y_C]$ is derived by invoking the process specified in subclause 8.5.3.2.2.2 with ($xInt_C$, $yInt_C$), ($xFrac_C$, $yFrac_C$) and $refPicLX_{Cr}$ given as input.

8.5.3.2.2.1 Luma sample interpolation process

Inputs to this process are:

– a luma location in full-sample units ($xInt_L$, $yInt_L$),

– a luma location in fractional-sample units ($xFrac_L$, $yFrac_L$),

– the luma reference sample array $refPicLX_L$.

Output of this process is a predicted luma sample value $predSampleLX_L[x_L, y_L]$

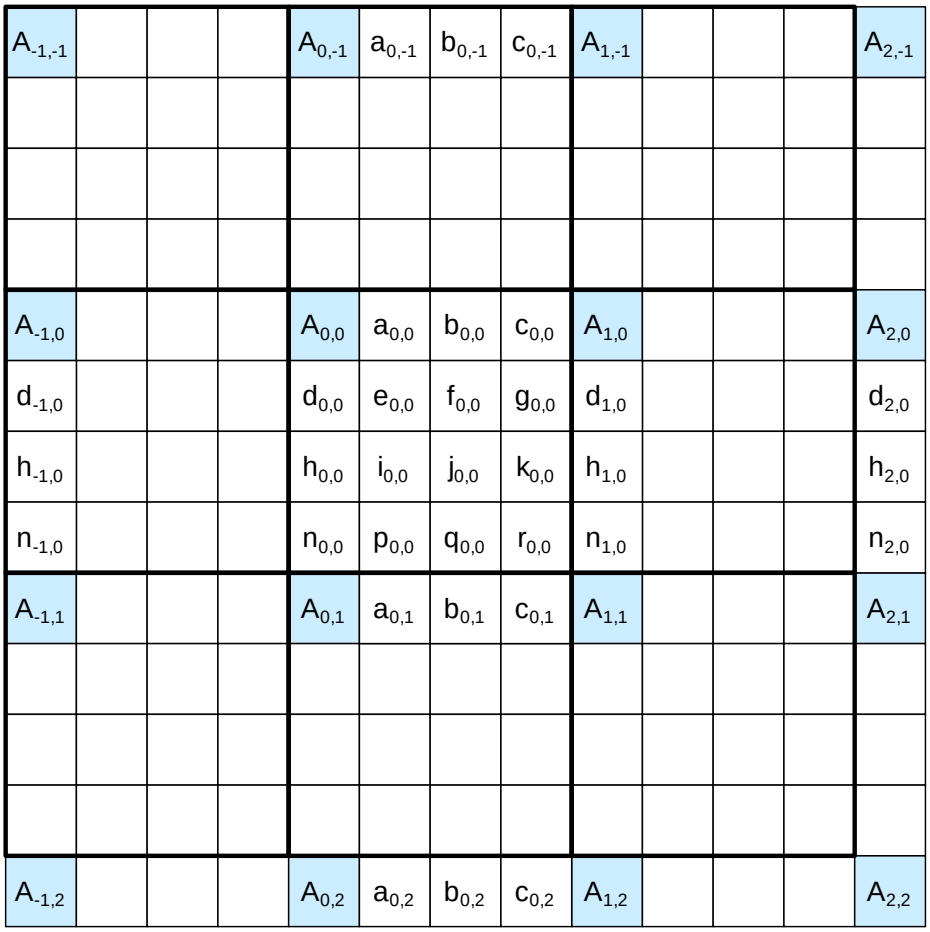


Figure 8-4 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for quarter sample luma interpolation

In Figure 8-4, the positions labelled with upper-case letters $A_{i,j}$ within shaded blocks represent luma samples at full-sample locations inside the given two-dimensional array $refPicLX_L$ of luma samples. These samples may be used for generating the predicted luma sample value $predSampleLX_L[x_L, y_L]$. The locations $(xA_{i,j}, yA_{i,j})$ for each of the corresponding luma samples $A_{i,j}$ inside the given array $refPicLX_L$ of luma samples are derived as follows:

$$xA_{i,j} = Clip3(0, pic\_width\_in\_luma\_samples - 1, xInt_L + i) \quad (8\text{-}186)$$
$$yA_{i,j} = Clip3(0, pic\_height\_in\_luma\_samples - 1, yInt_L + j) \quad (8\text{-}187)$$

The positions labelled with lower-case letters within un-shaded blocks represent luma samples at quarter-pel sample fractional locations. The luma location offset in fractional-sample units $(xFrac_L, yFrac_L)$ specifies which of the generated luma samples at full-sample and fractional-sample locations is assigned to the predicted luma sample value $predSampleLX_L[x_L, y_L]$. This assignment is done according to Table 8-7. The value of $predSampleLX_L[x_L, y_L]$ shall be the output.

Variables shift1, shift2 and shift3 are derived as follows.

– The variable shift1 is set equal to $BitDepth_Y - 8$, the variable shift2 is set equal to 6, and the variable shift3 is set equal to $14 - BitDepth_Y$.

Given the luma samples $A_{i,j}$ at full-sample locations $(xA_{i,j}, yA_{i,j})$, the luma samples "$a_{0,0}$" to "$r_{0,0}$" at fractional sample positions are derived by the following equations.

– The samples labelled $a_{0,0}$, $b_{0,0}$, $c_{0,0}$, $d_{0,0}$, $h_{0,0}$, and $n_{0,0}$ shall be derived by applying the 8-tap filter to the nearest integer position samples:

$$a_{0,0} = (-A_{-3,0} + 4*A_{-2,0} - 10*A_{-1,0} + 58*A_{0,0} + 17*A_{1,0} - 5*A_{2,0} + A_{3,0}) >> shift1 \quad (8\text{-}188)$$

$$b_{0,0} = (-A_{-3,0} + 4*A_{-2,0} - 11*A_{-1,0} + 40*A_{0,0} + 40*A_{1,0} - 11*A_{2,0} + 4*A_{3,0} - A_{4,0}) >> shift1 \quad (8\text{-}189)$$

$$c_{0,0} = (A_{-2,0} - 5*A_{-1,0} + 17*A_{0,0} + 58*A_{1,0} - 10*A_{2,0} + 4*A_{3,0} - A_{4,0}) >> shift1 \quad (8\text{-}190)$$

$$d_{0,0} = (-A_{0,-3} + 4*A_{0,-2} - 10*A_{0,-1} + 58*A_{0,0} + 17*A_{0,1} - 5*A_{0,2} + A_{0,3}) >> shift1 \quad (8\text{-}191)$$

$$h_{0,0} = (-A_{0,-3} + 4*A_{0,-2} - 11*A_{0,-1} + 40*A_{0,0} + 40*A_{0,1} - 11*A_{0,2} + 4*A_{0,3} - A_{0,4}) >> shift1 \quad (8\text{-}192)$$

$$n_{0,0} = (A_{0,-2} - 5*A_{0,-1} + 17*A_{0,0} + 58*A_{0,1} - 10*A_{0,2} + 4*A_{0,3} - A_{0,4}) >> shift1 \quad (8\text{-}193)$$

– The samples labelled $e_{0,0}$, $i_{0,0}$, $p_{0,0}$, $f_{0,0}$, $j_{0,0}$, $q_{0,0}$, $g_{0,0}$, $k_{0,0}$ and $r_{0,0}$ shall be derived by applying the 8-tap filter to the samples $a_{0,i}$, $b_{0,i}$ and $c_{0,i}$ where i = −3..4 in vertical direction:

$$e_{0,0} = (-a_{0,-3} + 4*a_{0,-2} - 10*a_{0,-1} + 58*a_{0,0} + 17*a_{0,1} - 5*a_{0,2} + a_{0,3}) >> shift2 \quad (8\text{-}194)$$

$$i_{0,0} = (-a_{0,-3} + 4*a_{0,-2} - 11*a_{0,-1} + 40*a_{0,0} + 40*a_{0,1} - 11*a_{0,2} + 4*a_{0,3} - a_{0,4}) >> shift2 \quad (8\text{-}195)$$

$$p_{0,0} = (a_{0,-2} - 5*a_{0,-1} + 17*a_{0,0} + 58*a_{0,1} - 10*a_{0,2} + 4*a_{0,3} - a_{0,4}) >> shift2 \quad (8\text{-}196)$$

$$f_{0,0} = (-b_{0,-3} + 4*b_{0,-2} - 10*b_{0,-1} + 58*b_{0,0} + 17*b_{0,1} - 5*b_{0,2} + b_{0,3}) >> shift2 \quad (8\text{-}197)$$

$$j_{0,0} = (-b_{0,-3} + 4*b_{0,-2} - 11*b_{0,-1} + 40*b_{0,0} + 40*b_{0,1} - 11*b_{0,2} + 4*b_{0,3} - b_{0,4}) >> shift2 \quad (8\text{-}198)$$

$$q_{0,0} = (b_{0,-2} - 5*b_{0,-1} + 17*b_{0,0} + 58*b_{0,1} - 10*b_{0,2} + 4*b_{0,3} - b_{0,4}) >> shift2 \quad (8\text{-}199)$$

$$g_{0,0} = (-c_{0,-3} + 4*c_{0,-2} - 10*c_{0,-1} + 58*c_{0,0} + 17*c_{0,1} - 5*c_{0,2} + c_{0,3}) >> shift2 \quad (8\text{-}200)$$

$$k_{0,0} = (-c_{0,-3} + 4*c_{0,-2} - 11*c_{0,-1} + 40*c_{0,0} + 40*c_{0,1} - 11*c_{0,2} + 4*c_{0,3} - c_{0,4}) >> shift2 \quad (8\text{-}201)$$

$$r_{0,0} = (c_{0,-2} - 5*c_{0,-1} + 17*c_{0,0} + 58*c_{0,1} - 10*c_{0,2} + 4*c_{0,3} - c_{0,4}) >> shift2 \quad (8\text{-}202)$$

Table 8-7 – Assignment of the luma prediction sample predSampleLX$_L$[x_L, y_L]

xFracL	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
yFracL	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
predSampleLX$_L$[x_L, y_L]	A << shift3	d	h	n	a	e	i	p	b	f	j	q	c	g	k	r

8.5.3.2.2.2 Chroma sample interpolation process

Inputs to this process are:

– a chroma location in full-sample units ($xInt_C$, $yInt_C$),

– a chroma location in fractional-sample units ($xFrac_C$, $yFrac_C$),

– the chroma reference sample array $refPicLX_C$.

Output of this process is a predicted chroma sample value $predSampleLX_C[x_C, y_C]$

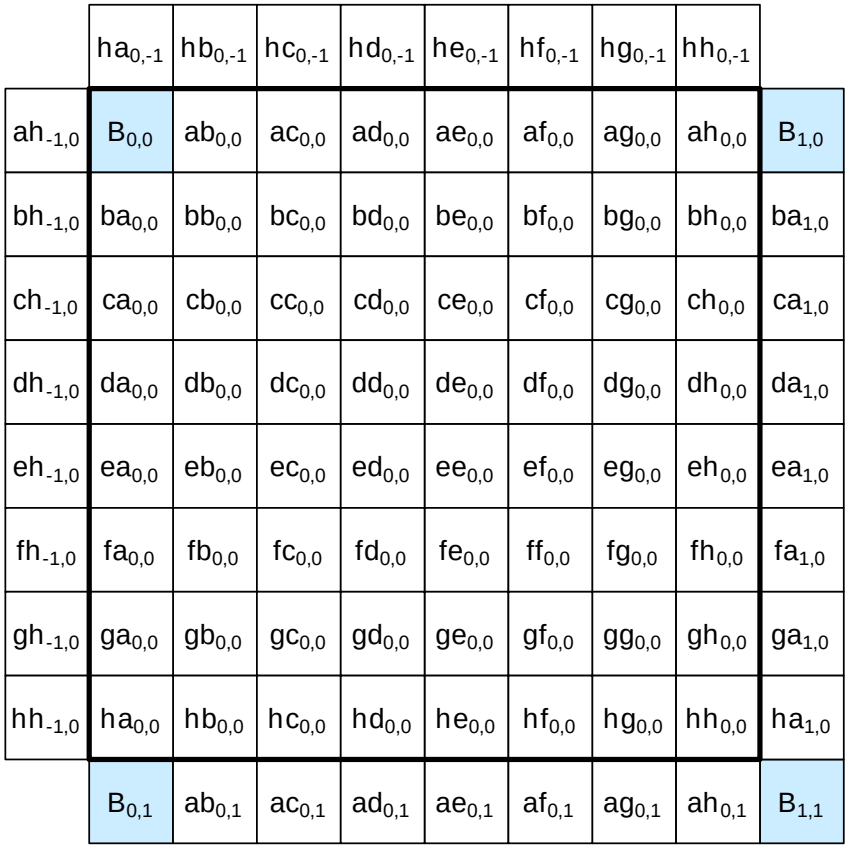


Figure 8-5 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for eighth sample chroma interpolation

In Figure 8-5, the positions labelled with upper-case letters $B_{i,j}$ within shaded blocks represent chroma samples at full-sample locations inside the given two-dimensional array $refPicLX_C$ of chroma samples. These samples may be used for generating the predicted chroma sample value $predSampleLX_C[\, x_C, y_C \,]$. The locations $(xB_{i,j}, yB_{i,j})$ for each of the corresponding chroma samples $B_{i,j}$ inside the given array $refPicLX_C$ of chroma samples are derived as follows:

$$xB_{i,j} = Clip3(0, (pic\_width\_in\_luma\_samples / SubWidthC) - 1, xInt_C + i) \quad (8\text{-}203)$$
$$yB_{i,j} = Clip3(0, (pic\_height\_in\_luma\_samples / SubHeightC) - 1, yInt_C + j) \quad (8\text{-}204)$$

The positions labelled with lower-case letters within un-shaded blocks represent chroma samples at eighth-pel sample fractional locations. The chroma location offset in fractional-sample units $(xFrac_C, yFrac_C)$ specifies which of the generated chroma samples at full-sample and fractional-sample locations is assigned to the predicted chroma sample value $predSampleLX_C[\, x_C, y_C \,]$. This assignment is done according to Table 8-8. The value of $predSampleLX_C[\, x_C, y_C \,]$ shall be the output.

Variables shift1, shift2 and shift3 are derived as follows.

– The variable shift1 is set equal to $BitDepth_C - 8$, the variable shift2 is set equal to 6, and the variable shift3 is set equal to $14 - BitDepth_C$.

Given the chroma samples $B_{i,j}$ at full-sample locations $(xB_{i,j}, yB_{i,j})$, the chroma samples "$ab_{0,0}$" to "$hh_{0,0}$" at fractional sample positions are derived by the following equations.

– The samples labelled $ab_{0,0}$, $ac_{0,0}$, $ad_{0,0}$, $ae_{0,0}$, $af_{0,0}$, $ag_{0,0}$, and $ah_{0,0}$ shall be derived by applying the 4-tap filter to the nearest integer position samples:

$$ab_{0,0} = (-2*B_{-1,0} + 58*B_{0,0} + 10*B_{1,0} - 2*B_{2,0}) >> shift1 \quad (8\text{-}205)$$

$$ac_{0,0} = (-4*B_{-1,0} + 54*B_{0,0} + 16*B_{1,0} - 2*B_{2,0}) >> shift1 \quad (8\text{-}206)$$

$$ad_{0,0} = (-6*B_{-1,0} + 46*B_{0,0} + 28*B_{1,0} - 4*B_{2,0}) >> shift1 \quad (8\text{-}207)$$

$$ae_{0,0} = (-4*B_{-1,0} + 36*B_{0,0} + 36*B_{1,0} - 4*B_{2,0}) >> shift1 \quad (8\text{-}208)$$

$$af_{0,0} = (-4*B_{-1,0} + 28*B_{0,0} + 46*B_{1,0} - 6*B_{2,0}) >> shift1 \quad (8\text{-}209)$$

$$ag_{0,0} = (-2*B_{-1,0} + 16*B_{0,0} + 54*B_{1,0} - 4*B_{2,0}) >> shift1 \quad (8\text{-}210)$$

$$ah_{0,0} = (-2*B_{-1,0} + 10*B_{0,0} + 58*B_{1,0} - 2*B_{2,0}) >> shift1 \quad (8\text{-}211)$$

– The samples labelled $ba_{0,0}$, $ca_{0,0}$, $da_{0,0}$, $ea_{0,0}$, $fa_{0,0}$, $ga_{0,0}$, and $ha_{0,0}$ shall be derived by applying the 4-tap filter to the nearest integer position samples:

$$ba_{0,0} = (-2*B_{0,-1} + 58*B_{0,0} + 10*B_{0,1} - 2*B_{0,2}) >> shift1 \quad (8\text{-}212)$$

$$ca_{0,0} = (-4*B_{0,-1} + 54*B_{0,0} + 16*B_{0,1} - 2*B_{0,2}) >> shift1 \quad (8\text{-}213)$$

$$da_{0,0} = (-6*B_{0,-1} + 46*B_{0,0} + 28*B_{0,1} - 4*B_{0,2}) >> shift1 \quad (8\text{-}214)$$

$$ea_{0,0} = (-4*B_{0,-1} + 36*B_{0,0} + 36*B_{0,1} - 4*B_{0,2}) >> shift1 \quad (8\text{-}215)$$

$$fa_{0,0} = (-4*B_{0,-1} + 28*B_{0,0} + 46*B_{0,1} - 6*B_{0,2}) >> shift1 \quad (8\text{-}216)$$

$$ga_{0,0} = (-2*B_{0,-1} + 16*B_{0,0} + 54*B_{0,1} - 4*B_{0,2}) >> shift1 \quad (8\text{-}217)$$

$$ha_{0,0} = (-2*B_{0,-1} + 10*B_{0,0} + 58*B_{0,1} - 2*B_{0,2}) >> shift1 \quad (8\text{-}218)$$

– The samples labelled $bX_{0,0}$, $cX_{0,0}$, $dX_{0,0}$, $eX_{0,0}$, $fX_{0,0}$, $gX_{0,0}$ and $hX_{0,0}$ for X being replaced by b, c, d, e, f, g and h, respectively, shall be derived by applying the 4-tap filter to the intermediate values $aX_{0,i}$ where i = −1..2 in vertical direction:

$$bX_{0,0} = (-2*aX_{0,-1} + 58*aX_{0,0} + 10*aX_{0,1} - 2*aX_{0,2}) >> shift2 \quad (8\text{-}219)$$

$$cX_{0,0} = (-4*aX_{0,-1} + 54*aX_{0,0} + 16*aX_{0,1} - 2*aX_{0,2}) >> shift2 \quad (8\text{-}220)$$

$$dX_{0,0} = (-6*aX_{0,-1} + 46*aX_{0,0} + 28*aX_{0,1} - 4*aX_{0,2}) >> shift2 \quad (8\text{-}221)$$

$$eX_{0,0} = (-4*aX_{0,-1} + 36*aX_{0,0} + 36*aX_{0,1} - 4*aX_{0,2}) >> shift2 \quad (8\text{-}222)$$

$$fX_{0,0} = (-4*aX_{0,-1} + 28*aX_{0,0} + 46*aX_{0,1} - 6*aX_{0,2}) >> shift2 \quad (8\text{-}223)$$

$$gX_{0,0} = (-2*aX_{0,-1} + 16*aX_{0,0} + 54*aX_{0,1} - 4*aX_{0,2}) >> shift2 \quad (8\text{-}224)$$

$$hX_{0,0} = (-2*aX_{0,-1} + 10*aX_{0,0} + 58*aX_{0,1} - 2*aX_{0,2}) >> shift2 \quad (8\text{-}225)$$

Table 8-8 – Assignment of the chroma prediction sample predSampleLX$_C$[x_C, y_C] for (X, Y) being replaced by (1, b), (2, c), (3, d), (4, e), (5, f), (6, g), and (7, h), respectively

xFracC	0	0	0	0	0	0	0	0	X	X	X	X	X	X	X	X
yFracC	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
predSampleLX$_C$[x_C, y_C]	B << shift3	ba	ca	da	ea	fa	ga	ha	aY	bY	cY	dY	eY	fY	gY	hY

8.5.3.2.3 Weighted sample prediction process

Inputs to this process are:

– a location (xB, yB) specifying the top-left sample of the current prediction block relative to the top left sample of the current coding block,

– the width and height of this prediction block, nPbW and nPbH,

– two (nPbW)x(nPbH) arrays predSamplesL0 and predSamplesL1,

– prediction list utilization flags, predFlagL0 and predFlagL1,

– reference indices, refIdxL0 and refIdxL1,

– a variable cIdx specifying colour component index.

Outputs of this process are:

– the (nPbW)x(nPbH) array predSamples of prediction sample values.

The variable bitDepth is derived as follows.

– If cIdx is equal to 0, bitDepth is set equal to $BitDepth_Y$.

– Otherwise, bitDepth is set equal to $BitDepth_C$.

In P slices, if the value of predFlagL0 is equal to 1, the following applies.

– If weighted_pred_flag is equal to 0, the array predSample of the prediction samples is derived by invoking the default weighted sample prediction process as specified in subclause 8.5.3.2.3.1 with the location (xB, yB), the width and height of this prediction block, nPbW and nPbH, two (nPbW)x(nPbH) arrays predSamplesL0 and predSamplesL1, prediction list utilization flags, predFlagL0 and predFlagL1, and the bit depth of samples, bitDepth given as input..

– Otherwise (weighted_pred_flag is equal to 1), the array predSample of the prediction samples is derived by invoking the weighted sample prediction process as specified in subclause 8.5.3.2.3.2 with the location (xB, yB), the width and height of this prediction block, nPbW and nPbH, two (nPbW)x(nPbH) arrays predSamplesL0 and predSamplesL1, prediction list utilization flags, predFlagL0 and predFlagL1, reference indices, refIdxL0 and refIdxL1, colour component index, cIdx, and the bit depth of samples, bitDepth given as input.

In B slices, if predFlagL0 or predFlagL1 is equal to 1, the following applies.

– If weighted_bipred_flag is equal to 0, the array predSample of the prediction samples is derived by invoking the default weighted sample prediction process as specified in subclause 8.5.3.2.3.1 with the location (xB, yB), the width and height of this prediction block, nPbW and nPbH, two (nPbW)x(nPbH) arrays predSamplesL0 and predSamplesL1, prediction list utilization flags, predFlagL0 and predFlagL1, and the bit depth of samples, bitDepth given as input.

– Otherwise, if weighted_bipred_flag is equal to 1, the array predSample of the prediction samples is derived by invoking the weighted sample prediction process as specified in subclause 8.5.3.2.3.2 with the location (xB, yB), the width and height of this prediction block, nPbW and nPbH, two (nPbW)x(nPbH) arrays predSamplesL0 and predSamplesL1, prediction list utilization flags, predFlagL0 and predFlagL1, reference indices, refIdxL0 and refIdxL1, colour component index, cIdx, and the bit depth of samples, bitDepth given as input.

8.5.3.2.3.1 Default weighted sample prediction process

Inputs to this process are:

– a location (xB, yB) specifying the top-left sample of the current prediction block relative to the top left sample of the current coding block,

– the width and height of this prediction block, nPbW and nPbH,

– two (nPbW)x(nPbH) arrays predSamplesL0 and predSamplesL1,

– prediction list utilization flags, predFlagL0 and predFlagL1,

– a bit depth of samples, bitDepth.

Outputs of this process are:

– the (nPbW)x(nPbH) array predSamples of prediction sample values.

Variables shift1, shift2, offset1 and offset2 are derived as follows.

– The variable shift1 is set equal to 14 − bitDepth and the variable shift2 is set equal to 15 − bitDepth,

– The variable offset1 is derived as follows.

  – If shift1 is greater than 0, offset1 set equal to 1 << (shift1 − 1).

  – Otherwise (shift1 is equal to 0), offset1 is set equal to 0.

– The variable offset2 is set equal to 1 << (shift2 − 1).

Depending on the value of predFlagL0 and predFlagL1, the prediction samples predSamples[x][y] with x = 0..(nPbW)−1 and y = 0..(nPbH)−1 are derived as follows.

– If predFlagL0 is equal to 1 and predFlagL1 is equal to 0,

$$predSamples[x][y] = Clip3(0, (1 << bitDepth) - 1, (predSamplesL0[x][y] + offset1) >> shift1) \quad (8\text{-}226)$$

– Otherwise, if predFlagL0 is equal to 0 and predFlagL1 is equal to 1,

$$predSamples[x][y] = Clip3(0, (1 << bitDepth) - 1, (predSamplesL1[x][y] + offset1) >> shift1) \quad (8\text{-}227)$$

– Otherwise,

$$predSamples[x][y] = Clip3(0, (1 << bitDepth) - 1, (predSamplesL0[x][y] + predSamplesL1[x][y] + offset2) >> shift2) \quad (8\text{-}228)$$

8.5.3.2.3.2 Weighted sample prediction process

Inputs to this process are:

– a location (xB, yB) specifying the top-left sample of the current prediction block relative to the top left sample of the current coding block,

– the width and height of this prediction block, nPbW and nPbH,

– two (nPbW)x(nPbH) arrays predSamplesL0 and predSamplesL1,

– prediction list utilization flags, predFlagL0 and predFlagL1,

– reference indices, refIdxL0 and refIdxL1,

– a variable cIdx specifying colour component index,

– a bit depth of samples, bitDepth.

Outputs of this process are:

– the (nPbW)x(nPbH) array predSamples of prediction sample values.

The variables shift1 is set equal to 14 − bitDepth.

The variables log2WD, o0, o1, and w0, w1 are derived as follows.

– If cIdx is equal to 0 for luma samples,

$$log2WD = luma\_log2\_weight\_denom + shift1 \quad (8\text{-}229)$$

$$w0 = LumaWeightL0[refIdxL0] \quad (8\text{-}230)$$

$$w1 = LumaWeightL1[refIdxL1] \quad (8\text{-}231)$$

$$o0 = luma\_offset\_l0[refIdxL0] * (1 << (bitDepth - 8)) \quad (8\text{-}232)$$

$$o1 = luma\_offset\_l1[refIdxL1] * (1 << (bitDepth - 8)) \quad (8\text{-}233)$$

– Otherwise (cIdx is not equal to 0 for chroma samples),

$$log2WD = ChromaLog2WeightDenom + shift1 \quad (8\text{-}234)$$

$$w0 = ChromaWeightL0[refIdxL0][cIdx - 1] \quad (8\text{-}235)$$

$$w1 = ChromaWeightL1[refIdxL1][cIdx - 1] \quad (8\text{-}236)$$

$$o0 = ChromaOffsetL0[refIdxL0][cIdx - 1] * (1 << (bitDepth - 8)) \quad (8\text{-}237)$$

$$o1 = ChromaOffsetL1[refIdxL1][cIdx - 1] * (1 << (bitDepth - 8)) \quad (8\text{-}238)$$

The prediction sample predSamples[x][y] with x = 0..(nPbW)−1 and y = 0..(nPbH)−1 are derived as follows:

– If the predFlagL0 is equal to 1 and predFlagL1 is equal to 0, the prediction samples are derived by:

```
if( log2WD >= 1 )
    predSamples[ x ][ y ] = Clip3( 0, ( 1 << bitDepth ) − 1,
                                   ( (predSamplesL0[ x ][ y ] * w0 + 2^(log2WD − 1)) >> log2WD ) + o0 )      (8-239)
else
    predSamples[ x ][ y ] = Clip3( 0, ( 1 << bitDepth ) − 1, predSamplesL0[ x ][ y ] * w0 + o0 )           (8-240)
```

– Otherwise, if the predFlagL0 is equal to 0 and predFlagL1 is equal to 1, the final predicted sample values predSamples [x][y] are derived by

```
if( log2WD >= 1 )
    predSamples[ x ][ y ] = Clip3( 0, ( 1 << bitDepth ) − 1,
                                   ( (predSamplesL1[ x ][ y ] * w1 + 2^(log2WD − 1)) >> log2WD ) + o1 )      (8-241)
else
    predSamples[ x ][ y ] = Clip3( 0, ( 1 << bitDepth ) − 1, predSamplesL1[ x ][ y ] * w1 + o1 )           (8-242)
```

– Otherwise, the final predicted sample values predSamples[x][y] are derived by

```
predSamples[ x ][ y ] = Clip3( 0, ( 1 << bitDepth ) − 1,
                               ( predSamplesL0 [ x ][ y ] * w0 + predSamplesL1[ x ][ y ] * w1 +
                                 ((o0 + o1 + 1) << log2WD) ) >> (log2WD + 1) )                              (8-243)
```

8.5.4 Decoding process for the residual signal of coding units coded in inter prediction mode

Inputs to this process are:

– a luma location (xC, yC) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable log2CbSize specifying the size of the current luma coding block.

Outputs of this process are:

– a (nCS_L)x(nCS_L) array $resSamples_L$ of luma residual samples, where nCS_L is derived as specified below,

– a (nCS_C)x(nCS_C) array $resSamples_{Cb}$ of chroma residual samples for the component Cb, where nCS_C is derived as specified below,

– a (nCS_C)x(nCS_C) array $resSamples_{Cr}$ of chroma residual samples for the component Cr, where nCS_C is derived as specified below.

The variable nCS_L is set equal to 1 << log2CbSize and the variable nCS_C is set equal to (1 << log2CbSize) >> 1.

Let $resSamples_L$ be a (nCS_L)x(nCS_L) array of luma residual samples and let $resSamples_{Cb}$ and $resSamples_{Cr}$ be two (nCS_C)x(nCS_C) arrays of chroma residual samples.

Depending on rqt_root_cbf, the following applies:

– If rqt_root_cbf is equal to 0, all samples of the (nCS_L)x(nCS_L) array $resSamples_L$ and all samples of the two (nCS_C)x(nCS_C) arrays $resSamples_{Cb}$ and $resSamples_{Cr}$ are set equal to 0.

– Otherwise (rqt_root_cbf is equal to 1), the following ordered steps apply:

1. The decoding process for luma residual blocks as specified in subclause 8.5.4.1 below is invoked with the luma location (xC, yC), the luma location (xB0, yB0) set equal to (0, 0), the variable log2TrafoSize set equal to log2CbSize, the variable trafoDepth set equal to 0, the variable nCS set equal to nCSL, and the (nCSL)x(nCSL) array $resSamples_L$ as the inputs and the output is a modified version of the (nCSL)x(nCSL) array $resSamples_L$.

2. The decoding process for chroma residual blocks as specified in subclause 8.5.4.2 below is invoked with the luma location (xC, yC), the luma location (xB0, yB0) set equal to (0, 0), the variable log2TrafoSize set equal to log2CbSize, the variable trafoDepth set equal to 0, the variable cIdx set equal to 1, the variable nCS set equal to nCS_C, and the (nCS_C)x(nCS_C) array $resSamples_{Cb}$ as the inputs and the output is a modified version of the (nCS_C)x(nCS_C) array $resSamples_{Cb}$.

3. The decoding process for chroma residual blocks as specified in subclause 8.5.4.2 below is invoked with the luma location (xC, yC), the luma location (xB0, yB0) set equal to (0, 0), the variable log2TrafoSize set equal to log2CbSize, the variable trafoDepth set equal to 0, the variable cIdx set equal to 2, the variable nCS set equal to nCS_C, and the (nCS_C)x(nCS_C) array $resSamples_{Cr}$ as the inputs and the output is a modified version of the (nCS_C)x(nCS_C) array $resSamples_{Cr}$.

8.5.4.1 Decoding process for luma residual blocks

Inputs to this process are:

– a luma location (xC, yC) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a luma location (xB0, yB0) specifying the top-left sample of the current luma block relative to the top-left sample of the current luma coding block,

– a variable log2TrafoSize specifying the size of the current luma block,

– a variable trafoDepth specifying the hierarchy depth of the current luma block relative to the luma coding block,

– a variable nCS specifying the size of the current luma coding block,

– a (nCS)x(nCS) array resSamples of luma residual samples.

Output of this process is:

– a modified version of the (nCS)x(nCS) array of luma residual samples.

Depending split_transform_flag[xC + xB0][yC + yB0][trafoDepth], the following applies:

– If split_transform_flag[xC + xB0][yC + yB0][trafoDepth] is equal to 1, the following ordered steps apply:

2. The variables xB1 and yB1 are derived as follows.
   – The variable xB1 is set equal to xB0 + ((1 << log2TrafoSize) >> 1).
   – The variable yB1 is set equal to yB0 + ((1 << log2TrafoSize) >> 1).
3. The decoding process for luma residual blocks as specified in this subclause is invoked with the luma location (xC, yC), the luma location (xB0, yB0), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the variable nCS, and the (nCS)x(nCS) array resSamples as the inputs and the output is a modified version of the (nCS)x(nCS) array resSamples.
4. The decoding process for luma residual blocks as specified in this subclause is invoked with the luma location (xC, yC), the luma location (xB1, yB0), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the variable nCS, and the (nCS)x(nCS) array resSamples as the inputs and the output is a modified version of the (nCS)x(nCS) array resSamples.
5. The decoding process for luma residual blocks as specified in this subclause is invoked with the luma location (xC, yC), the luma location (xB0, yB1), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the variable nCS, and the (nCS)x(nCS) array resSamples as the inputs and the output is a modified version of the (nCS)x(nCS) array resSamples.
6. The decoding process for luma residual blocks as specified in this subclause is invoked with the luma location (xC, yC), the luma location (xB1, yB1), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the variable nCS, and the (nCS)x(nCS) array resSamples as the inputs and the output is a modified version of the (nCS)x(nCS) array resSamples.

– Otherwise (split_transform_flag[xC + xB0][yC + yB0][trafoDepth] is equal to 0), the following ordered steps apply:

1. The variable nT is set equal to 1 << log2TrafoSize.
2. The scaling and transformation process as specified in subclause 8.6.2 is invoked with the luma location (xC + xB0, yC +yB0), the variable trafoDepth, the variable cIdx set equal to 0 and the transform size trafoSize set equal to nT as the inputs and the output is a (nT)x(nT) array transformBlock.
3. The (nC)x(nC) residual sample array of the current coding block resSamples is modified as follows.

$$resSamples[xB0 + i, yB0 + j] = transformBlock[i, j], \text{ with } i = 0..nT - 1, j = 0..nT - 1 \quad (8\text{-}244)$$

8.5.4.2 Decoding process for chroma residual blocks

Inputs to this process are:

– a luma location (xC, yC) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a luma location (xB0, yB0) specifying the top-left luma sample of the current chroma block relative to the top-left sample of the current luma coding block,

– a variable log2TrafoSize specifying the size of the current chroma block in luma samples,

– a variable trafoDepth specifying the hierarchy depth of the current chroma block relative to the chroma coding block,

– a variable cIdx specifying the chroma component of the current block,

– a variable nCS specifying the size of the current chroma coding block,

– a (nCS)x(nCS) array resSamples of chroma residual samples.

Output of this process is:

– a modified version of the (nCS)x(nCS) array of chroma residual samples.

The variable splitChromaFlag is derived as follows:

– If split_transform_flag[xB0][yB0][trafoDepth] is equal to 1 and log2TrafoSize is greater than 3, splitChromaFlag is set equal to 1.

– Otherwise (split_transform_flag[xB0][yB0][trafoDepth] is equal to 0 or log2TrafoSize is equal to 3), splitChromaFlag is set equal to 0.

Depending splitChromaFlag, the following applies:

– If splitChromaFlag is equal to 1, the following ordered steps apply:

1. The variables xB1 and yB1 are derived as follows.
   – The variable xB1 is set equal to xB0 + ((1 << log2TrafoSize) >> 1).
   – The variable yB1 is set equal to yB0 + ((1 << log2TrafoSize) >> 1).
2. The decoding process for residual chroma blocks as specified in this subclause is invoked with the luma location (xC, yC), the luma location (xB0, yB0), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the variable cIdx, the variable nCS, and the (nCS)x(nCS) array resSamples as the inputs and the output is a modified version of the (nCS)x(nCS) array resSamples.
3. The decoding process for residual chroma blocks as specified in this subclause is invoked with the luma location (xC, yC), the luma location (xB1, yB0), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the variable cIdx, the variable nCS, and the (nCS)x(nCS) array resSamples as the inputs and the output is a modified version of the (nCS)x(nCS) array resSamples.
4. The decoding process for residual chroma blocks as specified in this subclause is invoked with the luma location (xC, yC), the luma location (xB0, yB1), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the variable cIdx, the variable nCS, and the (nCS)x(nCS) array resSamples as the inputs and the output is a modified version of the (nCS)x(nCS) array resSamples.
5. The decoding process for residual chroma blocks as specified in this subclause is invoked with the luma location (xC, yC), the luma location (xB1, yB1), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the variable cIdx, the variable nCS, and the (nCS)x(nCS) array resSamples as the inputs and the output is a modified version of the (nCS)x(nCS) array resSamples.

– Otherwise (splitChromaFlag is equal to 0), the following ordered steps apply:

1. The variable nT is set equal to (1 << log2TrafoSize) >> 1.
2. The scaling and transformation process as specified in subclause 8.6.2 is invoked with the luma location (xC + xB0, yC +yB0), the variable trafoDepth, the variable cIdx and the transform size trafoSize set equal to nT as the inputs and the output is a (nT)x(nT) array transformBlock.
3. The (nC)x(nC) residual sample array of the current coding block resSamples is modified as follows.

   resSamples[(xC+xB0)/2+i, (yC+yB0)/2+j] = transformBlock[i, j], with i = 0..nT − 1, j = 0..nT − 1 (8-245)

8.6 Scaling, transformation and array construction process prior to deblocking filter process

8.6.1 Derivation process for quantization parameters

Input of this process is:

– a luma location (xC, yC) specifying the top-left sample of the current luma coding block relative to the top left luma sample of the current picture.

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 (xC − (xC & ((1 << Log2MinCuQpDeltaSize) − 1))) and (yC − (yC & ((1 << Log2MinCuQpDeltaSize) − 1))), respectively. The luma size of a quantization group, Log2MinCuQpDeltaSize, determines the luma size of the smallest area inside a coding tree block that shares the same qP_{Y_PRED}.

The predicted luma quantization parameter qP_{Y_PRED} is derived by the following ordered steps:

1. The variable qP_{Y_PREV} is derived as follows.
   - If one or more of the following conditions are true, qP_{Y_PREV} is set equal to $SliceQP_Y$.
     - The current quantization group is the first quantization group in a slice.
     - The current quantization group is the first quantization group in a tile.
     - The current quantization group is the first quantization group in a coding tree block row and entropy_coding_sync_enabled_flag is equal to 1.
   - Otherwise, qP_{Y_PREV} is set equal to the luma quantization parameter QP_Y of the the last coding unit in the previous quantization group in decoding order.
2. The availability derivation process for a block in z-scan order as specified in subclause 6.4.1 is invoked with the location (xCurr, yCurr) set equal to (xB, yB) and the neighbouring location (xN, yN) set equal to (xQG−1, yQG) as the input and the output is assigned to availableA. The variable qP_{Y_A} is derived as follows.
   - If availableA is equal to FALSE or the coding tree block address of the coding tree block containing the luma coding block covering (xQG − 1, yQG) ctbAddrA is not equal to CtbAddrInTS, qP_{Y_A} is set equal to qP_{Y_PREV}.

     [Ed. (BB): Insert correct derivation of ctbAddrA using MinTbAddrZS[][].]
   - Otherwise, qP_{Y_A} is set equal to the luma quantization parameter QP_Y of the coding unit containing the luma coding block covering (xQG − 1, yQG).
3. The availability derivation process for a block in z-scan order as specified in subclause 6.4.1 is invoked with the location (xCurr, yCurr) set equal to (xB, yB) and the neighbouring location (xN, yN) set equal to (xQG, yQG−1) as the input and the output is assigned to availableB. The variable qP_{Y_B} is derived as follows.
   - If availableB is equal to FALSE or the coding tree block address of the coding tree block containing the luma coding block covering (xQG, yQG − 1) ctbAddrB is not equal to CtbAddrInTS, qP_{Y_B} is set equal to qP_{Y_PREV}.

     [Ed. (BB): Insert correct derivation of ctbAddrA using MinTbAddrZS[][].]
   - Otherwise, qP_{Y_B} is set equal to the luma quantization parameter QP_Y of the coding unit containing the luma coding block covering (xQG, yQG − 1).
4. The predicted luma quantization parameter qP_{Y_PRED} is derived as:

$$qP_{Y\_PRED} = (qP_{Y\_A} + qP_{Y\_B} + 1) >> 1 \quad (8\text{-}246)$$

The variable QP_Y is derived as

$$QP_Y = ((qP_{Y\_PRED} + CuQpDelta + 52 + 2 * QpBdOffset_Y)\%(52 + QpBdOffset_Y)) - QpBdOffset_Y \quad (8\text{-}247)$$

The luma quantization parameter QP'_Y is derived as

$$QP'_Y = QP_Y + QpBdOffset_Y \quad (8\text{-}248)$$

The variables qP_{Cb} and qP_{Cr} are set equal to the value of QP_C as specified in Table 8-9 based on the index qPi equal to qPi_{Cb} and qPi_{Cr} derived as:

$$qPi_{Cb} = Clip3(-QpBdOffset_C, 57, QP_Y + pps\_cb\_qp\_offset + slice\_cb\_qp\_offset) \quad (8\text{-}249)$$

$$qPi_{Cr} = Clip3(-QpBdOffset_C, 57, QP_Y + pps\_cr\_qp\_offset + slice\_cr\_qp\_offset) \quad (8\text{-}250)$$

The chroma quantization parameters for Cb and Cr components, QP'_{Cb} and QP'_{Cr} are derived as:

$$QP'_{Cb} = qP_{Cb} + QpBdOffset_C \quad (8\text{-}251)$$

$$QP'_{Cr} = qP_{Cr} + QpBdOffset_C \quad (8\text{-}252)$$

Table 8-9 – Specification of QP_C as a function of qPi

qPi	<30	30	31	32	33	34	35	36	37	38	39	40	41	42	43	>43
QP_C	= qPi	29	30	31	32	33	33	34	34	35	35	36	36	37	37	= qPi − 6

8.6.2 Scaling and transformation process

Inputs to this process are:

– a luma location (xT, yT) specifying the top-left sample of the current luma transform block relative to the top-left luma sample of the current picture,

– a variable trafoDepth specifying the hierarchy depth of the current block relative to the coding block,

– a variable cIdx specifying the colour component of the current block,

– a variable nT specifying the size of the current transform block.

Output of this process is:

– the (nT)x(nT) array of residual samples r with elements r[x][y].

The quantization parameter qP is derived as follows.

– If cIdx is equal to 0,

$$qP = QP'_Y \quad (8\text{-}253)$$

– Otherwise, if cIdx is equal to 1,

$$qP = QP'_{Cb} \quad (8\text{-}254)$$

– Otherwise (cIdx is equal to 2),

$$qP = QP'_{Cr} \quad (8\text{-}255)$$

The (nT)x(nT) array of residual samples r is derived as specified as follows:

– If cu_transquant_bypass_flag is equal to 1, the (nT)x(nT) array r is set equal to the (nT)x(nT) array of transform coefficients TransCoeffLevel[xT][yT][cIdx].

– Otherwise, the following ordered steps apply:

1. The scaling process for transform coefficients as specified in subclause 8.6.3 is invoked with the transform block location (xT, yT), the size of the transform block nT, the colour component variable cIdx and the quantization parameter qP as the inputs and the output is a (nT)x(nT) array of scaled transform coefficients d.
2. The (nT)x(nT) array of residual samples r is derived as follows.
   - If transform_skip_flag[xT][yT][cIdx] is equal to 1, the residual sample array values r[x][y] with x = 0..nT − 1, y = 0..nT − 1 are derived as:

$$r[x][y] = (d[x][y] << 7) \quad (8\text{-}256)$$

   - Otherwise (transform_skip_flag[xT][yT][cIdx] is equal to 0), the transformation process for scaled transform coefficients as specified in subclause 8.6.4 is invoked with the transform block location (xT, yT), the size of the transform block nT, the colour component variable cIdx, and the (nT)x(nT) array of scaled transform coefficients d as the inputs and the output is a (nT)x(nT) array of residual samples r.
3. The variable bdShift is derived as:

$$bdShift = (cIdx == 0) ? 20 - BitDepth_Y : 20 - BitDepth_C \quad (8\text{-}257)$$

4. The residual sample values r[x][y] with x = 0..nT − 1, y = 0..nT − 1 are modified as follows.

$$r[x][y] = (r[x][y] + (1 << (bdShift − 1))) >> bdShift \quad (8\text{-}258)$$

8.6.3 Scaling process for transform coefficients

Inputs of this process are:

– a luma location (xT, yT) specifying the top-left sample of the current luma transform block relative to the top-left luma sample of the current picture,

– a variable nT specifying the size of the current transform block,

– a variable cIdx specifying the colour component of the current block,

– a variable qP specifying the quantization parameter.

Output of this process is:

– the (nT)x(nT) array d of scaled transform coefficients with elements d[x][y].

[Ed. (BB): In general we should replace all subscript matrix a_{ij} notations by the a[j][i] array notations to avoid confusion. Then, the corresponding definition in 5.9 Variables, syntax elements, and tables can be removed.]

The variable bdShift is derived as follows:

– If cIdx is equal to 0,

$$bdShift = BitDepth_Y + Log2(nT) − 5 \quad (8\text{-}259)$$

– Otherwise,

$$bdShift = BitDepth_C + Log2(nT) − 5 \quad (8\text{-}260)$$

The list levelScale[ ] is specified as levelScale[k] = { 40, 45, 51, 57, 64, 72 } with k = 0..5.

For the derivation of the scaled transform coefficients d[x][y] with x = 0..nT − 1, y = 0..nT − 1, the following applies.

– The scaling factor m[x][y] is derived as follows.

  – If scaling_list_enable_flag is equal to 0,

$$m[x][y] = 16 \quad (8\text{-}261)$$

  – Otherwise (scaling_list_enable_flag is equal to 1),

$$m[x][y] = ScalingFactor[sizeId][matrixId][x][y] \quad (8\text{-}262)$$

Where sizeId is specified in Table 7-3 for the size of the quantization matrix equal to (nT)x(nT) and matrixId is specified in Table 7-4 for sizeId, CuPredMode[xT][yT] and cIdx, respectively.

– The scaled transform coefficient d[x][y] is derived as follows.

$$d[x][y] = Clip3(−32768, 32767, ((TransCoeffLevel[xT][yT][cIdx][x][y] * m[x][y] * levelScale[qP%6] << (qP/6)) + (1 << (bdShift − 1))) >> bdShift) \quad (8\text{-}263)$$

8.6.4 Transformation process for scaled transform coefficients

Inputs of this process are:

– a luma location (xT, yT) specifying the top-left sample of the current luma transform block relative to the top-left luma sample of the current picture,

– a variable nT specifying the size of the current transform block,

– a variable cIdx specifying the colour component of the current block,

– a (nT)x(nT) array d of scaled transform coefficients with elements d[x][y].

Output of this process is:

– the (nT)x(nT) array r of residual samples with elements r[x][y].

Depending on CuPredMode[xT][yT], nT and cIdx, the variable trType is derived as follows.

– If CuPredMode[xT][yT] is equal to MODE_INTRA, nT is equal to 4, and cIdx is equal to 0, trType is set equal to 1.

– Otherwise, trType is set equal to 0.

The (nT)x(nT) array r of residual samples is derived as follows.

1. Each (vertical) column of scaled transform coefficients d[x][y] with x = 0..nT − 1, y = 0..nT − 1 is transformed to e[x][y] with x = 0..nT − 1, y = 0..nT − 1 by invoking the one-dimensional transformation process as specified in subclause 8.6.4.1 for each column x = 0..nT − 1 with the size of the transform block nT, the list d[x][y] with y = 0..nT − 1 and the transform type variable trType as the inputs and the output is the list e[x][y] with y = 0..nT − 1.

2. The intermediate sample values g[x][y] with x = 0..nT − 1, y = 0..nT − 1 are derived by

$$g[x][y] = Clip3(-32768, 32767, (e[x][y] + 64) >> 7) \quad (8\text{-}264)$$

3. Each (horizontal) row of the resulting array g[x][y] with x = 0..nT − 1, y = 0..nT − 1 is transformed to r[x][y] with x = 0..nT − 1, y = 0..nT − 1 by invoking the one-dimensional transformation process as specified in subclause 8.6.4.1 for each row y = 0..nT − 1 with the size of the transform block nT, the list g[x][y] with x = 0..nT − 1 and the transform type variable trType as the inputs and the output is the list r[x][y] with x = 0..nT − 1.

8.6.4.1 Transformation process

Inputs of this process are:

– a variable nT specifying the sample size of scaled transform coefficients,

– a list of scaled transform coefficients x with elements x[j], with j = 0..nT − 1.

– a transform type variable trType

Output of this process is:

– the list of transformed samples y with elements y[i], with i = 0..nT − 1.

Depending on trType, the following applies:

– If trType is equal to 1, the following transform matrix multiplication applies.

$$y[i] = \sum_j(transMatrix[i][j] * x[j]) \text{ with } i = 0..nT - 1, j = 0..nT - 1, \quad (8\text{-}265)$$

where the transform coefficient array transMatrix is specified as:

transMatrix = (8-266)

```
{
{29  55  74  84}
{74  74   0 -74}
{84 -29 -74  55}
{55 -84  74 -29}
}
```

– Otherwise (trType is equal to 0), the following transform matrix multiplication applies.

$$y[i] = \sum_j(transMatrix[i][k] * x[j]) \text{ with } i = 0..nT - 1, j = 0..nT - 1, \quad (8\text{-}267)$$

where k = (1 << (5 − Log2(nT))) * j and the transform coefficient array transMatrix is specified as:

transMatrix[m][n] = transMatrixCol0to15[m][n] with m = 0..15, n = 0...31 (8-268)

transMatrixCol0to15 = (8-269)

```
{
{64  64  64  64  64  64  64  64  64  64  64  64  64  64  64  64}
{90  90  88  85  82  78  73  67  61  54  46  38  31  22  13   4}
{90  87  80  70  57  43  25   9  -9 -25 -43 -57 -70 -80 -87 -90}
{90  82  67  46  22  -4 -31 -54 -73 -85 -90 -88 -78 -61 -38 -13}
{89  75  50  18 -18 -50 -75 -89 -89 -75 -50 -18  18  50  75  89}
{88  67  31 -13 -54 -82 -90 -78 -46  -4  38  73  90  85  61  22}
{87  57   9 -43 -80 -90 -70 -25  25  70  90  80  43  -9 -57 -87}
{85  46 -13 -67 -90 -73 -22  38  82  88  54  -4 -61 -90 -78 -31}
{83  36 -36 -83 -83 -36  36  83  83  36 -36 -83 -83 -36  36  83}
{82  22 -54 -90 -61  13  78  85  31 -46 -90 -67   4  73  88  38}
{80   9 -70 -87 -25  57  90  43 -43 -90 -57  25  87  70  -9 -80}
{78  -4 -82 -73  13  85  67 -22 -88 -61  31  90  54 -38 -90 -46}
{75 -18 -89 -50  50  89  18 -75 -75  18  89  50 -50 -89 -18  75}
{73 -31 -90 -22  78  67 -38 -90 -13  82  61 -46 -88  -4  85  54}
{70 -43 -87   9  90  25 -80 -57  57  80 -25 -90  -9  87  43 -70}
{67 -54 -78  38  85 -22 -90   4  90  13 -88 -31  82  46 -73 -61}
{64 -64 -64  64  64 -64 -64  64  64 -64 -64  64  64 -64 -64  64}
{61 -73 -46  82  31 -88 -13  90  -4 -90  22  85 -38 -78  54  67}
{57 -80 -25  90  -9 -87  43  70 -70 -43  87   9 -90  25  80 -57}
{54 -85  -4  88 -46 -61  82  13 -90  38  67 -78 -22  90 -31 -73}
{50 -89  18  75 -75 -18  89 -50 -50  89 -18 -75  75  18 -89  50}
{46 -90  38  54 -90  31  61 -88  22  67 -85  13  73 -82   4  78}
{43 -90  57  25 -87  70   9 -80  80  -9 -70  87 -25 -57  90 -43}
{38 -88  73  -4 -67  90 -46 -31  85 -78  13  61 -90  54  22 -82}
{36 -83  83 -36 -36  83 -83  36  36 -83  83 -36 -36  83 -83  36}
{31 -78  90 -61   4  54 -88  82 -38 -22  73 -90  67 -13 -46  85}
{25 -70  90 -80  43   9 -57  87 -87  57  -9 -43  80 -90  70 -25}
{22 -61  85 -90  73 -38  -4  46 -78  90 -82  54 -13 -31  67 -88}
{18 -50  75 -89  89 -75  50 -18 -18  50 -75  89 -89  75 -50  18}
{13 -38  61 -78  88 -90  85 -73  54 -31   4  22 -46  67 -82  90}
{ 9 -25  43 -57  70 -80  87 -90  90 -87  80 -70  57 -43  25  -9}
{ 4 -13  22 -31  38 -46  54 -61  67 -73  78 -82  85 -88  90 -90}
},
```

transMatrix[m][n] = transMatrixCol16to31[m − 16][n] with m = 16..31, n = 0..31, (8-270)

transMatrixCol16to31 = (8-271)

```
{
{ 64  64  64  64  64  64  64  64  64  64  64  64  64  64  64  64}
{ -4 -13 -22 -31 -38 -46 -54 -61 -67 -73 -78 -82 -85 -88 -90 -90}
{-90 -87 -80 -70 -57 -43 -25  -9   9  25  43  57  70  80  87  90}
{ 13  38  61  78  88  90  85  73  54  31   4 -22 -46 -67 -82 -90}
{ 89  75  50  18 -18 -50 -75 -89 -89 -75 -50 -18  18  50  75  89}
{-22 -61 -85 -90 -73 -38   4  46  78  90  82  54  13 -31 -67 -88}
{-87 -57  -9  43  80  90  70  25 -25 -70 -90 -80 -43   9  57  87}
{ 31  78  90  61   4 -54 -88 -82 -38  22  73  90  67  13 -46 -85}
{ 83  36 -36 -83 -83 -36  36  83  83  36 -36 -83 -83 -36  36  83}
{-38 -88 -73  -4  67  90  46 -31 -85 -78 -13  61  90  54 -22 -82}
{-80  -9  70  87  25 -57 -90 -43  43  90  57 -25 -87 -70   9  80}
{ 46  90  38 -54 -90 -31  61  88  22 -67 -85 -13  73  82   4 -78}
{ 75 -18 -89 -50  50  89  18 -75 -75  18  89  50 -50 -89 -18  75}
{-54 -85   4  88  46 -61 -82  13  90  38 -67 -78  22  90  31 -73}
{-70  43  87  -9 -90 -25  80  57 -57 -80  25  90   9 -87 -43  70}
{ 61  73 -46 -82  31  88 -13 -90  -4  90  22 -85 -38  78  54 -67}
{ 64 -64 -64  64  64 -64 -64  64  64 -64 -64  64  64 -64 -64  64}
{-67 -54  78  38 -85 -22  90   4 -90  13  88 -31 -82  46  73 -61}
{-57  80  25 -90   9  87 -43 -70  70  43 -87  -9  90 -25 -80  57}
{ 73  31 -90  22  78 -67 -38  90 -13 -82  61  46 -88   4  85 -54}
{ 50 -89  18  75 -75 -18  89 -50 -50  89 -18 -75  75  18 -89  50}
{-78  -4  82 -73 -13  85 -67 -22  88 -61 -31  90 -54 -38  90 -46}
{-43  90 -57 -25  87 -70  -9  80 -80   9  70 -87  25  57 -90  43}
{ 82 -22 -54  90 -61 -13  78 -85  31  46 -90  67   4 -73  88 -38}
{ 36 -83  83 -36 -36  83 -83  36  36 -83  83 -36 -36  83 -83  36}
{-85  46  13 -67  90 -73  22  38 -82  88 -54  -4  61 -90  78 -31}
{-25  70 -90  80 -43  -9  57 -87  87 -57   9  43 -80  90 -70  25}
{ 88 -67  31  13 -54  82 -90  78 -46   4  38 -73  90 -85  61 -22}
{ 18 -50  75 -89  89 -75  50 -18 -18  50 -75  89 -89  75 -50  18}
{-90  82 -67  46 -22  -4  31 -54  73 -85  90 -88  78 -61  38 -13}
{ -9  25 -43  57 -70  80 -87  90 -90  87 -80  70 -57  43 -25   9}
{ 90 -90  88 -85  82 -78  73 -67  61 -54  46 -38  31 -22  13  -4}
}
```

8.6.5 Picture construction process prior to in-loop filter process

Inputs of this process are:

– a location (xB, yB) specifying the top-left luma sample of the current block relative to the top-left sample of the current picture component,

– a variable nS specifying the size of the current block,

– a variable cIdx specifying the colour component of the current block,

– a (nS)x(nS) array predSamples specifying the predicted samples of the current block,

– a (nS)x(nS) array resSamples specifying the residual samples of the current block.

Depending on the colour component cIdx, the following assignments are made.

– If cIdx is equal to 0, recSamples corresponds to the reconstructed picture sample array S_L and the function clipCidx1 corresponds to $Clip1_Y$.

– Otherwise, if cIdx is equal to 1, recSamples corresponds to the reconstructed chroma sample array S_{Cb} and the function clipCidx1 corresponds to $Clip1_C$.

– Otherwise (cIdx is equal to 2), recSamples corresponds to the reconstructed chroma sample array S_{Cr} and the function clipCidx1 corresponds to $Clip1_C$.

The (nS)x(nS) block of the reconstructed sample array recSamples at location (xB, yB) is derived as follows.

$$recSamples[xB+i][yB+j] = clipCidx1(predSamples[i][j] + resSamples[i][j]) \quad (8\text{-}272)$$
$$\text{with } i = 0..nS - 1,\ j = 0..nS - 1$$

8.7 In-loop filter process

8.7.1 General

The two in-loop filters, namely deblocking filter and sample adaptive offset filter, are applied as specified by the following ordered steps.

1. When slice_disable_deblocking_filter_flag is equal to 0, the following applies.
   – The deblocking filter process as specified in subclause 8.7.2 is invoked with the reconstructed picture sample arrays S_L, S_{Cb}, S_{Cr} as inputs and the modified reconstructed picture sample arrays S'_L, S'_{Cb}, S'_{Cr} after deblocking as outputs.
   – The arrays S'_L, S'_{Cb}, S'_{Cr} are assigned to the arrays S_L, S_{Cb}, S_{Cr} (which represent the decoded picture), respectively.
2. When sample_adaptive_offset_enabled_flag is equal to 1, the following applies.
   – The sample adaptive offset process as specified in subclause 8.7.3 is invoked with the reconstructed picture sample arrays S_L, S_{Cb}, S_{Cr} as inputs and the modified reconstructed picture sample arrays S'_L, S'_{Cb}, S'_{Cr} after sample adaptive offset as outputs.
   – The arrays S'_L, S'_{Cb}, S'_{Cr} are assigned to the arrays S_L, S_{Cb}, S_{Cr} (which represent the decoded picture), respectively.

8.7.2 Deblocking filter process

Inputs of this process are the reconstructed picture sample arrays prior to deblocking $recPicture_L$, $recPicture_{Cb}$ and $recPicture_{Cr}$.

Outputs of this process are the modified reconstructed picture sample arrays after deblocking $recPicture_L$, $recPicture_{Cb}$ and $recPicture_{Cr}$.

The vertical edges in a picture are filtered first. Then the horizontal edges in a picture are filtered with samples modified by deblocking filtering of vertical edges as the input. The vertical and horizontal edges in the coding tree blocks of each coding tree unit 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 shall be applied to all prediction block edges and transform block edges of a picture, except edges at the boundary of the picture, any edges for which the deblocking filter process is disabled by slice_disable_deblocking_filter_flag, any edges that coincide with tile boundaries when loop_filter_across_tiles_enabled_flag is equal to 0, and any edges that coincide with upper or left slice boundaries of a particular slice when slice_loop_filter_across_slices_enabled_flag is equal to 0. For the transform units and prediction units with luma block edges less than 8 samples in either vertical or horizontal direction, only the edges lying on the 8x8 sample grid are filtered.

The deblocking filter process is invoked as follows.

For each coding unit with luma coding block size log2CbSize and location of top left sample of the luma coding block (xC, yC), the vertical edges are filtered by the following ordered steps.

1. The luma coding block size nS is set equal to 1 << log2CbSize.
2. The variable filterLeftCbEdgeFlag is derived as follows.
   - If the left boundary of current luma coding block is the left boundary of the picture, or if the left boundary of current luma coding block is the left boundary of the tile and loop_filter_across_tiles_enabled_flag is equal to 0, or if the left boundary of current luma coding block is the left boundary of the slice and slice_loop_filter_across_slices_enabled_flag is equal to 0, the variable filterLeftCbEdgeFlag is set equal to 0.
   - Otherwise, the variable filterLeftCbEdgeFlag is set equal to 1.
3. All elements of the two-dimensional (nS)x(nS) array verEdgeFlags are initialized to zero.
4. The derivation process of transform block boundary specified in subclause 8.7.2.1 is invoked with the luma location (xC, yC), the luma location (xB, yB) set equal to (0, 0), the transform block size log2TrafoSize set equal to log2CbSize, the variable trafoDepth set equal to 0, the variable filterLeftCbEdgeFlag, and the variable edgeType set equal to EDGE_VER as the inputs and the modified array verEdgeFlags as output.
5. The derivation process of prediction block boundary specified in subclause 8.7.2.2 is invoked with the luma coding block size log2CbSize, the prediction partition mode PartMode, and the variable edgeType set equal to EDGE_VER as inputs, and the modified array verEdgeFlags as output.
6. The derivation process of the boundary filtering strength specified in subclause 8.7.2.3 is invoked with the reconstructed luma picture sample array prior to deblocking $recPicture_L$, the luma location (xC, yC), the luma coding block size log2CbSize, the variable edgeType set equal to EDGE_VER, and verEdgeFlags as inputs and an (nS)x(nS) array verBS as output.
7. The vertical edge filtering process for a coding unit as specified in subclause 8.7.2.4.1 is invoked with the reconstructed picture sample arrays prior to deblocking $recPicture_L$, $recPicture_{Cb}$ and $recPicture_{Cr}$, the luma location (xC, yC), the luma coding block size log2CbSize and the array verBS as inputs and the modified reconstructued picture sample arrays $recPicture_L$, $recPicture_{Cb}$ and $recPicture_{Cr}$ as output.

For each coding unit with luma coding block size log2CbSize and location of top left sample of the luma coding block (xC, yC), the horizontal edges are filtered by the following ordered steps.

1. The luma coding block size nS is set equal to 1 << log2CbSize.
2. The variable filterTopCbEdgeFlag is derived as follows.
   - If the top boundary of current luma coding block is the top boundary of the picture, or if the top boundary of current luma coding block is the top boundary of the tile and loop_filter_across_tiles_enabled_flag is equal to 0, or if the top boundary of current luma coding block is the top boundary of the slice and slice_loop_filter_across_slices_enabled_flag is equal to 0, the variable filterTopCbEdgeFlag is set equal to 0.
   - Otherwise, the variable filterTopCbEdgeFlag is set equal to 1.
3. All elements of the two-dimensional (nS)x(nS) array horEdgeFlags are initialized to zero.
4. The derivation process of transform block boundary specified in subclause 8.7.2.1 is invoked with the luma location (xC, yC), the luma location (xB, yB) set equal to (0, 0), the transform block size log2TrafoSize set equal to log2CbSize, the variable trafoDepth set equal to 0, the variable filterTopCbEdgeFlag, and the variable edgeType set equal to EDGE_HOR as the inputs and the modified array horEdgeFlags as output.

5. The derivation process of prediction block boundary specified in subclause 8.7.2.2 is invoked with the luma coding block size log2CbSize, the prediction partition mode PartMode, and the variable edgeType set equal to EDGE_HOR as inputs, and the modified array horEdgeFlags as output.
6. The derivation process of the boundary filtering strength specified in subclause 8.7.2.3 is invoked with the reconstructed luma picture sample array prior to deblocking $recPicture_L$, the luma location (xC, yC), the luma coding block size log2CbSize, the variable edgeType set equal to EDGE_HOR, and horEdgeFlags as inputs and an (nS)x(nS) array horBS as output.
7. The horizontal edge filtering process for a coding unit as specified in subclause 8.7.2.4.2 is invoked with the modified reconstructed picture sample arrays $recPicture_L$, $recPicture_{Cb}$ and $recPicture_{Cr}$, the luma location (xC, yC), the luma coding block size log2CbSize and the array horBS as inputs and the modified reconstructued picture sample arrays $recPicture_L$, $recPicture_{Cb}$ and $recPicture_{Cr}$ as output.

8.7.2.1 Derivation process of transform block boundary

Inputs of this process are:

– a luma location (xC, yC) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a luma location (xB0, yB0) specifying the top-left sample of the current luma block relative to the top-left sample of the current luma coding block,

– a variable log2TrafoSize specifying the size of the current block,

– a variable trafoDepth,

– a variable filterEdgeFlag,

– a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered.

Output of this process is:

– a two-dimensional (nS)x(nS) array edgeFlags.

Depending on split_transform_flag[xC + xB0][yC + yB0][trafoDepth], the following applies:

– If split_transform_flag[xC + xB0][yC + yB0][trafoDepth] is equal to 1, the following ordered steps apply:

1. The variables xB1 and yB1 are derived as follows.
   – The variable xB1 is set equal to xB0 + ((1 << log2TrafoSize) >> 1).
   – The variable yB1 is set equal to yB0 + ((1 << log2TrafoSize) >> 1).
2. The deriviation process of transform block boundary as specified in this subclause is invoked with the luma location (xC, yC), the luma location (xB0, yB0), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth1 set equal to trafoDepth + 1, the variable filterEdgeFlag and the variable edgeType as inputs and the output is the modified version of array edgeFlags.
3. The deriviation process of transform block boundary as specified in this subclause is invoked with the luma location (xC, yC), the luma location (xB1, yB0), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth1 set equal to trafoDepth + 1, the variable filterEdgeFlag and the variable edgeType as inputs and the output is the modified version of array edgeFlags.
4. The deriviation process of transform block boundary as specified in this subclause is invoked with the luma location (xC, yC), the luma location (xB0, yB1), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth1 set equal to trafoDepth + 1, the variable filterEdgeFlag and the variable edgeType as inputs and the output is the modified version of array edgeFlags.
5. The deriviation process of transform block boundary as specified in this subclause is invoked with the luma location (xC, yC), the luma location (xB1, yB1), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth1 set equal to trafoDepth + 1, the variable filterEdgeFlag and the variable edgeType as inputs and the output is the modified version of array edgeFlags.

– Otherwise (split_transform_flag[xC + xB0][yC + yB0][trafoDepth] is equal to 0), the following applies:
  – If edgeType is equal to EDGE_VER, the value of edgeFlags[xB0][yB0 + k] for k = 0..(1 << log2TrafoSize) − 1 is derived as follows.
    – If xB0 is equal to 0, edgeFlags[xB0][yB0 + k] is set equal to filterEdgeFlag.

- Otherwise edgeFlags[xB0][yB0 + k] is set equal to 1.

- Otherwise (edgeType is equal to EDGE_HOR), the value of edgeFlags[xB0 + k][yB0] for k = 0..(1 << log2TrafoSize) − 1 is derived as follows.
  - If yB0 is equal to 0, edgeFlags[xB0 + k][yB0] is set equal to filterEdgeFlag.
  - Otherwise edgeFlags[xB0 + k][yB0] is set equal to 1.

8.7.2.2 Derivation process of prediction block boundary

Inputs of this process are:

- a variable log2CbSize specifying the luma coding block size,
- a prediction partition mode PartMode,
- a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered.

Output of this process is:

- a two-dimensional (nS)x(nS) array edgeFlags.

Depending on edgeType and PartMode, the following applies for k = 0.. (1 << log2CbSize) − 1:

- If edgeType is equal to EDGE_VER,
  - When PartMode is equal to PART_Nx2N or PART_NxN,
    edgeFlags[1 << (log2CbSize − 1)][k] is set equal to 1.
  - When PartMode is equal to PART_nLx2N,
    edgeFlags[(1 << (log2CbSize − 1)) − (1 << (log2CbSize − 2))][k] is set equal to 1.
  - When PartMode is equal to PART_nRx2N,
    edgeFlags[(1 << (log2CbSize − 1)) + (1 << (log2CbSize − 2))][k] is set equal to 1.
- Otherwise (edgeType is equal to EDGE_HOR),
  - When PartMode is equal to PART_2NxN or PART_NxN,
    edgeFlags[k][1 << (log2CbSize − 1)] is set equal to 1.
  - When PartMode is equal to PART_2NxnU,
    edgeFlags[k][(1 << (log2CbSize − 1)) − (1 << (log2CbSize − 2))] is set equal to 1.
  - When PartMode is equal to PART_2NxnD,
    edgeFlags[k][(1 << (log2CbSize − 1)) + (1 << (log2CbSize − 2))] is set equal to 1.

8.7.2.3 Derivation process of boundary filtering strength

Inputs of this process are:

- a luma picture sample array $recPicture_L$,
- a luma location (xC, yC) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
- a variable log2CbSize specifying the size of the current luma coding block,
- a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered,
- a two-dimensional array of size (nS)x(nS), edgeFlags.

Output of this process is:

- a two-dimensional array of size (nS)x(nS), bS specifying the boundary filtering strength.

The boundary filtering strength array bS for the current coding unit is derived as follows.

The variables xD_i, yD_j, xN and yN are derived as follows.

- If edgeType is equal to EDGE_VER, xD_i is set equal to (i << 3), yD_j is set equal to (j << 2), xN is set equal to (1 << (log2CbSize − 3)) − 1 and yN is set equal to (1 << (log2CbSize − 2)) − 1.
- Otherwise (edgeType is equal to EDGE_HOR), xD_i is set equal to (i << 2), yD_j is set equal to (j << 3), xN is set equal to (1 << (log2CbSize − 2)) − 1 and yN is set equal to (1 << (log2CbSize − 3)) − 1.

For xD_i with i = 0..xN, the following applies.

For yD_j with j = 0..yN, the following applies.

- If edgeFlags[xD_i][yD_j] is equal to 1, the sample values are derived as follows.
  - If edgeType is equal to EDGE_VER, sample $p_0 = recPicture_L[xC + xD_i - 1][yC + yD_j]$ and $q_0 = recPicture_L[xC + xD_i][yC + yD_j]$.
  - Otherwise (edgeType is equal to EDGE_HOR), sample $p_0 = recPicture_L[xC + xD_i][yC + yD_j - 1]$ and $q_0 = recPicture_L[xC + xD_i][yC + yD_j]$.

  Depending on p_0 and q_0, the variable bS[xD_i][yD_j] is derived as follows.
  - If the sample p_0 or q_0 is in the luma coding block of a coding unit coded with intra prediction mode, the variable bS[xD_i][yD_j] is set equal to 2.
  - Otherwise, if the block edge is also a transform block edge and the sample p_0 or q_0 is in a luma transform block which contains one or more non-zero transform coefficient levels, the variable bS[xD_i][yD_j] is set equal to 1.
  - Otherwise, the following applies.
    - If one or more of the following conditions are true, the variable bS[xD_i][yD_j] is set equal to 1.
      - For the prediction of the luma prediction block containing the sample p_0 different reference pictures or a different number of motion vectors are used than for the prediction of the luma prediction block containing the sample q_0.

        NOTE 1 – The determination of whether the reference pictures used for the two luma prediction blocks 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 luma prediction block with lop left luma sample covering (xB, yB), is equal to PredFlagL0[xB, yB] + PredFlagL1[xB, yB].
      - One motion vector is used to predict the luma prediction block containing the sample p_0 and one motion vector is used to predict the luma prediction block containing the sample q_0 and the absolute difference between the horizontal or vertical component of the motion vectors used is greater than or equal to 4 in units of quarter luma samples.
      - Two motion vectors and two different reference pictures are used to predict the luma prediction block containing the sample p_0 and two motion vectors for the same two reference pictures are used to predict the luma prediction block containing the sample q_0 and the absolute difference between the horizontal or vertical component of the two motion vectors used in the prediction of the two luma prediction blocks for the same reference picture is greater than or equal to 4 in units of quarter luma samples,
      - Two motion vectors for the same reference picture are used to predict the luma prediction block containing the sample p_0 and two motion vectors for the same reference picture are used to predict the luma prediction block containing the sample q_0 and all 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 luma prediction bocks is greater than or equal to 4 in quarter 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 luma prediction blocks is greater than or equal to 4 in units of quarter luma samples,
        - The absolute difference between the horizontal or vertical component of list 0 motion vector used in the prediction of the luma prediction block containing the sample p_0 and the list 1 motion vector used in the prediction of the luma prediction block containing the sample q_0 is greater than or equal to 4 in units of quarter luma samples or the absolute difference between the horizontal or vertical component of the list 1 motion vector used in the prediction of the luma prediction block containing the sample p_0 and list 0 motion vector used in the prediction of the luma prediction block containing the sample q_0 is greater than or equal to 4 in units of quarter luma samples.
    - Otherwise (none of the conditions above is true), the variable bS[xD_i][yD_j] is set equal to 0.
- Otherwise (edgeFlags[xD_i][yD_j] is equal to 0), the variable bS[xD_i][yD_j] is set equal to 0.

8.7.2.4 Edge filtering process

8.7.2.4.1 Vertical edge filtering process

Inputs of this process are:

– picture sample arrays $recPicture_L$, $recPicture_{Cb}$ and $recPicture_{Cr}$.

– a luma location (xC, yC) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable log2CbSize specifying the size of the current luma coding block,

– an array bS specifying the boundary filtering strength.

Outputs of this process are:

– the modified picture sample arrays $recPicture_L$, $recPicture_{Cb}$ and $recPicture_{Cr}$.

The filtering process for edges in the luma coding block of the current coding unit consists of the following ordered steps:

1. The variable nD is set equal to 1 << (log2CbSize − 3).
2. For xD_k set equal to k << 3, k = 0..nD − 1, the following applies.

   For yD_m set equal to m << 2, m = 0..nD*2 − 1, the following applies.

   – When bS[xD_k][yD_m] is greater than 0, the following ordered steps apply.

     a. The decision process for luma block edges as specified in subclause 8.7.2.4.3 is invoked with the luma picture sample array $recPicture_L$, the location of the luma coding block (xC, yC), the luma location of the block (xD_k, yD_m), a variable edgeType set equal to EDGE_VER, and the boundary filtering strength bS[xD_k][yD_m] as inputs, the decisions dE, dEp, dEq, and the variables β, t_C as outputs.

     b. The filtering process for luma block edges as specified in subclause 8.7.2.4.4 is invoked with the luma picture sample array $recPicture_L$, the location of the luma coding block (xC, yC), the luma location of the block (xD_k, yD_m), a variable edgeType set equal to EDGE_VER, the decisions dE, dEp, dEq, and the variables β, t_C as inputs and the modified luma picture sample array $recPicture_L$ as output.

The filtering process for edges in the chroma coding blocks of current coding unit consists of the following ordered steps:

1. The variable nD is set equal to 1 << (log2CbSize − 3).
2. For xD_k set equal to k << 2, k = 0..nD − 1, the following applies.

   For yD_m set equal to m << 2, m = 0..nD − 1, the following applies.

   – When bS[xD_k*2][yD_m*2] is greater than 1 and ((xD_k >> 3) << 3) is equal to xD_k, the following ordered steps apply.

     a. The filtering process for chroma block edges as specified in subclause 8.7.2.4.5 is invoked with the chroma picture sample array $recPicture_{Cb}$, the location of the chroma coding block (xC/2, yC/2), the chroma location of the block (xD_k, yD_m), a variable edgeType set equal to EDGE_VER, the boundary filtering strength bS[xD_k*2][yD_m*2], and a variable cQpPicOffset set equal to pps_cb_qp_offset as inputs and the modified chroma picture sample array $recPicture_{Cb}$ as output.

     b. The filtering process for chroma block edges as specified in subclause 8.7.2.4.5 is invoked with the chroma picture sample array $recPicture_{Cr}$, the location of the chroma coding block (xC/2, yC/2), the chroma location of the block (xD_k, yD_m), a variable edgeType set equal to EDGE_VER, the boundary filtering strength bS[xD_k*2][yD_m*2], and a variable cQpPicOffset set equal to pps_cr_qp_offset as inputs and the modified chroma picture sample array $recPicture_{Cr}$ as output.

8.7.2.4.2 Horizontal edge filtering process

Inputs of this process are:

– picture sample arrays $recPicture_L$, $recPicture_{Cb}$ and $recPicture_{Cr}$.

– a luma location (xC, yC) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a variable log2CbSize specifying the size of the current luma coding block,

– an array bS specifying the boundary filtering strength.

Outputs of this process are:

– the modified picture sample arrays $recPicture_L$, $recPicture_{Cb}$ and $recPicture_{Cr}$.

The filtering process for edges in the luma coding block of the current coding unit consists of the following ordered steps:

1. The variable nD is set equal to 1 << (log2CbSize − 3).
2. For yD_m set equal to m << 3, m = 0..nD − 1, the following applies.

   For xD_k set equal to k << 2, k = 0..nD*2 − 1, the following applies.

   – When bS[xD_k][yD_m] is greater than 0, the following ordered steps apply.

     a. The decision process for luma block edges as specified in subclause 8.7.2.4.3 is invoked with the luma picture sample array $recPicture_L$, the location of the luma coding block (xC, yC), the luma location of the block (xD_k, yD_m), a variable edgeType set equal to EDGE_HOR, and the boundary filtering strength bS[xD_k][yD_m] as inputs, the decisions dE, dEp, dEq, and the variables β, t_C as outputs.

     b. The filtering process for luma block edges as specified in subclause 8.7.2.4.4 is invoked with the luma picture sample array $recPicture_L$, the location of the luma coding block (xC, yC), the luma location of the block (xD_k, yD_m), a variable edgeType set equal to EDGE_HOR, the decisions dEp, dEp, dEq, and the variables β, t_C as inputs and the modified luma picture sample array $recPicture_L$ as output.

The filtering process for edges in the chroma coding blocks of current coding unit consists of the following ordered steps:

1. The variable nD is set equal to 1 << (log2CbSize − 3).
2. For yD_m set equal to m << 2, m = 0..nD − 1, the following applies.

   For xD_k set equal to k << 2, k = 0..nD*2 − 1, the following applies.

   – When bS[xD_k*2][yD_m*2] is greater than 1 and ((yD_m >> 3) << 3) is equal to yD_m, the following ordered steps apply.

     a. The filtering process for chroma block edges as specified in subclause 8.7.2.4.5 is invoked with the chroma picture sample array $recPicture_{Cb}$, the location of the chroma coding block (xC/2, yC/2), the chroma location of the block (xD_k, yD_m), a variable edgeType set equal to EDGE_HOR, the boundary filtering strength bS[xD_k*2][yD_m*2], and a variable cQpPicOffset set equal to pps_cb_qp_offset as inputs and the modified chroma picture sample array $recPicture_{Cb}$ as output.

     b. The filtering process for chroma block edges as specified in subclause 8.7.2.4.5 is invoked with the chroma picture sample array $recPicture_{Cr}$, the location of the chroma coding block (xC/2, yC/2), the chroma location of the block (xD_k, yD_m), a variable edgeType set equal to EDGE_HOR, the boundary filtering strength bS[xD_k*2][yD_m*2], and a variable cQpPicOffset set equal to pps_cr_qp_offset as inputs and the modified chroma picture sample array $recPicture_{Cr}$ as output.

8.7.2.4.3 Decision process for luma block edges

Inputs of this process are:

– a luma picture sample array $recPicture_L$,

– a luma location (xC, yC) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a luma location (xB, yB) specifying the top-left sample of the current luma block relative to the top left sample of the current luma 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.

Outputs of this process are:

– the variables dE, dEp, dEq containing decisions,

– the variables β, t_C.

If edgeType is equal to EDGE_VER, the sample values $p_{i,k}$ and $q_{i,k}$ with i = 0..3 and k = 0, 3 are derived as follows:

$$q_{i,k} = recPicture_L[\ xC + xB + i\ ][\ yC + yB + k\ ] \quad (8\text{-}273)$$

$$p_{i,k} = recPicture_L[\ xC + xB - i - 1\ ][\ yC + yB + k\ ] \quad (8\text{-}274)$$

Otherwise (edgeType is equal to EDGE_HOR), the sample values $p_{i,k}$ and $q_{i,k}$ with i = 0..3 and k = 0, 3 are derived as follows:

$$q_{i,k} = recPicture_L[\ xC + xB + k\ ][\ yC + yB + i\ ] \quad (8\text{-}275)$$

$$p_{i,k} = recPicture_L[\ xC + xB + k\ ][\ yC + yB - i - 1\ ] \quad (8\text{-}276)$$

The variables QP_Q and QP_P are set equal to the QP_Y values of the coding units which include the coding blocks containing the sample $q_{0,0}$ and $p_{0,0}$, respectively.

A variable qP_L is derived as follows:

$$qP_L = (\ (\ QP_Q + QP_P + 1\ ) >> 1\ ) \quad (8\text{-}277)$$

The value of the variable β′ is determined as specified in Table 8-10 based on the luma quantization parameter Q derived as:

$$Q = Clip3(\ 0, 51, qP_L + (\ slice\_beta\_offset\_div2 << 1\ )\ ) \quad (8\text{-}278)$$

where slice_beta_offset_div2 is the value of the syntax element slice_beta_offset_div2 for the slice that contains sample $q_{0,0}$.

The variable β is derived as:

$$\beta = \beta' * (1 << (\ BitDepthY - 8\ )\ ) \quad (8\text{-}279)$$

The value of the variable t_C' is determined as specified as Table 8-10 based on the luma quantization parameter Q derived as:

$$Q = Clip3(\ 0, 53, qP_L + 2*(\ bS - 1) + (\ slice\_tc\_offset\_div2 << 1)\ ) \quad (8\text{-}280)$$

where slice_tc_offset_div2 is the value of the syntax element slice_tc_offset_div2 for the slice that contains sample $q_{0,0}$.

The variable t_C is derived as:

$$t_C = t_C' * (\ 1 << (\ BitDepthY - 8\ )\ ) \quad (8\text{-}281)$$

Depending on edgeType, the following applies:

– If edgeType is equal to EDGE_VER, the following ordered steps apply:

1. The variables dpq0, dpq3, dp, dq, and d are derived as follows:

$$dp0 = Abs(\ p_{2,0} - 2*p_{1,0} + p_{0,0}\ ) \quad (8\text{-}282)$$

$$dp3 = Abs(\ p_{2,3} - 2*p_{1,3} + p_{0,3}\ ) \quad (8\text{-}283)$$

$$dq0 = Abs(\ q_{2,0} - 2*q_{1,0} + q_{0,0}\ ) \quad (8\text{-}284)$$

$$dq3 = Abs(\ q_{2,3} - 2*q_{1,3} + q_{0,3}\ ) \quad (8\text{-}285)$$

$$dpq0 = dp0 + dq0 \quad (8\text{-}286)$$

$$dpq3 = dp3 + dq3 \quad (8\text{-}287)$$

$$dp = dp0 + dp3 \quad (8\text{-}288)$$

$$dq = dq0 + dq3 \quad (8\text{-}289)$$

$$d = dpq0 + dpq3 \quad (8\text{-}290)$$

2. The variables dE, dEp and dEq are set equal to 0.
3. When d is less than β, the following ordered steps apply:
   a. The variable dpq is set equal to 2*dpq0.
   b. For the sample location (xC + xB, yC + yB), the decision process for a luma sample as specified in subclause 8.7.2.4.6 is invoked with sample values $p_{i,0}$, $q_{i,0}$ with i = 0..3, the variables dpq, β and t_C as inputs and the output is assigned to the decision dSam0.
   c. The variable dqp is set equal to 2*dpq3.
   d. For the sample location (xC + xB, yC + yB + 3), the decision process for a luma sample as specified in subclause 8.7.2.4.6 is invoked with sample values $p_{i,3}$, $q_{i,3}$ with i = 0..3, the variables dpq, β and t_C as inputs and the output is assigned to the decision dSam3.
   e. The variable dE is set equal to 1.
   f. When dSam0 is equal to 1 and dSam3 is equal to 1, the variable dE is set equal to 2.
   g. When dp is less than (β + (β >> 1)) >> 3, the variable dEp is set equal to 1.
   h. When dq is less than (β + (β >> 1)) >> 3, the variable dEq is set equal to 1.

– Otherwise (edgeType is equal to EDGE_HOR), the following ordered steps apply:

1. The variables dpq0, dpq3, dp, dq, and d are derived as follows:

$$dp0 = Abs(p_{2,0} - 2*p_{1,0} + p_{0,0}) \quad (8\text{-}291)$$

$$dp3 = Abs(p_{2,3} - 2*p_{1,3} + p_{0,3}) \quad (8\text{-}292)$$

$$dq0 = Abs(q_{2,0} - 2*q_{1,0} + q_{0,0}) \quad (8\text{-}293)$$

$$dq3 = Abs(q_{2,3} - 2*q_{1,3} + q_{0,3}) \quad (8\text{-}294)$$

$$dpq0 = dp0 + dq0 \quad (8\text{-}295)$$

$$dpq3 = dp3 + dq3 \quad (8\text{-}296)$$

$$dp = dp0 + dp3 \quad (8\text{-}297)$$

$$dq = dq0 + dq3 \quad (8\text{-}298)$$

$$d = dpq0 + dpq3 \quad (8\text{-}299)$$

2. The variables dE, dEp and dEq are set equal to 0.
3. When d is less than β, the following ordered steps apply:
   a. The variable dqp is set equal to 2*dpq0.
   b. For the sample location (xC + xB, yC + yB), the decision process for a luma sample as specified in subclause 8.7.2.4.6 is invoked with sample values $p_{i,0}$, $q_{i,0}$ with i = 0..3, the variables dpq, β and t_C as inputs and the output is assigned to the decision dSam0.
   c. The variable dqp is set equal to 2*dpq3.
   d. For the sample location (xC + xB + 3, yC + yB), the decision process for a luma sample as specified in subclause 8.7.2.4.6 is invoked with sample values $p_{i,3}$, $q_{i,3}$ with i = 0..3, the variables dpq, β and t_C as inputs and the output is assigned to the decision dSam3.
   e. The variable dE is set equal to 1.
   f. When dSam0 is equal to 1 and dSam3 is equal to 1, the variable dE is set equal to 2.
   g. When dp is less than (β + (β >> 1)) >> 3, the variable dEp is set equal to 1.

h. When dq is less than (β + (β >> 1)) >> 3, the variable dEq is set equal to 1.

Table 8-10 – Derivation of threshold variables β′ and t_C′ from input Q

Q	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
β′	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	7	8
t_C′	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Q	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
β′	9	10	11	12	13	14	15	16	17	18	20	22	24	26	28	30	32	34	36
t_C′	1	1	1	1	1	1	1	1	2	2	2	2	3	3	3	3	4	4	4
Q	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53			
β′	38	40	42	44	46	48	50	52	54	56	58	60	62	64	-	-			
t_C′	5	5	6	6	7	8	9	10	11	13	14	16	18	20	22	24			

8.7.2.4.4 Filtering process for luma block edges

Inputs of this process are:

– a luma picture sample array $recPicture_L$,

– a luma location (xC, yC) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a luma location (xB, yB) specifying the top-left sample of the current luma block relative to the top left sample of the current luma coding block,

– a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered,

– variables dE, dEp, dEq containing decisions,

– variables β, t_C.

Output of this process is:

– the modified luma picture sample array $recPicture_L$.

Depending on edgeType, the following applies:

– If edgeType is equal to EDGE_VER, the following ordered steps apply:

1. The sample values $p_{i,k}$ and $q_{i,k}$ with i = 0..3 and k = 0..3 are derived as follows:

$$q_{i,k} = recPicture_L[\ xC + xB + i\ ][\ yC + yB + k\ ] \quad (8\text{-}300)$$

$$p_{i,k} = recPicture_L[\ xC + xB - i - 1\ ][\ yC + yB + k\ ] \quad (8\text{-}301)$$

2. When dE is not equal to 0, for each sample location (xC + xB, yC + yB + k), k = 0..3, the following ordered steps apply:

   a. The filtering process for a luma sample as specified in subclause 8.7.2.4.7 is invoked with the sample values $p_{i,k}$, $q_{i,k}$ with i = 0..3, the locations (xC + xB +i, yC + yB + k), (xC + xB − i − 1, yC + yB + k) with i = 0..2, the decision dE, variables dEp and dEq, the variable t_C as inputs and the number of filtered samples nDp and nDq from each side of the block boundary, and the filtered sample values p_i' and q_j' as outputs.

   b. When nDp is greater than 0, the filtered sample values p_i' with i = 0..nDp − 1 replace the corresponding samples inside the sample array $recPicture_L$ as follows:

$$recPicture_L[\ xC + xB + k\ ][\ yC + yB - i - 1\ ] = p_i' \quad (8\text{-}302)$$

   c. When nDq is greater than 0, the filtered sample values q_j' with j = 0..nDq − 1 replace the corresponding samples inside the sample array $recPicture_L$ as follows:

$$recPicture_L[\ xC + xB + k\ ][\ yC + yB + j\ ] = q_j' \quad (8\text{-}303)$$

– Otherwise (edgeType is equal to EDGE_HOR), the following ordered steps apply:

1. The sample values $p_{i,k}$ and $q_{i,k}$ with i = 0..3 and k = 0..3 are derived as follows:

$$q_{i,k} = recPicture_L[\ xC + xB + k\ ][\ yC + yB + i\ ] \quad (8\text{-}304)$$

$$p_{i,k} = recPicture_L[\ xC + xB + k\ ][\ yC + yB - i - 1\ ] \quad (8\text{-}305)$$

2. When dE is not equal to 0, for each sample location (xC + xB + k, yC + yB), k = 0..3, the following ordered steps apply:

   a. The filtering process for a luma sample as specified in subclause 8.7.2.4.7 is invoked with the sample values $p_{i,k}$, $q_{i,k}$ with i = 0..3, the locations (xC + xB +k, yC + yB + i), (xC + xB +k, yC + yB − i − 1) with i = 0..2, decision dE, variables dEp and dEq, the variable t_C as inputs and the number of filtered samples nDp and nDq from each side of the block boundary and the filtered sample values p_i' and q_j' as outputs.

   b. When nDp is greater than 0, the filtered sample values p_i' with i = 0..nDp − 1 replace the corresponding samples inside the sample array $recPicture_L$ as follows:

   $$recPicture_L[\ xC + xB + k\ ][\ yC + yB - i - 1\ ] = p_i' \quad (8\text{-}306)$$

   c. When nDq is greater than 0, the filtered sample values q_j' with j = 0..nDq − 1 replace the corresponding samples inside the sample array $recPicture_L$ as follows:

   $$recPicture_L[\ xC + xB + k\ ][\ yC + yB + j\ ] = q_j' \quad (8\text{-}307)$$

8.7.2.4.5 Filtering process for chroma block edges

Inputs of this process are:

– a chroma picture sample array s′,

– a chroma location (xC, yC) 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 (xB, yB) 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 bS specifying the boundary filtering strength,

– a variable cQpPicOffset specifying the picture-level chroma QP offset.

Output of this process is:

– the modified chroma picture sample array s′.

If edgeType is equal to EDGE_VER, the values p_i and q_i with i = 0..1 and k = 0..3 are derived as follows:

$$q_{i,k} = s'[\ xC + xB + i\ ][\ yC + yB + k\ ] \quad (8\text{-}308)$$

$$p_{i,k} = s'[\ xC + xB - i - 1\ ][\ yC + yB + k\ ] \quad (8\text{-}309)$$

Otherwise (edgeType is equal to EDGE_HOR), the sample values p_i and q_i with i = 0..1 and k = 0..3 are derived as follows:

$$q_{i,k} = s'[\ xC + xB + k\ ][\ yC + yB + i\ ] \quad (8\text{-}310)$$

$$p_{i,k} = s'[\ xC + xB + k\ ][\ yC + yB - i - 1\ ] \quad (8\text{-}311)$$

The variables QP_Q and QP_P are set equal to the QP_Y values of the coding units which include the coding blocks containing the sample $q_{0,0}$ and $p_{0,0}$, respectively.

The variable QP_C is determined as specified in Table 8-9 based on the index qPi derived as:

$$qPi = (\ (\ QP_Q + QP_P + 1\ ) >> 1\ ) + cQpPicOffset \quad (8\text{-}312)$$

NOTE – The variable cQpPicOffset provides an adjustment for the value of pps_cb_qp_offset or pps_cr_qp_offset, according to whether the filtered chroma component is the Cb or Cr component. However, to avoid the need to vary the amount of the adjustment within the picture, the filtering process does not include an adjustment for the value of slice_cb_qp_offset or slice_cb_qp_offset.

The value of the variable t_C' is determined as specified as Table 8-10 based on the chroma quantization parameter Q derived as:

$$Q = Clip3(\ 0, 53, QP_C + 2 * (\ bS - 1\ ) + (\ slice\_tc\_offset\_div2 << 1\ )\ ) \quad (8\text{-}313)$$

where slice_tc_offset_div2 is the value of the syntax element slice_tc_offset_div2 for the slice that contains sample $q_{0,0}$.

The variable t_C is derived as:

$$t_C = t_C' * (1 << (BitDepthC - 8)) \quad (8\text{-}314)$$

Depending on edgeType, the following applies:

– If edgeType is equal to EDGE_VER, for each sample location (xC + xB, yC + yB + k), k = 0..3, the following ordered steps apply:

1. The filtering process for a chroma sample as specified in subclause 8.7.2.4.8 is invoked with the sample values $p_{i,k}$, $q_{i,k}$, with i = 0..1, the locations (xC + xB − 1, yC + yB + k) and (xC + xB, yC + yB + k), and the variable t_C as inputs and the filtered sample values p_0' and q_0' as outputs.
2. The filtered sample values p_0' and q_0' replace the corresponding samples inside the sample array s′ as follows:

$$s'[xC + xB][yC + yB + k] = q_0' \quad (8\text{-}315)$$

$$s'[xC + xB - 1][yC + yB + k] = p_0' \quad (8\text{-}316)$$

– Otherwise (edgeType is equal to EDGE_HOR), for each sample location (xC + xB + k, yC + yB), k = 0..3, the following ordered steps apply:

1. The filtering process for a chroma sample as specified in subclause 8.7.2.4.8 is invoked with the sample values $p_{i,k}$, $q_{i,k}$, with i = 0..1, the locations (xC + xB +k, yC + yB − 1) and (xC + xB +k, yC + yB) and the variable t_C as inputs and the filtered sample values p_0' and q_0' as outputs.
2. The filtered sample values p_0' and q_0' replace the corresponding samples inside the sample array s′ as follows:

$$s'[xC + xB +k][yC + yB] = q_0' \quad (8\text{-}317)$$

$$s'[xC + xB +k][yC + yB - 1] = p_0' \quad (8\text{-}318)$$

8.7.2.4.6 Decision process for a luma sample

Inputs of this process are:

– sample values, p_i and q_i with i = 0..3,

– variables dpq, β and t_C.

Output of this process is:

– a variable dSam containing a decision

The variable dSam is specified as follows:

– If dpq is less than (β >> 2), Abs($p_3 - p_0$) + Abs($q_0 - q_3$) is less than (β >> 3) and Abs($p_0 - q_0$) is less than (5*t_C + 1) >> 1, dSam is set equal to 1.

– Otherwise, dSam is set equal to 0.

8.7.2.4.7 Filtering process for a luma sample

Inputs of this process are:

– luma sample values, p_i and q_i with i = 0..3,

– luma locations of p_i and q_i, (xP_i, yP_i) and (xQ_i, yQ_i) with i = 0..2,

– a variable dE,

– variables dEp and dEq containing decisions to filter samples p1 and q1 respectively,

– a variable t_C.

Output of this process is:

– number of filtered samples nDp and nDq,

– filtered sample values, p_i' and q_j' with i = 0..nDp − 1, j = 0..nDq − 1

Depending on dE, the following applies:

– If the variable dE is equal to 2, the following strong filtering applies while nDp and nDq are set equal to 3:

$$p_0' = Clip3(p_0 - 2*t_C, p_0 + 2*t_C, (p_2 + 2*p_1 + 2*p_0 + 2*q_0 + q_1 + 4) >> 3) \quad (8\text{-}319)$$

$p_1' = Clip3(p_1-2*t_C, p_1+2*t_C, (p_2 + p_1 + p_0 + q_0 + 2) >> 2)$ (8-320)

$p_2' = Clip3(p_2-2*t_C, p_2+2*t_C, (2*p_3 + 3*p_2 + p_1 + p_0 + q_0 + 4) >> 3)$ (8-321)

$q_0' = Clip3(q_0-2*t_C, q_0+2*t_C, (p_1 + 2*p_0 + 2*q_0 + 2*q_1 + q_2 + 4) >> 3)$ (8-322)

$q_1' = Clip3(q_1-2*t_C, q_1+2*t_C, (p_0 + q_0 + q_1 + q_2 + 2) >> 2)$ (8-323)

$q_2' = Clip3(q_2-2*t_C, q_2+2*t_C, (p_0 + q_0 + q_1 + 3*q_2 + 2*q_3 + 4) >> 3)$ (8-324)

– Otherwise, nDp and nDq are set equal to 0 and the following weak filtering applies:

$\Delta = (9 * (q_0 - p_0) - 3 * (q_1 - p_1) + 8) >> 4$ (8-325)

  – When Abs(Δ) is less than t_C*10, the following ordered steps apply:

    – The filtered sample values p_0' and q_0' are specified as follows:

$\Delta = Clip3(-t_C, t_C, \Delta)$ (8-326)

$p_0' = Clip1_Y(p_0 + \Delta)$ (8-327)

$q_0' = Clip1_Y(q_0 - \Delta)$ (8-328)

    – When dEp is equal to 1, the filtered sample value p_1' is specified as follows:

$\Delta p = Clip3(-(t_C >> 1), t_C >> 1, (((p_2 + p_0 + 1) >> 1) - p_1 + \Delta) >>1)$ (8-329)

$p_1' = Clip1_Y(p_1 + \Delta p)$ (8-330)

    – When dEq is equal to 1, the filtered sample value q_1' is specified as follows:

$\Delta q = Clip3(-(t_C >> 1), t_C >> 1, (((q_2 + q_0 + 1) >> 1) - q_1 - \Delta) >>1)$ (8-331)

$q_1' = Clip1_Y(q_1 + \Delta q)$ (8-332)

    – nDp is set equal to dEp+1 and nDq is set equal to dEq+1.

When nDp is greater than 0 and one or more of the following conditions are true for i = 0..nDp−1, nDp is set equal to 0 [Ed. (GJS): It may be preferable to restructure this logic so that instead of computing a filtered p_i' and then replacing it with an unfiltered p_i, we would just not bother computing the filtered p_i' in the first place.]

– pcm_loop_filter_disable_flag is equal to 1 and pcm_flag[xP_i][yP_i] is equal to 1.

– cu_transquant_bypass_flag of the coding unit which includes the coding block containing the sample p_i is equal to 1.

When nDq is greater than 0 and one or more of the following conditions are true for j = 0..nDq−1, nDq is set equal to 0.

– pcm_loop_filter_disable_flag is equal to 1 and pcm_flag[xQ_j][yQ_j].

– cu_transquant_bypass_flag of the coding unit which includes the coding block containing the sample q_j is equal to 1.

8.7.2.4.8 Filtering process for a chroma sample

Inputs of this process are:

– chroma sample values, p_i and q_i with i = 0..1,

– chroma locations of p_0 and q_0, (xP_0, yP_0) and (xQ_0, yQ_0),

– a variable t_C.

Output of this process is:

– The filtered sample values, p_0' and q_0'.

The filtered sample values p_0' and q_0' are derived by

$$\Delta = \text{Clip3}(-t_C, t_C, ((((q_0 - p_0) << 2) + p_1 - q_1 + 4) >> 3)) \quad (8\text{-}333)$$

$$p_0' = \text{Clip1}_C(p_0 + \Delta) \quad (8\text{-}334)$$

$$q_0' = \text{Clip1}_C(q_0 - \Delta) \quad (8\text{-}335)$$

When one or more of the following conditions are true, the filtered sample value, p_0' is substituted by the corresponding input sample value p_0.

– pcm_loop_filter_disable_flag is equal to 1 and pcm_flag[2 * xP_0][2 * yP_0] is equal to 1.

– cu_transquant_bypass_flag of the coding unit which includes the coding block containing the sample p_0 is equal to 1.

When one or more of the following conditions are true, the filtered sample value, q_0' is substituted by the corresponding input sample value q_0.

– pcm_loop_filter_disable_flag is equal to 1 and pcm_flag[2 * xQ_0][2 * yQ_0] is equal to 1.

– cu_transquant_bypass_flag of the coding unit which includes the coding block containing the sample q_0 is equal to 1.

8.7.3 Sample adaptive offset process

8.7.3.1 General

Inputs of this process are the reconstructed picture sample arrays prior to sample adaptive offset recPicture_L, recPicture_{Cb} and recPicture_{Cr}.

Outputs of this process are the modified reconstructed picture sample arrays after sample adaptive offset saoPicture_L, saoPicture_{Cb} and saoPicture_{Cr}.

This process is performed on a coding tree block basis after the completion of the deblocking filter process for the decoded picture.

The sample values in the modified reconstructed picture sample arrays after sample adaptive offset saoPicture_L, saoPicture_{Cb} and saoPicture_{Cr} are initially set equal to the sample values in the reconstructed picture sample arrays prior to sample adaptive offset recPicture_L, recPicture_{Cb} and recPicture_{Cr}.

For every coding tree unit with coding tree block location (rx, ry), where rx = 0..PicWidthInCtbsY − 1 and ry = 0..PicHeightInCtbsY − 1, the following applies:

– When slice_sao_luma_flag of the current slice is equal to 1, the coding tree block modification process as specified in subclause 8.7.3.2 is invoked with recPicture set equal to recPicture_L, cIdx set equal to 0, (rx, ry) and nS set equal to (1 << Log2CtbSizeY) as inputs and the modified luma picture sample array saoPicture_L as output.

– When slice_sao_chroma_flag of the current slice is equal to 1, the coding tree block modification process as specified in subclause 8.7.3.2 is invoked with recPicture set equal to recPicture_{Cb}, cIdx set equal to 1, (rx, ry) and nS set equal to (1 << (Log2CtbSizeY − 1)) as inputs and the modified chroma picture sample array saoPicture_{Cb} as output.

– When slice_sao_chroma_flag of the current slice is equal to 1, the coding tree block modification process as specified in subclause 8.7.3.2 is invoked with recPicture set equal to recPicture_{Cr}, cIdx set equal to 2, (rx, ry) and nS set equal to (1 << (Log2CtbSizeY − 1)) as inputs and the modified chroma picture sample array saoPicture_{Cr} as output.

8.7.3.2 Coding tree block modification process

Inputs to this process are:

– picture sample array recPicture for the colour component cIdx,

– a variable cIdx specifying colour component index,

– a pair of variables (rx, ry) specifying the coding tree block location,

– a coding tree block size nS.

Output of this process is a modified picture sample array saoPicture for the colour component cIdx.

The variable bitDepth is derived as follows.

– If cIdx is equal to 0, bitDepth is set equal to $BitDepth_Y$.

– Otherwise, bitDepth is set equal to $BitDepth_C$.

The variables xC and yC are set equal to rx*nS and ry*nS, respectively.

For i = 0..nS−1 and j = 0..nS−1, depending on the value of pcm_loop_filter_disable_flag, pcm_flag[xC + i][yC + j], and cu_transquant_bypass_flag of the coding unit which includes the coding block covering recPicture[xC + i][yC + j], the following applies:

– If one or more of the following conditions are true, saoPicture[xC + i][yC + j] is not modified.

  − pcm_loop_filter_disable_flag and pcm_flag[xC + i][yC + j] are both equal to 1.

  − cu_transquant_bypass_flag is equal to 1.

  − SaoTypeIdx[cIdx][rx][ry] is 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 8-11 based on SaoEoClass[cIdx][rx][ry].

  2. The variable edgeIdx is derived as follows.

     − If one or more of the following conditions for (xS, xS) = (xC + i + hPos[k], yC + j + vPos[k]), k = 0..1 are true, edgeIdx is set equal to 0.

       − The sample at location (xS, yS) is outside picture boundary

       − The sample at location (xS, yS) belongs to a different slice and one of the following two conditions is true:

         − MinTbAddrZS[xS >> Log2MinTrafoSize][yS >> Log2MinTrafoSize] is less than MinTbAddrZS[(xC + i)>>Log2MinTrafoSize][(yC + j)>>Log2MinTrafoSize] and slice_loop_filter_across_slices_enabled_flag in the slice which the sample recPicture[xC + i][yC + j] belongs to is equal to 0.

         − MinTbAddrZS[(xC + i)>>Log2MinTrafoSize][(yC + j)>>Log2MinTrafoSize] is less than MinTbAddrZS[xS >> Log2MinTrafoSize][yS >> Log2MinTrafoSize] and slice_loop_filter_across_slices_enabled_flag in the slice which the sample recPicture[xS][yS] belongs to is equal to 0.

       − loop_filter_across_tiles_enabled_flag is equal to 0 and the sample at location (xS, yS) belongs to a different tile.

     − Otherwise, edgeIdx is derived as follows.

$$edgeIdx = 2 + \sum_k (Sign(recPicture[xC + i][yC + j] - recPicture[xC + i + hPos[k]][yC + j + vPos[k]])) \text{ with } k = 0..1 \quad (8\text{-}336)$$

       When edgeIdx is equal to 0, 1, or 2, it is modified as follows.

$$edgeIdx = (edgeIdx == 2) ? 0 : (edgeIdx + 1) \quad (8\text{-}337)$$

  3. The modified picture sample array saoPicture[xC + i][yC + j] is derived as follows.

$$saoPicture[xC + i][yC + j] = Clip3(0, (1 << bitDepth) - 1, recPicture[xC + i][yC + j] + SaoOffsetVal[cIdx][rx][ry][edgeIdx]) \quad (8\text{-}338)$$

− 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 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
```
(8-339)

4. The variable bandIdx is set equal to bandTable[recPicture[xC + i][yC + j] >> bandShift].
5. The modified picture sample array saoPicture[xC + i][yC + j] is derived as follows.

$$saoPicture[xC + i][yC + j] = Clip3(0, (1 << bitDepth) - 1, recPicture[xC + i][yC + j] + SaoOffsetVal[cIdx][rx][ry][bandIdx]) \quad (8\text{-}340)$$

Table 8-11 – 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

9 Parsing process

Inputs to this process are bits from the RBSP.

Outputs of this process are syntax element values.

This process is invoked when the descriptor of a syntax element in the syntax tables in subclause 7.3 is equal to ue(v), se(v) (see subclause 9.1), or ae(v) (see subclause 9.2).

9.1 Parsing process for 0-th order Exp-Golomb codes

This process is invoked when the descriptor of a syntax element in the syntax tables in subclause 7.3 is equal to ue(v) 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. The parsing process for these syntax elements begins with reading the bits starting at the current location in the bitstream up to and including the first non-zero bit, and counting the number of leading bits that are equal to 0. This process is specified as follows:

```
leadingZeroBits = −1
for( b = 0; !b; leadingZeroBits++ )                                    (9-1)
    b = read_bits( 1 )
```

The variable codeNum is then assigned as follows:

$$codeNum = 2^{leadingZeroBits} - 1 + read\_bits(leadingZeroBits) \quad (9\text{-}2)$$

where the value returned from read_bits(leadingZeroBits) is interpreted as a binary representation of an unsigned integer with most significant bit written first.

Table 9-1 illustrates the structure of the Exp-Golomb code by separating the bit string into "prefix" and "suffix" bits. The "prefix" bits are those bits that are parsed in the above pseudo-code for the computation of leadingZeroBits, and are shown as either 0 or 1 in the bit string column of Table 9-1. The "suffix" bits are those bits that are parsed in the computation of codeNum and are shown as x_i in Table 9-1, with i being in the range 0 to leadingZeroBits − 1, inclusive. Each x_i can take on values 0 or 1.

Table 9-1 – Bit strings with "prefix" and "suffix" bits and assignment to codeNum ranges (informative)

Bit string form	Range of codeNum
1	0
0 1 x_0	1..2
0 0 1 x_1 x_0	3..6
0 0 0 1 x_2 x_1 x_0	7..14
0 0 0 0 1 x_3 x_2 x_1 x_0	15..30
0 0 0 0 0 1 x_4 x_3 x_2 x_1 x_0	31..62
…	…

Table 9-2 illustrates explicitly the assignment of bit strings to codeNum values.

Table 9-2 – Exp-Golomb bit strings and codeNum in explicit form and used as ue(v) (informative)

Bit string	codeNum
1	0
0 1 0	1
0 1 1	2
0 0 1 0 0	3
0 0 1 0 1	4
0 0 1 1 0	5
0 0 1 1 1	6
0 0 0 1 0 0 0	7
0 0 0 1 0 0 1	8
0 0 0 1 0 1 0	9
…	…

Depending on the descriptor, the value of a syntax element is derived as follows.

– If the syntax element is coded as ue(v), the value of the syntax element is equal to codeNum.

– Otherwise, if the syntax element is coded as se(v), the value of the syntax element is derived by invoking the mapping process for signed Exp-Golomb codes as specified in subclause 9.1.1 with codeNum as the input.

9.1.1 Mapping process for signed Exp-Golomb codes

Input to this process is codeNum as specified in subclause 9.1.

Output of this process is a value of a syntax element coded as se(v).

The syntax element is assigned to the codeNum by ordering the syntax element by its absolute value in increasing order and representing the positive value for a given absolute value with the lower codeNum. Table 9-3 provides the assignment rule.

Table 9-3 – Assignment of syntax element to codeNum for signed Exp-Golomb coded syntax elements se(v)

codeNum	syntax element value
0	0
1	1
2	−1
3	2
4	−2
5	3
6	−3
k	$(-1)^{k+1}$ Ceil(k÷2)

9.2 CABAC parsing process for slice segment data

This process is invoked when parsing syntax elements with descriptor ae(v) in subclauses 7.3.9 to 7.3.9.11.

Inputs to this process are a request for a value of a syntax element and values of prior parsed syntax elements.

Output of this process is the value of the syntax element.

When one or more of the following conditions are true, the initialization process of the CABAC parsing process is invoked as specified in subclause 9.2.1

– when starting the parsing of the slice segment data of a slice segment in subclause 7.3.9

– when starting the parsing of the coding tree unit syntax in subclause 7.3.9.2 and the coding tree unit is the first coding tree unit in a tile

– when starting the parsing of the coding tree unit syntax in subclause 7.3.9.2, entropy_coding_sync_enabled_flag is equal to 1 and the associated luma coding tree block is the first luma coding tree block in a row

The parsing of syntax elements proceeds as follows:

For each requested value of a syntax element a binarization is derived as described in subclause 9.2.2.

The binarization for the syntax element and the sequence of parsed bins determines the decoding process flow as described in subclause 9.2.3. For each bin of the binarization of the syntax element, which is indexed by the variable binIdx, a context index ctxIdx is derived as specified in subclause 9.2.3.1. For each ctxIdx the arithmetic decoding process is invoked as specified in subclause 9.2.3.2.

The resulting sequence ($b_0..b_{binIdx}$) of parsed bins is compared to the set of bin strings given by the binarization process after decoding of each bin. When the sequence matches a bin string in the given set, the corresponding value is assigned to the syntax element.

In case the request for a value of a syntax element is processed for the syntax element pcm_flag and the decoded value of pcm_flag is equal to 1, the decoding engine is initialized as specified in subclause 9.2.1.4.

When ending the parsing of the coding tree unit syntax in subclause 7.3.9.2, the memorization process for context variables is applied as follows.

– When entropy_coding_sync_enabled_flag is equal to 1 and CtbAddrInRS % PicWidthInCtbsY is equal to 1, the memorization process for context variables as specified in subclause 9.2.1.2 is invoked with TableStateIdxWPP and TableMPSValWPP as output.

– When dependent_slice_segment_flag is equal to 1, and end_of_slice_segment_flag is equal to 1, the memorization process for context variables as specified in subclause 9.2.1.2 is invoked with TableStateIdxDS and TableMPSValDS as output.

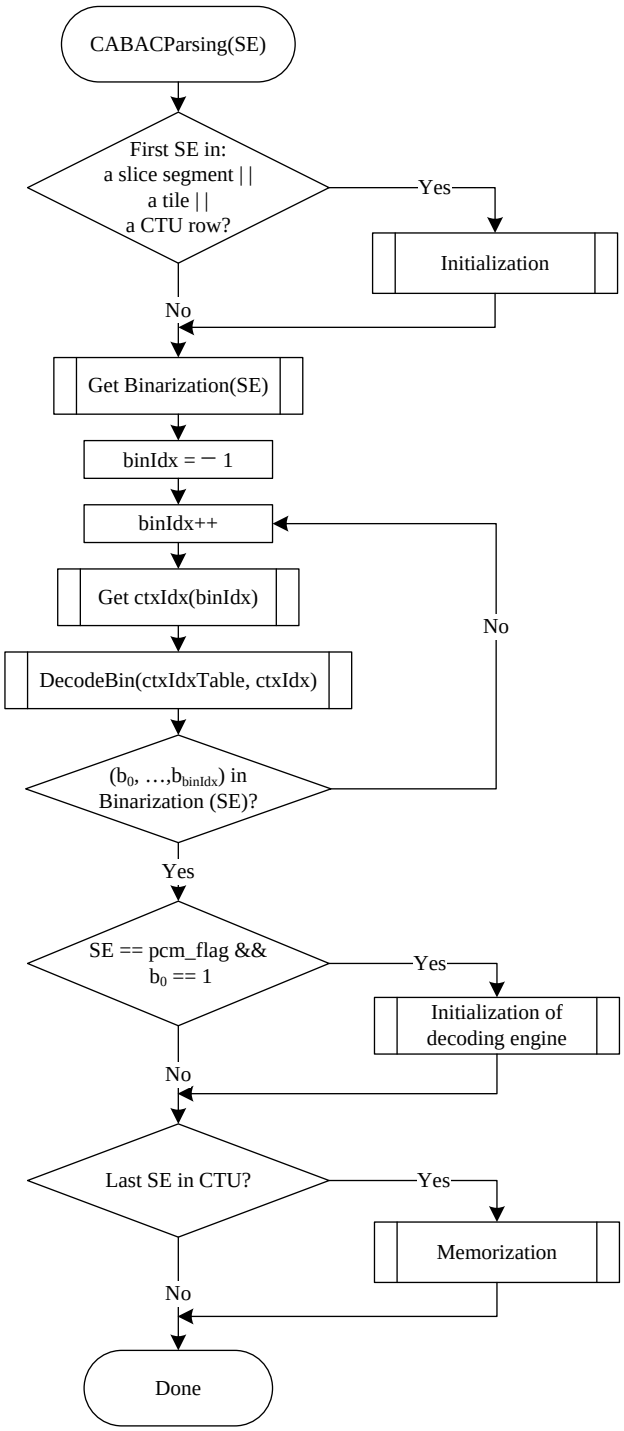


Figure 9-1 – Illustration of CABAC parsing process for a syntax element SE (informative)

9.2.1 Initialization process

This process is invoked when one or more of the follwing conditions are true.

– when starting the parsing of the slice segment data of a slice segment in subclause 7.3.9

– when starting the parsing of the coding tree unit syntax in subclause 7.3.9.2 and the coding tree unit is the first coding tree unit in a tile

– when starting the parsing of the coding tree unit syntax in subclause 7.3.9.2, entropy_coding_sync_enabled_flag is equal to 1 and the associated luma coding tree block is the first luma coding tree block in a row

Outputs of this process are initialized CABAC internal variables.

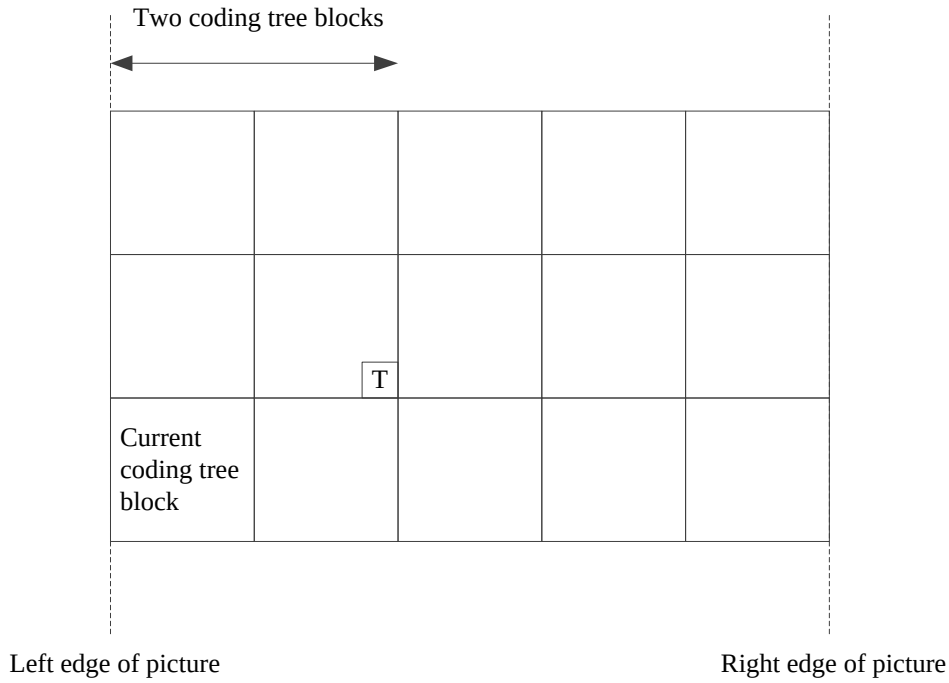


Figure 9-2 – Spatial neighbour T that is used to invoke the coding tree block availability derivation process relative to the current coding tree block (informative)

The context variables of the arithmetic decoding engine are initialized as follows.

– If entropy_coding_sync_enabled_flag is equal to 1, and CtbAddrInRS % PicWidthInCtbsY is equal to 0, the following applies.

  – The location (xT, yT) of the top-left luma sample of the spatial neighbouring block T (Figure 9-2) is derived using the location (x0, y0) of the top-left luma sample of the current coding tree block as follows.

$$(xT, yT) = (x0 + 2 << Log2CtbSizeY - 1, y0 - 1) \quad (9\text{-}3)$$

  – The availability derivation process for a block in z-scan order as specified in subclause 6.4.1 is invoked with the location (xCurr, yCurr) set equal to (x0, y0) and the neighbouring location (xN, yN) set equal to (xT, yT) as the input and the output is assigned to availableFlagT.

  – The the synchronization process for context variables is invoked as follows.

    – If availableFlagT is equal to 1, the synchronization process for context variables as specified in subclause 9.2.1.3 is invoked with TableStateIdxWPP and TableMPSValWPP as input.

    – Otherwise, the initialization process for context variables is invoked as specified in subclause 9.2.1.1.

– Otherwise, if CtbAddrInRS is equal to slice_segment_address and dependent_slice_segment_flag is equal to 1, the synchronization process for context variables as specified in subclause 9.2.1.3 is invoked with TableStateIdxDS and TableMPSValDS as input.

– Otherwise the initialization process for context variables is invoked as specified in subclause 9.2.1.1.

The initialization process for the arithmetic decoding engine is invoked as specified in subclause 9.2.1.4.

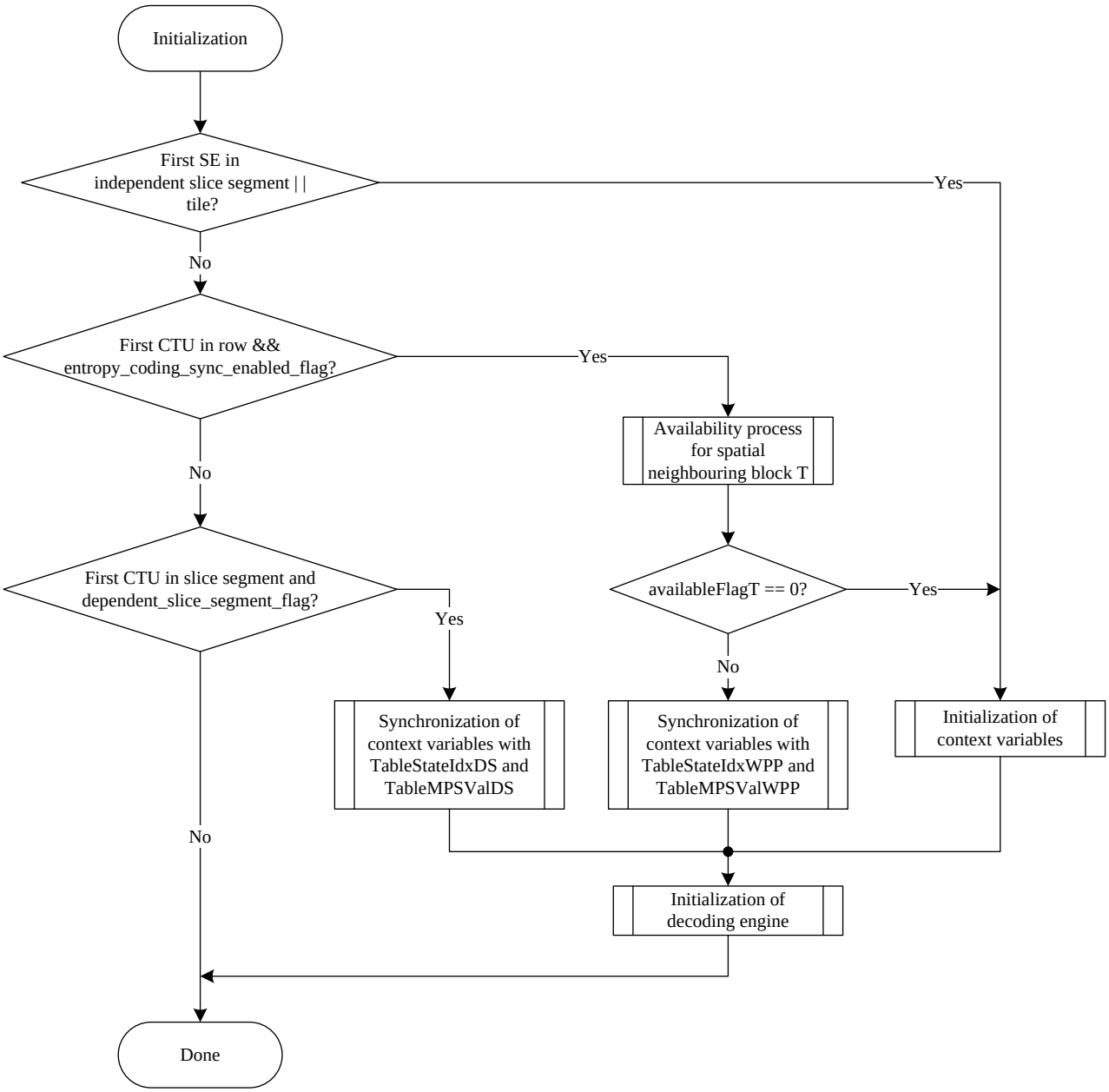


Figure 9-3 – Illustration of CABAC initialization process (informative)

9.2.1.1 Initialization process for context variables

Outputs of this process are the initialized CABAC context variables indexed by ctxIdxTable and ctxIdx.

Table 9-5 to Table 9-31 contain the values of the 8 bit variable initValue used in the initialization of context variables that are assigned to all syntax elements in subclauses 7.3.9 to 7.3.9.11 except for end_of_slice_segment_flag, end_of_sub_stream_one_bit and pcm_flag.

For each context variable, the two variables pStateIdx and valMPS are initialized.

NOTE 1 – The variable pStateIdx corresponds to a probability state index and the variable valMPS corresponds to the value of the most probable symbol as further described in subclause 9.2.3.2.

From the 8 bit table entry initValue, the two 4 bit variables slopeIdx and intersecIdx are derived according to the following pseudo-code process:

```
slopeIdx = initValue >> 4
intersecIdx = initValue & 15
```

Slope m and Intersec n are derived from the indices as follows:

```
m = slopeIdx*5 − 45
n = ( intersecIdx << 3 ) − 16
```

The two values assigned to pStateIdx and valMPS for the initialization are derived from $SliceQP_Y$, which is derived in Equation 7-48. Given the variable m and n, the initialization is specified by the following pseudo-code process:

```
preCtxState = Clip3( 1, 126, ( ( m * Clip3( 0, 51, SliceQPY ) ) >> 4 ) + n )
valMPS = ( preCtxState  <=  63) ? 0 : 1
pStateIdx = valMPS ? (preCtxState − 64) : (63 − preCtxState)                    (9-4)
```

In Table 9-4, 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 initialisation. For P and B slice type, the derivation of initType depends also on the value of the cabac_init_flag syntax element. The variable initType is derived as follows:

```
if( slice_type = = I )
    initType = 0
else if(slice_type = = P )
    initType = cabac_init_flag ? 2 : 1
else
    initType = cabac_init_flag ? 1 : 2
```

Table 9-4 – Association of ctxIdx and syntax elements for each initializationType in the initialization process

	Syntax element	ctxIdxTable	initType		
			0	1	2
sao( )	sao_merge_left_flag sao_merge_up_flag	Table 9-5	0	1	2
	sao_type_idx_luma sao_type_idx_chroma	Table 9-6	0	1	2
coding_quadtree( )	split_cu_flag	Table 9-7	0..2	3..5	6..8
coding_unit( )	cu_transquant_bypass_flag	Table 9-8	0	1	2
	cu_skip_flag	Table 9-9		0..2	3..5
	cu_qp_delta_abs	Table 9-10	0..1	2..3	4..5
	pred_mode_flag	Table 9-11		0	1
	part_mode	Table 9-12	0	1..4	5..8
prediction_unit( )	prev_intra_luma_pred_flag	Table 9-13	0	1	2
	intra_chroma_pred_mode	Table 9-14	0	1	2
	merge_flag	Table 9-15		0	1
	merge_idx	Table 9-16		0	1
	inter_pred_idc	Table 9-17		0..4	5..9
	ref_idx_l0, ref_idx_l1	Table 9-18		0..1	2..3
	abs_mvd_greater0_flag	Table 9-19		0	2
	abs_mvd_greater1_flag	Table 9-19		1	3
	mvp_l0_flag, mvp_l1_flag	Table 9-20		0	1
transform_tree( )	rqt_root_cbf	Table 9-21		0	1
	split_transform_flag	Table 9-22	0..2	3..5	6..8
	cbf_luma	Table 9-23	0..1	2..3	4..5
	cbf_cb, cbf_cr	Table 9-24	0..3	4..7	8..11
residual_coding( )	transform_skip_flag[ ][ ][0]	Table 9-25	0	1	2
	transform_skip_flag[ ][ ][1] transform_skip_flag[ ][ ][2]	Table 9-25	3	4	5
	last_significant_coeff_x_prefix	Table 9-26	0..17	18..35	36..53
	last_significant_coeff_y_prefix	Table 9-27	0..17	18..35	36..53
	coded_sub_block_flag	Table 9-28	0..3	4..7	8..11
	significant_coeff_flag	Table 9-29	0..41	42..83	84..125
	coeff_abs_level_greater1_flag	Table 9-30	0..23	24..47	48..71
	coeff_abs_level_greater2_flag	Table 9-31	0..5	6..11	12..17

NOTE 2 – ctxIdxTable equal to 0 and ctxIdx equal to 0 are associated with end_of_slice_segment_flag, end_of_sub_stream_one_bit and pcm_flag. The decoding process specified in subclause 9.2.3.2.4 applies to ctxIdxTable equal to 0 and ctxIdx equal to 0. This decoding process, however, may also be implemented by using the decoding process specified in subclause 9.2.3.2.1. In this case, the initial values associated with ctxIdxTable equal to 0 and ctxIdx equal to 0 are specified to be pStateIdx = 63 and valMPS = 0, where pStateIdx = 63 represents a non-adapting probability state.

Table 9-5 – Values of variable initValue for sao_merge_left_flag and sao_merge_up_flagctxIdx

Initialization variable	sao_merge_left_flag, sao_merge_up_flag ctxIdx		
	0	1	2
initValue	153	153	153

Table 9-6 – Values of variable initValue for sao_type_idx_luma and sao_type_idx_chroma ctxIdx

Initialization variable	sao_type_idx_luma, sao_type_idx_chroma ctxIdx		
	0	1	2
initValue	200	185	160

Table 9-7 – Values of variable initValue for split_cu_flag ctxIdx

Initialization variable	split_cu_flag ctxIdx								
	0	1	2	3	4	5	6	7	8
initValue	139	141	157	107	139	126	107	139	126

Table 9-8 – Values of variable initValue for cu_transquant_bypass_flag ctxIdx

Initialization variable	cu_transquant_bypass_flag ctxIdx		
	0	1	2
initValue	154	154	154

Table 9-9 – Values of variable initValue for cu_skip_flag ctxIdx

Initialization variable	cu_skip_flag ctxIdx					
	0	1	2	3	4	5
initValue	197	185	201	197	185	201

Table 9-10 – Values of variable initValue for cu_qp_delta_abs ctxIdx

Initialization variable	cu_qp_delta_abs ctxIdx					
	0	1	2	3	4	5
initValue	154	154	154	154	154	154

Table 9-11 – Values of variable initValue for pred_mode_flag

Initialization variable	pred_mode_flag ctxIdx	
	0	1
initValue	149	134

Table 9-12 – Values of variable initValue for part_mode

Initialization variable	part_mode ctxIdx								
	0	1	2	3	4	5	6	7	8
initValue	184	154	139	154	154	154	139	154	154

Table 9-13 – Values of variable initValue for prev_intra_luma_pred_flag ctxIdx

Initialization variable	prev_intra_luma_pred_flag ctxIdx		
	0	1	2
initValue	184	154	183

Table 9-14 – Values of variable initValue for intra_chroma_pred_mode ctxIdx

Initialization variable	intra_chroma_pred_mode ctxIdx		
	0	1	2
initValue	63	152	152

Table 9-15 – Value of variable initValue for merge_flag ctxIdx

Initialization variable	merge_flag ctxIdx	
	0	1
initValue	110	154

Table 9-16 – Values of variable initValue for merge_idx ctxIdx

Initialization variable	merge_idx ctxIdx	
	0	1
initValue	122	137

Table 9-17 – Values of variable initValue for inter_pred_idc ctxIdx

Initialization variable	inter_pred_idc ctxIdx									
	0	1	2	3	4	5	6	7	8	9
initValue	95	79	63	31	31	95	79	63	31	31

Table 9-18 – Values of variable initValue for ref_idx_l0, ref_idx_l1 ctxIdx

Initialization variable	ref_idx_l0, ref_idx_l1 ctxIdx			
	0	1	2	3
initValue	153	153	153	153

Table 9-19 – Values of variable initValue for abs_mvd_greater0_flag and abs_mvd_greater1_flag ctxIdx

Initialization variable	abs_mvd_greater0_flag, abs_mvd_greater1_flag ctxIdx			
	0	1	2	3
initValue	140	198	169	198

Table 9-20 – Values of variable initValue for mvp_l0_flag, mvp_l1_flag ctxIdx

Initialization variable	mvp_l0_flag, mvp_l1_flag ctxIdx	
	0	1
initValue	168	168

Table 9-21 – Values of variable initValue for rqt_root_cbf ctxIdx

Initialization variable	rqt_root_cbf ctxIdx	
	0	1
initValue	79	79

Table 9-22 – Values of variable initValue for split_transform_flag ctxIdx

Initialization variable	split_transform_flag ctxIdx								
	0	1	2	3	4	5	6	7	8
initValue	153	138	138	124	138	94	224	167	122

Table 9-23 – Values of variable initValue for cbf_luma ctxIdx

Initialization variable	cbf_luma ctxIdx					
	0	**1**	**2**	**3**	**4**	**5**
initValue	111	141	153	111	153	111

Table 9-24 – Values of variable initValue for cbf_cb and cbf_cr ctxIdx

Initialization variable	cbf_cb and cbf_cr ctxIdx											
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
initValue	94	138	182	154	149	107	167	154	149	92	167	154

Table 9-25 – Values of variable initValue for transform_skip_flag ctxIdx

Initialization variable	transform_skip_flag ctxIdx					
	0	**1**	**2**	**3**	**4**	**5**
initValue	139	139	139	139	139	139

Table 9-26 – Values of variable initValue for last_significant_coeff_x_prefix ctxIdx

Initialization variable	last_significant_coefficient_x_prefix ctxIdx																	
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
initValue	110	110	124	125	140	153	125	127	140	109	111	143	127	111	79	108	123	63
	18	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**
initValue	125	110	94	110	95	79	125	111	110	78	110	111	111	95	94	108	123	108
	36	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**	**51**	**52**	**53**
initValue	125	110	124	110	95	94	125	111	111	79	125	126	111	111	79	108	123	93

Table 9-27 – Values of variable initValue for last_significant_coeff_y_prefix ctxIdx

Initialization variable	last_significant_coefficient_y_prefix ctxIdx																	
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
initValue	110	110	124	125	140	153	125	127	140	109	111	143	127	111	79	108	123	63
	18	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**
initValue	125	110	94	110	95	79	125	111	110	78	110	111	111	95	94	108	123	108
	36	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**	**51**	**52**	**53**
initValue	125	110	124	110	95	94	125	111	111	79	125	126	111	111	79	108	123	93

Table 9-28 – Values of variable initValue for coded_sub_block_flag ctxIdx

Initialization variable	coded_sub_block_flag ctxIdx											
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
initValue	91	171	134	141	121	140	61	154	121	140	61	154

Table 9-29 – Values of variable initValue for significant_coeff_flag ctxIdx

Initialization variable	significant_coeff_flag ctxIdx															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
initValue	111	111	125	110	110	94	124	108	124	107	125	141	179	153	125	107
	16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
initValue	125	141	179	153	125	107	125	141	179	153	125	140	139	182	182	152
	32	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
initValue	136	152	136	153	136	139	111	136	139	111	155	154	139	153	139	123
	48	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
initValue	123	63	153	166	183	140	136	153	154	166	183	140	136	153	154	166
	64	**65**	**66**	**67**	**68**	**69**	**70**	**71**	**72**	**73**	**74**	**75**	**76**	**77**	**78**	**79**
initValue	183	140	136	153	154	170	153	123	123	107	121	107	121	167	151	183
	80	**81**	**82**	**83**	**84**	**85**	**86**	**87**	**88**	**89**	**90**	**91**	**92**	**93**	**94**	**95**
initValue	140	151	183	140	170	154	139	153	139	123	123	63	124	166	183	140
	96	**97**	**98**	**99**	**100**	**101**	**102**	**103**	**104**	**105**	**106**	**107**	**108**	**109**	**110**	**111**
initValue	136	153	154	166	183	140	136	153	154	166	183	140	136	153	154	170
	112	**113**	**114**	**115**	**116**	**117**	**118**	**119**	**120**	**121**	**122**	**123**	**124**	**125**		
initValue	153	138	138	122	121	122	121	167	151	183	140	151	183	140		

Table 9-30 – Values of variable initValue for coeff_abs_level_greater1_flag ctxIdx

Initialization variable	coeff_abs_level_greater1_flag ctxIdx															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
initValue	140	92	137	138	140	152	138	139	153	74	149	92	139	107	122	152
	16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
initValue	140	179	166	182	140	227	122	197	154	196	196	167	154	152	167	182
	32	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
initValue	182	134	149	136	153	121	136	137	169	194	166	167	154	167	137	182
	48	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
initValue	154	196	167	167	154	152	167	182	182	134	149	136	153	121	136	122
	64	**65**	**66**	**67**	**68**	**69**	**70**	**71**								
initValue	169	208	166	167	154	152	167	182								

Table 9-31 – Values of variable initValue for coeff_abs_level_greater2_flag ctxIdx

Initialization variable	coeff_abs_level_greater2_flag ctxIdx											
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
initValue	138	153	136	167	152	152	107	167	91	122	107	167
	12	**13**	**14**	**15**	**16**	**17**						
initValue	107	167	91	107	107	167						

9.2.1.2 Memorization process for context variables

Inputs of this process are the CABAC context variables indexed by ctxIdxTable and ctxIdx.

Output of this process are variables tableStateSync and tableMPSSync containing the values of the variables pStateIdx and valMPS used in the initialization process of context variables that are assigned to all syntax elements in subclauses 7.3.9 to 7.3.9.11 except for end_of_slice_segment_flag, end_of_sub_stream_one_bit and pcm_flag.

For each context variable, the corresponding entries pStateIdx and valMPS of tables tableStateSync and tableMPSSync are initialized to the corresponding pStateIdx and valMPS.

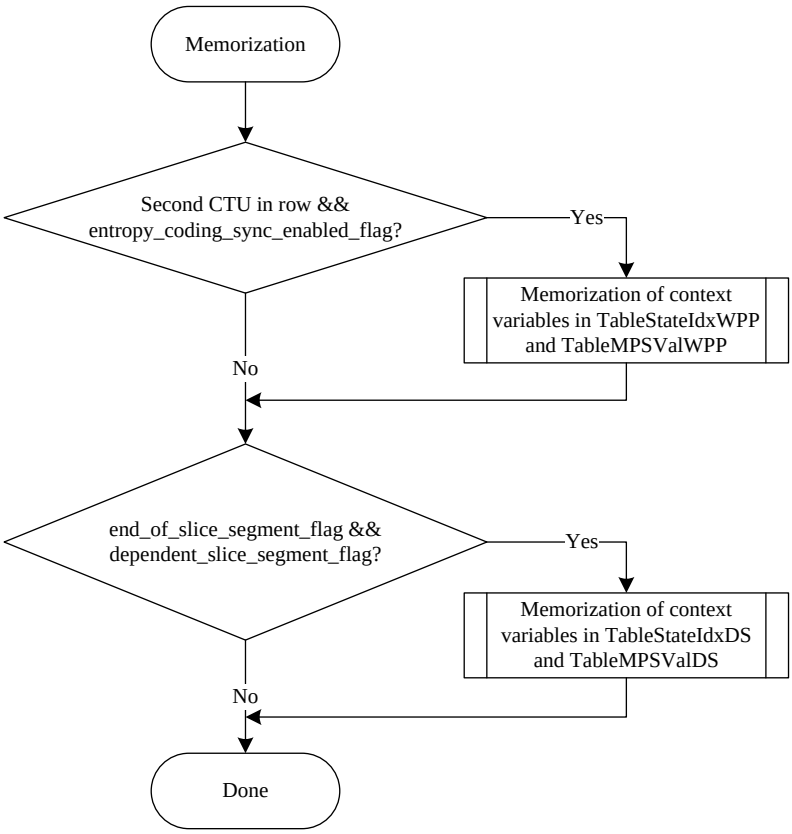


Figure 9-4 – Illustration of CABAC memorization process (informative)

9.2.1.3 Synchronization process for context variables

Inputs of this process are variables tableStateSync and tableMPSSync containing the values of the variables pStateIdx and valMPS used in the memorization process of context variables that are assigned to all syntax elements in subclauses 7.3.9 to 7.3.9.11 except for end_of_slice_segment_flag, end_of_sub_stream_one_bit and pcm_flag.

Outputs of this process are the initialized CABAC context variables indexed by ctxIdxTable and ctxIdx..

For each context variable, the corresponding context variables pStateIdx and valMPS are initialized to the corresponding entries pStateIdx and valMPS of tables tableStateSync and tableMPSSync.

9.2.1.4 Initialization process for the arithmetic decoding engine

This process is invoked before decoding the first coding tree block of a slice segment.

Outputs of this process are the initialized decoding engine registers codIRange and codIOffset both in 16 bit register precision.

The status of the arithmetic decoding engine is represented by the variables codIRange and codIOffset. In the initialization procedure of the arithmetic decoding process, codIRange is set equal to 510 and codIOffset is set equal to the value returned from read_bits(9) interpreted as a 9 bit binary representation of an unsigned integer with most significant bit written first.

The bitstream shall not contain data that result in a value of codIOffset being equal to 510 or 511.

NOTE – The description of the arithmetic decoding engine in this Specification utilizes 16 bit register precision. However, a minimum register precision of 9 bits is required for storing the values of the variables codIRange and codIOffset after invocation of the arithmetic decoding process (DecodeBin) as specified in subclause 9.2.3.2. The arithmetic decoding process for a binary decision (DecodeDecision) as specified in subclause 9.2.3.2.1 and the decoding process for a binary decision before termination (DecodeTerminate) as specified in subclause 9.2.3.2.4 require a minimum register precision of 9 bits for the variables codIRange and codIOffset. The bypass decoding process for binary decisions (DecodeBypass) as specified in subclause 9.2.3.2.3 requires a minimum register precision of 10 bits for the variable codIOffset and a minimum register precision of 9 bits for the variable codIRange.

9.2.2 Binarization process

Input to this process is a request for a syntax element.

Output of this process is the binarization of the syntax element, maxBinIdxCtx, ctxIdxOffset, and bypassFlag.

Table 9-32 specifies the type of binarization process, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset associated with each syntax element.

The specification of the unary (U) binarization process, the truncated unary (TU) binarization process, the truncated Rice (TR), the k-th order Exp-Golomb (EGk) binarization process, and the fixed-length (FL) binarization process are given in subclauses 9.2.2.1 to 9.2.2.5, respectively. Other binarizations are specified in subclauses 9.2.2.6 to 9.2.2.9.

The binarizations for the syntax element coeff_abs_level_remaining as specified in subclause 9.2.2.8 consist of bin strings given by a concatenation of prefix and suffix bit strings. For these binarization processes, the prefix and the suffix bit string are separately indexed using the binIdx variable as specified further in subclause 9.2.3. The two sets of prefix bit strings and suffix bit strings are referred to as the binarization prefix part and the binarization suffix part, respectively.

Associated with each binarization or binarization part of a syntax element is a specific value of the context index table (ctxIdxTable) variable and the related context index offset (ctxIdxOffset) variable and a specific value of the maxBinIdxCtx variable as given in Table 9-32. When two values for each of these variables are specified for one syntax element in Table 9-32, the value in the upper row is related to the prefix part while the value in the lower row is related to the suffix part of the binarization of the corresponding syntax element.

The use of the DecodeBypass process and the variable bypassFlag is derived as follows.

– If no value is assigned to ctxIdxOffset for the corresponding binarization or binarization part in Table 9-32 marked with "na", all bins of the bit strings of the corresponding binarization or of the binarization prefix/suffix part are decoded by invoking the DecodeBypass process as specified in subclause 9.2.3.2.3. In such a case, bypassFlag is set equal to 1, where bypassFlag is used to indicate that for parsing the value of the bin from the bitstream the DecodeBypass process is applied.

– Otherwise, for each possible value of binIdx up to the specified value of maxBinIdxCtx given in Table 9-32, a specific value of the variable ctxIdx is further specified in subclause 9.2.3. In such a case, bypassFlag is set equal to 0.

The possible values of the context index ctxIdx vary depending on the value of ctxIdxTable. The value assigned to ctxIdxOffset specifies the lower value of the range of ctxIdx assigned to the corresponding binarization or binarization part of a syntax element.

ctxIdxTable = 0 and ctxIdx = ctxIdxOffset = 0 are assigned to the syntax elements end_of_slice_segment_flag, end_of_sub_stream_one_bit and pcm_flag as further specified in subclause 9.2.3.1. For parsing the value of the corresponding bin from the bitstream, the arithmetic decoding process for decisions before termination (DecodeTerminate) as specified in subclause 9.2.3.2.4 is applied.

Table 9-32 – Syntax elements and associated types of binarization, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset

Syntax element	initType	Type of binarization	maxBinIdxCtx	ctxIdxTable	ctxIdxOffset
sao_merge_left_flag sao_merge_up_flag	0	FL, cMax = 1	0	Table 9-5	0
	1		0	Table 9-5	1
	2		0	Table 9-5	2
sao_type_idx_luma sao_type_idx_chroma	0	TU, cMax = 2	0	Table 9-6	0
	1		0	Table 9-6	1
	2		0	Table 9-6	2
sao_band_position	na	FL, cMax = 31	na	na	na, (Bypass)
sao_offset_abs	na	TU, cMax = (1<< (Min(bitDepth, 10) − 5)) − 1	na	na	na, (Bypass)
sao_offset_sign	na	FL, cMax = 1	na	na	na, (Bypass)
sao_eo_class_luma sao_eo_class_chroma	na	FL, cMax = 3	na	na	na, (Bypass)
end_of_slice_segment_flag	all	FL, cMax = 1	0	0	0
end_of_sub_stream_one_bit	all	FL, cMax = 1	0	0	0
split_cu_flag	0	FL, cMax = 1	0	Table 9-7	0
	1		0	Table 9-7	3
	2		0	Table 9-7	6
cu_transquant_bypass_flag	0	FL, cMax = 1	0	Table 9-8	0
	1		0	Table 9-8	1
	2		0	Table 9-8	2
cu_skip_flag	1	FL, cMax = 1	0	Table 9-9	0
	2		0	Table 9-9	3
cu_qp_delta_abs	0	prefix and suffix as specified in subclause 9.2.2.6	prefix: 1 suffix: na	prefix: Table 9-10 suffix: na	prefix: 0 suffix: na, (Bypass)
	1		prefix: 1 suffix: na	prefix: Table 9-10 suffix: na	prefix: 2 suffix: na, (Bypass)
	2		prefix: 1 suffix: na	prefix: Table 9-10 suffix: na	prefix: 4 suffix: na, (Bypass)
cu_qp_delta_sign	na	FL, cMax = 1	na	na	na, (Bypass)
pred_mode_flag	1	FL, cMax = 1	0	Table 9-11	0
	2		0	Table 9-11	1
part_mode	0	as specified in subclause 9.2.2.7	0	Table 9-11	0
	1		3	Table 9-11	1
	2		3	Table 9-11	5
pcm_flag	all	FL, cMax = 1	0	0	0

Table 9-32 – Syntax elements and associated types of binarization, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset

Syntax element	initType	Type of binarization	maxBinIdxCtx	ctxIdxTable	ctxIdxOffset
prev_intra_luma_pred_flag	0	FL, cMax = 1	0	Table 9-13	0
	1		0	Table 9-13	1
	2		0	Table 9-13	2
mpm_idx	na	TU, cMax = 2	na	na	na, (Bypass)
rem_intra_luma_pred_mode	na	FL, cMax = 31	na	na	na, (Bypass)
intra_chroma_pred_mode	0	prefix and suffix as specified in subclause 9.2.2.9	prefix: 0 suffix: na	prefix: Table 9-14 suffix: na	prefix: 0 suffix: na, (Bypass)
	1		prefix: 0 suffix: na	prefix: Table 9-14 suffix: na	prefix: 1 suffix: na, (Bypass)
	2		prefix: 0 suffix: na	prefix: Table 9-14 suffix: na	prefix: 2 suffix: na, (Bypass)
merge_flag	1	FL, cMax = 1	0	Table 9-15	0
	2		0	Table 9-15	1
merge_idx	1	TU, cMax = MaxNumMergeCand − 1	0	Table 9-16	0
	2		0	Table 9-16	1
inter_pred_idc	1	as specified in subclause 9.2.2.10	1	Table 9-17	0
	2		1	Table 9-17	5
ref_idx_l0	1	TU, cMax = num_ref_idx_l0_active_minus1	2	Table 9-18	0
	2		2	Table 9-18	2
ref_idx_l1	1	TU, cMax = num_ref_idx_l1_active_minus1	2	Table 9-18	0
	2		2	Table 9-18	2
abs_mvd_greater0_flag[ ]	1	FL, cMax = 1	0	Table 9-19	0
	2		0	Table 9-19	1
abs_mvd_greater1_flag[ ]	1	FL, cMax = 1	0	Table 9-19	2
	2		0	Table 9-19	3
abs_mvd_minus2[ ]	na	EG1	na	na	na, (Bypass)
mvd_sign_flag[ ]	na	FL, cMax = 1	na	na	na, (Bypass)
mvp_l0_flag	1	FL, cMax = 1	0	Table 9-20	0
	2		0	Table 9-20	1
mvp_l1_flag	1	FL, cMax = 1	0	Table 9-20	0
	2		0	Table 9-20	1
rqt_root_cbf	1	FL, cMax = 1	0	Table 9-21	0
	2		0	Table 9-21	1

Table 9-32 – Syntax elements and associated types of binarization, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset

Syntax element	initType	Type of binarization	maxBinIdxCtx	ctxIdxTable	ctxIdxOffset
split_transform_flag	0	FL, cMax = 1	0	Table 9-22	0
	1		0	Table 9-22	3
	2		0	Table 9-22	6
cbf_luma	0	FL, cMax = 1	0	Table 9-23	0
	1		0	Table 9-23	2
	2		0	Table 9-23	4
cbf_cb, cbf_cr	0	FL, cMax = 1	0	Table 9-24	0
	1		0	Table 9-24	4
	2		0	Table 9-24	8
transform_skip_flag[ ][ ][0]	0	FL, cMax = 1	0	Table 9-25	0
	1		0	Table 9-25	1
	2		0	Table 9-25	2
transform_skip_flag[ ][ ][1] transform_skip_flag[ ][ ][2]	0	FL, cMax = 1	0	Table 9-25	3
	1		0	Table 9-25	4
	2		0	Table 9-25	5
last_significant_coeff_x_prefix	0	TU, cMax = (log2TrafoSize << 1) − 1	8	Table 9-26	0
	1		8	Table 9-26	18
	2		8	Table 9-26	36
last_significant_coeff_y_prefix	0	TU, cMax = (log2TrafoSize << 1) − 1	8	Table 9-27	0
	1		8	Table 9-27	18
	2		8	Table 9-27	36
last_significant_coeff_x_suffix	na	FL, cMax = (last_significant_coeff_x_prefix >> 1) − 1	na	na	na, (Bypass)
last_significant_coeff_y_suffix	na	FL, cMax = (last_significant_coeff_y_prefix >> 1) − 1	na	na	na, (Bypass)
coded_sub_block_flag	0	FL, cMax = 1	0	Table 9-28	0
	1		0	Table 9-28	4
	2		0	Table 9-28	8
significant_coeff_flag	0	FL, cMax = 1	0	Table 9-29	0
	1		0	Table 9-29	42
	2		0	Table 9-29	84
coeff_abs_level_greater1_flag	0	FL, cMax = 1	0	Table 9-30	0
	1		0	Table 9-30	24
	2		0	Table 9-30	48

Table 9-32 – Syntax elements and associated types of binarization, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset

Syntax element	initType	Type of binarization	maxBinIdxCtx	ctxIdxTable	ctxIdxOffset
coeff_abs_level_greater2_flag	0	FL, cMax = 1	0	Table 9-31	0
	1		0	Table 9-31	6
	2		0	Table 9-31	12
coeff_abs_level_remaining	na	prefix and suffix as specified in subclause 9.2.2.8	prefix: na suffix: na	prefix: na suffix: na	prefix: na, (Bypass) suffix: na, (Bypass)
coeff_sign_flag	na	FL, cMax = 1	na	na	na, (Bypass)

9.2.2.1 Unary (U) binarization process

Input to this process is a request for a U binarization for a syntax element.

Output of this process is the U binarization of the syntax element.

The bin string of a syntax element having (unsigned integer) value synVal is a bit string of length synVal + 1 indexed by binIdx. The bins for binIdx less than synVal are equal to 1. The bin with binIdx equal to synVal is equal to 0.

Table 9-33 illustrates the bin strings of the unary binarization for a syntax element.

Table 9-33 – Bin string of the unary binarization (informative)

Value of syntax element	Bin string					
0 (I_NxN)	0					
1	1	0				
2	1	1	0			
3	1	1	1	0		
4	1	1	1	1	0	
5	1	1	1	1	1	0
…						
binIdx	0	1	2	3	4	5

9.2.2.2 Truncated unary (TU) binarization process

Input to this process is a request for a TU binarization for a syntax element and cMax.

Output of this process is the TU binarization of the syntax element.

For syntax element (unsigned integer) values less than cMax, the U binarization process as specified in subclause 9.2.2.1 is invoked. For the syntax element value equal to cMax the bin string is a bit string of length cMax with all bins being equal to 1.

NOTE – TU binarization is always invoked with a cMax value equal to the largest possible value of the syntax element being decoded.

9.2.2.3 Truncated Rice (TR) binarization process

Input to this process is a request for a TR binarization for a syntax element, cRiceParam and cTRMax.

Output of this process is the TR binarization of the syntax element.

A TR bin string is a concatenation of a prefix bit string and (when present) a suffix bit string. The prefix of the binarization is specified by invoking the TU binarization process for the prefix part of the value specified by synVal >> cRiceParam with cMax = cTRMax >> cRiceParam. When cTRMax is greater than synVal, the suffix of the TR bin string is present and is specified by the binary representation of synVal − ((synVal >> cRiceParam) << cRiceParam).

NOTE – For the input parameter cRiceParam = 0 the TR binarization is exactly the TU binarization.

9.2.2.4 k-th order Exp-Golomb (EGk) binarization process

Input to this process is a request for an EGk binarization for a syntax element.

Output of this process is the EGk binarization of the syntax element.

The bin string of the EGk binarization process of a syntax element synVal is specified by a process equivalent to the following pseudo-code:

```
absV = Abs( synVal )
stopLoop = 0
do {
    if( absV  >=  ( 1 << k ) ) {
        put( 1 )
        absV = absV − ( 1 << k )
        k++
    } else {
        put( 0 )                                                    (9-5)
        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 0-th order as specified in subclause 9.1.

9.2.2.5 Fixed-length (FL) binarization process

Input to this process is a request for a FL binarization for a syntax element and cMax.

Output of this process is the FL binarization of the syntax element.

FL binarization is constructed by using a fixedLength-bit unsigned integer bin string of the syntax element value, where fixedLength = Ceil(Log2(cMax + 1)). The indexing of bins for the FL binarization is such that the binIdx = 0 relates to the most significant bit with increasing values of binIdx towards the least significant bit.

9.2.2.6 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 consists of a prefix part and (when present) a suffix part. The prefix of the binarization is specified by invoking the TU binarization process for the prefix part of the value specified by Min(synVal, 5) with cMax = 5. When prefix is greater than 4, the suffix bin string is derived using the EGk binarization as specified in subclause 9.2.2.4 for the suffix part (cu_qp_delta_abs − 4) with the Exp-Golomb order k set equal to 0.

9.2.2.7 Binarization process for part_mode

Input to this process is a request for a binarization for the syntax element part_mode a luma location (xC, yC) specifying the top-left sample of the current luma coding block relative to the top left luma sample of the current picture, and a variable cLog2CbSize specifying the current luma coding block size.

Output of this process is the binarization of the syntax element.

The binarization for part_mode is given by Table 9-34 depending on CuPredMode[xC][yC] and cLog2CbSize.

Table 9-34 – Binarization for part_mode

CuPredMode[xC][yC]	**part_mode**	**PartMode**	**Bin string**			
			cLog2CbSize > Log2MinCbSizeY		cLog2CbSize = = Log2MinCbSizeY	
			!amp_enabled_flag	amp_enabled_flag	cLog2CbSize = = 3	cLog2CbSize > 3
MODE_INTRA	0	PART_2Nx2N	-	-	1	1
	1	PART_NxN	-	-	0	0
MODE_INTER	0	PART_2Nx2N	1	1	1	1
	1	PART_2NxN	01	011	01	01
	2	PART_Nx2N	00	001	00	001
	3	PART_NxN	-	-	-	000
	4	PART_2NxnU	-	0100	-	-
	5	PART_2NxnD	-	0101	-	-
	6	PART_nLx2N	-	0000	-	-
	7	PART_nRx2N	-	0001	-	-

9.2.2.8 Binarization process for coeff_abs_level_remaining

Input to this process is a request for a binarization for the syntax element coeff_abs_level_remaining[n], and baseLevel.

Output of this process is the binarization of the syntax element.

The variables cLastAbsLevel and cLastRiceParam are derived as follows.

– If n is equal to 15, cLastAbsLevel and cLastRiceParam are set equal to 0.

– Otherwise (n is less than 15), cLastAbsLevel is set equal to baseLevel + coeff_abs_level_remaining[n + 1] and cLastRiceParam is set equal to the value of cRiceParam that has been derived during the invocation of the binarization process as specified in this subclause for the syntax element coeff_abs_level_remaining[n + 1] of the same transform block.

The variable cRiceParam is derived from cLastAbsLevel and cLastRiceParam as:

$$cRiceParam = Min(cLastRiceParam + (cLastAbsLevel > (3 * (1 << cLastRiceParam)) ? 1 : 0), 4) \quad (9\text{-}6)$$

The variable cTRMax is derived from cRiceParam as:

$$cTRMax = 4 << cRiceParam \quad (9\text{-}7)$$

The binarization of coeff_abs_level_remaining consists of a prefix part and (when present) a suffix part.

The prefix part of the binarization is derived by invoking the TR binarization process as specified in subclause 9.2.2.3 for the prefix part Min(cTRMax, coeff_abs_level_remaining[n]) with the variables cRiceParam and cTRMax as the inputs.

When the prefix bin string is equal to the bit string of length 4 with all bits equal to 1, the bin string consists of a prefix bin string and a suffix bin string. The suffix bin string is derived using the EGk binarization as specified in subclause 9.2.2.4 for the suffix part (coeff_abs_level_remaining[n] − cTRMax) with the Exp-Golomb order k set equal to cRiceParam + 1.

9.2.2.9 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 of the syntax element intra_chroma_pred_mode consists of a prefix part and (when present) a suffix part. Table 9-35 specifies the binarization of the prefix part and the suffix part.

Table 9-35 – Specification of prefix and suffix part for intra_chroma_pred_mode binarization

Value of intra_chroma_pred_mode	prefix	suffix
4	0	n/a
0	1	00
1	1	01
2	1	10
3	1	11

9.2.2.10 Binarization process for inter_pred_idc

Input to this process is a request for a binarization for the syntax element inter_pred_idc, the width and the height of the current luma prediction block nPbW and nPbH.

Output of this process is the binarization of the syntax element.

The binarization for inter_pred_idc is given by Table 9-36.

Table 9-36 – Binarization for inter_pred_idc

Value of inter_pred_idc	Name of inter_pred_idc	Bin string	
		(nPbW + nPbH) != 12	(nPbW + nPbH) = = 12
0	Pred_L0	00	0
1	Pred_L1	01	1
2	Pred_BI	1	-

9.2.3 Decoding process flow

Input to this process is a binarization of the requested syntax element, maxBinIdxCtx, bypassFlag, ctxIdxTable and ctxIdxOffset as specified in subclause 9.2.2.

Output of this process is the value of the syntax element.

This process specifies how each bit of a bit string is parsed for each syntax element.

After parsing each bit, the resulting bit string is compared to all bin strings of the binarization of the syntax element and the following applies.

– If the bit string is equal to one of the bin strings, the corresponding value of the syntax element is the output.

– Otherwise (the bit string is not equal to one of the bin strings), the next bit is parsed.

While parsing each bin, the variable binIdx is incremented by 1 starting with binIdx being set equal to 0 for the first bin.

When the binarization of the corresponding syntax element consists of a prefix and a suffix binarization part, the variable binIdx is set equal to 0 for the first bin of each part of the bin string (prefix part or suffix part). In this case, after parsing the prefix bit string, the parsing process of the suffix bit string related to the binarizations specified in subclauses 9.2.2.3, 9.2.2.6, 9.2.2.8 and 9.2.2.9 is invoked depending on the resulting prefix bit string as specified in subclauses 9.2.2.3, 9.2.2.6, 9.2.2.8 and 9.2.2.9.

Depending on the variable bypassFlag, the following applies.

– If bypassFlag is equal to 1, the bypass decoding process as specified in subclause 9.2.3.2.3 is applied for parsing the value of the bins from the bitstream.

– Otherwise (bypassFlag is equal to 0), the parsing of each bin is specified by the following two ordered steps:

1. Given binIdx, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset, ctxIdx is derived as specified in subclause 9.2.3.1.

2. Given ctxIdxTable and ctxIdx, the value of the bin from the bitstream as specified in subclause 9.2.3.2 is decoded.

9.2.3.1 Derivation process for ctxIdx

Inputs to this process are binIdx, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset.

Output of this process is ctxIdx.

Table 9-37 shows the assignment of ctxIdx increments (ctxIdxInc) to binIdx for all syntax elements with context coded bins.

The ctxIdx to be used with a specific binIdx is specified by first determining the ctxIdxTable and ctxIdxOffset associated with the syntax element. For each syntax element listed in Table 9-37, ctxIdxTable and ctxIdxOffset are specified in Table 9-32, the ctxIdx for a binIdx is the sum of ctxIdxOffset and ctxIdxInc, which is found in Table 9-37. When more than one value is listed in Table 9-37 for a binIdx, the assignment process for ctxIdxInc for that binIdx is further specified in the subclauses given in parenthesis of the corresponding table entry.

All bins with binIdx greater than maxBinIdxCtx are parsed using the value of ctxIdx being assigned to binIdx equal to maxBinIdxCtx.

All entries in Table 9-37 marked with "na" correspond to values of binIdx that do not occur for the corresponding syntax element.

All entries in Table 9-37 marked with "bypass" correspond to values of binIdx that are decoded by invoking the DecodeBypass process as specified in subclause 9.2.3.2.3.

Table 9-37 – Assignment of ctxIdxInc to syntax elements with context coded bins

Syntax element	binIdx					
	0	**1**	**2**	**3**	**4**	**>=5**
sao_merge_left_flag sao_merge_up_flag	0	na	na	na	na	na
sao_type_idx_luma sao_type_idx_chroma	0	bypass	na	na	na	na
split_cu_flag	0,1,2 (subclause 9.2.3.1.1)	na	na	na	na	na
cu_transquant_bypass_flag	0	na	na	na	na	na
cu_skip_flag	0,1,2 (subclause 9.2.3.1.1)	na	na	na	na	na
cu_qp_delta_abs	0	1	1	1	1	bypass
pred_mode_flag	0	na	na	na	na	na
part_mode cLog2CbSize = = Log2MinCbSizeY	0	1	2	bypass	na	na
part_mode cLog2CbSize > Log2MinCbSizeY	0	1	3	bypass	na	na
prev_intra_luma_pred_flag	0	na	na	na	na	na
intra_chroma_pred_mode	0	bypass	bypass	na	na	na
merge_flag	0	na	na	na	na	na
merge_idx	0	bypass	bypass	bypass	na	na
inter_pred_idc[x0][y0]	(nPbW + nPbH) != 12 ? CtDepth[x0][y0] : 4	4	na	na	na	na
ref_idx_l0, ref_idx_l1	0	1	bypass	bypass		na
abs_mvd_greater0_flag[ ]	0	na	na	na	na	na
abs_mvd_greater1_flag[ ]	0	na	na	na	na	na
mvp_l0_flag, mvp_l1_flag	0	na	na	na	na	na
rqt_root_cbf	0	na	na	na	na	na
split_transform_flag	5 − log2TrafoSize	na	na	na	na	na
cbf_luma	trafoDepth = = 0 ? 1 : 0	na	na	na	na	na
cbf_cb, cbf_cr	trafoDepth	na	na	na	na	na
transform_skip_flag[ ][ ][0]	0	na	na	na	na	na
transform_skip_flag[ ][ ][1] transform_skip_flag[ ][ ][2]	0	na	na	na	na	na
last_significant_coeff_x_prefix	0..17 (subclause 9.2.3.1.2)					
last_significant_coeff_y_prefix	0..17 (subclause 9.2.3.1.2)					
coded_sub_block_flag	0..3 (subclause 9.2.3.1.3)	na	na	na	na	na
significant_coeff_flag	0..41 (subclause 9.2.3.1.4)	na	na	na	na	na
coeff_abs_level_greater1_flag	0..23 (subclause 9.2.3.1.5)	na	na	na	na	na
coeff_abs_level_greater2_flag	0..5 (subclause 9.2.3.1.6)	na	na	na	na	na

9.2.3.1.1 Derivation process of ctxIdxInc using left and above syntax elements

Input to this process is the luma location (xC, yC) specifying the top-left luma sample of the current luma coding block relative to the top-left sample of the current picture.

Output of this process is ctxIdxInc.

The location (xL, yL) is set equal to (xC − 1, yC) and the variable availableL, specifying the availability of the coding block located directly to the left of the current coding block, is derived by invoking the availability derivation process for a block in z-scan order as specified in subclause 6.4.1 with the location (xCurr, yCurr) set equal to (xC, yC) and the neighbouring location (xN, yN) set equal to (xL, yL) as the input and the output is assigned to availableL.

The location (xA, yA) is set equal to (xC, yC − 1) and the variable availableA specifying the availability of the coding block located directly above the current coding block, is derived by invoking the availability derivation process for a block in z-scan order as specified in subclause 6.4.1 with the location (xCurr, yCurr) set equal to (xC, yC) and the neighbouring location (xN, yN) set equal to (xA, yA) as the input and the output is assigned to availableA.

The assignment of ctxIdxInc for the syntax elements split_cu_flag and cu_skip_flag is specified in Table 9-38.

Table 9-38 – Specification of ctxIdxInc using left and above syntax elements

Syntax element	condL	condA	ctxIdxInc
split_cu_flag	CtDepth[xL][yL] > CtDepth[xC][yC]	CtDepth[xA][yA] > CtDepth[xC][yC]	(condL && availableL) + (condA && availableA)
cu_skip_flag	cu_skip_flag[xL][yL]	cu_skip_flag[xA][yA]	(condL && availableL) + (condA && availableA)

9.2.3.1.2 Derivation process of ctxIdxInc for the syntax elements last_significant_coeff_x_prefix and last_significant_coeff_y_prefix

Inputs to this process are the binIdx, the colour component index cIdx, the transform block size log2TrafoSize.

Output of this process is ctxIdxInc.

The variables ctxOffset and ctxShift are derived as follows.

– If cIdx is equal to 0, ctxOffset is set equal to 3 * (log2TrafoSize − 2) + ((log2TrafoSize − 1) >> 2) and ctxShift is set equal to (log2TrafoSize + 1) >> 2.

– Otherwise (cIdx is greater than 0), ctxOffset is set equal to 15 and ctxShift is set equal to log2TrafoSize − 2.

The variable ctxIdxInc is derived as follows.

$$ctxIdxInc = (binIdx >> ctxShift) + ctxOffset \quad (9\text{-}8)$$

9.2.3.1.3 Derivation process of ctxIdxInc for the syntax element coded_sub_block_flag

Inputs to this process are the colour component index cIdx, the current sub-block scan position (xS, yS), the previously decoded bins of the syntax element coded_sub_block_flag and the size of the current transform block, log2TrafoSize.

Output of this process is ctxIdxInc.

The variable csbfCtx is derived using the current position (xS, yS), two previously decoded bins of the syntax element coded_sub_block_flag in scan order, and the size of the current transform block, log2TrafoSize, as follows.

– csbfCtx is initialized with 0 as follows.

$$csbfCtx = 0 \quad (9\text{-}9)$$

– When xS is less than (1 << (log2TrafoSize − 2)) − 1, csbfCtx is modified as follows.

$$csbfCtx \mathrel{+}= coded\_sub\_block\_flag[xS + 1][yS] \quad (9\text{-}10)$$

– When yS is less than (1 << (log2TrafoSize − 2)) − 1, csbfCtx is modified as follows.

$$csbfCtx \mathrel{+}= coded\_sub\_block\_flag[xS][yS + 1] \quad (9\text{-}11)$$

The context index increment ctxIdxInc is derived using the colour component index cIdx and csbfCtx as follows.

– If cIdx is equal to 0, ctxIdxInc is derived as follows.

$$ctxIdxInc = Min(csbfCtx, 1) \quad (9\text{-}12)$$

– Otherwise (cIdx is greater than 0), ctxIdxInc is derived as follows.

$$ctxIdxInc = 2 + Min(csbfCtx, 1) \quad (9\text{-}13)$$

9.2.3.1.4 Derivation process of ctxIdxInc for the syntax element significant_coeff_flag

Inputs to this process are the colour component index cIdx, the current coefficient scan position (xC, yC), the scan order index scanIdx, the transform block size log2TrafoSize.

Output of this process is ctxIdxInc.

The variable sigCtx depends on the current position (xC, yC), the colour component index cIdx, the transform block size and previsously decoded bins of the syntax element coded_sub_block_flag. For the derivation of sigCtx, the following applies.

– If log2TrafoSize is equal to 2, sigCtx is derived using ctxIdxMap[ ] specified in Table 9-39 as follows..

$$sigCtx = ctxIdxMap[(yC << 2) + xC] \quad (9\text{-}14)$$

– Otherwise, if xC + yC is equal to 0, sigCtx is derived as follows.

$$sigCtx = 0 \quad (9\text{-}15)$$

– Otherwise, sigCtx is derived using previous values of coded_sub_block_flag as follows.
  – The horizontal and vertical sub-block positions xS and yS are set equal to (xC >> 2) and (yC >> 2), respectively.
  – The variable prevCsbf is set equal to 0.
  – When xS is less than (1 << (log2TrafoSize − 2)) − 1, the following applies.

$$prevCsbf += coded\_sub\_block\_flag[xS + 1][yS] \quad (9\text{-}16)$$

  – When yS is less than (1 << (log2TrafoSize − 2)) − 1, the following applies.

$$prevCsbf += (coded\_sub\_block\_flag[xS][yS + 1] << 1) \quad (9\text{-}17)$$

  – The inner sub-block positions xP and yP are set equal to (xC & 3) and (yC & 3), respectively.
  – The variable sigCtx is derived as follows.
    – If prevCsbf is equal to 0, the following applies.

$$sigCtx = (xP + yP == 0) ? 2 : (xP + yP < 3) ? 1: 0 \quad (9\text{-}18)$$

    – Otherwise, if prevCsbf is equal to 1, the following applies.

$$sigCtx = (yP == 0) ? 2 : (yP == 1) ? 1: 0 \quad (9\text{-}19)$$

    – Otherwise, if prevCsbf is equal to 2, the following applies.

$$sigCtx = (xP == 0) ? 2 : (xP == 1) ? 1: 0 \quad (9\text{-}20)$$

    – Otherwise (prevCsbf is equal to 3), the following applies.

$$sigCtx = 2 \quad (9\text{-}21)$$

  – The variable sigCtx is modified as follows.
    – If cIdx is equal to 0, the following applies.
      – When (xS + yS) is greater than 0, the following applies.

$$sigCtx += 3 \quad (9\text{-}22)$$

– The variable sigCtx is modified as follows.
  – If log2TrafoSize is equal to 3, the following applies.

$$sigCtx \mathrel{+}= (scanIdx == 0) ? 9 : 15 \quad (9\text{-}23)$$

  – Otherwise, the following applies.

$$sigCtx \mathrel{+}= 21 \quad (9\text{-}24)$$

– Otherwise (cIdx is greater than 0), the following applies.
  – If log2TrafoSize is equal to 3, the following applies.

$$sigCtx \mathrel{+}= 9 \quad (9\text{-}25)$$

  – Otherwise, the following applies.

$$sigCtx \mathrel{+}= 12 \quad (9\text{-}26)$$

The context index increment ctxIdxInc is derived using the colour component index cIdx and sigCtx as follows.

– If cIdx is equal to 0, ctxIdxInc is derived as follows.

$$ctxIdxInc = sigCtx \quad (9\text{-}27)$$

– Otherwise (cIdx is greater than 0), ctxIdxInc is derived as follows.

$$ctxIdxInc = 27 + sigCtx \quad (9\text{-}28)$$

Table 9-39 – Specification of ctxIdxMap[i]

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
ctxIdxMap[i]	0	1	4	5	2	3	4	5	6	6	8	8	7	7	8

9.2.3.1.5 Derivation process of ctxIdxInc for the syntax element coeff_abs_level_greater1_flag

Inputs to this process are the colour component index cIdx, the current sub-block scan index i and the current coefficient scan index n within the current sub-block.

Output of this process is ctxIdxInc.

The variable ctxSet specifies the current context set and for its derivation the following applies.

– If this process is invoked for the first time for the current sub-block scan index i, the following applies.
  – The variable ctxSet is initialized as follows.
    – If the current sub-block scan index i is equal to 0 or cIdx is greater than 0, the following applies.

$$ctxSet = 0 \quad (9\text{-}29)$$

    – Otherwise (i is greater than 0 and cIdx is equal to 0), the following applies.

$$ctxSet = 2 \quad (9\text{-}30)$$

  – The variable lastGreater1Ctx is derived as follows.
    – If the current sub-block with scan index i is the first one to be processed in this subclause for the current transform block, the variable lastGreater1Ctx is set equal to 1.
    – Otherwise, the variable lastGreater1Ctx is set equal to the value of greater1Ctx that has been derived during the last invocation of the process specified in this subclause for the syntax element coeff_abs_level_greater1_flag for the previous sub-block with scan index i + 1.
  – When lastGreater1Ctx is equal to 0, ctxSet is incremented by one as follows.

$$ctxSet = ctxSet + 1 \quad (9\text{-}31)$$

- The variable greater1Ctx is set equal to 1.

– Otherwise (this process is not invoked for the first time for the current sub-block scan index i), the following applies.

  - The variable ctxSet is set equal to the variable ctxSet that has been derived during the last invocation of the process specified in this subclause.
  - The variable greater1Ctx is set equal to the variable greater1Ctx that has been derived during the last invocation of the process specified in this subclause.
  - When greater1Ctx is greater than 0, the variable lastGreater1Flag is set equal to the syntax element coeff_abs_level_greater1_flag that has been used during the last invocation of the process specified in this subclause and greater1Ctx is modifed as follows.
    - If lastGreater1Flag is equal to 1, greater1Ctx is set equal to 0.
    - Otherwise (lastGreater1Flag is equal to 0), greater1Ctx is incremented by 1.

The context index increment ctxIdxInc is derived using the current context set ctxSet and the current context greater1Ctx as follows.

$$ctxIdxInc = (ctxSet * 4) + Min(3, greater1Ctx) \quad (9\text{-}32)$$

When cIdx is greater than 0, ctxIdxInc is modified as follows.

$$ctxIdxInc = ctxIdxInc + 16 \quad (9\text{-}33)$$

9.2.3.1.6 Derivation process of ctxIdxInc for the syntax element coeff_abs_level_greater2_flag

Inputs to this process are the colour component index cIdx, the current sub-block scan index i and the current coefficient scan index n within the current sub-block.

Output of this process is ctxIdxInc.

The variable ctxSet specifies the current context set and is set to the variable ctxSet that has been derived in subclause 9.2.3.1.5 for the same subset i.

The context index increment ctxIdxInc is set equal to the variable ctxSet as follows.

$$ctxIdxInc = ctxSet \quad (9\text{-}34)$$

When cIdx is greater than 0, ctxIdxInc is modified as follows.

$$ctxIdxInc = ctxIdxInc + 4 \quad (9\text{-}35)$$

9.2.3.2 Arithmetic decoding process

Inputs to this process are the bypassFlag, ctxIdxTable, and ctxIdx as derived in subclause 9.2.3.1, and the state variables codIRange and codIOffset of the arithmetic decoding engine.

Output of this process is the value of the bin.

Figure 9-5 illustrates the whole arithmetic decoding process for a single bin. For decoding the value of a bin, the context index table ctxIdxTable and the ctxIdx is passed to the arithmetic decoding process DecodeBin(ctxIdxTable, ctxIdx), which is specified as follows.

– If bypassFlag is equal to 1, DecodeBypass( ) as specified in subclause 9.2.3.2.3 is invoked.

– Otherwise, if bypassFlag is equal to 0, ctxIdxTable is equal 0 and ctxIdx is equal to 0, DecodeTerminate( ) as specified in subclause 9.2.3.2.4 is invoked.

– Otherwise (bypassFlag is equal to 0, ctxIdxTable is not equal to 0 and ctxIdx is not equal to 0), DecodeDecision( ) as specified in subclause 9.2.3.2.1 is applied.

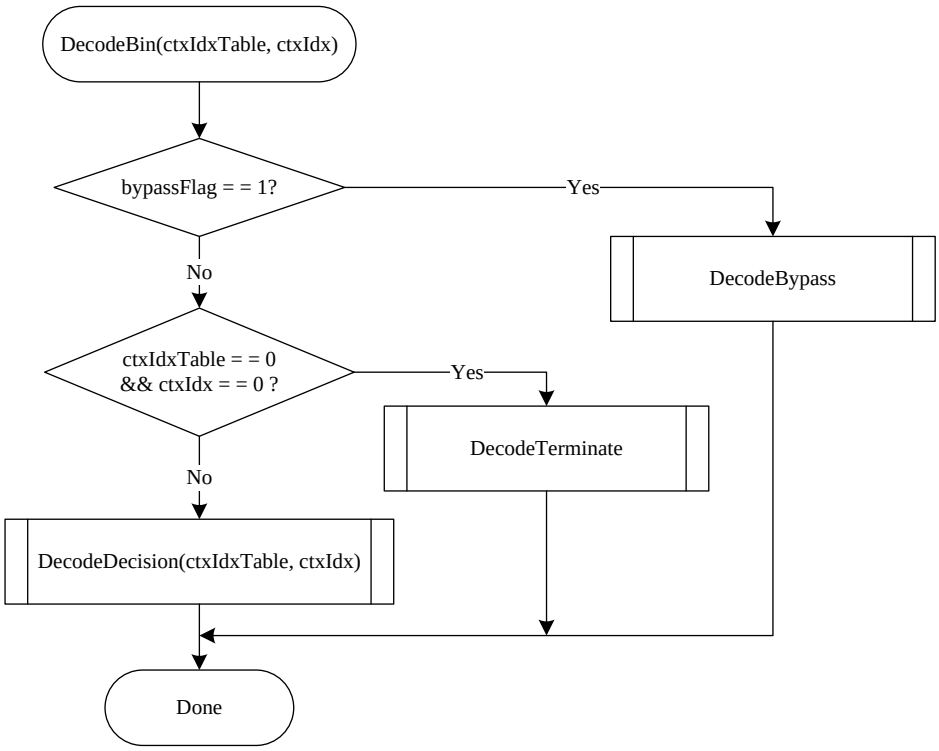


Figure 9-5 – Overview of the arithmetic decoding process for a single bin (informative)

NOTE – Arithmetic coding is based on the principle of recursive interval subdivision. Given a probability estimation p(0) and p(1) = 1 – p(0) of a binary decision (0, 1), an initially given code sub-interval with the range codIRange will be subdivided into two sub-intervals having range p(0) * codIRange and codIRange – p(0) * codIRange, respectively. Depending on the decision, which has been observed, the corresponding sub-interval will be chosen as the new code interval, and a binary code string pointing into that interval will represent the sequence of observed binary decisions. It is useful to distinguish between the most probable symbol (MPS) and the least probable symbol (LPS), so that binary decisions have to be identified as either MPS or LPS, rather than 0 or 1. Given this terminology, each context is specified by the probability p_{LPS} of the LPS and the value of MPS (valMPS), which is either 0 or 1.

The arithmetic core engine in this 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 { p_{LPS}(pStateIdx) | 0 <= pStateIdx < 64 } for the LPS probability p_{LPS}. The numbering of the states is arranged in such a way that the probability state with index pStateIdx = 0 corresponds to an LPS probability value of 0.5, with decreasing LPS probability towards higher state indices.
- The range codIRange representing the state of the coding engine is quantized to a small set $\{Q_1,\ldots,Q_4\}$ of pre-set quantization values prior to the calculation of the new interval range. Storing a table containing all 64x4 pre-computed product values of $Q_i * p_{LPS}$(pStateIdx) allows a multiplication-free approximation of the product codIRange * p_{LPS}(pStateIdx).
- For syntax elements or parts thereof for which an approximately uniform probability distribution is assumed to be given a separate simplified encoding and decoding bypass process is used.

9.2.3.2.1 Arithmetic decoding process for a binary decision

Inputs to this process are ctxIdxTable, ctxIdx, codIRange, and codIOffset.

Outputs of this process are the decoded value binVal, and the updated variables codIRange and codIOffset.

Figure 9-6 shows the flowchart for decoding a single decision (DecodeDecision):

1. The value of the variable codIRangeLPS is derived as follows.
   - Given the current value of codIRange, the variable qCodIRangeIdx is derived by

     qCodIRangeIdx =(codIRange >> 6) & 3 (9-36)

   - Given qCodIRangeIdx and pStateIdx associated with ctxIdxTable and ctxIdx, the value of the variable rangeTabLPS as specified in Table 9-40 is assigned to codIRangeLPS:

     codIRangeLPS = rangeTabLPS[pStateIdx][qCodIRangeIdx] (9-37)

2. The variable codIRange is set equal to codIRange − codIRangeLPS and the following applies.
   - If codIOffset is greater than or equal to codIRange, the variable binVal is set equal to 1 − valMPS, codIOffset is decremented by codIRange, and codIRange is set equal to codIRangeLPS.
   - Otherwise, the variable binVal is set equal to valMPS.

Given the value of binVal, the state transition is performed as specified in subclause 9.2.3.2.1.1. Depending on the current value of codIRange, renormalization is performed as specified in subclause 9.2.3.2.2.

9.2.3.2.1.1 State transition process

Inputs to this process are the current pStateIdx, the decoded value binVal and valMPS values of the context variable associated with ctxIdxTable and ctxIdx.

Outputs of this process are the updated pStateIdx and valMPS of the context variable associated with ctxIdx.

Depending on the decoded value binVal, the update of the two variables pStateIdx and valMPS associated with ctxIdx is derived as specified by the following pseudo-code:

```
if( binVal  = =  valMPS )
    pStateIdx = transIdxMPS( pStateIdx )
else {                                                                    (9-38)
    if( pStateIdx  = =  0 )
        valMPS = 1 – valMPS
    pStateIdx = transIdxLPS( pStateIdx )
}
```

Table 9-41 specifies the transition rules transIdxMPS( ) and transIdxLPS( ) after decoding the value of valMPS and 1 – valMPS, respectively.

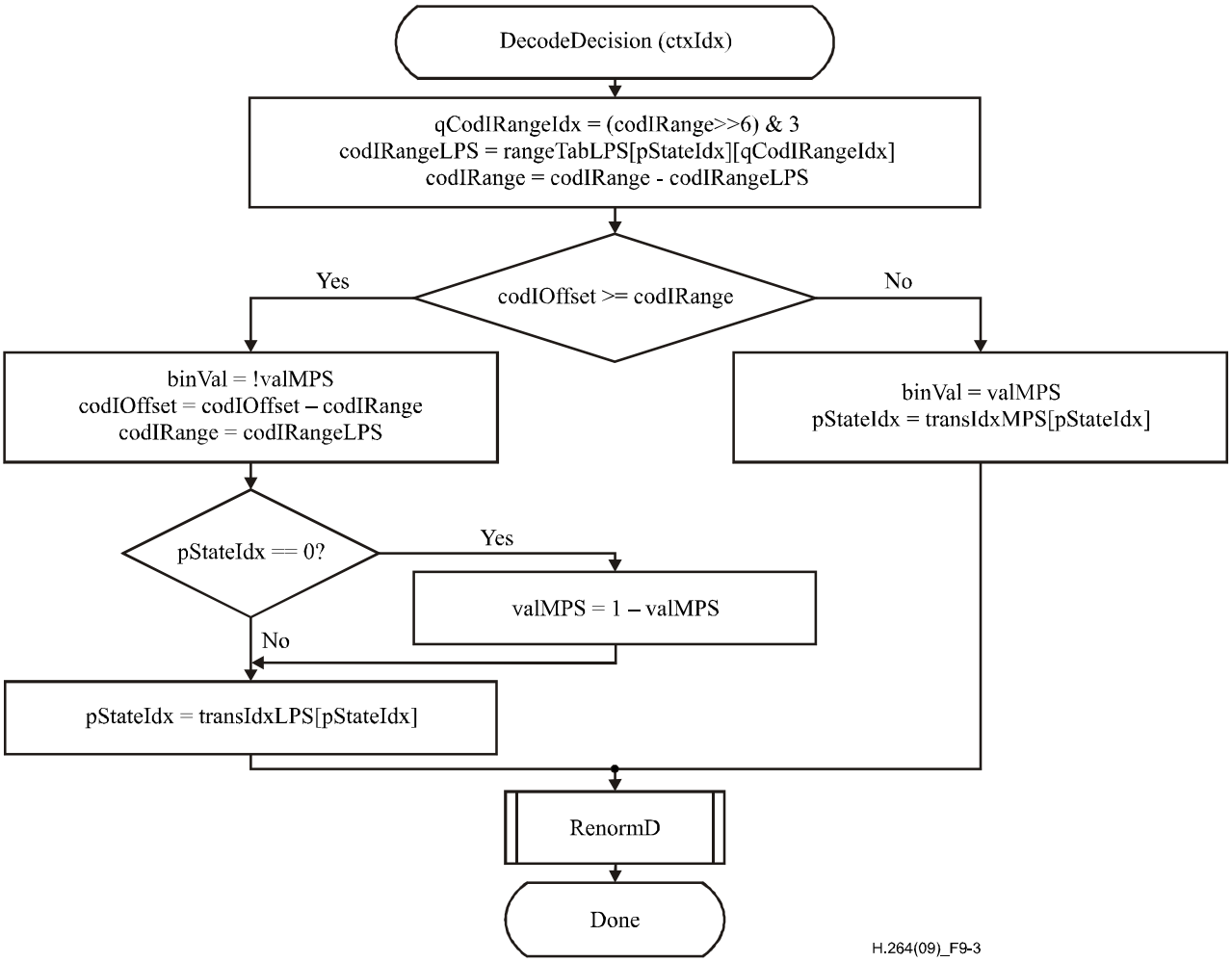


Figure 9-6 – Flowchart for decoding a decision

[Ed. (BB): add ctxIdxTable to the figure]

Table 9-40 – Specification of rangeTabLPS depending on pStateIdx and qCodIRangeIdx

pStateIdx	qCodIRangeIdx				pStateIdx	qCodIRangeIdx			
	0	1	2	3		0	1	2	3
0	128	176	208	240	**32**	27	33	39	45
1	128	167	197	227	**33**	26	31	37	43
2	128	158	187	216	**34**	24	30	35	41
3	123	150	178	205	**35**	23	28	33	39
4	116	142	169	195	**36**	22	27	32	37
5	111	135	160	185	**37**	21	26	30	35
6	105	128	152	175	**38**	20	24	29	33
7	100	122	144	166	**39**	19	23	27	31
8	95	116	137	158	**40**	18	22	26	30
9	90	110	130	150	**41**	17	21	25	28
10	85	104	123	142	**42**	16	20	23	27
11	81	99	117	135	**43**	15	19	22	25
12	77	94	111	128	**44**	14	18	21	24
13	73	89	105	122	**45**	14	17	20	23
14	69	85	100	116	**46**	13	16	19	22
15	66	80	95	110	**47**	12	15	18	21
16	62	76	90	104	**48**	12	14	17	20
17	59	72	86	99	**49**	11	14	16	19
18	56	69	81	94	**50**	11	13	15	18
19	53	65	77	89	**51**	10	12	15	17
20	51	62	73	85	**52**	10	12	14	16
21	48	59	69	80	**53**	9	11	13	15
22	46	56	66	76	**54**	9	11	12	14
23	43	53	63	72	**55**	8	10	12	14
24	41	50	59	69	**56**	8	9	11	13
25	39	48	56	65	**57**	7	9	11	12
26	37	45	54	62	**58**	7	9	10	12
27	35	43	51	59	**59**	7	8	10	11
28	33	41	48	56	**60**	6	8	9	11
29	32	39	46	53	**61**	6	7	9	10
30	30	37	43	50	**62**	6	7	8	9
31	29	35	41	48	**63**	2	2	2	2

Table 9-41 – State transition table

pStateIdx	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
transIdxLPS	0	0	1	2	2	4	4	5	6	7	8	9	9	11	11	12
transIdxMPS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
pStateIdx	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
transIdxLPS	13	13	15	15	16	16	18	18	19	19	21	21	22	22	23	24
transIdxMPS	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
pStateIdx	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
transIdxLPS	24	25	26	26	27	27	28	29	29	30	30	30	31	32	32	33
transIdxMPS	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
pStateIdx	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
transIdxLPS	33	33	34	34	35	35	35	36	36	36	37	37	37	38	38	63
transIdxMPS	49	50	51	52	53	54	55	56	57	58	59	60	61	62	62	63

9.2.3.2.2 Renormalization process in the arithmetic decoding engine

Inputs to this process are bits from slice segment data and the variables codIRange and codIOffset.

Outputs of this process are the updated variables codIRange and codIOffset.

A flowchart of the renormalization is shown in Figure 9-7. The current value of codIRange is first compared to 256 and further steps are specified as follows.

– If codIRange is greater than or equal to 256, no renormalization is needed and the RenormD process is finished;

– Otherwise (codIRange is less than 256), the renormalization loop is entered. Within this loop, the value of codIRange is doubled, i.e. left-shifted by 1 and a single bit is shifted into codIOffset by using read_bits(1).

The bitstream shall not contain data that result in a value of codIOffset being greater than or equal to codIRange upon completion of this process.

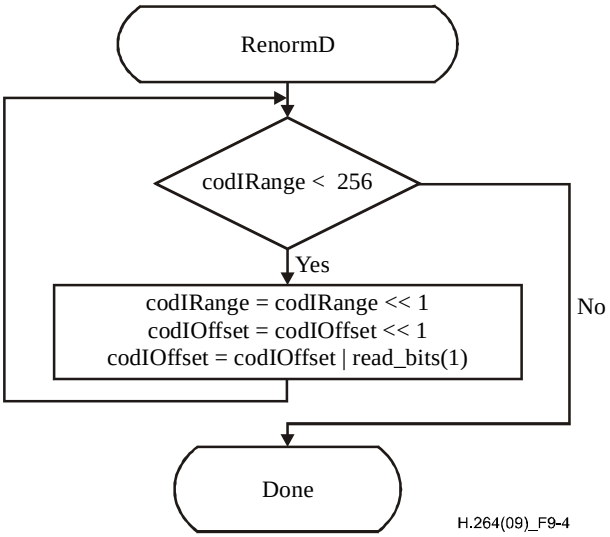


Figure 9-7 – Flowchart of renormalization

9.2.3.2.3 Bypass decoding process for binary decisions

Inputs to this process are bits from slice segment data and the variables codIRange and codIOffset.

Outputs of this process are the updated variable codIOffset and the decoded value binVal.

The bypass decoding process is invoked when bypassFlag is equal to 1. Figure 9-8 shows a flowchart of the corresponding process.

First, the value of codIOffset is doubled, i.e. left-shifted by 1 and a single bit is shifted into codIOffset by using read_bits(1). Then, the value of codIOffset is compared to the value of codIRange and further steps are specified as follows.

– If codIOffset is greater than or equal to codIRange, the variable binVal is set equal to 1 and codIOffset is decremented by codIRange.

– Otherwise (codIOffset is less than codIRange), the variable binVal is set equal to 0.

The bitstream shall not contain data that result in a value of codIOffset being greater than or equal to codIRange upon completion of this process.

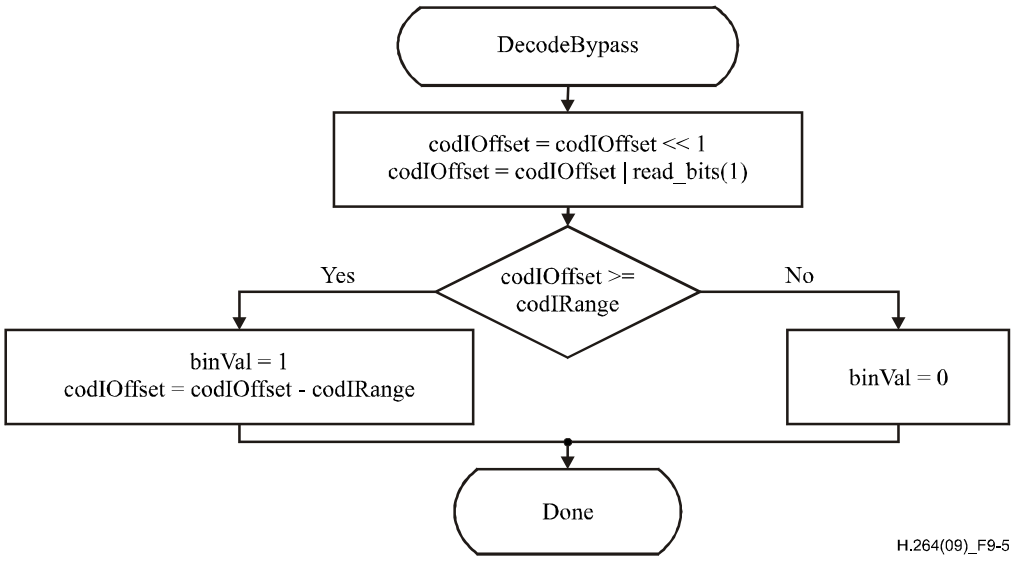


Figure 9-8 – Flowchart of bypass decoding process

9.2.3.2.4 Decoding process for binary decisions before termination

Inputs to this process are bits from slice segment data and the variables codIRange and codIOffset.

Outputs of this process are the updated variables codIRange and codIOffset, and the decoded value binVal.

This decoding process applies to decoding of end_of_slice_segment_flag, end_of_sub_stream_one_bit and pcm_flag corresponding to ctxIdxTable equal to 0 and ctxIdx equal to 0. Figure 9-9 shows the flowchart of the corresponding decoding process, which is specified as follows.

First, the value of codIRange is decremented by 2. Then, the value of codIOffset is compared to the value of codIRange and further steps are specified as follows.

– If codIOffset is greater than or equal to codIRange, the variable binVal is set equal to 1, no renormalization is carried out, and CABAC decoding is terminated. The last bit inserted in register codIOffset is equal to 1. When decoding end_of_slice_segment_flag, this last bit inserted in register codIOffset is interpreted as rbsp_stop_one_bit.

– Otherwise (codIOffset is less than codIRange), the variable binVal is set equal to 0 and renormalization is performed as specified in subclause 9.2.3.2.2.

NOTE – This procedure may also be implemented using DecodeDecision(ctxIdxTable, ctxIdx) with ctxIdxTable = 0 and ctxIdx = 0. In the case where the decoded value is equal to 1, seven more bits would be read by DecodeDecision(ctxIdxTable, ctxIdx) and a decoding process would have to adjust its bitstream pointer accordingly to properly decode following syntax elements.

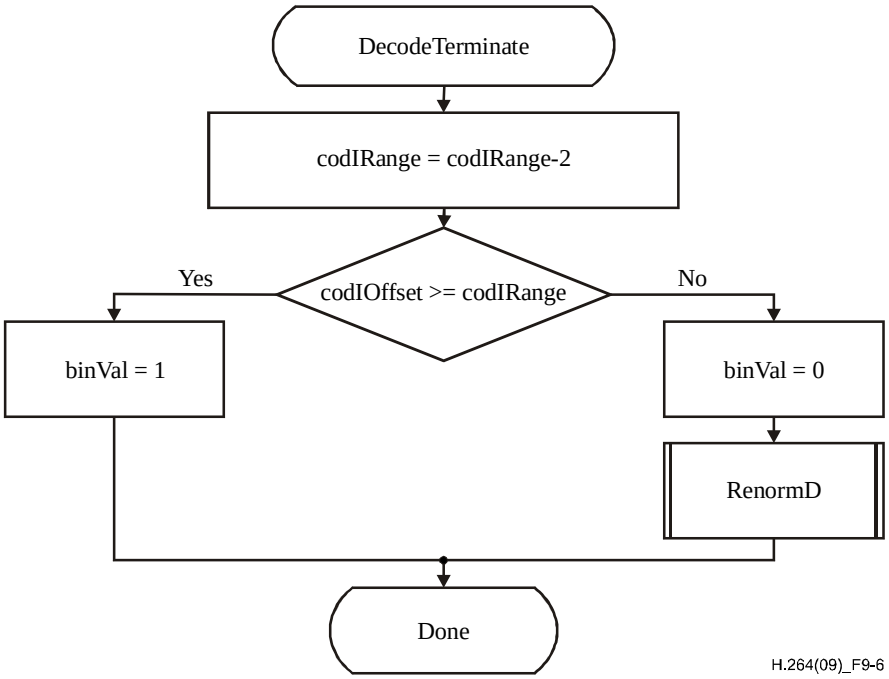


Figure 9-9 – Flowchart of decoding a decision before termination

9.2.4 Arithmetic encoding process (informative)

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

Inputs to this process are decisions that are to be encoded and written.

Outputs of this process are bits that are written to the RBSP.

This informative subclause describes an arithmetic encoding engine that matches the arithmetic decoding engine described in subclause 9.2.3.2. The encoding engine is essentially symmetric with the decoding engine, i.e. procedures are called in the same order. The following procedures are described in this section: InitEncoder, EncodeDecision, EncodeBypass, EncodeTerminate, which correspond to InitDecoder, DecodeDecision, DecodeBypass, and DecodeTerminate, respectively. The state of the arithmetic encoding engine is represented by a value of the variable codILow pointing to the lower end of a sub-interval and a value of the variable codIRange specifying the corresponding range of that sub-interval.

9.2.4.1 Initialization process for the arithmetic encoding engine (informative)

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

This process is invoked before encoding the first coding block of a slice segment, and after encoding any pcm_alignment_zero_bit and all pcm_sample_luma and pcm_sample_chroma data for a coding unit with pcm_flag equal to 1.

Outputs of this process are the values codILow, codIRange, firstBitFlag, bitsOutstanding, and BinCountsInNALunits of the arithmetic encoding engine.

In the initialization procedure of the encoder, codILow is set equal to 0, and codIRange is set equal to 510. Furthermore, firstBitFlag is set equal to 1 and the counter bitsOutstanding is set equal to 0.

Depending on whether the current slice segment is the first slice segment of a coded picture, the following applies.

– If the current slice segment is the first slice segment of a coded picture, the counter BinCountsInNALunits is set equal to 0.

– Otherwise (the current slice segment is not the first slice segment of a coded picture), the counter BinCountsInNALunits is not modified. The value of BinCountsInNALunits is the result of encoding all the slice segments of a coded picture that precede the current slice segment in decoding order. After initializing for the first slice segment of a coded picture as specified in this subclause, BinCountsInNALunits is incremented as specified in subclauses 9.2.4.2, 9.2.4.4, and 9.2.4.5.

NOTE – The minimum register precision required for storing the values of the variables codILow and codIRange after invocation of any of the arithmetic encoding processes specified in subclauses 9.2.4.2, 9.2.4.4, and 9.2.4.5 is 10 bits and 9 bits, respectively. The encoding process for a binary decision (EncodeDecision) as specified in subclause 9.2.4.2 and the encoding process for a binary decision before termination (EncodeTerminate) as specified in subclause 9.2.4.5 require a minimum register precision of 10 bits for the variable codILow and a minimum register precision of 9 bits for the variable codIRange. The bypass encoding process for binary decisions (EncodeBypass) as specified in subclause 9.2.4.4 requires a minimum register precision of 11 bits for the variable codILow and a minimum register precision of 9 bits for the variable codIRange. The precision required for the counters bitsOutstanding and BinCountsInNALunits should be sufficiently large to prevent overflow of the related registers. When maxBinCountInSlice denotes the maximum total number of binary decisions to encode in one slice segment and maxBinCountInPic denotes the maximum total number of binary decisions to encode a picture, the minimum register precision required for the variables bitsOutstanding and BinCountsInNALunits is given by Ceil(Log2(maxBinCountInSlice + 1)) and Ceil(Log2(maxBinCountInPic + 1)), respectively.

9.2.4.2 Encoding process for a binary decision (informative)

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

Inputs to this process are the context index ctxIdx, the value of binVal to be encoded, and the variables codIRange, codILow and BinCountsInNALunits.

Outputs of this process are the variables codIRange, codILow, and BinCountsInNALunits.

Figure 9-10 shows the flowchart for encoding a single decision. In a first step, the variable codIRangeLPS is derived as follows.

Given the current value of codIRange, codIRange is mapped to the index qCodIRangeIdx of a quantized value of codIRange by using Equation 9-36. The value of qCodIRangeIdx and the value of pStateIdx associated with ctxIdx are used to determine the value of the variable rangeTabLPS as specified in Table 9-40, which is assigned to codIRangeLPS. The value of codIRange − codIRangeLPS is assigned to codIRange.

In a second step, the value of binVal is compared to valMPS associated with ctxIdx. When binVal is different from valMPS, codIRange is added to codILow and codIRange is set equal to the value codIRangeLPS. Given the encoded decision, the state transition is performed as specified in subclause 9.2.3.2.1.1. Depending on the current value of codIRange, renormalization is performed as specified in subclause 9.2.4.3. Finally, the variable BinCountsInNALunits is incremented by 1.

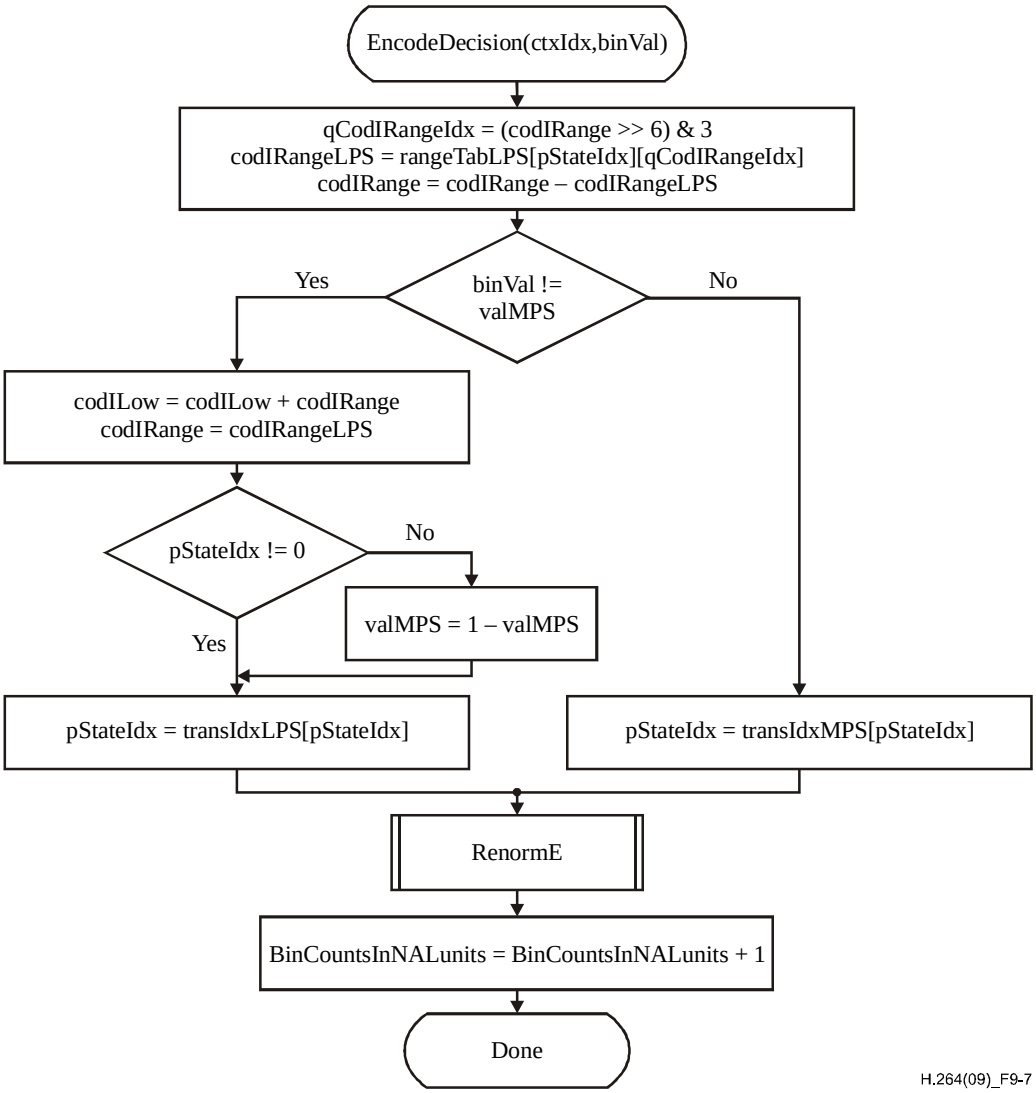


Figure 9-10 – Flowchart for encoding a decision

9.2.4.3 Renormalization process in the arithmetic encoding engine (informative)

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

Inputs to this process are the variables codIRange, codILow, firstBitFlag, and bitsOutstanding.

Outputs of this process are zero or more bits written to the RBSP and the updated variables codIRange, codILow, firstBitFlag, and bitsOutstanding.

Renormalization is illustrated in Figure 9-11.

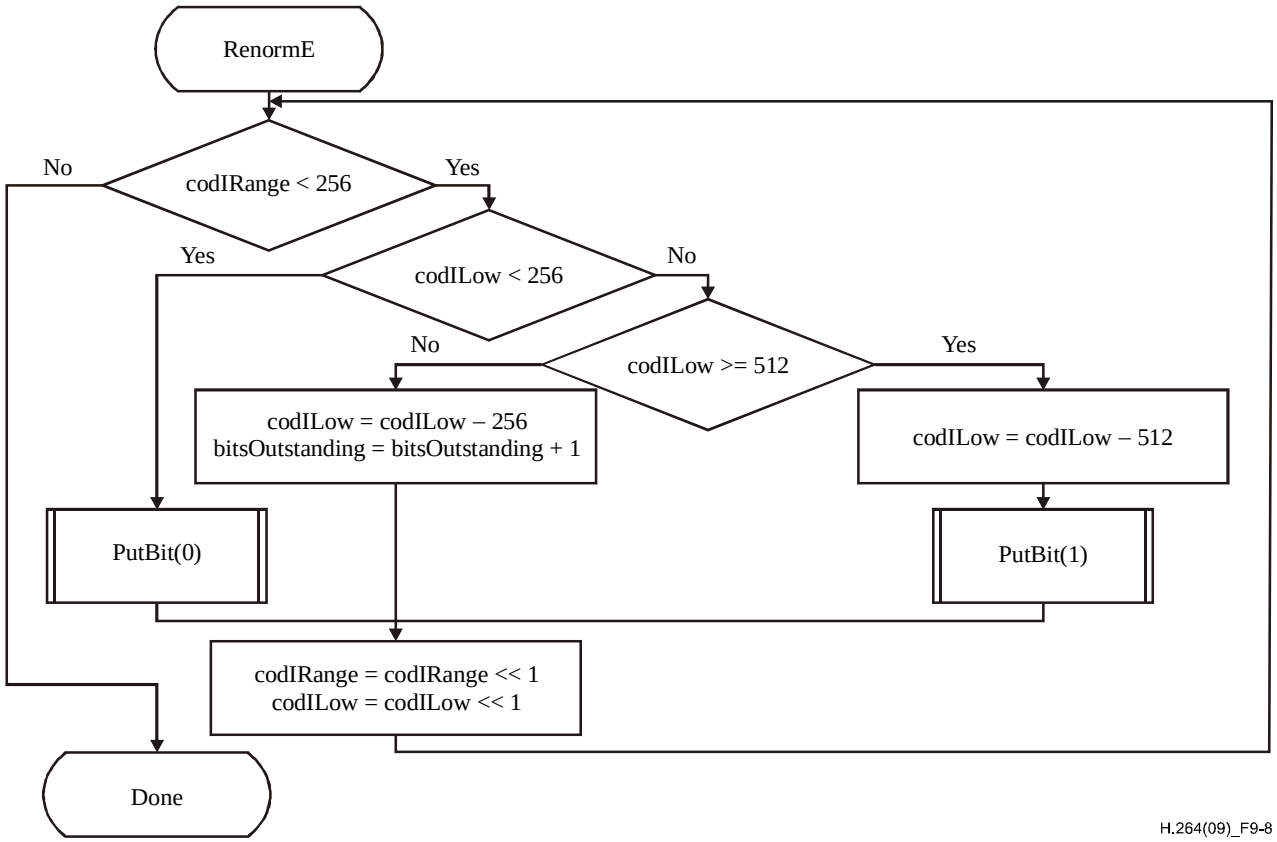


Figure 9-11 – Flowchart of renormalization in the encoder

The PutBit( ) procedure described in Figure 9-12 provides carry over control. It uses the function WriteBits(B, N) that writes N bits with value B to the bitstream and advances the bitstream pointer by N bit positions. This function assumes the existence of a bitstream pointer with an indication of the position of the next bit to be written to the bitstream by the encoding process.

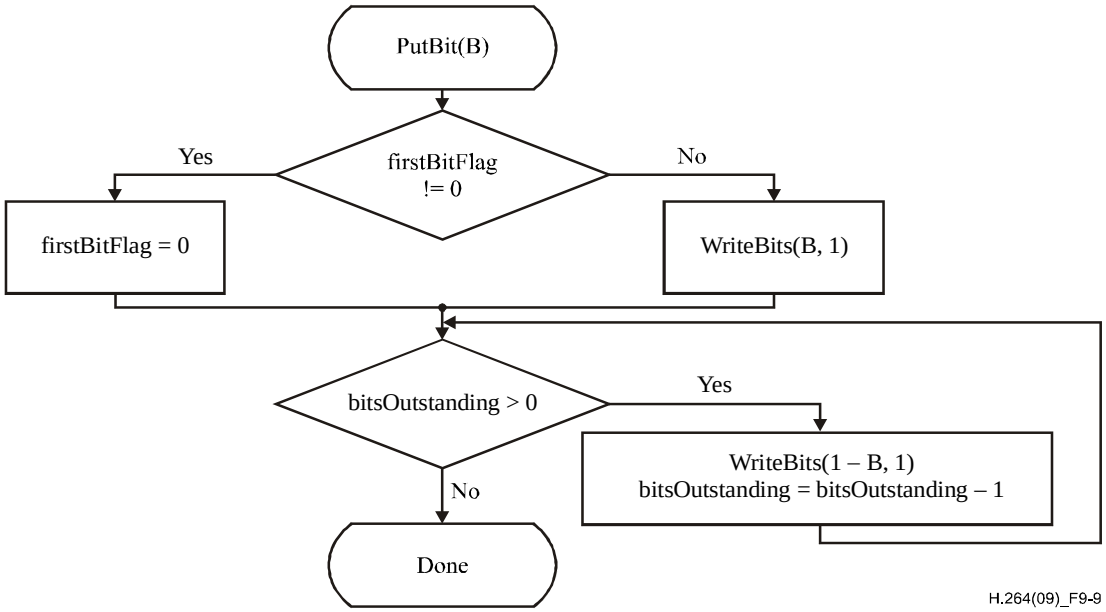


Figure 9-12 – Flowchart of PutBit(B)

9.2.4.4 Bypass encoding process for binary decisions (informative)

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

Inputs to this process are the variables binVal, codILow, codIRange, bitsOutstanding, and BinCountsInNALunits.

Output of this process is a bit written to the RBSP and the updated variables codILow, bitsOutstanding, and BinCountsInNALunits.

This encoding process applies to all binary decisions with bypassFlag equal to 1. Renormalization is included in the specification of this process as given in Figure 9-13.

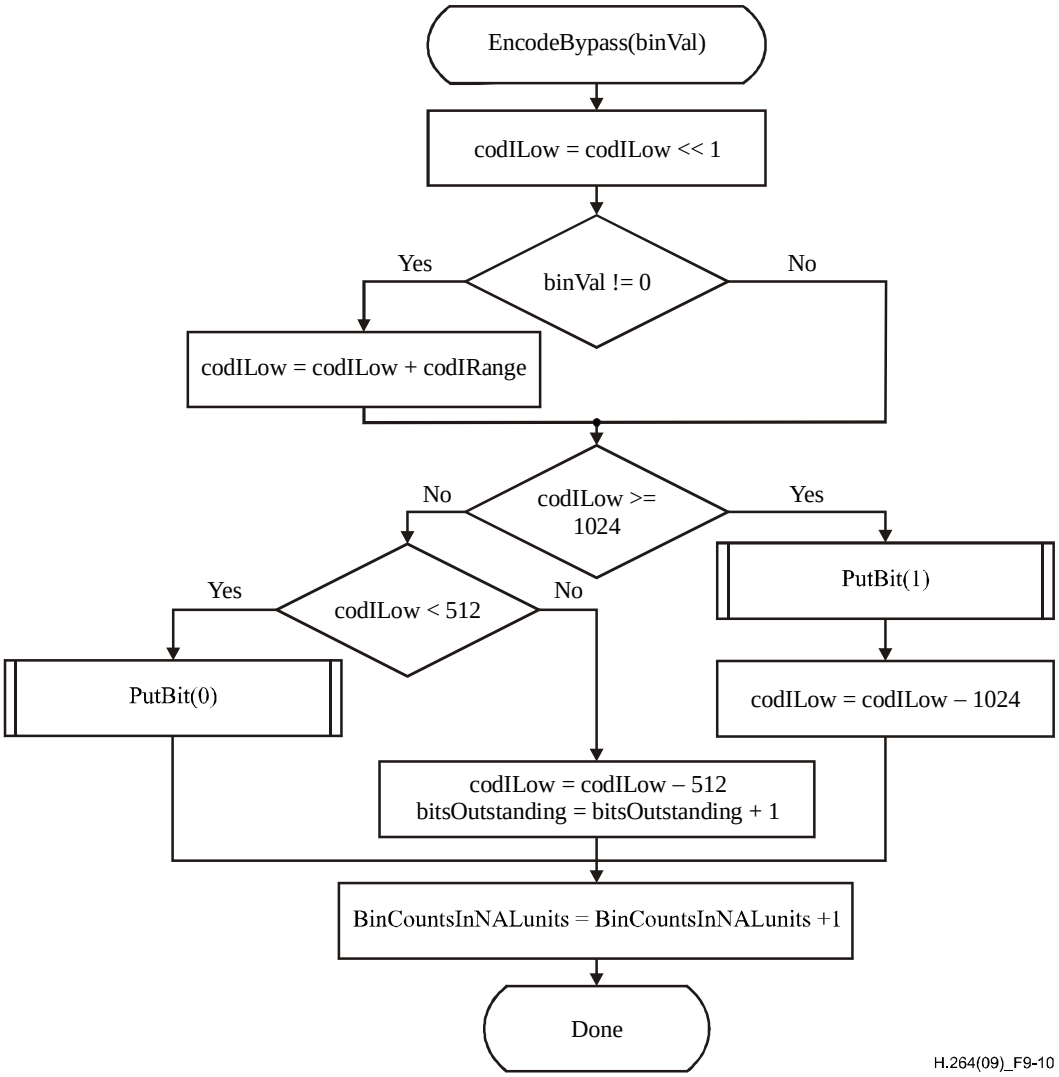


Figure 9-13 – Flowchart of encoding bypass

9.2.4.5 Encoding process for a binary decision before termination (informative)

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

Inputs to this process are the variables binVal, codIRange, codILow, bitsOutstanding, and BinCountsInNALunits.

Outputs of this process are zero or more bits written to the RBSP and the updated variables codILow, codIRange, bitsOutstanding, and BinCountsInNALunits.

This encoding routine shown in Figure 9-14 applies to encoding of end_of_slice_segment_flag, end_of_sub_stream_one_bit and pcm_flag, all associated with ctxIdx equal to 0.

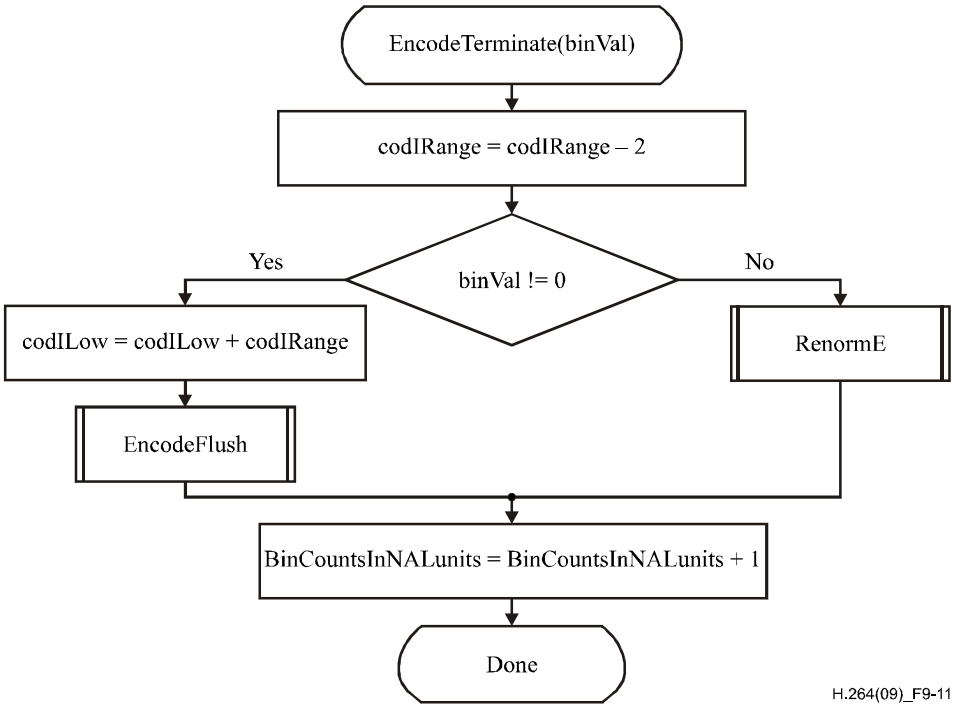


Figure 9-14 – Flowchart of encoding a decision before termination

When the value of binVal to encode is equal to 1, CABAC encoding is terminated and the flushing procedure shown in Figure 9-15 is applied. In this flushing procedure, the last bit written by WriteBits(B, N) is equal to 1. When encoding end_of_slice_segment_flag, this last bit is interpreted as the rbsp_stop_one_bit.

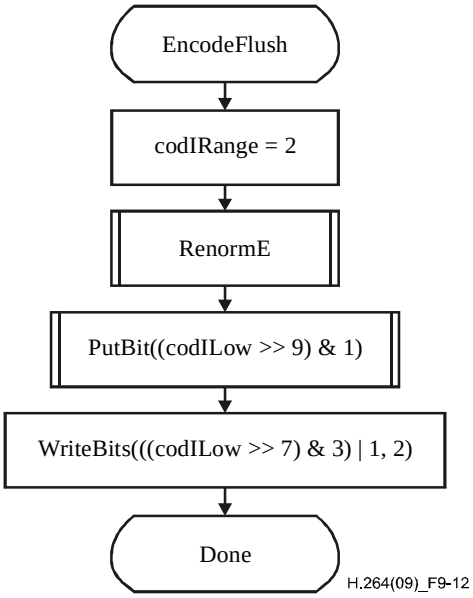


Figure 9-15 – Flowchart of flushing at termination

9.2.4.6 Byte stuffing process (informative)

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

This process is invoked after encoding the last coding block of the last slice segment of a picture and after encapsulation.

Inputs to this process are the number of bytes NumBytesInVclNALunits of all VCL NAL units of a picture, the number of minimum CUs PicSizeInMinCbsY in the picture, and the number of binary symbols BinCountsInNALunits resulting from encoding the contents of all VCL NAL units of the picture.

NOTE – The value of BinCountsInNALunits is the result of encoding all slice segments of a coded picture. After initializing for the first slice segment of a coded picture as specified in subclause 9.2.4.1, BinCountsInNALunits is incremented as specified in subclauses 9.2.4.2, 9.2.4.4, and 9.2.4.5.

Outputs of this process are zero or more bytes appended to the NAL unit.

Let the variable k be set equal to Ceil((Ceil(3 * (32 * BinCountsInNALunits − RawMinCuBits * PicSizeInMinCbsY) ÷ 1024) − NumBytesInVclNALunits) ÷ 3). Depending on the variable k the following applies.

– If k is less than or equal to 0, no cabac_zero_word is appended to the NAL unit.

– Otherwise (k is greater than 0), the 3-byte sequence 0x000003 is appended k times to the NAL unit after encapsulation, where the first two bytes 0x0000 represent a cabac_zero_word and the third byte 0x03 represents an emulation_prevention_three_byte.

10 Specification of bitstream subsets

Subclause 10.1 specifies the sub-bitstream extraction process.

10.1 Sub-bitstream extraction process

It is requirement of bitstream conformance that any sub-bitstream that is included in the output of the process specified in this subclause with tIdTarget equal to any value in the range of 0 to 6, inclusive, and with layerIdSetTarget containing the value 0 only shall be conforming to the requirements of bitstream conformance to this Specification.

NOTE – A conforming bitstream contains one or more coded slice segment NAL units with nuh_reserved_zero_6bits equal to 0 and TemporalId equal to 0.

Inputs to this process are a bitstream, a variable tIdTarget and a set layerIdSetTarget.

Output of this process is a sub-bitstream.

The sub-bitstream is derived by removing from the bitstream all NAL units with TemporalId greater than tIdTarget or nuh_reserved_zero_6bits not among the values included in layerIdSetTarget.

Annex A

Profiles, tiers and levels

(This annex forms an integral part of this Recommendation | International Standard)

A.1 Overview of profiles, tiers and levels

Profiles, tiers and levels specify restrictions on bitstreams and hence limits on the capabilities needed to decode the bitstreams. Profiles, tiers and levels may also be used to indicate interoperability points between individual decoder implementations.

NOTE 1 – This Specification does not include individually selectable "options" at the decoder, as this would increase interoperability difficulties.

Each profile specifies a subset of algorithmic features and limits that shall be supported by all decoders conforming to that profile.

NOTE 2 – Encoders are not required to make use of any particular subset of features supported in a profile.

Each level 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 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.

The profiles that are specified in subclause A.3 are also referred to as the profiles specified in Annex A.

A.2 Requirements on video decoder capability

Capabilities of video decoders conforming to this Specification are specified in terms of the ability to decode video streams conforming to the constraints of profiles and levels specified in this annex. For each such profile, the level supported for that profile shall also be expressed.

Specific values are specified in this annex for the syntax elements general_profile_idc, general_tier_flag, and general_level_idc. All other values of profile_idc, general_tier_flag, and level_idc are reserved for future use by ITU-T | ISO/IEC.

NOTE – Decoders should not infer that when a reserved value of general_profile_idc, general_tier_flag or general_level_idc falls between the values specified in this Specification that this indicates intermediate capabilities between the specified profiles or levels, as there are no restrictions on the method to be chosen by ITU-T | ISO/IEC for the use of such future reserved values. [Ed. (GJS): Do we really mean that? Perhaps we should say exactly the opposite of what this statement says.]

A.3 Profiles

A.3.1 General

All constraints for picture parameter sets that are specified are constraints for picture parameter sets that are activated in the bitstream. All constraints for sequence parameter sets that are specified are constraints for sequence parameter sets that are activated in the bitstream.

The variable RawCtuBits is derived as

$$\begin{aligned} RawCtuBits = {} & CtbSizeY * CtbSizeY * BitDepth_Y + \\ & 2 * (CtbWidthC * CtbHeightC) * BitDepth_C \end{aligned} \tag{A-1}$$

A.3.2 Main profile

Bitstreams conforming to the Main profile shall obey the following constraints:

– Sequence parameter sets shall have chroma_format_idc equal to 1 only.

– Sequence parameter sets shall have bit_depth_luma_minus8 equal to 0 only.

– Sequence parameter sets shall have bit_depth_chroma_minus8 equal to 0 only.

– Log2CtbSizeY shall be in the range from 4 to 6, inclusive.

– When a picture parameter set has tiles_enabled_flag is equal to 1, it shall have entropy_coding_sync_enabled_flag equal to 0.

– When a picture parameter set has tiles_enabled_flag is equal to 1, ColumnWidthInLumaSamples[i] shall be greater than or equal to 256 for all values of i in the range of 0 to num_tile_columns_minus1, inclusive, and RowHeightInLumaSamples[j] shall be greater than or equal to 64 for all values of j in the range of 0 to num_tile_rows_minus1, inclusive.

– The number of times read_bits(1) is called in subclauses 9.2.3.2.2 and 9.2.3.2.3 when parsing coding_tree_unit( ) data for any coding tree unit shall be less than or equal to 4 * RawCtuBits / 3.

– The level constraints specified for the Main profile in subclause A.4 shall be fulfilled.

Conformance of a bitstream to the Main profile is indicated by general_profile_idc being equal to 1 or general_profile_compatibility_flag[1] being equal to 1.

NOTE – When general_profile_compatibility_flag[1] is equal to 1, general_profile_compatibility_flag[2] should also be equal to 1.

Decoders conforming to the Main profile at a specific level (identified by a specific value of general_level_idc) shall be capable of decoding all bitstreams for which the all of following conditions apply:

– general_profile_compatibility_flag[1] is equal to 1.

– general_level_idc represents a level lower than or equal to the specified level.

– general_tier_flag represents a tier lower than or equal to the specified tier.

A.3.3 Main 10 profile

Bitstreams conforming to the Main 10 profile shall obey the following constraints:

– Sequence parameter sets shall have chroma_format_idc equal to 1 only.

– Sequence parameter sets shall have bit_depth_luma_minus8 in the range of 0 to 2, inclusive.

– Sequence parameter sets shall have bit_depth_chroma_minus8 in the range of 0 to 2, inclusive.

– Log2CtbSizeY shall be in the range from 4 to 6, inclusive.

– When a picture parameter set has tiles_enabled_flag is equal to 1, it shall have entropy_coding_sync_enabled_flag equal to 0.

– When a picture parameter set has tiles_enabled_flag is equal to 1, ColumnWidthInLumaSamples[i] shall be greater than or equal to 256 for all values of i in the range of 0 to num_tile_columns_minus1, inclusive, and RowHeightInLumaSamples[j] shall be greater than or equal to 64 for all values of j in the range of 0 to num_tile_rows_minus1, inclusive.

– The number of times read_bits(1) is called in subclauses 9.2.3.2.2 and 9.2.3.2.3 when parsing coding_tree_unit( ) data for any coding tree unit shall be less than or equal to 4 * RawCtuBits / 3.

– The level constraints specified for the Main 10 profile in subclause A.4 shall be fulfilled.

Conformance of a bitstream to the Main 10 profile is indicated by general_profile_idc being equal to 2 or general_profile_compatibility_flag[2] being equal to 1.

Decoders conforming to the Main 10 profile at a specific level (identified by a specific value of general_level_idc) shall be capable of decoding all bitstreams for which the all of following conditions apply:

– general_profile_compatibility_flag[1] is equal to 1 or general_profile_compatibility_flag[2] is equal to 1.

– general_level_idc represents a level lower than or equal to the specified level.

– general_tier_flag represents a tier lower than or equal to the specified tier.

A.3.4 Main Still Picture profile

Bitstreams conforming to the Main Still Picture profile shall obey the following constraints:

– The bitstream shall contain only one picture.

– Sequence parameter sets shall have chroma_format_idc equal to 1 only.

– Sequence parameter sets shall have bit_depth_luma_minus8 equal to 0 only.

– Sequence parameter sets shall have bit_depth_chroma_minus8 equal to 0 only.

– Sequence parameter sets shall have sps_max_dec_pic_buffering[sps_max_sub_layers_minus1] equal to 0 only.

– Log2CtbSizeY shall be in the range from 4 to 6, inclusive.

– When a picture parameter set has tiles_enabled_flag is equal to 1, it shall have entropy_coding_sync_enabled_flag equal to 0.

– When a picture parameter set has tiles_enabled_flag is equal to 1, ColumnWidthInLumaSamples[i] shall be greater than or equal to 256 for all values of i in the range of 0 to num_tile_columns_minus1, inclusive, and RowHeightInLumaSamples[j] shall be greater than or equal to 64 for all values of j in the range of 0 to num_tile_rows_minus1, inclusive.

– The number of times read_bits(1) is called in subclauses 9.2.3.2.2 and 9.2.3.2.3 when parsing coding_tree_unit( ) data for any coding tree unit shall be less than or equal to 4 * RawCtuBits / 3.

– The level constraints specified for the Main Still Picture profile in subclause A.4 shall be fulfilled.

Conformance of a bitstream to the Main Still Picture profile is indicated by general_profile_idc being equal to 3 or general_profile_compatibility_flag[3] being equal to 1.

Decoders conforming to the Main Still Picture profile at a specific level (identified by a specific value of general_level_idc) shall be capable of decoding all bitstreams for which the all of following conditions apply:

– general_profile_compatibility_flag[3] is equal to 1.

– general_level_idc represents a level lower than or equal to the specified level.

– general_tier_flag represents a tier lower than or equal to the specified tier.

A.4 Tiers and levels

A.4.1 General tier and level limits

For purposes of comparison of tier capabilities, the tier with general_tier_flag equal to 0 shall be 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 shall be considered to be a lower level than some other level if the level is listed a higher row of Table A-1 than the other level. [Ed. (GJS): If we follow the guidance in the note above, comparison of level values will not necessarily work properly when a reserved level value is used in the future.]

The following is specified for expressing the constraints in this annex.

– Let access unit n be the n-th access unit in decoding order, with the first access unit being access unit 0.

– Let picture n be the coded picture or the corresponding decoded picture of access unit n.

– Let the variable cpbBrVclFactor be equal to 1000.

– Let the variable cpbBrNalFactor be equal to 1100.

Bitstreams conforming to a profile at a specified level shall obey the following constraints for each bitstream conformance test as specified in Annex C:

a) PicSizeInSamplesY <= MaxLumaPS, where MaxLumaPS is specified in Table A-1.

b) pic_width_in_luma_samples <= Sqrt(MaxLumaPS * 8)

c) pic_height_in_luma_samples <= Sqrt(MaxLumaPS * 8)

d) sps_max_dec_pic_buffering[HighestTid] <= MaxDpbSize, where MaxDpbSize is derived as specified by the following:

```
if ( PicSizeInSamplesY <= ( MaxLumaPS >> 2 ) )
    MaxDpbSize = Min( 4 * maxDpbPicBuf, 16 )
else if ( PicSizeInSamplesY <= ( MaxLumaPS >> 1 ) )
    MaxDpbSize = Min( 2 * maxDpbPicBuf, 16 )
else if ( PicSizeInSamplesY <= ( ( 3 * MaxLumaPS ) >> 2 ) )
    MaxDpbSize = Min( (4 * maxDpbPicBuf) / 3,  16 )
else
    MaxDpbSize = maxDpbPicBuf
```

where MaxLumaPS is specified in Table A-1 and maxDpbPicBuf is equal to 6.

e) For level 5 and higher levels, the variable CtbSizeY shall be equal to 32 or 64.

f) The value of NumPocTotalCurr shall be less than or equal to 8.

g) The value of num_tile_columns_minus1 shall be less than MaxTileCols and num_tile_rows_minus1 shall be less than MaxTileRows, where MaxTileCols and MaxTileRows are as specified in Table A-1.

h) For the VCL HRD parameters, CpbSize[i] <= cpbBrVclFactor * MaxCPB for at least one value of i in the range of 0 to cpb_cnt_minus1[HighestTid], inclusive, where CpbSize[i] is specified in subclause E.2.3 based on parameters specified in subclause C.1 and MaxCPB is specified in Table A-1 in units of cpbBrVclFactor bits.

i) For the NAL HRD parameters, CpbSize[i] <= cpbBrNalFactor * MaxCPB for at least one value of i in the range of 0 to cpb_cnt_minus1[HighestTid], inclusive, where CpbSize[i] is specified in subclause E.2.3 based on parameters specified in subclause C.1 and MaxCPB is specified in Table A-1 in units of cpbBrNalFactor bits.

Table A-1 specifies the limits for each level. The use of the MinCR parameter column of Table A-1 is specified in subclause A.4.2.

A tier and level to which the bitstream conforms shall be indicated by the syntax element general_level_idc as follows.

– general_tier_flag equal to 0 indicates conformance to the Main tier, and general_tier_flag equal to 1 indicates conformance to the High tier, according to the tier constraint specifications in Table A-1. general_tier_flag shall be equal to 0 for levels below level 4 (corresponding to the entries in Table A-1 marked with "-").

– level_idc shall be set equal to a value of 30 times the level number specified in Table A-1.

[Ed. (YK): Setting general_level_idc to 30 times instead of 10 times the level number helps avoid the level 1b problem that occurred in AVC. (GJS): However, it only gives us a limited range of higher values.]

Table A-1 – General tier and level limits

Level	Max luma picture size MaxLumaPS (samples)	Max CPB size MaxCPB (1000 bits)		Max slice segments per picture MaxSliceSegmentsPerPicture	Max # of tile rows MaxTileRows	Max # of tile columns MaxTileCols
		Main tier	High tier			
1	36 864	350	-	16	1	1
2	122 880	1 500	-	16	1	1
2.1	245 760	3 000	-	20	1	1
3	552 960	6 000	-	30	2	2
3.1	983 040	10 000	-	40	3	3
4	2 228 224	12 000	30 000	75	5	5
4.1	2 228 224	20 000	50 000	75	5	5
5	8 912 896	25 000	100 000	200	11	10
5.1	8 912 896	40 000	160 000	200	11	10
5.2	8 912 896	60 000	240 000	200	11	10
6	35 651 584	60 000	240 000	600	22	20
6.1	35 651 584	120 000	480 000	600	22	20
6.2	35 651 584	240 000	800 000	600	22	20

Informative subclause A.4.3 shows the effect of these limits on picture rates for several example picture formats.

A.4.2 Profile-specific level limits for the Main and Main 10 profiles

The following is specified for expressing the constraints in this annex.

– Let the variable fR be set to 1 ÷ 300.

Bitstreams conforming to the Main or Main 10 profile at a specified tier and level shall obey the following constraints for each bitstream conformance test as specified in Annex C:

a) The nominal removal time of access unit n (with n > 0) from the CPB as specified in subclause C.2.3, satisfies the constraint that $t_{r,n}(n) - t_r(n - 1)$ is greater than or equal to Max(PicSizeInSamplesY ÷ MaxLumaSR, fR) for the value of PicSizeInSamplesY of picture n − 1, where MaxLumaSR is the value specified in Table A-2 that applies to picture n − 1.

b) The difference between consecutive output times of pictures from the DPB as specified in subclause C.3.3, satisfies the constraint that $\Delta t_{o,dpb}(n) >= Max(PicSizeInSamplesY \div MaxLumaSR, fR)$ for the value of PicSizeInSamplesY of picture n, where MaxLumaSR is the value specified in Table A-2 for picture n, provided that picture n is a picture that is output and is not the last picture of the bitstream that is output.

c) The removal time of access unit 0 shall satisfy the constraint that the number of slice segments in picture 0 is less than or equal to Min(MaxSliceSegmentsPerPicture * MaxLumaSR / MaxLumaPS * ($t_r(0) - t_{r,n}(0)$) + MaxSliceSegmentsPerPicture * PicSizeInSamplesY / MaxLumaPS, MaxSliceSegmentsPerPicture), for the value of PicSizeInSamplesY of picture 0, where MaxSliceSegmentsPerPicture, MaxLumaPS and MaxLumaSR are the values specified in Table A-1 and Table A-2, respectively, that apply to picture 0.

d) The difference between consecutive removal time of access units n and n − 1 (with n > 0) shall satisfy the constraint that the number of slice segments in picture n is less than or equal to Min(MaxSliceSegmentsPerPicture * MaxLumaSR / MaxLumaPS * ($t_r(n) - t_r(n - 1)$), MaxSliceSegmentsPerPicture), where MaxSliceSegmentsPerPicture, MaxLumaPS and MaxLumaSR are the values specified in Table A-1 and Table A-2, respectively, that apply to picture n.

e) For the VCL HRD parameters, BitRate[i] <= cpbBrVclFactor * MaxBR and CpbSize[i] <= cpbBrVclFactor * MaxCPB for at least one value of i in the range of 0 to cpb_cnt_minus1[HighestTid], inclusive, where BitRate[i] and CpbSize[i] are specified in subclause E.2.3 based on parameters specified in subclause C.1, MaxCPB is specified in Table A-1 in units of cpbBrVclFactor bits, and MaxBR is specified in Table A-2 in units of cpbBrVclFactor bits/s.

f) For the NAL HRD parameters, BitRate[i] <= cpbBrNalFactor * MaxBR and CpbSize[i] <= cpbBrNalFactor * MaxCPB for at least one value of i, where BitRate[i] and CpbSize[i] are specified in subclause E.2.3 based on parameters specified in subclause C.1, MaxCPB is specified in Table A-1 in units of cpbBrNalFactor bits, and MaxBR is specified in Table A-2 in units of cpbBrNalFactor bits/s.

g) The sum of the NumBytesInNALunit variables for access unit 0 is less than or equal to 1.5 * (Max(PicSizeInSamplesY, fR * MaxLumaSR) + MaxLumaSR * ($t_r(0) - t_{r,n}(0)$)) ÷ MinCR for the value of PicSizeInSamplesY of picture 0, where MaxLumaPR and MinCR are the values specified in Table A-1 that apply to picture 0.

h) The sum of the NumBytesInNALunit variables for access unit n with n > 0 is less than or equal to 1.5 * MaxLumaSR * ($t_r(n) - t_r(n - 1)$) ÷ MinCR, where MaxLumaSR and MinCR are the values specified in Table A-1 that apply to picture n.

i) The removal time of access unit 0 shall satisfy the constraint that the number of tiles in picture 0 is less than or equal to Min(MaxTileCols * MaxTileRows * 120 * ($t_r(0) - t_{r,n}(0)$) + MaxTileCols * MaxTileRows * PicSizeInSamplesY / MaxLumaPS, MaxTileCols * MaxTileRows), for the value of PicSizeInSamplesY of picture 0, where MaxTileCols and MaxTileRows are the values specified in Table A-1 that apply to picture 0.

j) The difference between consecutive removal time of access units n and n − 1 (with n > 0) shall satisfy the constraint that the number of tiles in picture n is less than or equal to Min(MaxTileCols * MaxTileRows * 120 * ($t_r(n) - t_r(n - 1)$), MaxTileCols * MaxTileRows), where MaxTileCols and MaxTileRows are the values specified in Table A-1 that apply to picture n.

[Ed. Note (KM): Check against Annex E. (GJS) What needs to be checked?]

k)

Table A-2 – Tier and level limits for the Main and Main 10 profiles

Level	Max luma sample rate MaxLumaSR (samples/sec)	Max bit rate MaxBR (1000 bits/s)		Min Compression Ratio MinCR
		Main tier	High tier	
1	552 960	128	-	2
2	3 686 400	1 500	-	2
2.1	7 372 800	3 000	-	2
3	16 588 800	6 000	-	2
3.1	33 177 600	10 000	-	2
4	66 846 720	12 000	30 000	4
4.1	133 693 440	20 000	50 000	4
5	267 386 880	25 000	100 000	6
5.1	534 773 760	40 000	160 000	8
5.2	1 069 547 520	60 000	240 000	8
6	1 069 547 520	60 000	240 000	8
6.1	2 139 095 040	120 000	480 000	8
6.2	4 278 190 080	240 000	800 000	6

A.4.3 Effect of level limits on picture rate for the Main and Main 10 profiles (informative)

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

Informative Tables A-3 and A-4 provide examples of maximum picture rates for the Main and Main 10 profiles for various picture formats when MinCbSizeY is equal to 64.

Table A-3 – Maximum picture rates (pictures per second) at level 1 to 4.3 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	**1024**	**768**	786 432	-	-	-	-	42.1	85.0	170.0
720p HD	**1280**	**720**	983 040	-	-	-	-	33.7	68.0	136.0
4VGA	**1280**	**960**	1 228 800	-	-	-	-	-	54.4	108.8
SXGA	**1280**	**1024**	1 310 720	-	-	-	-	-	51.0	102.0
525 16SIF	**1408**	**960**	1 351 680	-	-	-	-	-	49.4	98.9
16CIF	**1408**	**1152**	1 622 016	-	-	-	-	-	41.2	82.4
4SVGA	**1600**	**1200**	1 945 600	-	-	-	-	-	34.3	68.7
1080 HD	**1920**	**1080**	2 088 960	-	-	-	-	-	32.0	64.0
2Kx1K	**2048**	**1024**	2 097 152	-	-	-	-	-	31.8	63.7
2Kx1080	**2048**	**1080**	2 228 224	-	-	-	-	-	30.0	60.0
4XGA	**2048**	**1536**	3 145 728	-	-	-	-	-	-	-
16VGA	**2560**	**1920**	4 915 200	-	-	-	-	-	-	-
3616x1536 (2.35:1)	**3616**	**1536**	5 603 328	-	-	-	-	-	-	-
3672x1536 (2.39:1)	**3680**	**1536**	5 701 632	-	-	-	-	-	-	-
3840x2160 (4*HD)	**3840**	**2160**	8 355 840	-	-	-	-	-	-	-
4Kx2K	**4096**	**2048**	8 388 608	-	-	-	-	-	-	-
4096x2160	**4096**	**2160**	8 912 896	-	-	-	-	-	-	-
4096x2304 (16:9)	**4096**	**2304**	9 437 184	-	-	-	-	-	-	-
7680x4320	**7680**	**4320**	33 423 360	-	-	-	-	-	-	-
8192x4096	**8192**	**4096**	33 554 432	-	-	-	-	-	-	-
8192x4320	**8192**	**4320**	35 651 584	-	-	-	-	-	-	-

Table A-4 – 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	1024	768	786 432	300.0	300.0	300.0	300.0	300.0	300.0
720p HD	1280	720	983 040	272.0	300.0	300.0	300.0	300.0	300.0
4VGA	1280	960	1 228 800	217.6	300.0	300.0	300.0	300.0	300.0
SXGA	1280	1024	1 310 720	204.0	300.0	300.0	300.0	300.0	300.0
525 16SIF	1408	960	1 351 680	197.8	300.0	300.0	300.0	300.0	300.0
16CIF	1408	1152	1 622 016	164.8	300.0	300.0	300.0	300.0	300.0
4SVGA	1600	1200	1 945 600	137.4	274.8	300.0	300.0	300.0	300.0
1080 HD	1920	1080	2 088 960	128.0	256.0	300.0	300.0	300.0	300.0
2Kx1K	2048	1024	2 097 152	127.5	255.0	300.0	300.0	300.0	300.0
2Kx1080	2048	1080	2 228 224	120.0	240.0	300.0	300.0	300.0	300.0
4XGA	2048	1536	3 145 728	85.0	170.0	300.0	300.0	300.0	300.0
16VGA	2560	1920	4 915 200	54.4	108.8	217.6	217.6	300.0	300.0
3616x1536 (2.35:1)	3616	1536	5 603 328	47.7	95.4	190.8	190.8	300.0	300.0
3672x1536 (2.39:1)	3680	1536	5 701 632	46.8	93.7	187.5	187.5	300.0	300.0
3840x2160 (4*HD)	3840	2160	8 355 840	32.0	64.0	128.0	256.0	300.0	300.0
4Kx2K	4096	2048	8 388 608	31.8	63.7	127.5	127.5	255.0	300.0
4096x2160	4096	2160	8 912 896	30.0	60.0	120.0	120.0	240.0	300.0
4096x2304 (16:9)	4096	2304	9 437 184	-	-	-	113.3	226.6	300.0
7680x4320	7680	4320	33 423 360	-	-	-	32.0	64.0	128.0
8192x4096	8192	4096	33 554 432	-	-	-	31.8	63.7	127.5
8192x4320	8192	4320	35 651 584	-	-	-	30.0	60.0	120.0

The following should be noted in regard to the examples shown in Tables A-3 and A-4:

– This Specification 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 pic_width_in_luma_samples and 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.

– As used in the examples, "525" refers to typical use for environments using 525 analogue scan lines (of which approximately 480 lines contain the visible picture region), and "625" refers to environments using 625 analogue scan lines (of which approximately 576 lines contain the visible picture region).

– XGA is also known as (aka) XVGA, 4SVGA aka UXGA, 16XGA aka 4Kx3K, CIF aka 625 SIF, 625 HHR aka 2CIF aka half 625 D-1, aka half 625 ITU-R BT.601, 525 SD aka 525 D-1 aka 525 ITU-R BT.601, 625 SD aka 625 D-1 aka 625 ITU-R BT.601.

Annex B

Byte stream format

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies syntax and semantics of a byte stream format specified for use by applications that deliver some or all of the NAL unit stream as an ordered stream of bytes or bits within which the locations of NAL unit boundaries need to be identifiable from patterns in the data, such as Rec. ITU-T H.222.0 | ISO/IEC 13818-1 systems or Rec. ITU-T H.320 systems. For bit-oriented delivery, the bit order for the byte stream format is specified to start with the MSB of the first byte, proceed to the LSB of the first byte, followed by the MSB of the second byte, etc.

The byte stream format consists of a sequence of byte stream NAL unit syntax structures. Each byte stream NAL unit syntax structure contains one start code prefix followed by one nal_unit(NumBytesInNALunit) syntax structure. It may (and under some circumstances, it shall) also contain an additional zero_byte syntax element. It may also contain one or more additional trailing_zero_8bits syntax elements. When it is the first byte stream NAL unit in the bitstream, it may also contain one or more additional leading_zero_8bits syntax elements.

B.1 Byte stream NAL unit syntax and semantics

B.1.1 Byte stream NAL unit syntax

byte_stream_nal_unit(NumBytesInNALunit) {	C	Descriptor
while(next_bits(24) != 0x000001 && next_bits(32) != 0x00000001)		
leading_zero_8bits /* equal to 0x00 */		f(8)
if(next_bits(24) != 0x000001)		
zero_byte /* equal to 0x00 */		f(8)
start_code_prefix_one_3bytes /* equal to 0x000001 */		f(24)
nal_unit(NumBytesInNALunit)		
while(more_data_in_byte_stream( ) && next_bits(24) != 0x000001 && next_bits(32) != 0x00000001)		
trailing_zero_8bits /* equal to 0x00 */		f(8)
}		

B.1.2 Byte stream NAL unit semantics

The order of byte stream NAL units in the byte stream shall follow the decoding order of the NAL units contained in the byte stream NAL units (see subclause 7.4.1.4). The content of each byte stream NAL unit is associated with the same access unit as the NAL unit contained in the byte stream NAL unit (see subclause 7.4.1.4.3).

leading_zero_8bits is a byte equal to 0x00.

NOTE – The leading_zero_8bits syntax element can only be present in the first byte stream NAL unit of the bitstream, because (as shown in the syntax diagram of subclause B.1.1) any bytes equal to 0x00 that follow a NAL unit syntax structure and precede the four-byte sequence 0x00000001 (which is to be interpreted as a zero_byte followed by a start_code_prefix_one_3bytes) will be considered to be trailing_zero_8bits syntax elements that are part of the preceding byte stream NAL unit.

zero_byte is a single byte equal to 0x00.

When 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( ) is equal to VPS_NUT, SPS_NUT or PPS_NUT,

– The byte stream NAL unit syntax structure contains the first NAL unit of an access unit in decoding order, as specified by subclause 7.4.1.4.3.

start_code_prefix_one_3bytes is a fixed-value sequence of 3 bytes equal to 0x000001. This syntax element is called a start code prefix.

trailing_zero_8bits is a byte equal to 0x00.

B.2 Byte stream NAL unit decoding process

Input to this process consists of an ordered stream of bytes consisting of a sequence of byte stream NAL unit syntax structures.

Output of this process consists of a sequence of NAL unit syntax structures.

At the beginning of the decoding process, the decoder 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 (if present), moving the current position in the byte stream forward one byte at a time, until the current position in the byte stream is such that the next four bytes in the bitstream form the four-byte sequence 0x00000001.

The decoder then performs the following step-wise process repeatedly to extract and decode each NAL unit syntax structure in the byte stream until the end of the byte stream has been encountered (as determined by unspecified means) and the last NAL unit in the byte stream has been decoded:

1. When the next four bytes in the bitstream form the four-byte sequence 0x00000001, the next byte in the byte stream (which is a zero_byte syntax element) is extracted and discarded and the current position in the byte stream is set equal to the position of the byte following this discarded byte.
2. The next three-byte sequence in the byte stream (which is a start_code_prefix_one_3bytes) is extracted and discarded and the current position in the byte stream is set equal to the position of the byte following this three-byte sequence.
3. NumBytesInNALunit is set equal to the number of bytes starting with the byte at the current position in the byte stream up to and including the last byte that precedes the location of 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.
4. NumBytesInNALunit bytes are removed from the bitstream and the current position in the byte stream is advanced by NumBytesInNALunit bytes. This sequence of bytes is nal_unit(NumBytesInNALunit) and is decoded using the NAL unit decoding process.
5. When the current position in the byte stream is not at the end of the byte stream (as determined by unspecified means) and the next bytes in the byte stream do not start with a three-byte sequence equal to 0x000001 and the next bytes in the byte stream do not start with a four byte sequence equal to 0x00000001, the decoder extracts and discards each trailing_zero_8bits syntax element, moving the current position in the byte stream forward one byte at a time, until the current position in the byte stream is such that the next bytes in the byte stream form the four-byte sequence 0x00000001 or the end of the byte stream has been encountered (as determined by unspecified means).

B.3 Decoder byte-alignment recovery (informative)

This subclause does not form an integral part of this 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 subclause.

A decoder is said to have byte-alignment with a bitstream when the decoder is able to determine whether or not the positions of data in the bitstream are byte-aligned. When a decoder does not have byte alignment with the encoder's byte stream, the decoder may examine the incoming bitstream for the binary pattern '00000000 00000000 00000000 00000001' (31 consecutive bits equal to 0 followed by a bit equal to 1). The bit immediately following this pattern is the first bit of an aligned byte following a start code prefix. Upon detecting this pattern, the decoder will be byte aligned with the encoder and positioned at the start of a NAL unit in the byte stream.

Once byte aligned with the encoder, the decoder can examine the incoming byte stream for subsequent three-byte sequences 0x000001 and 0x000003.

When the three-byte sequence 0x000001 is detected, this is a start code prefix.

When the three-byte sequence 0x000003 is detected, the third byte (0x03) is an emulation_prevention_three_byte to be discarded as specified in subclause 7.4.1.

When an error in the bitstream syntax is detected (e.g. a non-zero value of the forbidden_zero_bit or one of the three-byte or four-byte sequences that are prohibited in subclause 7.4.1), the decoder may consider the detected condition as an indication that byte alignment may have been lost and may discard all bitstream data until the detection of byte alignment at a later position in the bitstream as described in this subclause.

Annex C

Hypothetical reference decoder

(This annex forms an integral part of this Recommendation | International Standard)

C.1 General

This annex specifies the hypothetical reference decoder (HRD) and its use to check bitstream and decoder conformance.

Two types of bitstreams or bitstream subsets are subject to HRD conformance checking for this Specification. The first type, called a Type I bitstream, is a NAL unit stream containing only the VCL NAL units and NAL units with nal_unit_type equal to FD_NUT (filler data NAL units) for all access units 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 access units in the bitstream, at least one of the following:

– additional non-VCL NAL units other than filler data NAL units,

– all leading_zero_8bits, zero_byte, start_code_prefix_one_3bytes, and trailing_zero_8bits syntax elements that form a byte stream from the NAL unit stream (as specified in Annex B).

Figure C-1 shows the types of bitstream conformance points checked by the HRD.

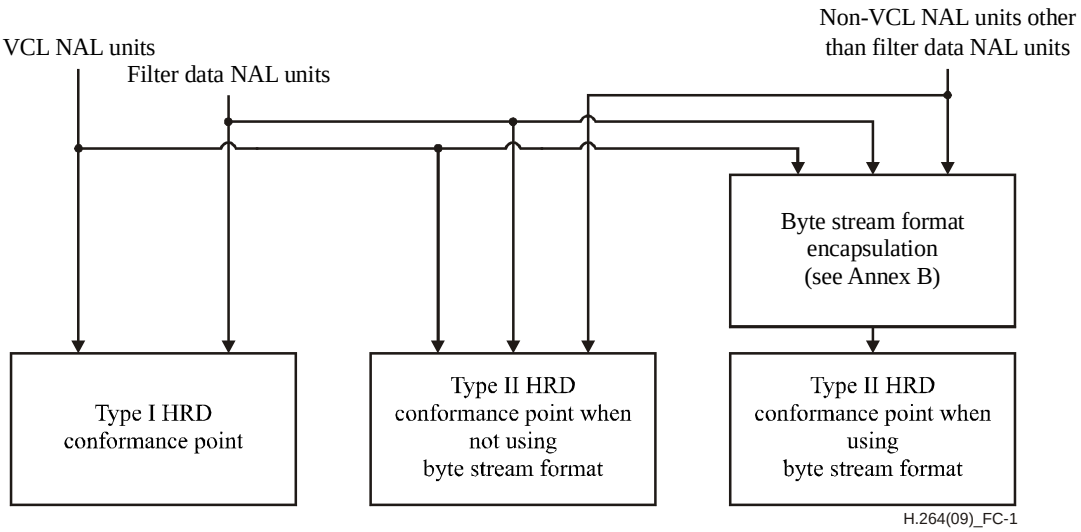


Figure C-1 – Structure of byte streams and NAL unit streams for HRD conformance checks [Ed. (KJS): Text renders poorly on screen – redraw the figure. (BB): change filter data NAL units to filler data NAL units when redrawing.]

The syntax elements of non-VCL NAL units (or their default values for some of the syntax elements), required for the HRD, are specified in the semantic subclauses of clause 7, Annexes D and E.

Two types of HRD parameter sets (NAL HRD parameters and VCL HRD parameters) are used. The HRD parameter sets are signalled through the hrd_parameters( ) syntax structure, which may be part of the sequence parameter set syntax structure or the video parameter set syntax structure.

Multiple tests may be needed for checking the conformance of a bitstream, which is referred to as the bitstream under test in the following. For each test, the following steps apply in the order listed:

1. An operation point under test, denoted as TargetOp, is selected. The OpLayerIdSet of TargetOp contains the set of nuh_reserved_zero_6bits values present in the bitstream subset associated with TargetOp, which is a subset of nuh_reserved_zero_6bits values present in the bitstream under test. The OpTid of TargetOp is equal to the highest TemporalId present in the bitstream subset associated with TargetOp.
2. TargetDecLayerIdSet is set to OpLayerIdSet of TargetOp, and HighestTid is set to OpTid of TargetOp, and BitstreamToDecode is set to the bitstream subset associated with TargetOp, i.e. the output of the sub-bitstream

extraction process as specified in subclause 10.1 with the bitstream under test, HighestTid and TargetDecLayerIdSet as inputs.

3. The hrd_parameters( ) syntax structure and the sub_layer_hrd_parameters( ) syntax structure applicable to TargetOp are selected. If TargetDecLayerIdSet contains all nuh_reserved_zero_6bits values present in the bitstream under test, the hrd_parameters( ) syntax structure in the active sequence parameter set (or provided through an external means not specified in this Specification) is selected. Otherwise, the hrd_parameters( ) syntax structure in the active video parameter set (or provided through some external means not specified in this Specification) that applies to TargetOp is selected. Within the selected hrd_parameters( ) syntax structure, if BitstreamToDecode is a Type I bitstream, the sub_layer_hrd_parameters(HighestTid) syntax structure that immediately follows the condition "if(vcl_hrd_parameters_present_flag)" is selected and the variable NalHrdModeFlag is set equal to 0; otherwise (BitstreamToDecode is a Type II bitstream), the sub_layer_hrd_parameters(HighestTid) syntax structure that immediately follows either the condition "if(vcl_hrd_parameters_present_flag)" (in this case the variable NalHrdModeFlag is set equal to 0) or the condition "if(nal_hrd_parameters_present_flag)" (in this case the variable NalHrdModeFlag is set equal to 1) is selected. When BitstreamToDecode is a Type II bitstream and NalHrdModeFlag is equal to 0, all non-VLC NAL units except for filler data NAL units are discarded from BitstreamToDecode, and the remaining bitstream is assigned to BitstreamToDecode.

   [Ed. (BB): Better define two sets of syntax elements one with "nal_" and one with "vcl_" prefix, instead of using the variable NalHrdModeFlag.]

4. An access unit associated with a buffering period SEI message (present in BitstreamToDecode in a scalable nesting SEI message or available through external means not specified in this Specification) applicable to TargetOp is selected as the HRD initialization point and referred to as access unit 0.
5. For each access unit in BitstreamToDecode starting from access unit 0, the buffering period SEI message (present in BitstreamToDecode in a scalable nesting SEI message or available through external means not specified in this Specification) that is associated with the access unit and applies to TargetOp is selected, the picture timing SEI message (present in BitstreamToDecode in a scalable nesting SEI message or available through external means not specified in this Specification) that is associated with the access unit and applies to TargetOp is selected, and when SubPicCpbFlag is equal to 1 and sub_pic_cpb_params_in_pic_timing_sei_flag is equal to 0, the decoding unit information SEI messages (present in BitstreamToDecode in a scalable nesting SEI message or available through external means not specified in this Specification) that are associated with decoding units in the access unit and apply to TargetOp are selected.
6. A value of SchedSelIdx is selected. The selected SchedSelIdx shall be in the range of 0 to cpb_cnt_minus1[HighestTid], inclusive, where cpb_cnt_minus1[HighestTid] is found in the sub_layer_hrd_parameters(HighestTid) syntax structure as selected above.
7. When the coded picture in access unit 0 has nal_unit_type equal to CRA_NUT or BLA_W_LP, and rap_cpb_params_present_flag in the selected buffering period SEI message is equal to 1, either of the following applies for selection of the initial CPB removal delay and delay offset.
   - The default initial CPB removal delay and delay offset represented by initial_cpb_removal_delay[SchedSelIdx] and initial_cpb_removal_offset[SchedSelIdx] are selected depending on NalHrdModeFlag as specified under step 3 above, the variable DefaultInitCpbParamsFlag is set equal to 1.
   - The alternative initial CPB removal delay and delay offset represented by initial_alt_cpb_removal_delay[SchedSelIdx] and initial_alt_cpb_removal_offset[SchedSelIdx] selected depending on NalHrdModeFlag as specified under step 3 above, the variable DefaultInitCpbParamsFlag is set equal to 0, and the RASL access units associated with access unit 0 are discarded from BitstreamToDecode and the remaining bitstream is still assigned to BitstreamToDecode.
8. When sub_pic_cpb_params_present_flag in the selected hrd_parameters( ) syntax structure is equal to 1, the CPB is scheduled to operate either at the access unit level (in which case the variable SubPicCpbFlag is set equal to 0) or at the sub-picture level (in which case the variable SubPicCpbFlag is set equal to 1).

For each operation point under test, the number of bitstream conformance tests to be performed is equal to n0 * n1 * (n2 * 2 + n3) *n4, where the values of n0, n1, n2, n3, and n4 are specified as follows:

- n0 is derived as follows.
  - If BitstreamToDecode is a Type I bitstream, n0 is equal to 1.
  - Otherwise (BitstreamToDecode is a Type II bitstream), n0 is equal to 2.
- n1 is equal to cpb_cnt_minus1[HighestTid] + 1.
- n2 is the number of access units in BitstreamToDecode that each is associated with a buffering period SEI message applicable to TargetOp and for each of which both of the following conditions are true:

  - nal_unit_type is equal to CRA_NUT or BLA_W_LP for the VCL NAL units;
  - The associated buffering period SEI message applicable to TargetOp has rap_cpb_params_present_flag equal to 1.
- n3 is the number of access units in BitstreamToDecode BitstreamToDecode that each is associated with a buffering period SEI message applicable to TargetOp and for each of which one or both of the following conditions are true:
  - nal_unit_type is equal to neither CRA_NUT nor BLA_W_LP for the VCL NAL units;
  - The associated buffering period SEI message applicable to TargetOp has rap_cpb_params_present_flag equal to 0.
- n4 is derived as follows:
  - If sub_pic_cpb_params_present_flag in the selected hrd_parameters( ) syntax structure is equal to 0, n4 is equal to 1;
  - Otherwise, n4 is equal to 2.

When BitstreamToDecode is a Type II bitstream, if the sub_layer_hrd_parameters(HighestTid) syntax structure that immediately follows the condition "if(vcl_hrd_parameters_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 sub_layer_hrd_parameters(HighestTid) syntax structure that immediately follows the condition "if(nal_hrd_parameters_present_flag)" is selected, the test is conducted at the Type II conformance point shown in Figure C-1, and all NAL units (of a Type II NAL unit stream) or all bytes (of a byte stream) are counted for the input bit rate and CPB storage.

NOTE 1 – NAL HRD parameters established by a value of SchedSelIdx for the Type II conformance point shown in Figure C-1 are sufficient to also establish VCL HRD conformance for the Type I conformance point shown in Figure C-1 for the same values of InitCpbRemovalDelay[SchedSelIdx], BitRate[SchedSelIdx], and CpbSize[SchedSelIdx] for the VBR case (cbr_flag[SchedSelIdx] equal to 0). This is because the data flow into the Type I conformance point is a subset of the data flow into the Type II conformance point and because, for the VBR case, the CPB is allowed to become empty and stay empty until the time a next picture is scheduled to begin to arrive. For example, when decoding a coded video sequence conforming to one or more of the profiles specified in Annex A using the decoding process specified in clauses 2-9, when NAL HRD parameters are provided for the Type II conformance point that not only fall within the bounds set for NAL HRD parameters for profile conformance in item f) of subclause A.4.2 but also fall within the bounds set for VCL HRD parameters for profile conformance in item e) of subclause A.4.2, conformance of the VCL HRD for the Type I conformance point is also assured to fall within the bounds of item e) of subclause A.4.2.

All video parameter sets, sequence parameter sets and picture parameter sets referred to in the VCL NAL units, and the corresponding buffering period, picture timing and decoding unit 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, D, and E, the specification for "presence" of non-VCL NAL units that contain video parameter sets, sequence parameter sets, picture parameter sets. buffering period SEI messages, picture timing SEI messages, or decoding unit 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 by this Specification. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

NOTE 2 – 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 3 – When HRD information is contained within the bitstream, it is possible to verify the conformance of a bitstream to the requirements of this subclause based solely on information contained in the bitstream. When the HRD information is not present in the bitstream, as is the case for all "stand-alone" Type I bitstreams, conformance can only be verified when the HRD data is supplied by some other means not specified in this Specification.

The HRD contains a coded picture buffer (CPB), an instantaneous decoding process, a decoded picture buffer (DPB), and output cropping as shown in Figure C-2.

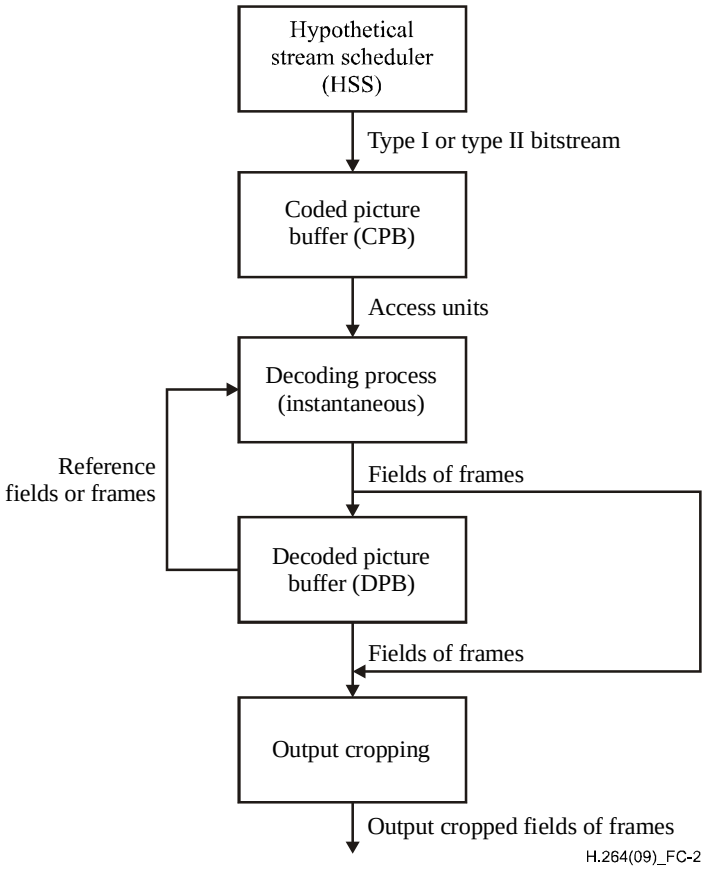


Figure C-2 – HRD buffer model

[Ed. (JS): Remove "fields" from figure. (DF) Rename "frames" to "pictures"? (GJS): Yes, and I believe our current model now always stores the current picture, so the bypass around the DPB should be removed. (YK): And, "Access units" -> "decoding units"; "Reference fields or frames" -> "reference pictures"; "Fields of frames" (first instance) -> "decoded decoding units"; "Fields of frames" (second instance) -> "decoded pictures"; "Output cropped fields of frames" -> "Output cropped pictures"; (MH): There should be two arrows out from DPB, Output cropping (for pics with PicOutputFlag equal to 1) and Removal (for pics with PicOutputFlag equal to 0). (GJS): Be careful with the term "reference pictures". I personally don't see a need for an arrow to show removal of pictures with PicOutputFlag equal to 0. Arrows are to show what comes out of a process, and those just don't come out. It may be confusing to show them coming out, unless the arrow leads to a picture of a trash can.]

For each bitstream conformance test, the CPB size (number of bits) is CpbSize[SchedSelIdx] as specified in subclause E.2.3, where SchedSelIdx and the HRD parameters are specified above in this subclause. The DPB size (number of picture storage buffers) is sps_max_dec_pic_buffering[HighestTid].

The variable SubPicCpbPreferredFlag is either specified by external means, or when not specified by external means, set to 0.

When the value of the variable SubPicCpbFlag has not been set by step 8 above in this subclause, it is derived as follows:

$$\text{SubPicCpbFlag} = \text{SubPicCpbPreferredFlag} \;\&\&\; \text{sub\_pic\_cpb\_params\_present\_flag} \qquad \text{(C-1)}$$

If SubPicCpbFlag is equal to 0, the CPB operates at access unit level and each decoding unit is an access unit. Otherwise the CPB operates at sub-picture level and each decoding unit is a subset of an access unit.

The HRD operates as follows. Data associated with decoding units that flow into the CPB according to a specified arrival schedule are delivered by the HSS. The data associated with each decoding unit are removed and decoded instantaneously by the instantaneous decoding process at the CPB removal time of the decoding unit. 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 subclause C.2, the instantaneous decoder operation is specified in clauses 2-9, the operation of the DPB is specified in subclause C.3, and the output cropping is specified in subclause C.3.3 and subclause .C.5.3.

HSS and HRD information concerning the number of enumerated delivery schedules and their associated bit rates and buffer sizes is specified in subclauses E.1.2 and E.2.2. The HRD is initialized as specified by the buffering period SEI message specified in subclauses D.1.2 and D.2.2. The removal timing of decoding units from the CPB and output timing of decoded pictures from the DPB are specified in the picture timing SEI message and/or in the decoding unit information SEI message specified in subclauses D.1.3, D.1.21, D.2.3, and D.2.21. All timing information relating to a specific decoding unit shall arrive prior to the CPB removal time of the decoding unit.

The requirements for bitstream conformance are specified in subclause C.4, and the HRD is used to check conformance of bitstreams as specified above in this subclause and to check conformance of decoders as specified in subclause C.5.

NOTE 4 – 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 done with real values, so that no rounding errors can propagate. For example, the number of bits in a CPB just prior to or after removal of a decoding unit is not necessarily an integer.

The variable t_c is derived as follows and is called a clock tick:

$$t_c = num\_units\_in\_tick \div time\_scale \quad (C\text{-}1)$$

The variable t_{c_sub} is derived as follows and is called a sub-picture clock tick:

$$t_{c\_sub} = t_c \div (tick\_divisor\_minus2 + 2) \quad (C\text{-}2)$$

[Ed. (GJS): Instead of sending num_units_in_tick and requiring it to be an integer multiple of the tick divisor, it would make more sense to send the tick divisor and a multiplier m use that to compute num_units_in_tick = m times the tick divisor.]

The following is specified for expressing the constraints in this annex:

– Let access unit n be the n-th access unit in decoding order with the first access unit being access unit 0 (i.e. the 0-th access unit).

– Let picture n be the coded picture or the decoded picture of access unit n.

– Let decoding unit m be the m-th decoding unit in decoding order with the first decoding unit being decoding unit 0.

[Ed. (KJS): Numbering relative to beginning of bitstream or access unit? (GJS): I think it's relative to the first decoding unit in the access unit that initializes the HRD.]

C.2 Operation of coded picture buffer (CPB)

C.2.1 General

The specifications in this subclause apply independently to each set of CPB parameters that is present and to both the Type I and Type II conformance points shown in Figure C-1, and the set of CPB parameters is selected as specified in subclause C.1.

C.2.2 Timing of decoding unit arrival

The HRD is initialized at access unit 0 selected as specified in subclause C.1. Prior to HRD initialization, the CPB is empty.

NOTE 1 – After initialization, the HRD is not initialized again by subsequent buffering period SEI messages.

Each access unit is referred to as access unit n, where the number n identifies the particular access unit. The access unit that is associated with the buffering period SEI message that initializes the CPB is referred to as access unit 0. The value of n is incremented by 1 for each subsequent access unit in decoding order.

Each decoding unit is referred to as decoding unit m, where the number m identifies the particular decoding unit. The first decoding unit in decoding order in access unit 0 is referred to as decoding unit 0. The value of m is incremented by 1 for each subsequent decoding unit in decoding order.

When sub_pic_cpb_params_present_flag is equal to 1, the following process in this subclause is invoked first with the variable subPicParamsFlag set equal to 0 and a decoding unit being considered as an access unit during the invocation of the process, for derivation of the initial and final arrival times for access unit n, and then invoked with subPicParamsFlag set equal to 1 and a decoding unit being considered as a subset of an access unit during the invocation of the process, for derivation of the initial and final arrival times for the decoding units in access unit n.

The variables InitCpbRemovalDelay[SchedSelIdx] and InitCpbRemovalDelayOffset[SchedSelIdx] are derived as follows:

– If one or more of the following conditions are true, InitCpbRemovalDelay[SchedSelIdx] and InitCpbRemovalDelayOffset[SchedSelIdx] are set equal to the values of the buffering period SEI message syntax elements initial_alt_cpb_removal_delay[SchedSelIdx] and initial_alt_cpb_removal_offset[SchedSelIdx], respectively, which are selected depending on NalHrdModeFlag as specified in subclause C.1.

  – Access unit 0 is a BLA access unit for which the coded picture has nal_unit_type equal to BLA_W_DLP or BLA_N_LP, and the value of rap_cpb_params_present_flag of the buffering period SEI message is equal to 1.

  – Access unit 0 is a BLA access unit for which the coded picture has nal_unit_type equal to BLA_W_LP or is a CRA access unit, and the value of rap_cpb_params_present_flag of the buffering period SEI message is equal to 1, and one or more of the following conditions are true.

    – UseAltCpbParamsFlag is equal to 1

    – DefaultInitCpbParamsFlag is equal to 0

  – The values of sub_pic_cpb_params_present_flag and subPicParamsFlag are both equal to 1.

    NOTE 2 – The values of sub_pic_cpb_params_present_flag and rap_cpb_params_present_flag cannot be both equal to 1.

– Otherwise, InitCpbRemovalDelay[SchedSelIdx] and InitCpbRemovalDelayOffset[SchedSelIdx] are set equal to the values of the buffering period SEI message syntax elements initial_cpb_removal_delay[SchedSelIdx] and initial_cpb_removal_offset[SchedSelIdx], respectively, which are selected depending on NalHrdModeFlag as specified in subclause C.1.

The time at which the first bit of decoding unit m begins to enter the CPB is referred to as the initial arrival time $t_{ai}(m)$.

The initial arrival time of decoding unit m is derived as follows.

– If the decoding unit is decoding unit 0 (i.e. m = 0), $t_{ai}(0) = 0$,

– Otherwise (the decoding unit is decoding unit m with m > 0), the following applies:

  – If cbr_flag[SchedSelIdx] is equal to 1, the initial arrival time for decoding unit m, is equal to the final arrival time (which is derived below) of decoding unit m − 1, i.e.

$$t_{ai}(m) = t_{af}(m - 1) \tag{C-3}$$

  – Otherwise (cbr_flag[SchedSelIdx] is equal to 0), the initial arrival time for decoding unit m is derived by

$$t_{ai}(m) = \mathrm{Max}(t_{af}(m - 1), t_{ai,earliest}(m)) \tag{C-4}$$

    where $t_{ai,earliest}(m)$ is derived as follows.

    – If decoding unit m is not the first decoding unit of a subsequent buffering period, $t_{ai,earliest}(m)$ is derived as

$$t_{ai,earliest}(m) = t_{r,n}(m) - (\mathrm{InitCpbRemovalDelay}[\mathrm{SchedSelIdx}] + \mathrm{InitCpbRemovalDelayOffset}[\mathrm{SchedSelIdx}]) \div 90000 \tag{C-5}$$

      with $t_{r,n}(m)$ being the nominal removal time of decoding unit m from the CPB as specified in subclause C.2.3.

    – Otherwise (decoding unit m is the first decoding unit of a subsequent buffering period), $t_{ai,earliest}(m)$ is derived as

$$t_{ai,earliest}(m) = t_{r,n}(m) - (\mathrm{InitCpbRemovalDelay}[\mathrm{SchedSelIdx}] \div 90000) \tag{C-6}$$

The final arrival time for decoding unit m is derived by

$$t_{af}(m) = t_{ai}(m) + b(m) \div \mathrm{BitRate}[\mathrm{SchedSelIdx}] \tag{C-7}$$

where b(m) is the size in bits of decoding unit 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 SchedSelIdx, BitRate[SchedSelIdx], and CpbSize[SchedSelIdx] are constrained as follows.

– If the content of the selected hrd_parameters( ) syntax structures for the access unit containing decoding unit m and the previous access unit differ, the HSS selects a value SchedSelIdx1 of SchedSelIdx from among the values of SchedSelIdx provided in the selected hrd_parameters( ) syntax structures for the access unit containing decoding unit m that results in a BitRate[SchedSelIdx1] or CpbSize[SchedSelIdx1] for the access unit containing decoding

unit m. The value of BitRate[SchedSelIdx1] or CpbSize[SchedSelIdx1] may differ from the value of BitRate[SchedSelIdx0] or CpbSize[SchedSelIdx0] for the value SchedSelIdx0 of SchedSelIdx that was in use for the previous access unit.

– Otherwise, the HSS continues to operate with the previous values of SchedSelIdx, BitRate[SchedSelIdx] and CpbSize[SchedSelIdx].

When the HSS selects values of BitRate[SchedSelIdx] or CpbSize[SchedSelIdx] that differ from those of the previous access unit, the following applies.

– The variable BitRate[SchedSelIdx] comes into effect at time t_{ai}(m)

– The variable CpbSize[SchedSelIdx] comes into effect as follows.

  – If the new value of CpbSize[SchedSelIdx] is greater than the old CPB size, it comes into effect at time t_{ai}(m).

  – Otherwise, the new value of CpbSize[SchedSelIdx] comes into effect at the CPB removal time of the last decoding unit of the access unit containing decoding unit m.

NOTE 3 – When SubPicCpbFlag is equal to 0, each decoding unit is an access unit, hence the initial and final CPB arrival times of access unit n are the initial and final CPB arrival times of decoding unit n.

C.2.3 Timing of decoding unit removal and decoding of decoding unit

The variables InitCpbRemovalDelay[SchedSelIdx] and InitCpbRemovalDelayOffset[SchedSelIdx] are derived as follows.

– If one or more of the following conditions are true, InitCpbRemovalDelay[SchedSelIdx] and InitCpbRemovalDelayOffset[SchedSelIdx] are set equal to the values of the buffering period SEI message syntax elements initial_alt_cpb_removal_delay[SchedSelIdx] and initial_alt_cpb_removal_offset[SchedSelIdx], respectively, which are selected depending on NalHrdModeFlag as specified in subclause C.1.

  – Access unit 0 is a BLA access unit for which the coded picture has nal_unit_type equal to BLA_W_DLP or BLA_N_LP, and the value of rap_cpb_params_present_flag of the buffering period SEI message is equal to 1;

  – Access unit 0 is a BLA access unit for which the coded picture has nal_unit_type equal to BLA_W_LP or is a CRA access unit, and the value of rap_cpb_params_present_flag of the buffering period SEI message is equal to 1, and one or more of the following condtions are true;

    – UseAltCpbParamsFlag is equal to 1;

    – DefaultInitCpbParamsFlag is equal to 0.

– Otherwise, InitCpbRemovalDelay[SchedSelIdx] and InitCpbRemovalDelayOffset[SchedSelIdx] are set equal to the values of the buffering period SEI message syntax elements initial_cpb_removal_delay[SchedSelIdx] and initial_cpb_removal_offset[SchedSelIdx], respectively, which are selected depending on NalHrdModeFlag as specified in subclause C.1.

The nominal removal time of the access unit n from the CPB is specified as follows.

– If access unit n is the access unit with n being equal to 0 (the access unit that initializes the HRD), the nominal removal time of the access unit from the CPB is specified by

$$t_{r,n}(0) = \text{InitCpbRemovalDelay}[\text{SchedSelIdx}] \div 90000 \quad \text{(C-8)}$$

– Otherwise, the following applies.

  – When access unit n is the first access unit of a buffering period that does not initialize the HRD, the nominal removal time of the access unit n from the CPB is specified by

$$t_{r,n}(n) = t_{r,n}(n_b) + t_c * (\text{au\_cpb\_removal\_delay\_minus1}(n) + 1) \quad \text{(C-9)}$$

    where $t_{r,n}(n_b)$ is the nominal removal time of the first access unit of the previous buffering period, and au_cpb_removal_delay_minus1(n) is the value of au_cpb_removal_delay_minus1 in the picture timing SEI message, selected as specified in subclause C.1, associated with access unit n.

  – When access unit n is the first access unit of a buffering period, n_b is set equal to n at the nominal removal time $t_{r,n}(n)$ of the access unit n.

  – When access unit n is not the first access unit of a buffering period, $t_{r,n}(n)$ is given by Equation C-9, where $t_{r,n}(n_b)$ is the nominal removal time of the first access unit of the current buffering period.

When sub_pic_cpb_params_present_flag is equal to 1, the following applies.

– The variable CpbRemovalDelay(m) is derived as follows.

  – If sub_pic_cpb_params_in_pic_timing_sei_flag is equal to 0, the variable CpbRemovalDelay(m) is set to the value of du_spt_cpb_removal_delay in the decoding unit information SEI message, selected as specified in subclause C.1, associated with decoding unit m.

  – Otherwise, if du_common_cpb_removal_delay_flag is equal to 0, the variable CpbRemovalDelay(m) is set to the value of du_cpb_removal_delay_minus1[i] + 1 for decoding unit m in the picture timing SEI message, selected as specified in subclause C.1, associated with access unit n, where the value of i is 0 for the first num_nalus_in_du_minus1[0] + 1 consecutive decoding units in the access unit that contains decoding unit m, the value of i is 1 for the subsequent num_nalus_in_du_minus1[1] + 1 decoding units in the same access unit, the value of i is 2 for the subsequent num_nalus_in_du_minus1[2] + 1 decoding units in the same access unit, etc.

  – Otherwise, the variable CpbRemovalDelay(m) is set to the value of du_common_cpb_removal_delay_minus1 + 1 in the picture timing SEI message, selected as specified in subclause C.1, associated with access unit n.

– The nominal removal time of decoding unit m from the CPB is specified as follows, where $t_{r,n}(n)$ is the nominal removal time of access unit n.

  – If decoding unit m is the last decoding unit in access unit n, the nominal removal time of decoding unit m $t_{r,n}(m)$ is set to $t_{r,n}(n)$.

  – Otherwise, (i.e. decoding unit m is not the last decoding unit in access unit n), the nominal removal time of decoding unit m $t_{r,n}(m)$ is derived as follows.

$$\begin{aligned}&\text{if(sub\_pic\_cpb\_params\_in\_pic\_timing\_sei\_flag)}\\&\quad t_{r,n}(m) = t_{r,n}(m + 1) - t_{c\_sub} * \text{CpbRemovalDelay(m)}\\&\text{else}\\&\quad t_{r,n}(m) = t_{r,n}(n) - t_{c\_sub} * \text{CpbRemovalDelay(m)}\end{aligned} \tag{C-10}$$

The removal time of access unit n from the CPB is specified as follows, where $t_{af}(m)$ and $t_{r,n}(m)$ are the final arrival time and nominal removal time of the last decoding unit in access unit n.

$$\begin{aligned}&\text{if(!low\_delay\_hrd\_flag || } t_{r,n}(n) >= t_{af}(n))\\&\quad t_r(n) = t_{r,n}(n)\\&\text{else if(sub\_pic\_cpb\_params\_present\_flag)}\\&\quad t_r(n) = t_{r,n}(n) + \text{Max}((t_{c\_sub} * \text{Ceil}((t_{af}(m) - t_{r,n}(m)) \div t_{c\_sub})),\\&\qquad\qquad (t_c * \text{Ceil}((t_{af}(n) - t_{r,n}(n)) \div t_c)))\\&\text{else}\\&\quad t_r(n) = t_{r,n}(n) + t_c * \text{Ceil}((t_{af}(n) - t_{r,n}(n)) \div t_c)\end{aligned} \tag{C-11}$$

When SubPicCpbFlag is equal to 1, the removal time of decoding unit m from the CPB is specified as follows.

– If low_delay_hrd_flag is equal to 0 or $t_{r,n}(m) >= t_{af}(m)$, the removal time of decoding unit m is specified by

$$t_r(m) = t_{r,n}(m) \tag{C-12}$$

– Otherwise, if decoding unit m is not the last decoding unit of access unit n, the removal time of decoding unit m is specified by

$$t_r(m) = t_{r,n}(m) + t_{c\_sub} * \text{Ceil}((t_{af}(m) - t_{r,n}(m)) \div t_{c\_sub}) \tag{C-13}$$

– Otherwise, if decoding unit m is the last decoding unit of access unit n, the removal time of decoding unit m is specified by

$$t_r(m) = t_{r,n}(n) \tag{C-14}$$

NOTE – When low_delay_hrd_flag is equal to 1 and $t_{r,n}(m) < t_{af}(m)$, the size of decoding unit m, b(m), is so large that it prevents removal at the nominal removal time.

At the CPB removal time of decoding unit m, the decoding unit is instantaneously decoded.

Picture n is considered as decoded when the last decoding unit of the picture is decoded.

C.3 Operation of the decoded picture buffer (DPB)

C.3.1 General

The specifications in this subclause apply independently to each set of DPB parameters selected as specified in subclause C.1.

The decoded picture buffer contains picture storage buffers. Each of the picture storage buffers may contain a decoded picture that is marked as "used for reference" or is held for future output. Prior to initialization, the DPB is empty (the DPB fullness is set to zero). The following steps of the subclauses of this subclause happen in the sequence as listed below.

C.3.2 Removal of pictures from the DPB

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 decoding unit of access unit n (containing the current picture) and proceeds as follows.

The decoding process for reference picture set as specified in subclause 8.3.2 is invoked.

When the current picture is an IDR or a BLA picture, the following applies:

1. When the IDR or BLA picture is not the first picture decoded and the value of pic_width_in_luma_samples or pic_height_in_luma_samples or sps_max_dec_pic_buffering[HighestTid] derived from the active sequence parameter set is different from the value of pic_width_in_luma_samples or pic_height_in_luma_samples or sps_max_dec_pic_buffering[HighestTid] derived from the sequence parameter set that was active for the preceding picture, respectively, no_output_of_prior_pics_flag is inferred to be equal to 1 by the HRD, regardless of the actual value of no_output_of_prior_pics_flag.

NOTE – Decoder implementations should try to handle picture or DPB size changes more gracefully than the HRD in regard to changes in pic_width_in_luma_samples, pic_height_in_luma_samples, or sps_max_dec_pic_buffering[HighestTid].

2. When no_output_of_prior_pics_flag is equal to 1 or is inferred to be equal to 1, all picture storage buffers in the DPB are emptied without output of the pictures they contain, and DPB fullness is set 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 PicOutputFlag equal to 0 or its DPB output time is less than or equal to the CPB removal time of the first decoding unit (denoted as decoding unit m) of the current picture n; i.e. $t_{o,dpb}(k) <= t_r(m)$

When a picture is removed from the DPB, the DPB fullness is decremented by one.

C.3.3 Picture output

The following happens instantaneously at the CPB removal time of access unit n, $t_r(n)$.

When picture n has PicOutputFlag equal to 1, its DPB output time $t_{o,dpb}(n)$ is derived by

$$t_{o,dpb}(n) = t_r(n) + t_c * pic\_dpb\_output\_delay(n) \quad (C\text{-}15)$$

where pic_dpb_output_delay(n) is the value of pic_dpb_output_delay specified in the picture timing SEI message associated with access unit n.

The output of the current picture is specified as follows.

– If PicOutputFlag is equal to 1 and $t_{o,dpb}(n) = t_r(n)$, the current picture is output.

– Otherwise, if PicOutputFlag is equal to 0, the current picture is not output, but will be stored in the DPB as specified in subclause C.3.4.

– Otherwise (PicOutputFlag is equal to 1 and $t_{o,dpb}(n) > t_r(n)$), the current picture is output later and will be stored in the DPB (as specified in subclause C.3.4) and is output at time $t_{o,dpb}(n)$ unless indicated not to be output by the decoding or inference of no_output_of_prior_pics_flag equal to 1 at a time that precedes $t_{o,dpb}(n)$.

When output, the picture shall be cropped, using the conformance cropping window specified in the active sequence parameter set.

When picture n is a picture that is output and is not the last picture of the bitstream that is output, the value of $\Delta t_{o,dpb}(n)$ is defined as:

$$\Delta t_{o,dpb}(n) = t_{o,dpb}(n_n) - t_{o,dpb}(n) \quad \text{(C-16)}$$

where n_n indicates the picture that follows picture n in output order and has PicOutputFlag equal to 1.

C.3.4 Current decoded picture marking and storage

The following happens instantaneously at the CPB removal time of access unit n, $t_r(n)$.

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".

C.4 Bitstream conformance

A bitstream of coded data conforming to this Specification shall fulfil all requirements specified in this subclause.

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 a RAP picture, i.e. an IDR picture or a CRA picture or a BLA picture.

The bitstream is tested by the HRD for conformance as specified below:

For each current picture that is decoded, let the variables maxPicOrderCnt and minPicOrderCnt be set equal to the maximum and the minimum, respectively, of the PicOrderCntVal values of the following pictures:

– The current picture.

– The previous picture in decoding order that has TemporalId equal to 0.

– The short-term reference pictures in the reference picture set of the current picture.

– All pictures n that have PicOutputFlag equal to 1 and $t_r(n) < t_r(\text{currPic})$ and $t_{o,dpb}(n) >= t_r(\text{currPic})$, where currPic is the current picture.

All of the following conditions shall be fulfilled for each of the bitstream conformance tests.

1. For each access unit n, with n>0, associated with a buffering period SEI message, with $\Delta t_{g,90}(n)$ specified by

$$\Delta t_{g,90}(n) = 90000 * (t_{r,n}(n) - t_{af}(n - 1)) \quad \text{(C-17)}$$

the value of InitCpbRemovalDelay[SchedSelIdx] shall be constrained as follows.

– If cbr_flag[SchedSelIdx] is equal to 0,

$$\text{InitCpbRemovalDelay[SchedSelIdx]} <= \text{Ceil}(\Delta t_{g,90}(n)) \quad \text{(C-18)}$$

– Otherwise (cbr_flag[SchedSelIdx] is equal to 1),

$$\text{Floor}(\Delta t_{g,90}(n)) <= \text{InitCpbRemovalDelay[SchedSelIdx]} <= \text{Ceil}(\Delta t_{g,90}(n)) \quad \text{(C-19)}$$

NOTE 4 – The exact number of bits in the CPB at the removal time of each picture may depend on which buffering period SEI message is selected to initialize the HRD. Encoders must take this into account to ensure that all specified constraints must be obeyed regardless of which buffering period SEI message is selected to initialize the HRD, as the HRD may be initialized at any one of the buffering period SEI messages.

2. A CPB overflow is specified as the condition in which the total number of bits in the CPB is larger than the CPB size. The CPB shall never overflow.

3. A CPB underflow is specified as the condition in which the nominal CPB removal time of decoding unit m $t_{r,n}(m)$ is less than the final CPB arrival time of decoding unit m $t_{af}(m)$ for at least one value of m. When low_delay_hrd_flag is equal to 0, the CPB shall never underflow.

4. When low_delay_hrd_flag is equal to 1, a CPB underflow may occur at decoding unit m. In this case, the final CPB arrival time of access unit n containing decoding unit m $t_{af}(n)$ shall be greater than the nominal CPB removal time of access unit n containing decoding unit m $t_{r,n}(n)$. [Ed. (GJS): Does "shall" make sense here?]

5. The nominal removal times of pictures from the CPB (starting from the second picture in decoding order), shall satisfy the constraints on $t_{r,n}(n)$ and $t_r(n)$ expressed in subclauses A.4.1 through A.4.2.

6. For each current picture that is decoded, after invocation of the process for removal of pictures from the DPB as specified in subclause 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 PicOutputFlag equal to 1 and $t_r(n) < t_r(currPic)$, where currPic is the current picture, shall be less than or equal to Max(0, sps_max_dec_pic_buffering[HighestTid] − 1).

7. All reference pictures shall be present in the DPB when needed for prediction. Each picture that has PicOutputFlag 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 subclause C.3.

8. For each current picture that is decoded, the he value of maxPicOrderCnt − minPicOrderCnt shall be less than MaxPicOrderCntLsb / 2.

9. The value of $\Delta_{to,dpb}(n)$ as given by Equation C-16, which is the difference between the output time of a picture and that of the first picture following it in output order and having PicOutputFlag equal to 1, shall satisfy the constraint expressed in subclause A.4.1 for the profile, tier and level specified in the bitstream using the decoding process specified in clauses 2–9.

10. For each current picture, when sub_pic_cpb_params_in_pic_timing_sei_flag is equal to 1, the following relationship shall apply, where $t_{r,n}(n)$ is the nominal CPB removal time of the current access unit and $t_{r,n}(m)$ is the nominal CPB removal time of the first decoding unit in the current access unit in decoding order.

$$t_{r,n}(n) - t_{r,n}(m) == t_{c\_sub} * \sum_{i=0}^{num\_decoding\_units\_minus1} du\_cpb\_removal\_delay\_minus1[i]+1 \quad \text{(C-20)}$$

C.5 Decoder conformance

C.5.1 General

A decoder conforming to this Specification shall fulfil all requirements specified in this subclause.

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 subclause C.4, in the manner specified in Annex A, provided that all video parameter sets, sequence parameter sets and picture parameter sets referred to in the VCL NAL units, and appropriate buffering period and picture timing SEI messages are conveyed to the decoder, in a timely manner, either in the bitstream (by non-VCL NAL units), or by external means not specified by this 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 by subclause C.4 are delivered by a hypothetical stream scheduler (HSS) both to the HRD and to the decoder under test (DUT). All pictures output by the HRD shall also be output by the DUT and, for each picture output by the HRD, the values of all samples that are output by the DUT for the corresponding picture shall be equal to the values of the samples output by the HRD. [Ed. (YK): So a conforming decoder is allowed to output some arbitrary junk pictures in addition to those pictures output by the HRD. Should it actually be further restricted that a conforming decoder may output some additional pictures, but only those included in the bitstream, and decoded and output using the same decoding process and output process for the pictures that are also output by the HRD? (GJS) Yes, that it what is intended, and the text should be modified to express that – and it should only be allowed to output those pictures that have PicOutputFlag equal to 1 and only those corresponding to the applicable (parsed or inferred) value of no_output_of_prior_pics_flag.]

For output timing decoder conformance, the HSS operates as described above, with delivery schedules selected only from the subset of values of SchedSelIdx for which the bit rate and CPB size are restricted as specified in Annex A 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 buffering period SEI messages are present with cpb_cnt_minus1[HighestTid] greater than 0, the decoder shall be capable of decoding the bitstream as delivered from the HSS operating using an "interpolated" delivery schedule specified as having peak bit rate r, CPB size c(r), and initial CPB removal delay (f(r) ÷ r) as follows:

$$\alpha = (r - BitRate[SchedSelIdx - 1]) \div (BitRate[SchedSelIdx] - BitRate[SchedSelIdx - 1]), \quad \text{(C-21)}$$

$$c(r) = \alpha * CpbSize[SchedSelIdx] + (1 - \alpha) * CpbSize[SchedSelIdx-1], \quad \text{(C-22)}$$

$$f(r) = \alpha * InitCpbRemovalDelay[SchedSelIdx] * BitRate[SchedSelIdx] + (1 - \alpha) * InitCpbRemovalDelay[SchedSelIdx - 1] * BitRate[SchedSelIdx - 1] \quad (C\text{-}23)$$

for any SchedSelIdx > 0 and r such that BitRate[SchedSelIdx − 1] <= r <= BitRate[SchedSelIdx] such that r and c(r) are within the limits as specified in Annex A for the maximum bit rate and buffer size for the specified profile, tier and level.

NOTE 1 – InitCpbRemovalDelay[SchedSelIdx] can be different from one buffering period to another and have to be re-calculated.

For output timing decoder conformance, an HRD as described above is used and the timing (relative to the delivery time of the first bit) of picture output is the same for both 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 coded picture buffer of the DUT could be as small as the size of the largest decoding unit.

– A modified HRD as described below is used, and the HSS delivers the bitstream to the HRD by one of the schedules specified in the bitstream 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[SchedSelIdx] as specified in subclause E.2.3, where SchedSelIdx and the HRD parameters are selected as specified in subclause C.1. The DPB size is given by sps_max_dec_pic_buffering[HighestTid]. 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 subclauses C.5.2 through C.5.4.

C.5.2 Operation of the output order DPB

The decoded picture buffer contains picture storage buffers. Each of the picture storage buffers contains a decoded picture that is marked as "used for reference" or is held for future output. At HRD initialization, the DPB is empty. The following steps happen in the order as listed below.

C.5.3 Output and removal of pictures from the DPB

The output and 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 when the first decoding unit of the access unit containing the current picture is removed from the CPB and proceeds as follows.

The decoding process for reference picture set as specified in subclause 8.3.2 is invoked.

– If the current picture is an IDR or a BLA picture, the following applies.

1. When the IDR or BLA picture is not the first picture decoded and the value of pic_width_in_luma_samples or pic_height_in_luma_samples or sps_max_dec_pic_buffering[HighestTid] for any possible value of i derived from the active sequence parameter set is different from the value of pic_width_in_luma_samples or pic_height_in_luma_samples or sps_max_dec_pic_buffering[HighestTid] derived from the sequence parameter set that was active for the preceding picture, respectively, no_output_of_prior_pics_flag is inferred to be equal to 1 by the HRD, regardless of the actual value of no_output_of_prior_pics_flag.

   NOTE – Decoder implementations should try to handle picture or DPB size changes more gracefully than the HRD in regard to changes in pic_width_in_luma_samples, pic_height_in_luma_samples or sps_max_dec_pic_buffering[HighestTid].

2. When no_output_of_prior_pics_flag is equal to 1 or is inferred to be equal to 1, all picture storage buffers in the DPB are emptied without output of the pictures they contain.

3. When no_output_of_prior_pics_flag is not equal to 1 and is not inferred to be equal to 1, 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 subclause C.5.3.1.

– Otherwise (the current picture is not an IDR or a BLA picture), picture storage buffers containing a picture which are marked as "not needed for output" and "unused for reference" are emptied (without output). When one or more of the following conditions are true, the "bumping" process specified in subclause C.5.3.1 is invoked repeatedly until there is an empty picture storage buffer to store the current decoded picture.

1. The number of pictures in the DPB that are marked as "needed for output" is greater than sps_max_num_reorder_pics[HighestTid],

2. The number of pictures in the DPB is equal to sps_max_dec_pic_buffering[HighestTid].

C.5.3.1 "Bumping" process

The "bumping" process is invoked in the following cases.

– The current picture is an IDR or a BLA picture and no_output_of_prior_pics_flag is not equal to 1 and is not inferred to be equal to 1, as specified in subclause C.5.3.

– The current picture is neither an IDR picture nor a BLA picture, and the number of pictures in the DPB that are marked as "needed for output" is greater than sps_max_num_reorder_pics[HighestTid], as specified in subclause C.5.3.

– The current picture is neither an IDR picture nor a BLA picture, and the number of pictures in the DPB is equal to sps_max_dec_pic_buffering[HighestTid], as specified in subclause C.5.3.

The "bumping" process consists of the following ordered steps:

1. The picture that is first for output is selected as the one having the smallest value of PicOrderCntVal of all pictures in the DPB marked as "needed for output".
2. The picture is cropped, using the conformance cropping window specified in the active sequence parameter set for the picture, the cropped picture is output, and the picture is marked as "not needed for output".
3. If the picture storage buffer that included the picture that was cropped and output contains a picture marked as "unused for reference", the picture storage buffer is emptied.

C.5.4 Picture decoding, marking and storage

The following happens instantaneously when the last decoding unit of access unit n containing the current picture is removed from the CPB.

The current picture is considered as decoded after the last decoding unit of the picture is decoded. The current decoded picture is stored in an empty picture storage buffer in the DPB, and the following applies.

– If the current decoded picture has PicOutputFlag equal to 1, it is marked as "needed for output".

– Otherwise (the current decoded picture has PicOutputFlag equal to 0), it is marked as "not needed for output".

The current decoded picture is marked as "used for short-term reference".

Annex D

Supplemental enhancement information

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies syntax and semantics for SEI message payloads.

SEI messages assist in processes related to decoding, display or other purposes. However, SEI messages are not required for constructing the luma or chroma samples by the decoding process. Conforming decoders are not required to process this information for output order conformance to this Specification (see Annex C for the specification of conformance). Some SEI message information is required to check bitstream conformance and for output timing decoder conformance.

In subclause C.5.2, specification for presence of SEI messages are also satisfied when those messages (or some subset of them) are conveyed to decoders (or to the HRD) by other means not specified by this Specification. When present in the bitstream, SEI messages shall obey the syntax and semantics specified in subclause 7.3.7 and this annex. When the content of an SEI message is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the SEI message is not required to use the same syntax specified in this annex. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

D.1 SEI payload syntax

D.1.1 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 = = 2)	
pan_scan_rect(payloadSize)	
else if(payloadType = = 3)	
filler_payload(payloadSize)	
else if(payloadType = = 4)	
user_data_registered_itu_t_t35(payloadSize)	
else if(payloadType = = 5)	
user_data_unregistered(payloadSize)	
else if(payloadType = = 6)	
recovery_point(payloadSize)	
else if(payloadType = = 9)	
scene_info(payloadSize)	
else if(payloadType = = 15)	
full_frame_snapshot(payloadSize)	
else if(payloadType = = 16)	
progressive_refinement_segment_start(payloadSize)	
else if(payloadType = = 17)	
progressive_refinement_segment_end(payloadSize)	
else if(payloadType = = 19)	
film_grain_characteristics(payloadSize)	
else if(payloadType = = 22)	
post_filter_hint(payloadSize)	
else if(payloadType = = 23)	
tone_mapping_info(payloadSize)	
else if(payloadType = = 45)	
frame_packing_arrangement(payloadSize)	
else if(payloadType = = 47) [Ed. (GJS): Check numbering w.r.t. AVC.]	
display_orientation(payloadSize)	
else if(payloadType = = 128)	
sop_description(payloadSize)	
else if(payloadType = = 129)	
active_parameter_sets(payloadSize)	
else if(payloadType = = 130)	
decoding_unit_info(payloadSize)	
else if(payloadType = = 131)	
tl0_index(payloadSize)	
else if(payloadType = = 133)	
scalable_nesting(payloadSize)	
else if(payloadType = = 134)	

region_refresh_info(payloadSize)	
else	
reserved_sei_message(payloadSize)	
else /* nal_unit_type = = SUFFIX_SEI_NUT */	
if(payloadType = = 132)	
decoded_picture_hash(payloadSize)	
else	
reserved_sei_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)
}	
}	

D.1.2 Buffering period SEI message syntax

buffering_period(payloadSize) {	**Descriptor**
bp_seq_parameter_set_id	ue(v)
if(!sub_pic_cpb_params_present_flag)	
rap_cpb_params_present_flag	u(1)
if(NalHrdBpPresentFlag) {	
for(i = 0; i <= CpbCnt; i++) {	
initial_cpb_removal_delay[i **]**	u(v)
initial_cpb_removal_offset[i **]**	u(v)
if(sub_pic_cpb_params_present_flag || rap_cpb_params_present_flag) {	
initial_alt_cpb_removal_delay[i **]**	u(v)
initial_alt_cpb_removal_offset[i **]**	u(v)
}	
}	
}	
if(VclHrdBpPresentFlag) {	
for(i = 0; i <= CpbCnt; i++) {	
initial_cpb_removal_delay[i **]**	u(v)
initial_cpb_removal_offset[i **]**	u(v)
if(sub_pic_cpb_params_present_flag || rap_cpb_params_present_flag) {	
initial_alt_cpb_removal_delay[i **]**	u(v)
initial_alt_cpb_removal_offset[i **]**	u(v)
}	
}	
}	
}	

D.1.3 Picture timing SEI message syntax

pic_timing(payloadSize) {	**Descriptor**
if(frame_field_info_present_flag) {	
pic_struct	u(4)
progressive_source_idc	u(2)
duplicate_flag	u(1)
}	
au_cpb_removal_delay_minus1	u(v)
pic_dpb_output_delay	u(v)
if(sub_pic_cpb_params_present_flag && sub_pic_cpb_params_in_pic_timing_sei_flag) {	
num_decoding_units_minus1	ue(v)
du_common_cpb_removal_delay_flag	u(1)
if(du_common_cpb_removal_delay_flag)	
du_common_cpb_removal_delay_minus1	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)	
du_cpb_removal_delay_minus1[i]	u(v)
}	
}	
}	

D.1.4 Pan-scan rectangle SEI message syntax

The syntax table is specified in subclause D.1.3 of Rec. ITU-T H.264 | ISO/IEC 14496-10.

D.1.5 Filler payload SEI message syntax

The syntax table is specified in subclause D.1.4 of Rec. ITU-T H.264 | ISO/IEC 14496-10.

D.1.6 User data registered by Rec. ITU-T T.35 SEI message syntax

The syntax table is specified in subclause D.1.5 of Rec. ITU-T H.264 | ISO/IEC 14496-10.

D.1.7 User data unregistered SEI message syntax

The syntax table is specified in subclause D.1.6 of Rec. ITU-T H.264 | ISO/IEC 14496-10.

D.1.8 Recovery point SEI message syntax

recovery_point(payloadSize) {	**Descriptor**
recovery_poc_cnt	se(v)
exact_match_flag	u(1)
broken_link_flag	u(1)
}	

[Ed. (YK): The broken_link_flag is not useful anymore after the introduction of BLA pictures, and therefore should be removed from the recovery point SEI message. (GJS): It seems potentially redundant, but perhaps not harmful to keep.]

D.1.9 Scene information SEI message syntax

The syntax table is specified in subclause D.1.10 of Rec. ITU-T H.264 | ISO/IEC 14496-10.

D.1.10 Full-frame snapshot SEI message syntax

The syntax table is specified in subclause D.1.16 of Rec. ITU-T H.264 | ISO/IEC 14496-10.

D.1.11 Progressive refinement segment start SEI message syntax

The syntax table is specified in subclause D.1.17 of Rec. ITU-T H.264 | ISO/IEC 14496-10.

D.1.12 Progressive refinement segment end SEI message syntax

The syntax table is specified in subclause D.1.18 of Rec. ITU-T H.264 | ISO/IEC 14496-10.

D.1.13 Film grain characteristics SEI message syntax

The syntax table is specified in subclause D.1.20 of Rec. ITU-T H.264 | ISO/IEC 14496-10.

D.1.14 Post-filter hint SEI message syntax

The syntax table is specified in subclause D.1.23 of Rec. ITU-T H.264 | ISO/IEC 14496-10.

D.1.15 Tone mapping information SEI message syntax

tone_mapping_info(payloadSize) {	**Descriptor**
tone_map_id	ue(v)
tone_map_cancel_flag	u(1)
if(!tone_map_cancel_flag) {	
tone_map_repetition_period	ue(v)
coded_data_bit_depth	u(8)
target_bit_depth	u(8)
model_id	ue(v)
if(model_id = = 0) {	
min_value	u(32)
max_value	u(32)
} else if(model_id = = 1) {	
sigmoid_midpoint	u(32)
sigmoid_width	u(32)
} else if(model_id = = 2)	
for(i = 0; i < (1 << target_bit_depth); i++)	
start_of_coded_interval[i]	u(v)
else if(model_id = = 3) {	
num_pivots	u(16)
for(i = 0; i < num_pivots; i++) {	
coded_pivot_value[i]	u(v)
target_pivot_value[i]	u(v)
}	
} else if(model_id = = 4) {	
camera_iso_sensitivity_idc	u(8)
if(camera_iso_sensitivity_idc = = Extended_ISO)	
camera_iso_sensitivity	u(v)
exposure_index_idc	u(8)
if(exposure_index_idc = = Extended_ISO)	
exposure_index_rating	u(v)
sign_image_exposure_value	u(1)
image_exposure_value0	u(v)
image_exposure_value1	u(v)
ref_screen_lw	u(v)
max_image_white_level	u(v)
black_level_code_value	u(8)
white_level_code_value	u(8)
max_white_level_code_value	u(8)
}	
}	
}	

D.1.16 Frame packing arrangement SEI message syntax

frame_packing_arrangement(payloadSize) {	**Descriptor**
frame_packing_arrangement_id	ue(v)
frame_packing_arrangement_cancel_flag	u(1)
if(!frame_packing_arrangement_cancel_flag) {	
frame_packing_arrangement_type	u(7)
quincunx_sampling_flag	u(1)
content_interpretation_type	u(6)
spatial_flipping_flag	u(1)
frame0_flipped_flag	u(1)
field_views_flag	u(1)
current_frame_is_frame0_flag	u(1)
frame0_self_contained_flag	u(1)
frame1_self_contained_flag	u(1)
if (!quincunx_sampling_flag && frame_packing_arrangement_type != 5) {	
frame0_grid_position_x	u(4)
frame0_grid_position_y	u(4)
frame1_grid_position_x	u(4)
frame1_grid_position_y	u(4)
}	
frame_packing_arrangement_reserved_byte	u(8)
frame_packing_arrangement_repetition_period	ue(v)
}	
upsampled_aspect_ratio_flag	u(1)
}	

D.1.17 Display orientation SEI message syntax

display_orientation(payloadSize) {	**Descriptor**
display_orientation_cancel_flag	u(1)
if(!display_orientation_cancel_flag) {	
hor_flip	u(1)
ver_flip	u(1)
anticlockwise_rotation	u(16)
display_orientation_repetition_period	ue(v)
display_orientation_extension_flag	u(1)
}	
}	

D.1.18 SOP description SEI message syntax

sop_description(payloadSize) {	**Descriptor**
sop_seq_parameter_set_id	ue(v)
num_pics_in_sop_minus1	ue(v)
for(i = 0; i <= num_pics_in_sop_minus1; i++) {	
sop_desc_nal_ref_flag[i]	u(1)
sop_desc_temporal_id[i]	u(3)
st_rps_idx[i]	ue(v)
if(i > 0)	
poc_delta[i]	se(v)
}	
}	

D.1.19 Decoded picture hash SEI message syntax

decoded_picture_hash(payloadSize) {	**Descriptor**
hash_type	u(8)
for(cIdx = 0; cIdx < (chroma_format_idc = = 0 ? 1 : 3); cIdx++)	
if(hash_type = = 0)	
for(i = 0; i < 16; i++)	
picture_md5[cIdx][i]	b(8)
else if(hash_type = = 1)	
picture_crc[cIdx]	u(16)
else if(hash_type = = 2)	
picture_checksum[cIdx]	u(32)
}	

D.1.20 Active parameter sets SEI message syntax

active_parameter_sets(payloadSize) {	**Descriptor**
active_vps_id	u(4)
num_sps_ids_minus1	ue(v)
for(i = 0; i <= num_sps_ids_minus1; i++)	
active_seq_parameter_set_id[i]	ue(v)
}	

D.1.21 Decoding unit information SEI message syntax

decoding_unit_info(payloadSize) {	**Descriptor**
decoding_unit_idx	ue(v)
if(!sub_pic_cpb_params_in_pic_timing_sei_flag)	
du_spt_cpb_removal_delay	u(v)
}	

D.1.22 Temporal level zero index SEI message syntax

tl0_index(payloadSize) {	**Descriptor**
tl0_idx	u(8)
rap_idx	u(8)
}	

D.1.23 Scalable nesting SEI message syntax

scalable_nesting(payloadSize) {	**Descriptor**
bitstream_subset_flag	u(1)
nesting_op_flag	u(1)
if(nesting_op_flag) {	
default_op_flag	u(1)
nesting_num_ops_minus1	ue(v)
for(i = default_op_flag; i <= nesting_num_ops_minus1; i++) {	
nesting_max_temporal_id_plus1[i]	u(3)
nesting_op_idx[i]	ue(v)
}	
} else {	
all_layers_flag	u(1)
if(!all_layers_flag) {	
nesting_no_op_max_temporal_id_plus1	u(3)
nesting_num_layers_minus1	ue(v)
for(i = 0; i <= nesting_num_layers_minus1; i++)	
nesting_layer_id[i]	u(6)
}	
}	
while(!byte_aligned( ))	
nesting_zero_bit /* equal to 0 */	u(1)
do	
sei_message( )	
while(more_rbsp_data( ))	
}	

D.1.24 Region refresh information SEI message syntax

region_refresh_info(payloadSize) {	**Descriptor**
refreshed_region_flag	u(1)
}	

D.1.25 Reserved SEI message syntax

reserved_sei_message(payloadSize) {	**Descriptor**
for(i = 0; i < payloadSize; i++)	
reserved_sei_message_payload_byte	b(8)
}	

D.2 SEI payload semantics

D.2.1 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.

The semantics and persistence scope for each SEI message are specified in the semantics specification for each particular SEI message.

NOTE – Persistence information for prefix and suffix SEI messages is informatively summarized in Tables D-1 and D-2.

Table D-1 – Persistence scope of prefix SEI messages (informative)

SEI message	Persistence scope
buffering period	The remainder of the bitstream
Picture timing	The access unit containing the SEI message
Pan-scan rectangle	Specified by the syntax of the SEI message
Filler payload	The access unit containing the SEI message
User data registered	Unspecified
User data unregistered	Unspecified
Recovery point	Specified by the syntax of the SEI message
Scene information	The access unit containing the SEI message and up to but not including the next access unit, in decoding order, that contains a scene information SEI message [Ed. (GJS): Seems incorrect. That's a big problem for a splicer if it reaches beyond the end of the coded video sequence.]
Full-frame snapshot	The access unit containing the SEI message
Progressive refinement segment start	Specified by the syntax of the SEI message
Progressive refinement segment end	The access unit containing the SEI message
Film grain characteristics	Specified by the syntax of the SEI message
Post-filter hint	The access unit containing the SEI message [Ed. (GJS): Current semantics of this message are vague about scope.]
Tone mapping information	Specified by the syntax of the SEI message
Frame packing arrangement	Specified by the syntax of the SEI message
Display orientation	Specified by the syntax of the SEI message
Structure of pictures	The SOP [Ed. (GJS): Undefined term] containing the access unit that contains the SEI message
Field indication	The access unit containing the SEI message [Ed. (GJS): Check this.]
Active parameter sets	The coded video sequence containing the SEI message
Decoding unit information	The decoding unit containing the SEI message
Temporal level zero index	The access unit containing the SEI message
Scalable nesting	Depending on the nested SEI messages. Each nested SEI message has the same persistence scope as if the SEI message was not nested
Region refresh information	The set of VCL NAL units within the access unit starting from the VCL NAL unit following the SEI message up to but not including the VCL NAL unit following the next SEI NAL unit containing a region refresh information SEI message (if any)

Table D-2 – Persistence scope of suffix SEI messages (informative)

SEI message	Persistence scope
Decoded picture hash	The access unit containing the SEI message

D.2.2 Buffering period SEI message semantics

A buffering period 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 access unit in decoding order.

The following applies for the buffering period SEI message syntax and semantics:

– The syntax elements initial_cpb_removal_delay_length_minus1 and sub_pic_cpb_params_present_flag, and the variables NalHrdBpPresentFlag and VclHrdBpPresentFlag are found in or derived from syntax elements found in the hrd_parameters( ) syntax structure that is applicable to at least one of the operation points to which the buffering period SEI message applies.

– The variables CpbSize[i], BitRate[i] and CpbCnt are derived from syntax elements found in the sub_layer_hrd_parameters( ) syntax structure that is applicable to at least one of the operation points to which the buffering period SEI message applies.

– Any two operation points that the buffering period SEI message applies to having different OpTid values tIdA and tIdB indicate that the values of cpb_cnt_minus1[tIdA] and cpb_cnt_minus1[tIdB] coded in the hrd_parameters( ) syntax structure(s) applicable to the two operation points are identical.

– Any two operation points that the buffering period SEI message applies to having different OpLayerIdSet values layerIdSetA and layerIdSetB indicate that the values of nal_hrd_parameters_present_flag and vcl_hrd_parameters_present_flag, respectively, for the two hrd_parameters( ) syntax structures applicable to the two operation points are identical.

– 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 buffering period SEI message applies.

If NalHrdBpPresentFlag or VclHrdBpPresentFlag are equal to 1, a buffering period SEI message applicable to the specified operation points may be associated with any access unit in the coded video sequence, and a buffering period SEI message applicable to the specified operation points shall be associated with each RAP access unit, and with each access unit associated with a recovery point SEI message. Otherwise (NalHrdBpPresentFlag and VclHrdBpPresentFlag are both equal to 0), no access unit in the coded video sequence shall be associated with a buffering period SEI message applicable to the specified operation points.

NOTE 1 – For some applications, frequent presence of buffering period SEI messages may be desirable.

When an SEI NAL unit that contains a buffering period SEI message and has nuh_reserved_zero_6bits equal to 0 is present, the SEI NAL unit shall precede, in decoding order, the first VCL NAL unit in the access unit.

[Ed. (YK): This restriction should only apply to SEI NAL units with nuh_reserved_zero_6bits equal to 0. Since nuh_reserved_zero_6bits is required to be equal to 0 in this version of this Specification, now we don't have to mention nuh_reserved_zero_6bits equal to 0, but it may be safer if we do say it now, though it is not needed. Similar for other SEI messages below.]

[Ed. (KJS): Consider parsing dependency of this message on SPS/VUI.]

bp_seq_parameter_set_id specifies the sps_seq_parameter_set_id for the sequence parameter set that is active for the coded picture associated with the buffering period SEI message. The value of bp_seq_parameter_set_id shall be equal to the value of pps_seq_parameter_set_id in the picture parameter set referenced by the slice_pic_parameter_set_id of the slice segment headers of the coded picture associated with the buffering period SEI message. The value of bp_seq_parameter_set_id shall be in the range of 0 to 15, inclusive.

rap_cpb_params_present_flag equal to 1 specifies the presence of the initial_alt_cpb_removal_delay[i] and initial_alt_cpb_removal_offset[i] syntax elements. When not present, the value of rap_cpb_params_present_flag is inferred to be equal to 0. When the associated picture is neither a CRA picture nor a BLA picture, the value of rap_cpb_params_present_flag shall be equal to 0.

initial_cpb_removal_delay[i] and **initial_alt_cpb_removal_delay**[i] specify the default and the alternative initial CPB removal delays, respectively, for the i-th CPB. The syntax elements have a length in bits given by initial_cpb_removal_delay_length_minus1 + 1, and are in units of a 90 kHz clock. The values of the syntax elements shall not be equal to 0 and shall be less than or equal to 90000 * (CpbSize[i] ÷ BitRate[i]), the time-equivalent of the CPB size in 90 kHz clock units.

initial_cpb_removal_offset[i] and **initial_alt_cpb_removal_offset**[i] specify the default and the alternative initial CPB removal offsets, respectively, for the i-th CPB to specify the initial delivery time of coded data units to the CPB. The syntax elements have a length in bits given by initial_cpb_removal_delay_length_minus1 + 1 and are in units of a 90 kHz clock. These syntax elements are not used by decoders and may be needed only for the delivery scheduler (HSS) specified in Annex C.

NOTE 2 – Encoders are recommended not to include the alternative initial CPB removal delay and delay offset parameters in buffering period SEI messages associated with CRA or BLA pictures that have associated RASL and RADL pictures that are interleaved in decoding order.

Over the entire coded video sequence, the sum of initial_cpb_removal_delay[i] and initial_cpb_removal_offset[i] shall be constant for each value of i, and the sum of initial_alt_cpb_removal_delay[i] and initial_alt_cpb_removal_offset[i] shall be constant for each value of i.

D.2.3 Picture timing SEI message semantics

The picture timing SEI message provides CPB removal delay and DPB output delay information for the access unit associated with the SEI message.

The following applies for the picture timing SEI message syntax and semantics:

– The syntax elements and variable sub_pic_cpb_params_present_flag, sub_pic_cpb_params_in_pic_timing_sei_flag, au_cpb_removal_delay_length_minus1, dpb_output_delay_length_minus1, du_cpb_removal_delay_length_minus1, and CpbDpbDelaysPresentFlag are found in or derived from syntax elements found in the hrd_parameters( ) syntax structure that is applicable to at least one of the operation points to which the picture timing 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 picture timing SEI message applies.

NOTE 1 – The syntax of the picture timing SEI message is dependent on the content of the hrd_parameters( ) syntax structures applicable to the operation points to which the picture timing SEI message applies. These hrd_parameters( ) syntax structures are in the video parameter set and/or the sequence parameter set that are active for the coded picture associated with the picture timing SEI message. When the picture timing SEI message is associated with a CRA access unit that is the first access unit in the bitstream, an IDR access unit, or a BLA access unit, unless it is preceded by a buffering period SEI message within the same access unit, the activation of the video parameter set and the sequence parameter set (and, for IDR or BLA pictures that are not the first picture in the bitstream, the determination that the coded picture is an IDR picture or a BLA picture) does not occur until the decoding of the first coded slice segment NAL unit of the coded picture. Since the coded slice segment NAL unit of the coded picture follows the picture timing SEI message in NAL unit order, there may be cases in which it is necessary for a decoder to store the RBSP containing the picture timing SEI message until determining the active video parameter set and the active sequence parameter set for the coded picture, and then perform the parsing of the picture timing SEI message.

The presence of picture timing SEI message in the bitstream is specified as follows.

– If frame_field_info_present_flag is equal to 1 or CpbDpbDelaysPresentFlag is equal to 1, one picture timing SEI message applicable to the specified operation points shall be associated with every access unit of the coded video sequence.

– Otherwise, no access unit of the coded video sequence shall be associated with picture timing SEI messages applicable to the specified operation points.

When an SEI NAL unit that contains a picture timing SEI message and has nuh_reserved_zero_6bits equal to 0 is present, the SEI NAL unit shall precede, in decoding order, the first VCL NAL unit in the access unit.

pic_struct indicates whether a picture should be displayed as a frame or as one or more fields and, for the display of frames when fixed_pic_rate_within_cvs_flag is equal to 1, may indicate a frame doubling or tripling repetition period for displays that use a fixed frame refresh interval equal to $\Delta t_{e,dpb}(n)$ as given by Equation E-51. The interpretation of pic_struct is specified by Table D-3.

When present, it is a requirement of bitstream conformance that the value of pic_struct shall be constrained such that exactly one of the following conditions is true:

– The value of pic_struct is equal to 0, 7, or 8 for all pictures in the coded video sequence.

– The value of pic_struct is equal to 1, 2, 9, 10, 11, or 12 for all pictures in the coded video sequence.

– The value of pic_struct shall be equal to 3, 4, 5, or 6 for all pictures in the coded video sequence.

When fixed_pic_rate_within_cvs_flag is equal to 1, frame doubling is indicated by pic_struct equal to 7, which indicates that the frame should be displayed two times consecutively on displays with a frame refresh interval equal to $\Delta t_{e,dpb}(n)$ as given by Equation E-51, and frame tripling is indicated by pic_struct equal to 8, which indicates that the frame should be displayed three times consecutively on displays with a frame refresh interval equal to $\Delta t_{e,dpb}(n)$ as given by Equation E-51.

NOTE 2 – Frame doubling can be used to facilitate the display, for example, of 25 Hz progressive-scan video on a 50 Hz progressive-scan display or 30 Hz progressive-scan video on a 60 Hz progressive-scan display. Using frame doubling and frame tripling in alternating combination on every other frame can be used to facilitiate the display of 24 Hz progressive-scan video on a 60 Hz progressive-scan display.

The nominal vertical and horizontal sampling locations of samples in top and bottom fields for 4:2:0, 4:2:2, and 4:4:4 chroma formats are shown in Figure D-3, Figure D-4, and Figure D-5, respectively.

Association indicators for fields (pic_struct equal to 9 though 12) provide hints to associate fields of complementary parity together as frames. The parity of a field can be top or bottom, and the parity of two fields is considered complementary when the parity of one field is top and the parity of the other field is bottom.

When frame_field_info_present_flag is equal to 1, it is a requirement of bitstream conformance that the constraints specified in the third column of Table D-3 shall apply.

NOTE 3 – When frame_field_info_present_flag is equal to 0, then in many cases default values may be inferred or indicated by other means. In the absence of other indications of the intended display type of a picture, the decoder should infer the value of pic_struct as equal to 0 when frame_field_info_present_flag is equal to 0.

progressive_source_idc equal to 1 indicates that the scan type of the associated picture should be interpreted as progressive. progressive_source_idc equal to 0 indicates that the scan type of the associated picture should be interpreted as interlaced. progressive_source_idc equal to 2 indicates that the scan type of the associated picture is unknown. progressive_source_idc equal to 3 is reserved for future use by ITU-T | ISO/IEC. When frame_field_info_present_flag is equal to 0, in the absence of other indications of the appropriate interpretation of the associated picture, the value of progressive_source_idc should be inferred to be equal to 1.

duplicate_flag equal to 1 indicates that the current picture is indicated to be a duplicate of a previous picture in output order. duplicate_flag equal to 0 indicates that the current picture is not indicated to be a duplicate of a previous picture in output order.

NOTE 4 – The duplicate_flag should be used to mark coded pictures known to have originated from a repetition process such as 3:2 pull-down or other such duplication and picture rate interpolation methods. This flag would commonly be used when a video feed is encoded as a field sequence in a "transport pass-through" fashion, with known duplicate pictures tagged by setting duplicate_flag equal to 1.

NOTE 5 – When field_seq_flag is equal to 1 and duplicate_flag is equal to 1, this should be interpreted as an indication that the access unit contains a duplicated field of the previous field in output order with the same parity as the current field unless a pairing is otherwise indicated by the use of a pic_struct value in the range of 9 to 12, inclusive.

Table D-3 – Interpretation of pic_struct

Value	Indicated display of picture	Restrictions
0	(progressive) frame	field_seq_flag shall be 0
1	top field	field_seq_flag shall be 1
2	bottom field	field_seq_flag shall be 1
3	top field, bottom field, in that order	field_seq_flag shall be 0
4	bottom field, top field, in that order	field_seq_flag shall be 0
5	top field, bottom field, top field repeated, in that order	field_seq_flag shall be 0
6	bottom field, top field, bottom field repeated, in that order	field_seq_flag shall be 0
7	frame doubling	field_seq_flag shall be 0 fixed_pic_rate_within_cvs_flag shall be 1
8	frame tripling	field_seq_flag shall be 0 fixed_pic_rate_within_cvs_flag shall be 1
9	top field paired with previous bottom field in output order	field_seq_flag shall be 1
10	bottom field paired with previous top field in output order	field_seq_flag shall be 1
11	top field paired with next bottom field in output order	field_seq_flag shall be 1
12	bottom field paired with next top field in output order	field_seq_flag shall be 1
13..15	reserved for future use by ITU-T | ISO/IEC	

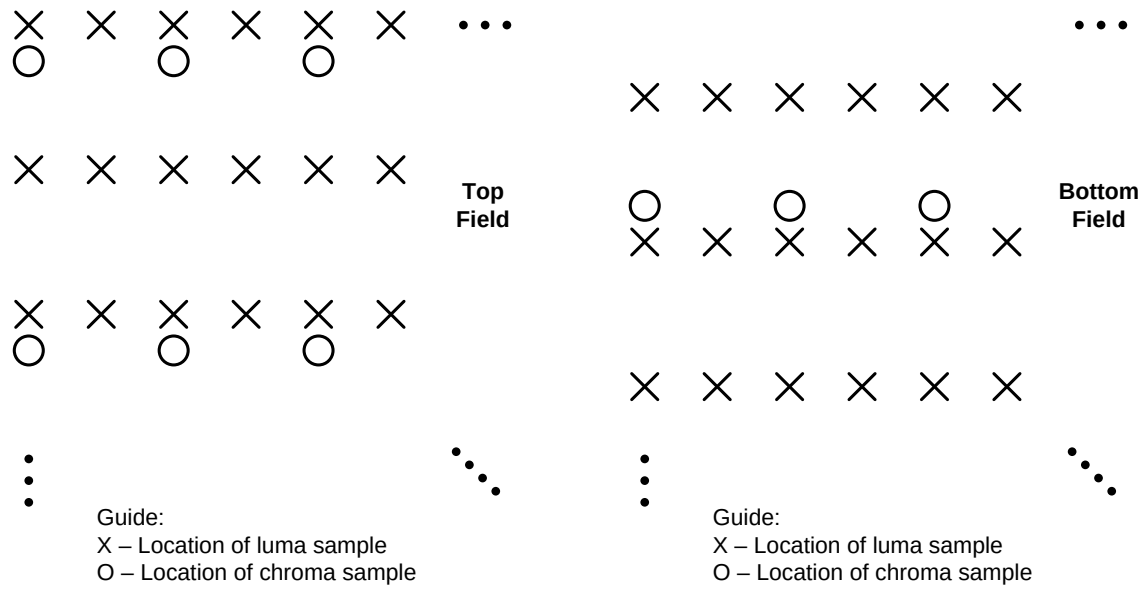


Figure D-3 – Nominal vertical and horizontal sampling locations of 4:2:0 samples in top and bottom fields

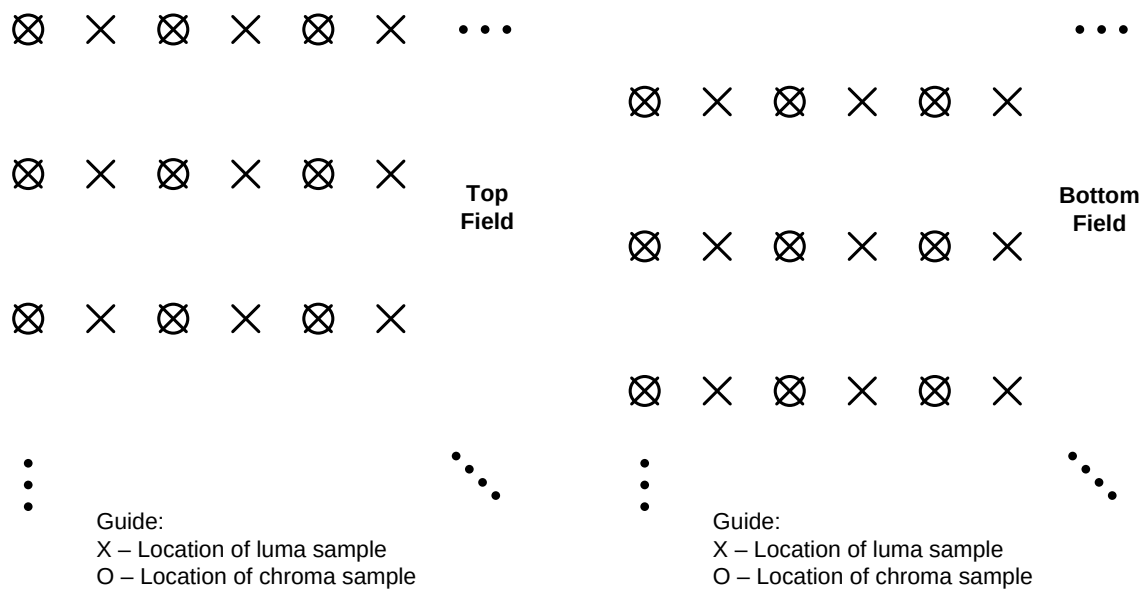


Figure D-4 – Nominal vertical and horizontal sampling locations of 4:2:2 samples in top and bottom fields

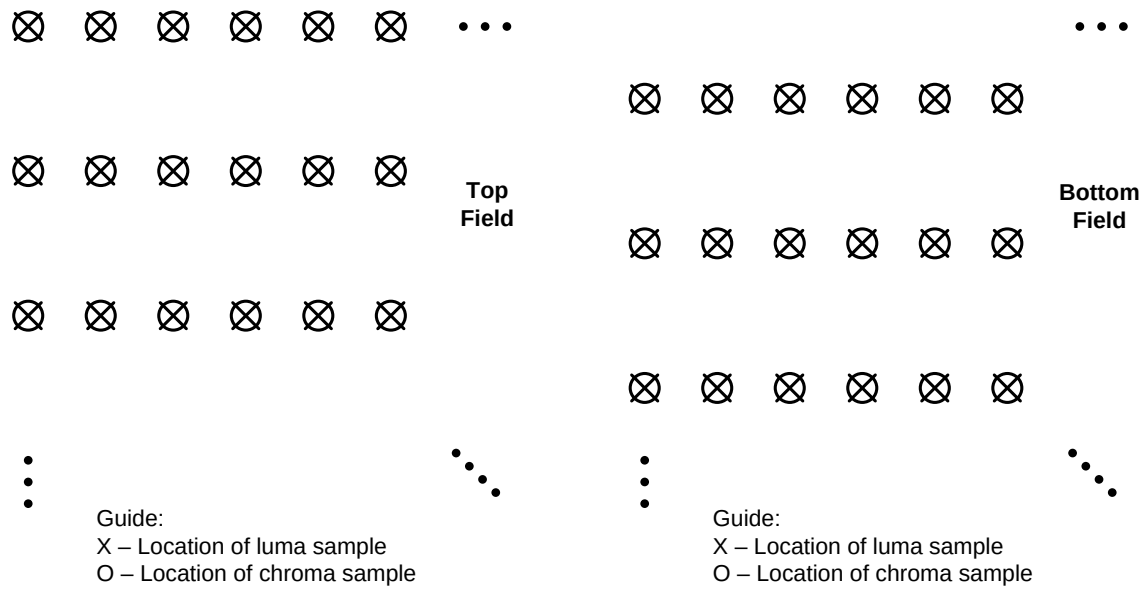


Figure D-5 – Nominal vertical and horizontal sampling locations of 4:4:4 samples in top and bottom fields

au_cpb_removal_delay_minus1 plus 1 specifies the number clock ticks between the nominal CPB removal time of the access unit associated with the picture timing SEI message and the preceding access unit in decoding order that contained a buffering period SEI message. This value is also used to calculate an earliest possible time of arrival of access unit data into the CPB for the HSS. The syntax element is a fixed length code whose length in bits is given by au_cpb_removal_delay_length_minus1 + 1.

NOTE 6 – The value of au_cpb_removal_delay_length_minus1 that determines the length (in bits) of the syntax element au_cpb_removal_delay_minus1 is the value of au_cpb_removal_delay_length_minus1 coded in the video parameter set or the sequence parameter set that is active for the coded picture associated with the picture timing SEI message, although au_cpb_removal_delay_minus1 specifies a number of clock ticks relative to the removal time of the preceding access unit containing a buffering period SEI message, which may be an access unit of a different coded video sequence.

pic_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 the last decoding unit in an access unit from the CPB before the decoded picture is output from the DPB.

NOTE 7 – A picture is not removed from the DPB at its output time when it is still marked as "used for short-term reference" or "used for long-term reference".

NOTE 8 – Only one pic_dpb_output_delay is specified for a decoded picture.

The length of the syntax element pic_dpb_output_delay is given in bits by dpb_output_delay_length_minus1 + 1. When sps_max_dec_pic_buffering[minTid] is equal to 1, where minTid is the minimum of the OpTid values of all operation points the picture timing SEI message applies to, pic_dpb_output_delay shall be equal to 0.

The output time derived from the pic_dpb_output_delay of any picture that is output from an output timing conforming decoder shall precede the output time derived from the pic_dpb_output_delay of all pictures in any subsequent coded video sequence 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, an IDR or BLA picture with no_output_of_prior_pics_flag equal to 1 or inferred to be equal to 1, the output times derived from pic_dpb_output_delay shall be increasing with increasing value of PicOrderCntVal relative to all pictures within the same coded video sequence.

num_decoding_units_minus1 plus 1 specifies the number of decoding units in the access unit the picture timing 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 element du_common_cpb_removal_delay_minus1 is present. du_common_cpb_removal_delay_flag equal to 0 specifies that the syntax element du_common_cpb_removal_delay_minus1 is not present.

du_common_cpb_removal_delay_minus1 plus 1 specifies the duration, in units of sub-picture clock ticks (see subclause E.2.2), between the nominal CPB removal times of any two consecutive decoding units in decoding order in

the access unit associated with the picture timing SEI message. This value is also used to calculate an earliest possible time of arrival of decoding unit data into the CPB for the HSS, as specified in Annex C. The syntax element is a fixed length code whose length in bits is given by du_cpb_removal_delay_length_minus1 + 1.

num_nalus_in_du_minus1[i] plus 1 specifies the number of NAL units in the i-th decoding unit of the access unit the picture timing 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 decoding unit of the access unit consists of the first num_nalus_in_du_minus1[0] + 1 consecutive NAL units in decoding order in the access unit. The i-th (with i greater than 0) decoding unit of the access unit consists of the num_nalus_in_du_minus1[i] + 1 consecutive NAL units immediately following the last NAL unit in the previous decoding unit of the access unit, in decoding order. There shall be at least one VCL NAL unit in each decoding unit. All non-VCL NAL units associated with a VCL NAL unit shall be included in the same decoding unit as the VCL NAL unit.

du_cpb_removal_delay_minus1[i] plus 1 specifies the duration, in units of sub-picture clock ticks, between the nominal CPB removal times of the (i + 1)-th decoding unit and the i-th decoding unit, in decoding order, in the access unit associated with the picture timing SEI message. The value of du_cpb_removal_delay_minus1[num_decoding_units_minus1] is inferred to be equal to 0. This value is also used to calculate an earliest possible time of arrival of decoding unit data into the CPB for the HSS, as specified in Annex C. The syntax element is a fixed length code whose length in bits is given by du_cpb_removal_delay_length_minus1 + 1.

D.2.4 Pan-scan rectangle SEI message semantics

The semantics specified in subclause D.2.3 of ITU-T Rec. H.264 | ISO/IEC 14496-10 apply.

When an SEI NAL unit that contains a pan-scan rectangle SEI message and has nuh_reserved_zero_6bits equal to 0 is present, the SEI NAL unit shall precede, in decoding order, the first VCL NAL unit in the access unit.

D.2.5 Filler payload SEI message semantics

The semantics specified in subclause D.2.4 of ITU-T Rec. H.264 | ISO/IEC 14496-10 apply.

D.2.6 User data registered by ITU-T Rec. T.35 SEI message semantics

The semantics specified in subclause D.2.5 of ITU-T Rec. H.264 | ISO/IEC 14496-10 apply.

D.2.7 User data unregistered SEI message semantics

The semantics specified in subclause D.2.6 of ITU-T Rec. H.264 | ISO/IEC 14496-10 apply.

D.2.8 Recovery point SEI message semantics

The recovery point SEI message assists a decoder in determining when the decoding process will produce acceptable pictures for display after the decoder initiates random access or after the encoder indicates a broken link in the coded video sequence. When the decoding process is started with the access unit in decoding order associated with the recovery point SEI message, all decoded pictures at or subsequent to the recovery point in output order specified in this SEI message are indicated to be correct or approximately correct in content. Decoded pictures produced by random access at or before the picture associated with the recovery point SEI message need not be correct in content until the indicated recovery point, and the operation of the decoding process starting at the picture associated with the recovery point SEI message may contain references to pictures unavailable in the decoded picture buffer.

In addition, by use of the broken_link_flag, the recovery point SEI message can indicate to the decoder the location of some pictures in the bitstream that can result in serious visual artefacts when displayed, even when the decoding process was begun at the location of a previous IDR access unit in decoding order.

NOTE 1 – The broken_link_flag can be used by encoders to indicate the location of a point after which the decoding process for the decoding of some pictures may cause references to pictures that, though available for use in the decoding process, are not the pictures that were used for reference when the bitstream was originally encoded (e.g. due to a splicing operation performed during the generation of the bitstream).

The recovery point is specified as a count in units of PicOrderCntVal increments subsequent to PicOrderCntVal of the current access unit at the position of the SEI message.

When random access is performed to start decoding from the access unit associated with the recovery point SEI message, picture, the associated picture is considered as the first picture in the bitstream, and the variables prevPicOrderCntLsb and prevPicOrderCntMsb used in derivation of PicOrderCntVal are both set to be equal to 0.

NOTE 2 – When HRD information is present in the bitstream, a buffering period SEI message should be associated with the access unit associated with the recovery point SEI message in order to establish initialization of the HRD buffer model after a random access.

Any sequence or picture parameter set RBSP that is referred to by a picture associated with a recovery point SEI message or by any picture following such a picture in decoding order shall be available to the decoding process prior to its activation, regardless of whether or not the decoding process is started at the beginning of the bitstream or with the access unit, in decoding order, that is associated with the recovery point SEI message.

When an SEI NAL unit that contains a recovery point SEI message and has nuh_reserved_zero_6bits equal to 0 is present, the SEI NAL unit shall precede, in decoding order, the first VCL NAL unit in the access unit.

recovery_poc_cnt specifies the recovery point of output pictures in output order. All decoded pictures in output order are indicated to be correct or approximately correct in content starting at the output order position of the picture that has PicOrderCntVal equal to the PicOrderCntVal of the current picture plus the value of recovery_poc_cnt. The value of recovery_poc_cnt shall be in the range of −MaxPicOrderCntLsb / 2 to MaxPicOrderCntLsb / 2 − 1.

exact_match_flag indicates whether decoded pictures at and subsequent to the specified recovery point in output order derived by starting the decoding process at the access unit associated with the recovery point SEI message will be an exact match to the pictures that would be produced by starting the decoding process at the location of a previous RAP access unit, if any, in the bitstream. The value 0 indicates that the match may not be exact and the value 1 indicates that the match will be exact. When exact_match_flag is equal to 1, it is a requirement of bitstream conformance that the decoded pictures at and subsequent to the specified recovery point in output order derived by starting the decoding process at the access unit associated with the recovery point SEI message shall be an exact match to the pictures that would be produced by starting the decoding process at the location of a previous RAP access unit, if any, in the bitstream.

NOTE 3 – When performing random access, decoders should infer all references to unavailable reference pictures as references to pictures containing only intra coding blocks and having sample values given by Y equal to (1 << ($BitDepth_Y$ − 1)), Cb and Cr both equal to (1 << ($BitDepth_C$ − 1)) (mid-level grey), regardless of the value of exact_match_flag.

When exact_match_flag is equal to 0, the quality of the approximation at the recovery point is chosen by the encoding process and is not specified by this Specification.

broken_link_flag indicates the presence or absence of a broken link in the NAL unit stream at the location of the recovery point SEI message and is assigned further semantics as follows:

– If broken_link_flag is equal to 1, pictures produced by starting the decoding process at the location of a previous RAP access unit may contain undesirable visual artefacts to the extent that decoded pictures at and subsequent to the access unit associated with the recovery point SEI message in decoding order should not be displayed until the specified recovery point in output order.

– Otherwise (broken_link_flag is equal to 0), no indication is given regarding any potential presence of visual artefacts.

[Ed. (YK): The broken_link_flag is not useful anymore after the introduction of BLA pictures, and therefore should be removed from the recovery point SEI message.]

Regardless of the value of the broken_link_flag, pictures subsequent to the specified recovery point in output order are specified to be correct or approximately correct in content.

D.2.9 Scene information SEI message semantics

The semantics specified in subclause D.2.10 of ITU-T Rec. H.264 | ISO/IEC 14496-10 apply.

When an SEI NAL unit that contains a scene information SEI message and has nuh_reserved_zero_6bits equal to 0 is present, the SEI NAL unit shall precede, in decoding order, the first VCL NAL unit in the access unit.

D.2.10 Full-frame snapshot SEI message semantics

The semantics specified in subclause D.2.16 of ITU-T Rec. H.264 | ISO/IEC 14496-10 apply.

When an SEI NAL unit that contains a full-frame SEI message and has nuh_reserved_zero_6bits equal to 0 is present, the SEI NAL unit shall precede, in decoding order, the first VCL NAL unit in the access unit.

D.2.11 Progressive refinement segment start SEI message semantics

The semantics specified in subclause D.2.17 of ITU-T Rec. H.264 | ISO/IEC 14496-10 apply.

When an SEI NAL unit that contains a progressive refinement segment start SEI message and has nuh_reserved_zero_6bits equal to 0 is present, the SEI NAL unit shall precede, in decoding order, the first VCL NAL unit in the access unit.

D.2.12 Progressive refinement segment end SEI message semantics

The semantics specified in subclause D.2.18 of ITU-T Rec. H.264 | ISO/IEC 14496-10 apply.

When an SEI NAL unit that contains a progressive refinement segment end SEI message and has nuh_reserved_zero_6bits equal to 0 is present, the SEI NAL unit shall precede, in decoding order, the first VCL NAL unit in the access unit.

D.2.13 Film grain characteristics SEI message semantics

The semantics specified in subclause D.2.20 of ITU-T Rec. H.264 | ISO/IEC 14496-10 apply.

When an SEI NAL unit that contains a film grain characteristics SEI message and has nuh_reserved_zero_6bits equal to 0 is present, the SEI NAL unit shall precede, in decoding order, the first VCL NAL unit in the access unit.

D.2.14 Post-filter hint SEI message semantics

The semantics specified in subclause D.2.23 of ITU-T Rec. H.264 | ISO/IEC 14496-10 apply.

When an SEI NAL unit that contains a post-filter hint SEI message and has nuh_reserved_zero_6bits equal to 0 is present, the SEI NAL unit shall precede, in decoding order, the first VCL NAL unit in the access unit. [Ed. (GJS): How is it necessary to say that?]

D.2.15 Tone mapping information SEI message semantics

This SEI message provides information to enable remapping of the colour samples of the output decoded pictures for customization to particular display environments. The remapping process maps coded sample values in the RGB colour space (specified in Annex E) to target sample values. The mappings are expressed in terms of the RGB colour space and should be applied to each RGB component separately or are expressed in terms of luma and should be applied to the luma component of the decoded image.

When an SEI NAL unit that contains a tone mapping information SEI message and has nuh_reserved_zero_6bits equal to 0 is present, the SEI NAL unit shall precede, in decoding order, the first VCL NAL unit in the access unit.

tone_map_id contains an identifying number that may be used to identify the purpose of the tone mapping model. The value of tone_map_id shall be in the range of 0 to $2^{32} - 2$, inclusive.

Values of tone_map_id from 0 to 255 and from 512 to $2^{31} - 1$ may be used as determined by the application. Values of tone_map_id from 256 to 511 and from 2^{31} to $2^{32} - 2$ are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore (remove from the bitstream and discard) all tone mapping information SEI messages containing a value of tone_map_id in the range of 256 to 511 or in the range of 2^{31} to $2^{32} - 2$, and bitstreams shall not contain such values.

NOTE 1 – The tone_map_id can be used to support tone mapping operations that are suitable for different display scenarios. For example, different values of tone_map_id may correspond to different display bit depths.

tone_map_cancel_flag equal to 1 indicates that the tone mapping information SEI message cancels the persistence of any previous tone mapping information SEI message in output order. tone_map_cancel_flag equal to 0 indicates that tone mapping information follows.

tone_map_repetition_period specifies the persistence of the tone mapping information SEI message and may specify a picture order count interval within which another tone mapping information SEI message with the same value of tone_map_id or the end of the coded video sequence shall be present in the bitstream. The value of tone_map_repetition_period shall be in the range of 0 to 16 384, inclusive.

tone_map_repetition_period equal to 0 specifies that the tone map information applies to the current decoded picture only.

tone_map_repetition_period equal to 1 specifies that the tone map information persists in output order until any of the following conditions are true:

– A new coded video sequence begins.

– A picture in an access unit containing a tone mapping information SEI message with the same value of tone_map_id is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic).

tone_map_repetition_period equal to 0 or equal to 1 indicates that another tone mapping information SEI message with the same value of tone_map_id may or may not be present.

tone_map_repetition_period greater than 1 specifies that the tone map information persists until any of the following conditions are true:

– A new coded video sequence begins.

– A picture in an access unit containing a tone mapping information SEI message with the same value of tone_map_id is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + tone_map_repetition_period.

tone_map_repetition_period greater than 1 indicates that another tone mapping information SEI message with the same value of tone_map_id shall be present for a picture in an access unit that is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + tone_map_repetition_period; unless the bitstream ends or a new coded video sequence begins without output of such a picture.

coded_data_bit_depth specifies the $BitDepth_Y$ of the luma component of the coded video sequence. It is used to identify the tone mapping information SEI message that is intended for use with the coded video sequence. If tone mapping information SEI messages are present that have coded_data_bit_depth that is not equal to $BitDepth_Y$, these refer to the hypothetical result of a transcoding operation performed to convert the coded video to the $BitDepth_Y$ corresponding to the value of coded_data_bit_depth.

The value of coded_data_bit_depth shall be in the range of 8 to 14, inclusive. Values of coded_data_bit_depth from 0 to 7 and from 15 to 255 are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore (remove from the bitstream and discard) all tone mapping SEI messages that contain a coded_data_bit_depth in the range of 0 to 7 or in the range of 15 to 255, and bitstreams shall not contain such values.

target_bit_depth specifies the bit depth of the output of the dynamic range mapping function (or tone mapping function) described by the tone mapping information SEI message. The tone mapping function specified with a particular target_bit_depth is suggested to be reasonable for all display bit depths that are less than or equal to the target_bit_depth.

The value of target_bit_depth shall be in the range of 1 to 16, inclusive, or 255 to indicate an unspecified value. Values of target_bit_depth equal to 0 and in the range of 17 to 254 are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore (remove from the bitstream and discard) all tone mapping SEI messages that contain a value of target_bit_depth equal to 0 or in the range of 17 to 254, and bitstreams shall not contain such values.

model_id specifies the model utilized for mapping the coded data into the target_bit_depth range. Values greater than 4 are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore (remove from the bitstream and discard) all tone mapping SEI messages that contain a value of model_id greater than 4, and bitstreams shall not contain such values.

NOTE 2 – A model_id of 0 corresponds to a linear mapping with clipping; a model_id of 1 corresponds to a sigmoidal mapping; a model_id of 2 corresponds to a user-defined table mapping, and a model_id of 3 corresponds to a piece-wise linear mapping, model_id of 4 corresponds to luminance dynamic range information.

min_value specifies the RGB sample value in the coded data that maps to the minimum value in the signalled target_bit_depth. It is used in combination with the max_value parameter. All values in the coded data that are less than or equal to min_value are mapped to this minimum value in the target_bit_depth representation.

max_value specifies the RGB sample value in the coded data that maps to the maximum value in the signalled target_bit_depth. It is used in combination with the min_value parameter. All values in the coded data that are larger than or equal to max_value are mapped to this maximum value in the target_bit_depth representation.

sigmoid_midpoint specifies the RGB sample value of the coded data that is mapped to the centre point of the target_bit_depth representation. It is used in combination with the sigmoid_width parameter.

sigmoid_width specifies the distance between two coded data values that approximately correspond to the 5% and 95% values of the target_bit_depth representation, respectively. It is used in combination with the sigmoid_midpoint parameter and is interpreted according to the following function:

$$f(i) = \mathrm{Round}\left(\frac{2^{target\_bit\_depth} - 1}{1 + \exp\left(\frac{-6 * (i - sigmoid\_midpoint)}{sigmoid\_width} \right)} \right) \tag{E-24}$$

where *f*(i) denotes the function that maps an RGB sample value i from the coded data to a resulting RGB sample value in the target_bit_depth representation.

start_of_coded_interval[i] specifies the beginning point of an interval in the coded data such that all RGB sample values that are greater than or equal to start_of_coded_interval[i] and less than start_of_coded_interval[i + 1] are mapped to i in the target bit depth representation. The value of start_of_coded_interval[$2^{target\_bit\_depth}$] is equal to $2^{coded\_bit\_depth}$. The number of bits used for the representation of the start_of_coded_interval is ((coded_data_bit_depth + 7) >> 3) << 3.

num_pivots specifies the number of pivot points in the piece-wise linear mapping function without counting the two default end points, (0, 0) and ($2^{coded\_data\_bit\_depth} - 1$, $2^{target\_bit\_depth} - 1$) .

coded_pivot_value[i] specifies the value in the coded_data_bit_depth corresponding to the i-th pivot point. The number of bits used for the representation of the coded_pivot_value is ((coded_data_bit_depth + 7) >> 3) << 3.

target_pivot_value[i] specifies the value in the reference target_bit_depth corresponding to the i-th pivot point. The number of bits used for the representation of the target_pivot_value is ((target_bit_depth + 7) >> 3) << 3.

camera_iso_sensitivity_idc specifies the camera sensitivity to light in ISO number (ISO 12232:2006). Table D-4 shows the mapping of camera_iso_sensitivity_idc to the camera sensitivity to light in ISO number. When camera_iso_sensitivity_idc indicates Extended_ISO, the ISO number is specified by camera_iso_sensitivity.

camera_iso_sensitivity specifies the camera sensitivity to light in ISO number (ISO 12232:2006).

exposure_index_idc specifies exposure index rating setting of the camera in ISO number (ISO 12232:2006). Table D-4 shows the mapping of exposure_index_idc to the exposure index rating setting of the camera in ISO number. When exposure_index_idc indicates Extended_ISO, the ISO number is specified by exposure_index_rating.

exposure_index_rating specifies exposure index rating setting of the camera in ISO number (ISO 12232:2006).

Table D-4 – Mapping of camera_iso_sensitivity_idc and exposure_index_idc to ISO numbers

camera_iso_sensitivity_idc or exposure_index_idc	ISO number
0	Unspecified
1	10
2	12
3	16
4	20
5	25
6	32
7	40
8	50
9	64
10	80
11	100
12	125
13	160
14	200
15	250
16	320
17	400
18	500
19	640
20	800
21	1000
22	1250
23	1600
24	2000
25	2500
26	3200
27	4000
28	5000
29	6400
30	8000
31-254	Reserved

255	Extended_ISO

sign_image_exposure_value specifies the sign of exposure value used in the image production process relative to exposure index rating of the camera. sign_image_exposure_value equal to 0 indicates that the exposure value is positive. sign_image_exposure_value equal to 1 indicates that the exposure value is negative.

image_exposure_value0 and **image_exposure_value1** specify the numerator and the denominator of the magnitude of the exposure value used in the image production process relative to exposure index rating of the camera.

The magnitude of the exposure value used in the image production process relative to exposure index rating of the camera is given by image_exposure_value0 ÷ image_exposure_value1.

NOTE 3 – For example, in case the exposure value is set to +1/2 at production stage, sign_image_exposure_value is set to 0, image_exposure_value0 may be set to 1 and image_exposure_value1 may be set to 2.

screen_lw specifies reference screen luminance setting for white level used for image production process in units of cd/m^2.

max_image_white_level specifies the luminance dynamic range of image expressed as an integer percentage in reference to the nominal white level.

black_level_code_value specifies the luma sample value of the coded data in which the black level is assigned.

whitle_level_code_value specifies luma sample value of the coded data in which the white level is assigned.

max_whitle_level_code_value specifies the luma sample value of the coded data in which the maximum video white level is assigned.

D.2.16 Frame packing arrangement SEI message semantics

This SEI message informs the decoder that the output cropped decoded picture contains samples of multiple distinct spatially packed constituent frames that are packed into one frame using an indicated frame packing arrangement scheme. This information can be used by the decoder to appropriately rearrange the samples and process the samples of the constituent frames appropriately for display or other purposes (which are outside the scope of this Specification).

When an SEI NAL unit that contains a frame packing arrangement SEI message and has nuh_reserved_zero_6bits equal to 0 is present, the SEI NAL unit shall precede, in decoding order, the first VCL NAL unit in the access unit.

This SEI message may be associated with pictures that are either frames or fields. The frame packing arrangement of the samples is specified in terms of the sampling structure of a frame in order to define a frame packing arrangement structure that is invariant with respect to whether a picture is a single field of such a packed frame or is a complete packed frame.

frame_packing_arrangement_id contains an identifying number that may be used to identify the usage of the frame packing arrangement SEI message. The value of frame_packing_arrangement_id shall be in the range of 0 to $2^{32} - 2$, inclusive.

Values of frame_packing_arrangement_id from 0 to 255 and from 512 to $2^{31} - 1$ may be used as determined by the application. Values of frame_packing_arrangement_id from 256 to 511 and from 2^{31} to $2^{32} - 2$ are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore (remove from the bitstream and discard) all frame packing arrangement SEI messages containing a value of frame_packing_arrangement_id in the range of 256 to 511 or in the range of 2^{31} to $2^{32} - 2$, and bitstreams shall not contain such values.

frame_packing_arrangement_cancel_flag equal to 1 indicates that the frame packing arrangement SEI message cancels the persistence of any previous frame packing arrangement SEI message in output order. frame_packing_arrangement_cancel_flag equal to 0 indicates that frame packing arrangement information follows.

frame_packing_arrangement_type indicates the type of packing arrangement of the frames as specified in Table D-8.

Table D-8 – Definition of frame_packing_arrangement_type

Value	Interpretation
0	Each component plane of the decoded frames contains a "checkerboard" based interleaving of corresponding planes of two constituent frames as illustrated in Figure D-1.
1	Each component plane of the decoded frames contains a column based interleaving of corresponding planes of two constituent frames as illustrated in Figure D-2 and Figure D-3.
2	Each component plane of the decoded frames contains a row based interleaving of corresponding planes of two constituent frames as illustrated in Figure D-4 and Figure D-5.
3	Each component plane of the decoded frames contains a side-by-side packing arrangement of corresponding planes of two constituent frames as illustrated in Figure D-6, Figure D-7, and Figure D-10.
4	Each component plane of the decoded frames contains a top-bottom packing arrangement of corresponding planes of two constituent frames as illustrated in Figure D-8 and Figure D-9.
5	The component planes of the decoded frames in output order form a temporal interleaving of alternating first and second constituent frames as illustrated in Figure D-11.
6	Each decoded frame constitutes a single frame without the use of a frame packing of multiple consituent frames (see NOTE 5).
7	Each component plane of the decoded frames contains a rectangular region frame packing arrangement of corresponding planes of two constituent frames as illustrated in Figure D-12.

NOTE 1 – Figure D-1 to Figure D-10 provide typical examples of rearrangement and upconversion processing for various packing arrangement schemes. Actual characteristics of the constituent frames are signalled in detail by the subsequent syntax elements of the frame packing arrangement SEI message. In Figure D-1 to Figure D-10, an upconversion processing is performed on each constituent frame to produce frames having the same resolution as that of the decoded frame. An example of the upsampling method to be applied to a quincunx sampled frame as shown in Figure D-1 or Figure D-10 is to fill in missing positions with an average of the available spatially neighbouring samples (the average of the values of the available samples above, below, to the left and to the right of each sample to be generated). The actual upconversion process to be performed, if any, is outside the scope of this Specification.

NOTE 2 – When the output time of the samples of constituent frame 0 differs from the output time of the samples of constituent frame 1 (i.e., when field_views_flag is equal to 1 or frame_packing_arrangement_type is equal to 5) and the display system in use presents two views simultaneously, the display time for constituent frame 0 should be delayed to coincide with the display time for constituent frame 1. (The display process is not specified in this Recommendation | International Standard.)

NOTE 3 – When field_views_flag is equal to 1 or frame_packing_arrangement_type is equal to 5, the value 0 for fixed_pic_rate_within_cvs_flag is not expected to be prevalent in industry use of this SEI message.

NOTE 4 – frame_packing_arrangement_type equal to 5 describes a temporal interleaving process of different views.

NOTE 5 – The value of frame_packing_arrangement_type equal to 6 is used to signal presence of 2D content (that is not frame packed) in 3D services that use such a mix of contents. The frame_packing_arrangement_type value of 6 should only be used with frame pictures and with content_interpretation_type equal to 0.

NOTE 6 – Figure D-12 provides an illustration of the rearrangement process for the frame_packing_arrangement_type value of 7.

All other values of frame_packing_arrangement_type are reserved for future use by ITU-T | ISO/IEC. It is a requirement of bitstream conformance that the bitstreams shall not contain such other values of frame_packing_arrangement_type.

quincunx_sampling_flag equal to 1 indicates that each colour component plane of each constituent frame is quincunx sampled as illustrated in Figure D-1 or Figure D-10, and quincunx_sampling_flag equal to 0 indicates that the colour component planes of each constituent frame are not quincunx sampled.

When frame_packing_arrangement_type is equal to 0, it is a requirement of bitstream conformance that quincunx_ sampling_flag shall be equal to 1. When frame_packing_arrangement_type is equal to 5, 6, or 7, it is a requirement of bitstream conformance that quincunx_sampling_flag shall be equal to 0.

NOTE 7 – For any chroma format (4:2:0, 4:2:2, or 4:4:4), the luma plane and each chroma plane is quincunx sampled as illustrated in Figure D-1 when quincunx_sampling_flag is equal to 1.

Let croppedWidth and croppedHeight be the width and height, respectively, of the cropped frame output from the decoder in units of luma samples, derived as follows:

$$\begin{aligned} \text{croppedWidth} = \text{pic\_width\_in\_luma\_samples} - \\ \text{SubWidthC} * (\text{conf\_win\_right\_offset} + \text{conf\_win\_left\_offset}) \end{aligned} \quad \text{(D-1)}$$

$$\text{croppedHeight} = \text{pic\_height\_in\_luma\_samples} - \text{SubHeightC} * (\text{conf\_win\_bottom\_offset} + \text{conf\_win\_top\_offset}) \quad \text{(D-2)}$$

When frame_packing_arrangement_type is equal to 7, it is a requirement of bitstream conformance that both of the following conditions shall be true:

– croppedWidth is an integer multiple of 3 * SubWidthC

– croppedHeight is an integer multiple of 3 * SubHeightC

Let oneThirdWidth and oneThirdHeight be derived as follows:

$$\text{oneThirdWidth} = \text{croppedWidth} / 3 \quad \text{(D-3)}$$

$$\text{oneThirdHeight} = \text{croppedHeight} / 3 \quad \text{(D-4)}$$

When frame_packing_arrangement_type is equal to 7, the frame packing arrangement is composed of four rectangular regions as illustrated in Figure D-12. The upper left region contains consituent frame 0 and has corner points (xA, yA) (at upper left), (xB − 1, yB) (at upper right), (xC − 1, yC − 1) (at lower right) and (xD, yD − 1) (at lower left). Constituent frame 1 is decomposed into three regions, denoted as R1, R2 and R3. Region R1 has corner points (xB, yB) (at upper left), (xL − 1, yL) (at upper right), (xI − 1, yI − 1) (at lower right) and (xC, yC − 1) (at lower left); region R2 has corner points (xD, yD) (at upper left), (xG − 1, yG) (at upper right), (xE, yE − 1) (at lower left) and (xF − 1, yF − 1) (at lower right); region R3 has corner points (xG, yG) (at upper left), (xC − 1, yC) (at upper right), (xF, yF − 1) (at lower left) and (xH − 1, yH − 1) (at lower right). The (x, y) locations of points (xA, yA) to (xL, yL) are calculated as follows in units of luma sample coordinate positions:

$$(\text{xA}, \text{yA}) = (\text{SubWidthC} * \text{conf\_win\_left\_offset}, \text{SubHeightC} * \text{conf\_win\_top\_offset}) \quad \text{(D-5)}$$

$$(\text{xB}, \text{yB}) = (\text{SubWidthC} * \text{conf\_win\_left\_offset} + 2 * \text{oneThirdWidth}, \text{SubHeightC} * \text{conf\_win\_top\_offset}) \quad \text{(D-6)}$$

$$(\text{xC}, \text{yC}) = (\text{SubWidthC} * \text{conf\_win\_left\_offset} + 2 * \text{oneThirdWidth}, \text{SubHeightC} * \text{conf\_win\_top\_offset} + 2 * \text{oneThirdHeight}) \quad \text{(D-7)}$$

$$(\text{xD}, \text{yD}) = (\text{SubWidthC} * \text{conf\_win\_left\_offset}, \text{SubHeightC} * \text{conf\_win\_top\_offset} + 2 * \text{oneThirdHeight}) \quad \text{(D-8)}$$

$$(\text{xE}, \text{yE}) = (\text{SubWidthC} * \text{conf\_win\_left\_offset}, \text{SubHeightC} * \text{conf\_win\_top\_offset} + \text{croppedHeight}) \quad \text{(D-9)}$$

$$(\text{xF}, \text{yF}) = (\text{SubWidthC} * \text{conf\_win\_left\_offset} + \text{oneThirdWidth}, \text{SubHeightC} * \text{conf\_win\_top\_offset} + \text{croppedHeight}) \quad \text{(D-10)}$$

$$(\text{xG}, \text{yG}) = (\text{SubWidthC} * \text{conf\_win\_left\_offset} + \text{oneThirdWidth}, \text{SubHeightC} * \text{conf\_win\_top\_offset} + 2 * \text{oneThirdHeight}) \quad \text{(D-11)}$$

$$(\text{xH}, \text{yH}) = (\text{SubWidthC} * \text{conf\_win\_left\_offset} + 2 * \text{oneThirdWidth}, \text{SubHeightC} * \text{conf\_win\_top\_offset} + \text{croppedHeight}) \quad \text{(D-12)}$$

$$(\text{xI}, \text{yI}) = (\text{SubWidthC} * \text{conf\_win\_left\_offset} + \text{croppedWidth}, \text{SubHeightC} * \text{conf\_win\_top\_offset} + 2 * \text{oneThirdHeight}) \quad \text{(D-13)}$$

$$(\text{xL}, \text{yL}) = (\text{SubWidthC} * \text{conf\_win\_left\_offset} + \text{croppedWidth}, \text{SubHeightC} * \text{conf\_win\_top\_offset} + \text{croppedHeight}) \quad \text{(D-14)}$$

When frame_packing_arrangement_type is equal to 7, constituent frame 0 is obtained by cropping from the decoded frames the region R0 enclosed by points A, B, C, and D, and constituent frame 1 is obtained by stacking vertically the regions R2 and R3, obtained by cropping the areas enclosed by points D, G, F, and E and G, C, F, and H, respectively, and then placing the resulting rectangle to the right of the region R1, obtained by cropping the area enclosed by points B, L, I, and C, as illustrated in Figure D-12.

content_interpretation_type indicates the intended interpretation of the constituent frames as specified in Table D-9. Values of content_interpretation_type that do not appear in Table D-9 are reserved for future specification by ITU-T | ISO/IEC.

When frame_packing_arrangement_type is not equal to 6, for each specified frame packing arrangement scheme, there are two constituent frames that are referred to as frame 0 and frame 1.

Table D-9 – Definition of content_interpretation_type

Value	Interpretation
0	Unspecified relationship between the frame packed constituent frames
1	Indicates that the two constituent frames form the left and right views of a stereo view scene, with frame 0 being associated with the left view and frame 1 being associated with the right view
2	Indicates that the two constituent frames form the right and left views of a stereo view scene, with frame 0 being associated with the right view and frame 1 being associated with the left view

NOTE 8 – The value 2 for content_interpretation_type is not expected to be prevalent in industry use of this SEI message. However, the value was specified herein for purposes of completeness.

spatial_flipping_flag equal to 1, when frame_packing_arrangement_type is equal to 3 or 4, indicates that one of the two constituent frames is spatially flipped relative to its intended orientation for display or other such purposes.

When frame_packing_arrangement_type is equal to 3 or 4 and spatial_flipping_flag is equal to 1, the type of spatial flipping that is indicated is as follows:

– If frame_packing_arrangement_type is equal to 3, the indicated spatial flipping is horizontal flipping.

– Otherwise (frame_packing_arrangement_type is equal to 4), the indicated spatial flipping is vertical flipping.

When frame_packing_arrangement_type is not equal to 3 or 4, it is a requirement of bitstream conformance that spatial_flipping_flag shall be equal to 0. When frame_packing_arrangement_type is not equal to 3 or 4, the value 1 for spatial_flipping_flag is reserved for future use by ITU-T | ISO/IEC. When frame_packing_arrangement_type is not equal to 3 or 4, decoders shall ignore the value 1 for spatial_flipping_flag.

frame0_flipped_flag, when spatial_flipping_flag is equal to 1, indicates which one of the two constituent frames is flipped.

When spatial_flipping_flag is equal to 1, frame0_flipped_flag equal to 0 indicates that frame 0 is not spatially flipped and frame 1 is spatially flipped, and frame0_flipped_flag equal to 1 indicates that frame 0 is spatially flipped and frame 1 is not spatially flipped.

When spatial_flipping_flag is equal to 0, it is a requirement of bitstream conformance that frame0_flipped_flag shall be equal to 0. When spatial_flipping_flag is equal to 0, the value 1 for spatial_flipping_flag is reserved for future use by ITU-T | ISO/IEC. When spatial_flipping_flag is equal to 0, decoders shall ignore the value of frame0_flipped_flag.

field_views_flag equal to 1 indicates that all pictures in the current coded video sequence are coded as complementary field pairs. All fields of a particular parity are considered a first constituent frame and all fields of the opposite parity are considered a second constituent frame. When frame_packing_arrangement_type is not equal to 2, it is a requirement of bitstream conformance that the field_views_flag shall be equal to 0. When frame_packing_arrangement_type is not equal to 2, the value 1 for field_views_flag is reserved for future use by ITU-T | ISO/IEC. When frame_ packing_arrangement_type is not equal to 2, decoders shall ignore the value of field_views_flag.

current_frame_is_frame0_flag equal to 1, when frame_packing_arrangement is equal to 5, indicates that the current decoded frame is constituent frame 0 and the next decoded frame in output order is constituent frame 1, and the display time of the constituent frame 0 should be delayed to coincide with the display time of constituent frame 1. current_frame_is_frame0_flag equal to 0, when frame_packing_arrangement is equal to 5, indicates that the current decoded frame is constituent frame 1 and the previous decoded frame in output order is constituent frame 0, and the display time of the constituent frame 1 should not be delayed for purposes of stereo-view pairing.

When frame_packing_arrangement_type is not equal to 5, the constituent frame associated with the upper-left sample of the decoded frame is considered to be consitutuent frame 0 and the other constituent frame is considered to be constituent frame 1. When frame_packing_arrangement_type is not equal to 5, it is a requirement of bitstream conformance that current_frame_is_frame0_flag shall be equal to 0. When frame_packing_arrangement_type is not equal to 5, the value 1 for current_frame_is_frame0_flag is reserved for future use by ITU-T | ISO/IEC. When frame_packing_ arrangement_type is not equal to 5, decoders shall ignore the value of current_frame_is_frame0_flag.

frame0_self_contained_flag equal to 1 indicates that no inter prediction operations within the decoding process for the samples of constituent frame 0 of the coded video sequence refer to samples of any constituent frame 1. frame0_self_contained_flag equal to 0 indicates that some inter prediction operations within the decoding process for the samples of constituent frame 0 of the coded video sequence may or may not refer to samples of some constituent frame 1. When frame_packing_arrangement_type is equal to 0 or 1, it is a requirement of bitstream conformance that frame0_self_contained_flag shall be equal to 0. When frame_packing_arrangement_type is equal to 0 or 1, the value 1 for frame0_self_contained_flag is reserved for future use by ITU-T | ISO/IEC. When frame_packing_arrangement_type is equal to 0 or 1, decoders shall ignore the value of frame0_self_contained_flag. Within a coded video sequence, the value of frame0_self_contained_flag in all frame packing arrangement SEI messages shall be the same.

frame1_self_contained_flag equal to 1 indicates that no inter prediction operations within the decoding process for the samples of constituent frame 1 of the coded video sequence refer to samples of any constituent frame 0. frame1_self_contained_flag equal to 0 indicates that some inter prediction operations within the decoding process for the samples of constituent frame 1 of the coded video sequence may or may not refer to samples of some constituent frame 0. When frame_packing_arrangement_type is equal to 0 or 1, it is a requirement of bitstream conformance that frame1_self_contained_flag shall be equal to 0. When frame_packing_arrangement_type is equal to 0 or 1, the value 1 for frame1_self_contained_flag is reserved for future use by ITU-T | ISO/IEC. When frame_packing_arrangement_type is equal to 0 or 1, decoders shall ignore the value of frame1_self_contained_flag. Within a coded video sequence, the value of frame1_self_contained_flag in all frame packing arrangement SEI messages shall be the same.

NOTE 9 – When frame0_self_contained_flag is equal to 1 or frame1_self_contained_flag is equal to 1, and frame_packing_arrangement_type is equal to 2, it is expected that the decoded frame should not be an MBAFF frame.

When quincunx_sampling_flag is equal to 0 and frame_packing_arrangement_type is not equal to 5, two (x, y) coordinate pairs are specified to determine the indicated luma sampling grid alignment for constituent frame 0 and constituent frame 1, relative to the upper left corner of the rectangular area represented by the samples of the corresponding constituent frame.

NOTE 10 – The location of chroma samples relative to luma samples can be indicated by the chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field syntax elements in the VUI parameters.

frame0_grid_position_x (when present) specifies the x component of the (x, y) coordinate pair for constituent frame 0.

frame0_grid_position_y (when present) specifies the y component of the (x, y) coordinate pair for constituent frame 0.

frame1_grid_position_x (when present) specifies the x component of the (x, y) coordinate pair for constituent frame 1.

frame1_grid_position_y (when present) specifies the y component of the (x, y) coordinate pair for constituent frame 1.

When quincunx_sampling_flag is equal to 0 and frame_packing_arrangement_type is not equal to 5 the (x, y) coordinate pair for each constituent frame is interpreted as follows:

- If the (x, y) coordinate pair for a constituent frame is equal to (0, 0), this indicates a default sampling grid alignment specified as follows:
  - If frame_packing_arrangement_type is equal to 1 or 3, the indicated position is the same as for the (x, y) coordinate pair value (4, 8), as illustrated in Figure D-2 and Figure D-6.
  - Otherwise (frame_packing_arrangement_type is equal to 2 or 4), the indicated position is the same as for the (x, y) coordinate pair value (8, 4), as illustrated in Figure D-4 and Figure D-8.
- Otherwise, if the (x, y) coordinate pair for a constituent frame is equal to (15, 15), this indicates that the sampling grid alignment is unknown or unspecified or specified by other means not specified in this Recommendation | International Standard.
- Otherwise, the x and y elements of the (x, y) coordinate pair specify the indicated horizontal and vertical sampling grid alignment positioning to the right of and below the upper left corner of the rectangular area represented by the corresponding constituent frame, respectively, in units of one sixteenth of the luma sample grid spacing between the samples of the columns and rows of the constituent frame that are present in the decoded frame (prior to any upsampling for display or other purposes).

NOTE 11 – The spatial location reference information frame0_grid_position_x, frame0_grid_position_y, frame1_grid_position_x, and frame1_grid_position_y is not provided when quincunx_sampling_flag is equal to 1 because the spatial alignment in this case is assumed to be such that constituent frame 0 and constituent frame 1 cover corresponding spatial areas with interleaved quincunx sampling patterns as illustrated in Figure D-1 and Figure D-10.

NOTE 12 – When frame_packing_arrangement_type is equal to 2 and field_views_flag is equal to 1, it is suggested that frame0_grid_position_y should be equal to frame1_grid_position_y.

frame_packing_arrangement_reserved_byte is reserved for future use by ITU-T | ISO/IEC. It is a requirement of bitstream conformance that the value of frame_packing_arrangement_reserved_byte shall be equal to 0. All other values

of frame_packing_arrangement_reserved_byte are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore (remove from the bitstream and discard) the value of frame_packing_arrangement_reserved_byte.

frame_packing_arrangement_repetition_period specifies the persistence of the frame packing arrangement SEI message and may specify a frame order count interval within which another frame packing arrangement SEI message with the same value of frame_packing_arrangement_id or the end of the coded video sequence shall be present in the bitstream. The value of frame_packing_arrangement_repetition_period shall be in the range of 0 to 16 384, inclusive.

frame_packing_arrangement_repetition_period equal to 0 specifies that the frame packing arrangement SEI message applies to the current decoded frame only.

frame_packing_arrangement_repetition_period equal to 1 specifies that the frame packing arrangement SEI message persists in output order until any of the following conditions are true:

– A new coded video sequence begins.

– A frame in an access unit containing a frame packing arrangement SEI message with the same value of frame_packing_arrangement_id is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic).

frame_packing_arrangement_repetition_period equal to 0 or equal to 1 indicates that another frame packing arrangement SEI message with the same value of frame_packing_arrangement_id may or may not be present.

frame_packing_arrangement_repetition_period greater than 1 specifies that the frame packing arrangement SEI message persists until any of the following conditions are true:

– A new coded video sequence begins.

– A frame in an access unit containing a frame packing arrangement SEI message with the same value of frame_packing_arrangement_id is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + frame_packing_arrangement_repetition_period.

frame_packing_arrangement_repetition_period greater than 1 indicates that another frame packing arrangement SEI message with the same value of frame_packing_arrangement_frames_id shall be present for a frame in an access unit that is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + frame_packing_arrangement_repetition_period; unless the bitstream ends or a new coded video sequence begins without output of such a frame.

upsampled_aspect_ratio_flag equal to 1 indicates that the sample aspect ratio (SAR) indicated by the VUI parameters of the sequence parameter set identifies the SAR of the samples after the application of an upconversion process to produce a higher resolution frame from each constituent frame as illustrated in Figure D-1 to Figure D-10. upsampled_aspect_ratio_flag equal to 0 indicates that the SAR indicated by the VUI parameters of the sequence parameter set identifies the SAR of the samples before the application of any such upconversion process.

NOTE 13 – The default display window parameters in the VUI parameters of the sequence parameter set can be used by an encoder to indicate to a decoder that does not interpret the frame packing arrangement SEI message that the default display window is an area within only one of the two constituent frames.

NOTE 14 – The SAR indicated in the VUI parameters should indicate the preferred display picture shape for the packed decoded frame output by a decoder that does not interpret the frame packing arrangement SEI message. When upsampled_aspect_ratio_flag is equal to 1, the SAR produced in each upconverted colour plane is indicated to be the same as the SAR indicated in the VUI parameters in the examples shown in Figure D-1 to Figure D-10. When upsampled_aspect_ratio_flag is equal to 0, the SAR produced in each colour plane prior to upconversion is indicated to be the same as the SAR indicated in the VUI parameters in the examples shown in Figure D-1 to Figure D-10.

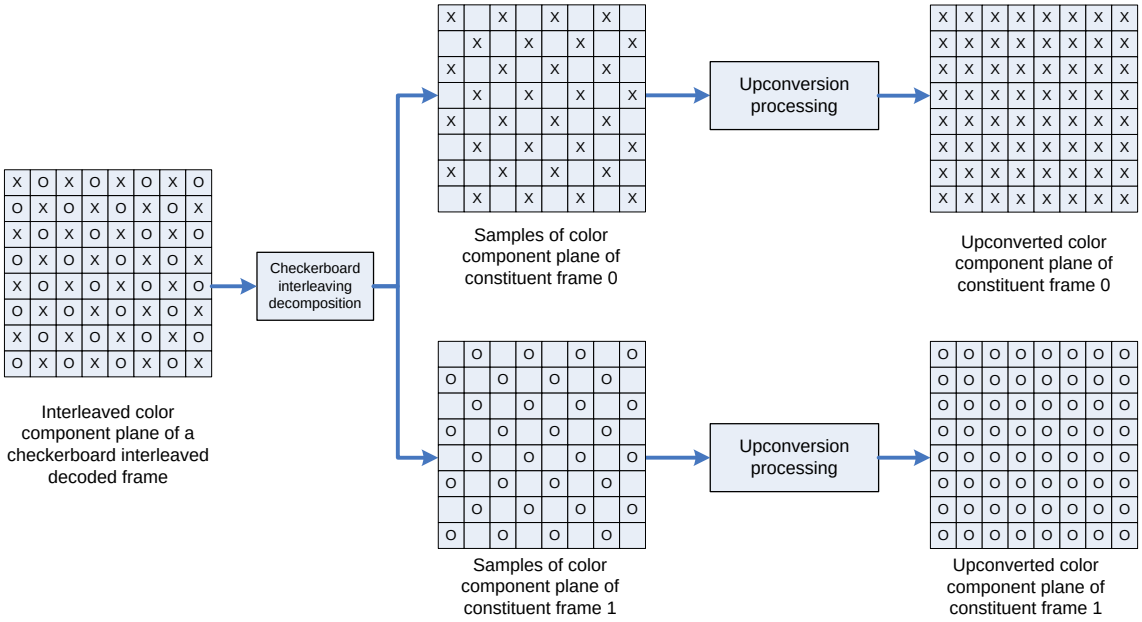


Figure D-1 – Rearrangement and upconversion of checkerboard interleaving (frame_packing_arrangement_type equal to 0)

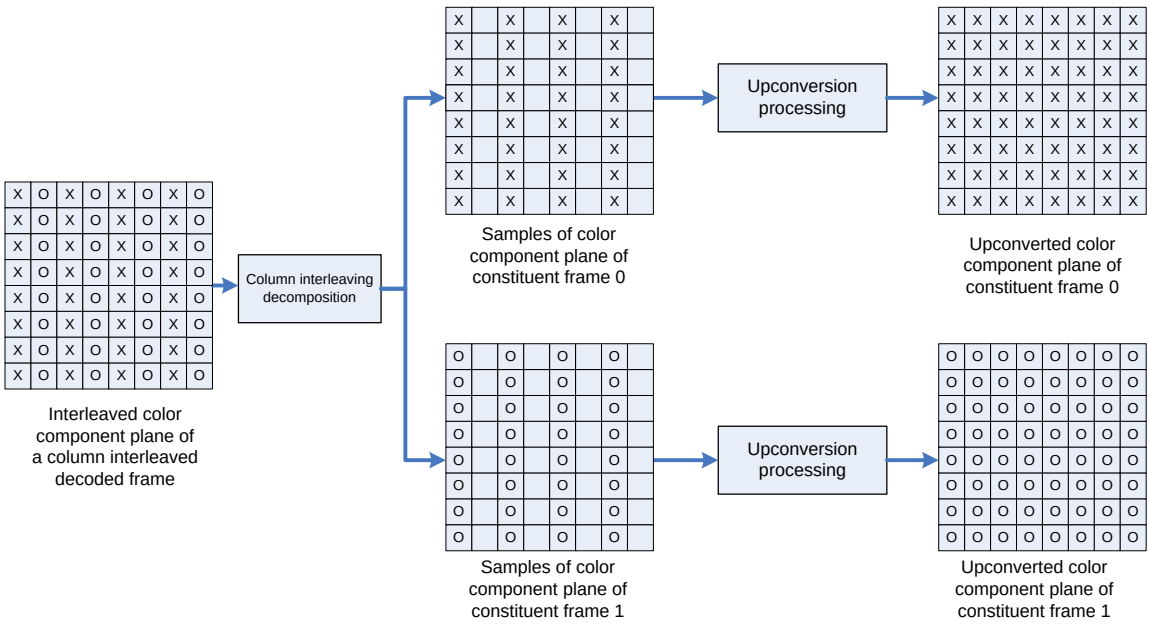


Figure D-2 – Rearrangement and upconversion of column interleaving with frame_packing_arrangement_type equal to 1, quincunx_sampling_flag equal to 0, and (x, y) equal to (0, 0) or (4, 8) for both constituent frames

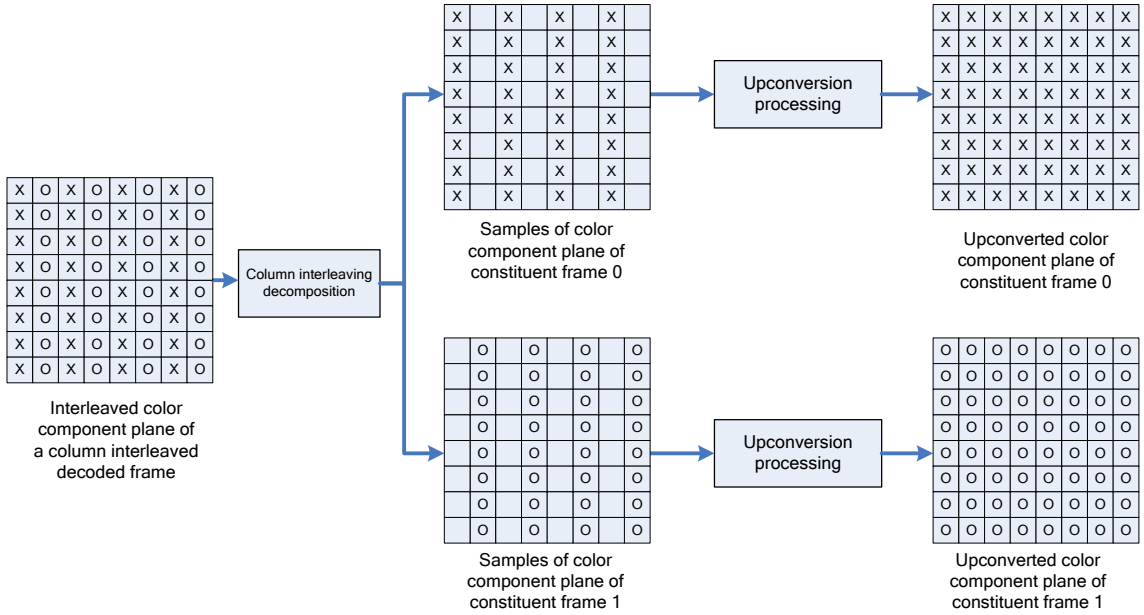


Figure D-3 – Rearrangement and upconversion of column interleaving with frame_packing_arrangement_type equal to 1, quincunx_sampling_flag equal to 0, (x, y) equal to (0, 0) or (4, 8) for constituent frame 0 and (x, y) equal to (12, 8) for constituent frame 1

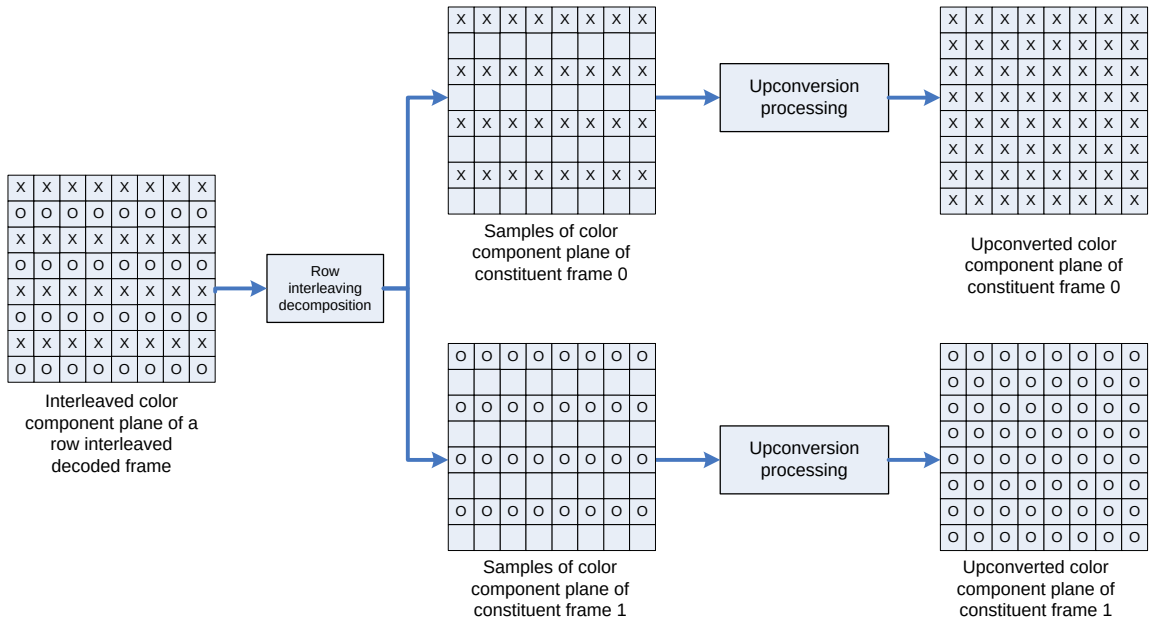


Figure D-4 – Rearrangement and upconversion of row interleaving with frame_packing_arrangement_type equal to 2, quincunx_sampling_flag equal to 0, and (x, y) equal to (0, 0) or (8, 4) for both constituent frames

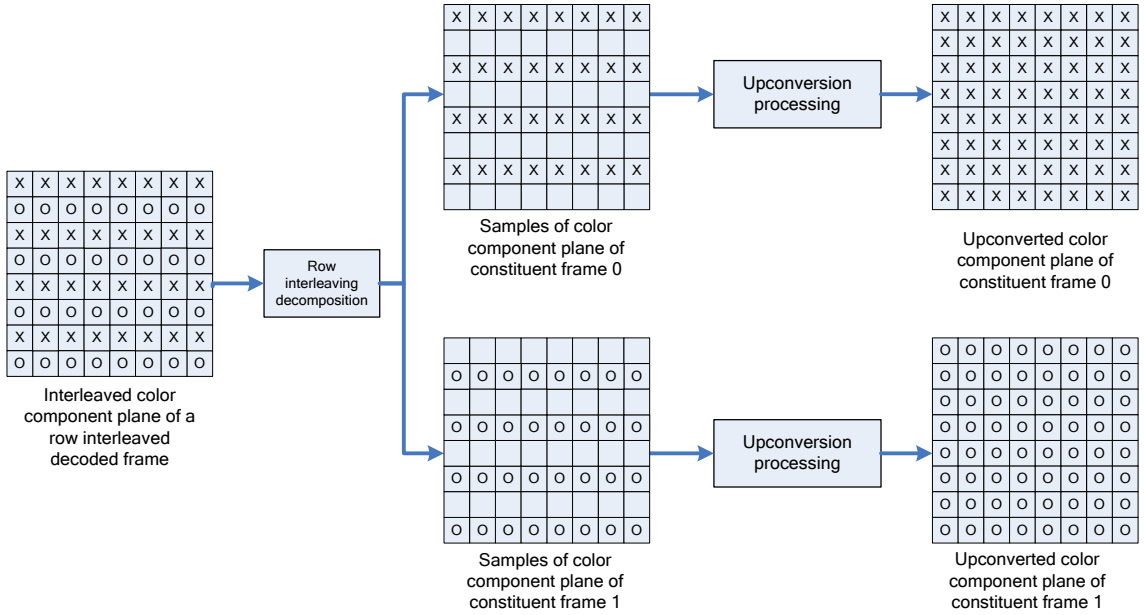


Figure D-5 – Rearrangement and upconversion of row interleaving with frame_packing_arrangement_type equal to 2, quincunx_sampling_flag equal to 0, (x, y) equal to (0, 0) or (8, 4) for constituent frame 0, and (x, y) equal to (8, 12) for constituent frame 1

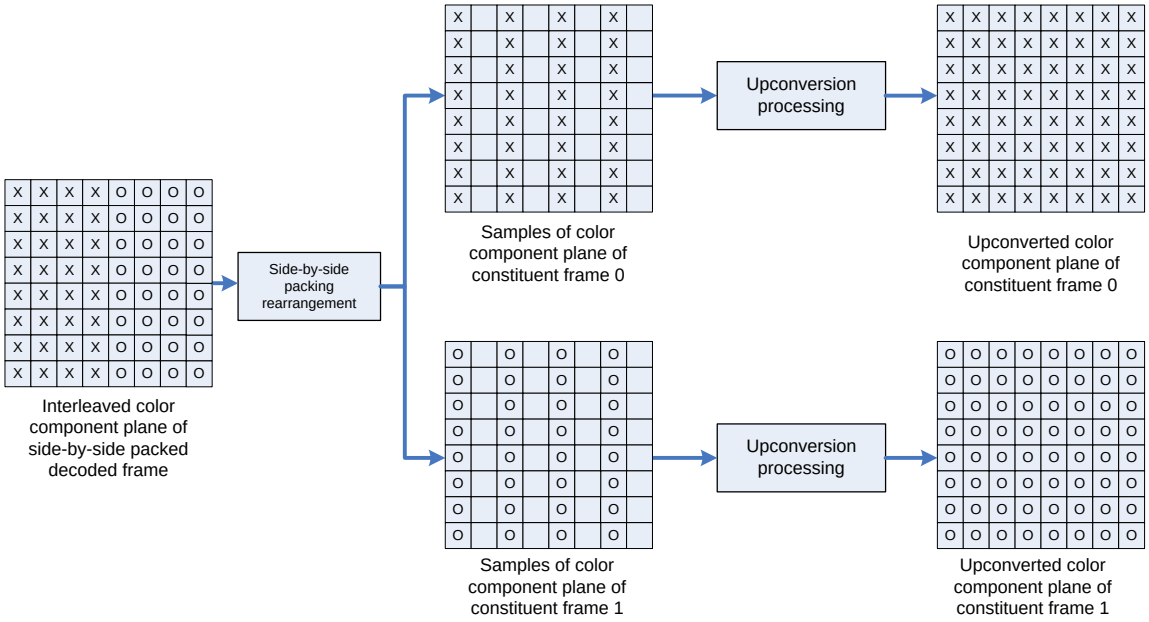


Figure D-6 – Rearrangement and upconversion of side-by-side packing arrangement with frame_packing_arrangement_type equal to 3, quincunx_sampling_flag equal to 0, and (x, y) equal to (0, 0) or (4, 8) for both constituent frames

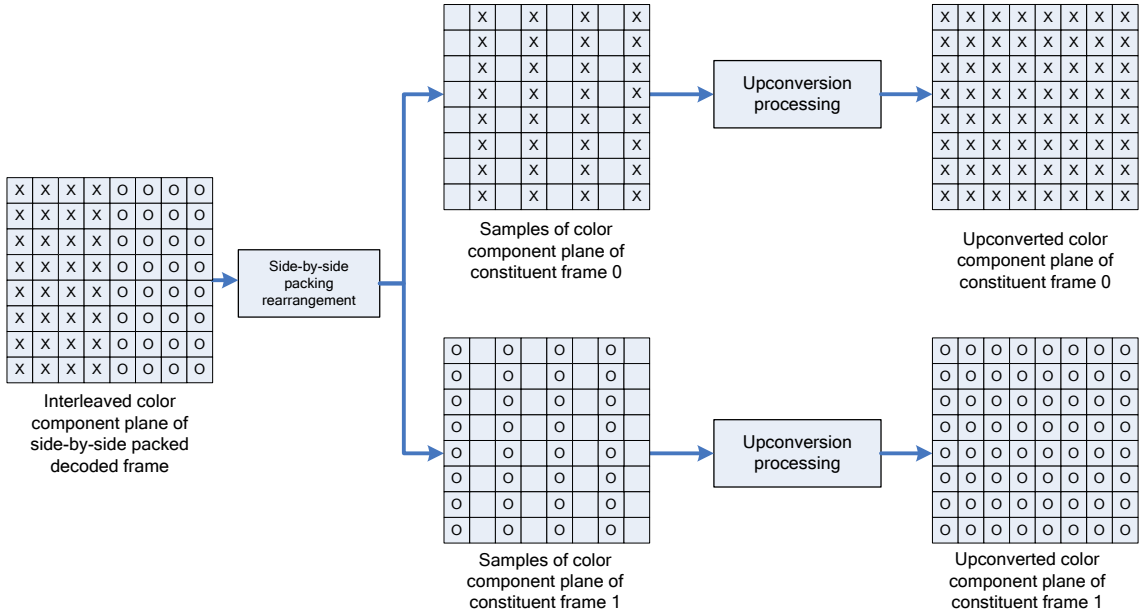


Figure D-7 – Rearrangement and upconversion of side-by-side packing arrangement with frame_packing_arrangement_type equal to 3, quincunx_sampling_flag equal to 0, (x, y) equal to (12, 8) for constituent frame 0, and (x, y) equal to (0, 0) or (4, 8) for constituent frame 1

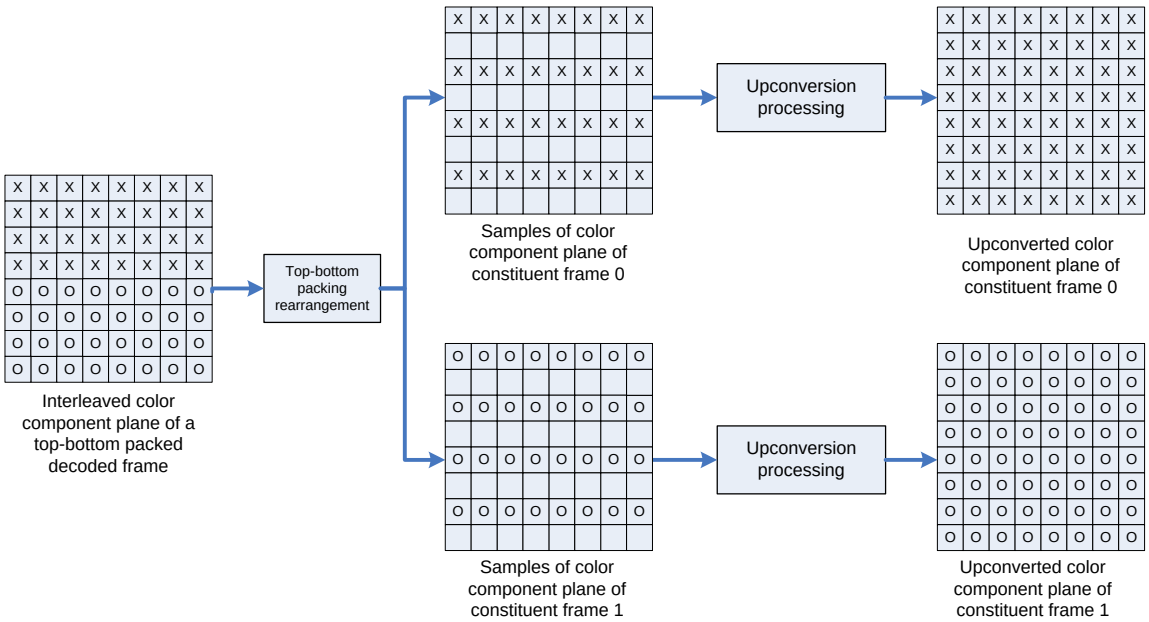


Figure D-8 – Rearrangement and upconversion of top-bottom packing arrangement with frame_packing_arrangement_type equal to 4, quincunx_sampling_flag equal to 0, and (x, y) equal to (0, 0) or (8, 4) for both constituent frames

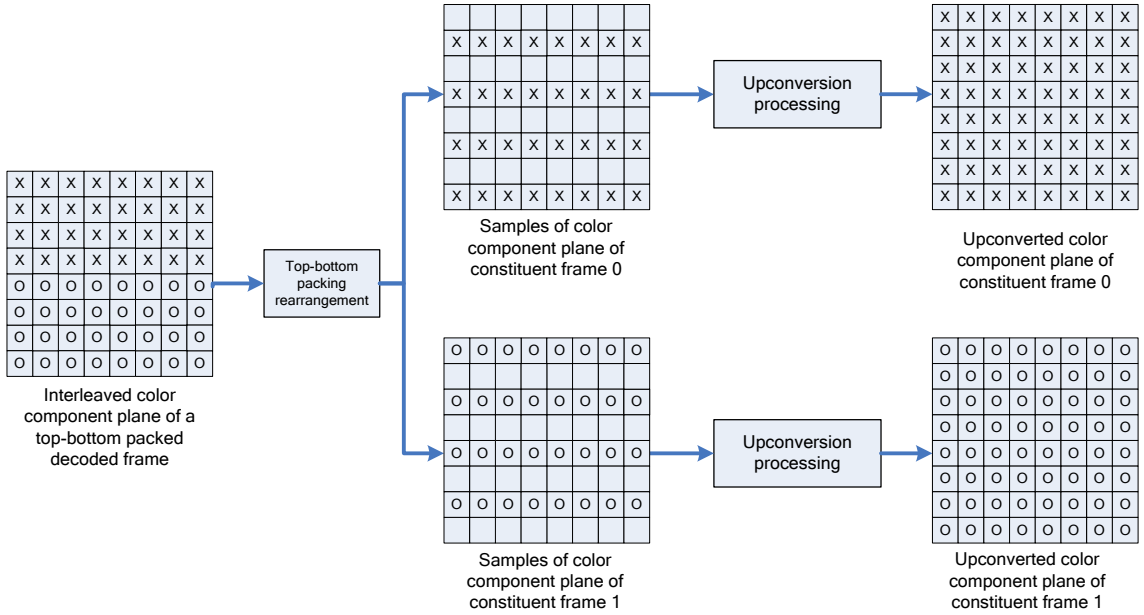


Figure D-9 – Rearrangement and upconversion of top-bottom packing arrangement with frame_packing_arrangement_type equal to 4, quincunx_sampling_flag equal to 0, (x, y) equal to (8, 12) for constituent frame 0, and (x, y) equal to (0, 0) or (8, 4) for constituent frame 1

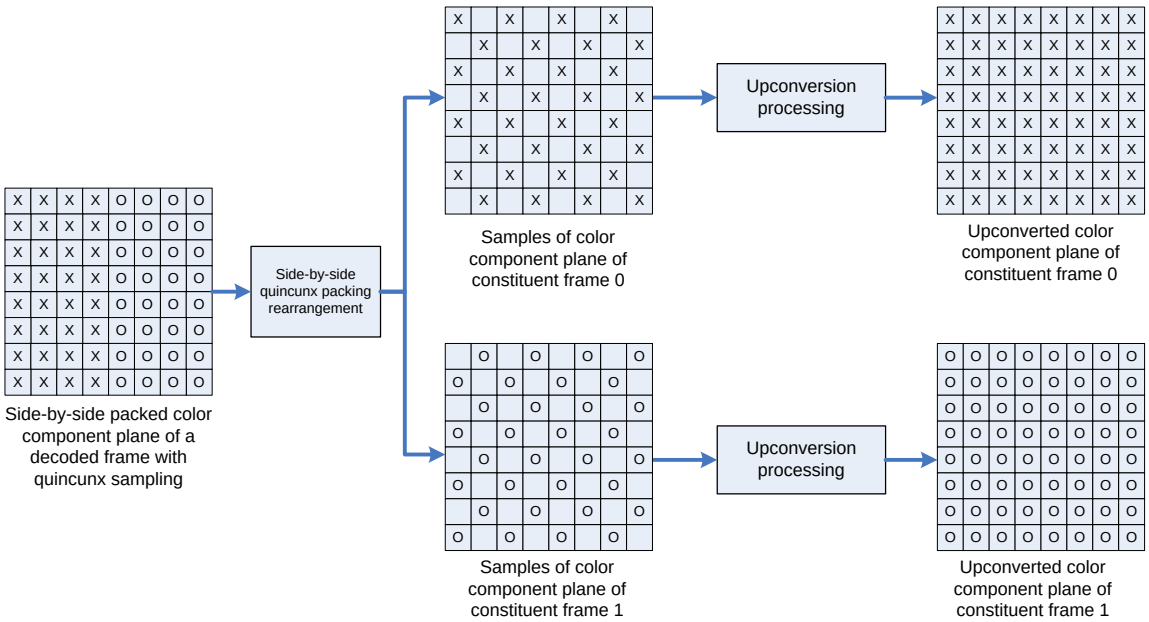


Figure D-10 – Rearrangement and upconversion of side-by-side packing arrangement with quincunx sampling (frame_packing_arrangement_type equal to 3 with quincunx_sampling_flag equal to 1)

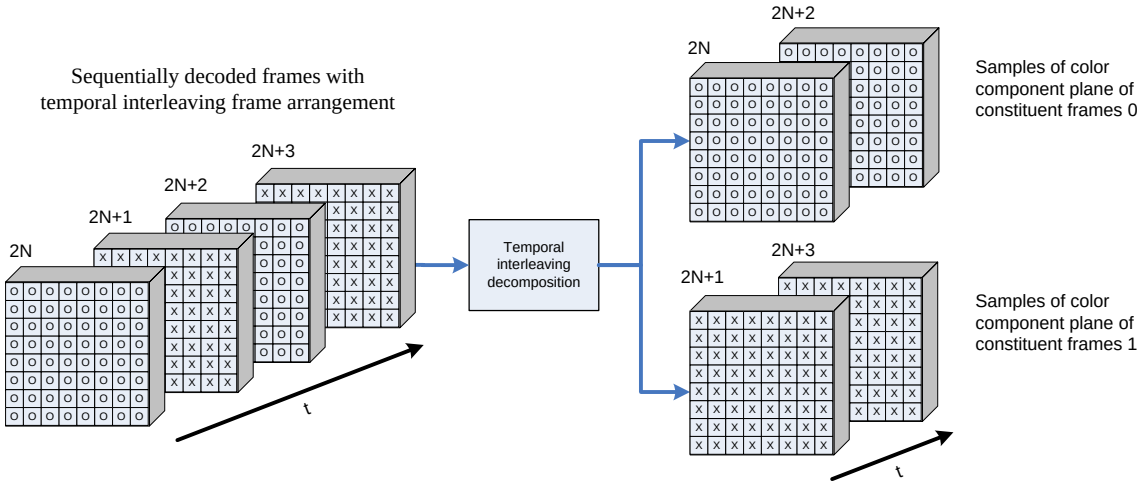


Figure D-11 – Rearrangement of a temporal interleaving frame arrangement (frame_packing_arrangement_type equal to 5)

Figure D-12 – Rearrangement and upconversion of rectangular region frame packing arrangement (frame_packing_arrangement_type equal to 7)

D.2.17 Display orientation SEI message semantics

This SEI message informs the decoder of a transformation that is recommended to be applied to the output decoded and cropped picture prior to display.

When an SEI NAL unit that contains a display orientation SEI message and has nuh_reserved_zero_6bits equal to 0 is present, the SEI NAL unit shall precede, in decoding order, the first VCL NAL unit in the access unit.

display_orientation_cancel_flag equal to 1 indicates that the SEI message cancels the persistence of any previous display orientation SEI message in output order. display_orientation_cancel_flag equal to 0 indicates that display orientation information follows.

hor_flip equal to 1 indicates that the cropped decoded picture should be flipped horizontally for display. hor_flip equal to 0 indicates that the decoded picture should not be flipped horizontally.

When hor_flip is equal to 1, the cropped decoded picture should be flipped as follows for each colour component Z = L, Cb, and Cr, letting dZ be the final cropped array of output samples for the component Z:

```
for( x = 0; x < croppedWidthInSamplesZ; x++ ) [Ed. (GJS): More properly account for cropping.]
    for( y = 0; y < croppedHeightInSamplesZ; y++ )
        dZ[ x ][ y ] = Z[ croppedWidthInSamplesZ − x − 1 ][ y ]
```

ver_flip equal to 1 indicates that the cropped decoded picture should be flipped vertically (in addition to any horizontal flipping when hor_flip is equal to 1) for display. ver_flip equal to 0 indicates that the decoded picture should not be flipped vertically.

When ver_flip is equal to 1, the cropped decoded picture should be flipped as follows for each colour component Z = L, Cb, and Cr, letting dZ be the final cropped array of output samples for the component Z:

```
for( x = 0; x < croppedWidthInSamplesZ; x++ )
    for( y = 0; y < croppedHeightInSamplesZ; y++ )
        dZ[ x ][ y ] = Z[ x ][ croppedWidthInSamplesZ − y − 1 ]
```

anticlockwise_rotation specifies the recommended anticlockwise rotation of the decoded picture (after applying horizontal and/or vertical flipping when hor_flip or ver_flip is set) prior to display. The decoded picture should be rotated by 360 * anticlockwise_rotation ÷ 2^{16} degrees (2 * π * anticlockwise_rotation ÷ 2^{16} radians, where π is Archimedes' Constant (3.141 592 653 589 793 …) in the anticlockwise direction prior to display. For example, anticlockwise_rotation equal to 0 indicates no rotation and anticlockwise_rotation equal to 16 384 indicates 90 degrees (π ÷ 2 radians) rotation in the anticlockwise direction.

NOTE – It is possible for equivalent transformations to be expressed in multiple ways using these syntax elements. For example, the combination of having both hor_flip and ver_flip equal to 1 with anticlockwise_rotation equal to 0 can alternatively be expressed by having both hor_flip and ver_flip equal to 1 with anticlockwise_rotation equal to 0x8000000, and the combination of hor_flip equal to 1 with ver_flip equal to 0 and anticlockwise_rotation equal to 0 can alternatively be expressed by having hor_flip equal to 0 with ver_flip equal to 1 and anticlockwise_rotation equal to 0x8000000.

display_orientation_repetition_period specifies the persistence of the display orientation SEI message and may specify a picture order count interval within which another display orientation SEI message or the end of the coded video sequence shall be present in the bitstream. The value of display_orientation_repetition_period shall be in the range 0 to 16 384, inclusive.

display_orientation_repetition_period equal to 0 specifies that the display orientation SEI message applies to the current decoded picture only.

display_orientation_repetition_period equal to 1 specifies that the display orientation SEI message persists in output order until one or more of the following conditions are true:

– A new coded video sequence begins.

– A picture in an access unit containing a display orientation SEI message is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic).

display_orientation_repetition_period greater than 1 specifies that the display orientation SEI message persists until one or more of the following conditions are true:

– A new coded video sequence begins.

– A picture in an access unit containing a display orientation SEI message is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + display_orientation_repetition_period.

display_orientation_repetition_period greater than 1 indicates that another display orientation SEI message shall be present for a picture in an access unit that is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + display_orientation_repetition_period; unless the bitstream ends or a new coded video sequence begins without output of such a picture. [Ed. (GJS): Check POC variable use.]

display_orientation_extension_flag equal to 0 indicates that no additional data follows within the post-filter hint SEI message. The value of display_orientation_extension_flag shall be equal to 0. The value of 1 for display_orientation_extension_flag is reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore all data that follows the value of 1 for display_orientation_extension_flag in a display orientation SEI message.

D.2.18 SOP description SEI message semantics

The SOP description SEI message indicates constraints that apply in the SOP that starts with the current access unit. [Ed.(GJS): Define SOP – what is it?]

The SOP description SEI message shall not be present in any access unit with TemporalId greater than 0.

When an SEI NAL unit that contains a SOP description SEI message and has nuh_reserved_zero_6bits equal to 0 is present, the SEI NAL unit shall precede, in decoding order, the first VCL NAL unit in the access unit.

sop_seq_parameter_set_id specifies the sps_seq_parameter_set_id value for the sequence parameter set. The value of sop_seq_parameter_set_id shall be in the range of 0 to 15, inclusive.

num_pics_in_sop_minus1 + 1 specifies the number of pictures in the SOP.

sop_desc_nal_ref_flag[i] equal to 0 indicates the i-th picture in decoding order within the SOP has a nal_unit_type equal to TRAIL_N, TSA_N, STSA_N, RADL_N, RASL_N, RSV_VCL_N10, RSV_VCL_N12, or RSV_VCL_N14. [Ed. TK. This was changed since nal_ref_flag has been removed. Perhaps sop_desc_nal_ref_flag[i] can be removed if no longer useful.]

sop_desc_temporal_id[i] specifies the TemporalId value of the i-th picture in decoding order within the SOP.

st_rps_idx[i] specifies the short-term reference picture set included in the sequence parameter set identified by sop_seq_parameter_set_id and used by the i-th picture in decoding order within the SOP.

poc_delta[i] specifies the difference between the picture order count values of the i-th picture in decoding order within the SOP and the (i−1)-th picture in decoding order within the SOP.

The bitstream may not contain all the pictures described in a SOP description. For example, the bitstream may have been subject to TemporalId based sub-bitstream extraction, but pictures that have TemporalId values no longer existing in the extracted bitstream may still be present in the SOP description SEI message. The following constraints specify that the pictures that are present in the bitstream must match to the information given in the SOP description SEI message. [Ed. (GJS): This paragraph seems normative. Improve phrasing. Does it contradict other statements within the annex?]

The variable picOrderCntExp[i] is derived as follows.

```
picOrderCntExp[ 0 ] = PicOrderCntVal of the current picture
for( i = 1; i <= num_pics_in_sop_minus1; i++ )
    picOrderCntExp[ i ] = picOrderCntExp[ i − 1 ] + poc_delta[ i ]
```

The variable nalRefFlag[j] is specified as follows for the j-th picture in decoding order starting from j equal to 0 for the first picture of the SOP containing the picture associated with the SEI message.

– If the j-th picture had nal_unit_type equal to TRAIL_N, TSA_N, STSA_N, RADL_N, RASL_N, RSV_VCL_N10, RSV_VCL_N12, or RSV_VCL_N14, nalRefFlag[j] is set to 0.

– Otherwise, nalRefFlag[j] is set to 1.

Let tId[j], stRpsIdx[j], and picOrderCntVal[j] be the values of TemporalId, st_rps_idx, and PicOrderCntVal that are in effect for the j-th picture in decoding order starting from j equal to 0 for the first picture of the SOP containing the picture associated with the SEI message. Let currSeqParamSet be the previous sequence parameter set RBSP in decoding order with sps_seq_parameter_set_id equal to sop_seq_parameter_set_id. long_term_ref_pics_present_flag shall be equal to 0 in currSeqParamSet. [Ed. (GJS): This seems to introduce some confusion with respect to uses of the same variable names in other places in the text. I suggest using subscripts rather than array symbols here.]

It is a requirement of bitstream conformance that when the SOP description SEI message, the following constraints shall apply for each picture i from picture 0 to picture num_pics_in_sop_minus1 when picOrderCntExp[i] is equal to picOrderCntVal[j], where j is greater than 0 and picOrderCntVal[j − 1] is less than or equal to picOrderCntExp[num_pics_in_sop_minus1]:

– nalRefFlag[j] shall be equal to sop_desc_nal_ref_flag[i].

– tId[j] shall be equal to sop_desc_temporal_id[i].

– stRpsIdx[j] shall be equal to st_rps_idx[i].

– currSeqParamSet shall be the active sequence parameter set RBSP for picture j.

D.2.19 Decoded picture hash SEI message semantics

This message provides a hash for each colour component of the decoded picture in the current access unit.

NOTE 1 – The decoded picture hash SEI message is a suffix SEI message.

Prior to computing the hash, the decoded picture data is arranged into one or three strings of bytes called pictureData[cIdx] of lengths dataLen[cIdx] according to the following pseudocode process:

```
for( cIdx = 0; cIdx < ( chroma_format_idc  = =  0 ) ? 1 : 3; cIdx++ ) {
    if (cIdx  = =  0) {
        compWidth[ cIdx ] = pic_width_in_luma_samples
        compHeight[ cIdx ] = pic_height_in_luma_samples
        compDepth[ cIdx ] = BitDepthY
    } else {
        compWidth[ cIdx ] = pic_width_in_luma_samples / SubWidthC
        compHeight[ cIdx ] = pic_height_in_luma_samples / SubHeightC
        compDepth[ cIdx ] = BitDepthC
    }
    iLen = 0
    for( i = 0; i < compWidth[ cIdx ] * compHeight[ cIdx ]; i++ ) {
        pictureData[ cIdx ][ iLen++ ] = component[ cIdx ][ i ] & 0xFF
        if( compDepth[ cIdx ] > 8 )
            pictureData[ cIdx ][ iLen++ ] = component[ cIdx ][ i ] >> 8
    }
```

```
        dataLen[ cIdx ] = iLen
    }
```

where component[cIdx][i] is an array in raster scan order of decoded sample values in two's complement representation.

hash_type indicates the method used to calculate the checksum according to Table D-10.

Table D-10 – Interpretation of hash_type

hash_type	Method
0	MD5 (RFC 1321)
1	CRC
2	Checksum
3..255	Reserved

[Ed. (GJS): Add normative reference to RFC 1321.]

picture_md5[cIdx **][** i **]** is the 16-byte MD5 hash of the colour component cIdx of the decoded picture. [Ed. (YK): Is colour component cIdx a precise description?] The value of picture_md5[cIdx][i] shall be equal to the value of digestVal[cIdx] obtained by performing the following pseudocode process using the MD5 functions defined in RFC 1321:

```
MD5Init( context )
MD5Update( context, pictureData[ cIdx ], dataLen[ cIdx ] )
MD5Final( digestVal[ cIdx ], context )
```

picture_crc[cIdx **]** is the cyclic redundancy check (CRC) of the colour component cIdx of the decoded picture. The value of picture_crc[cIdx] shall be equal to the value of crcVal[cIdx] obtained by performing the following pseudocode process:

```
crc = 0xFFFF
pictureData[ cIdx ][  dataLen[ cIdx ] ] = 0
pictureData[ cIdx ][  dataLen[ cIdx ] + 1 ] = 0
for( bitIdx = 0; bitIdx < ( dataLen[ cIdx ]  + 2 ) * 8; bitIdx++ ) {
    dataByte = pictureData[ cIdx ][ bitIdx >> 3 ]
    crcMsb = ( crc >> 15 ) & 1
    bitVal = ( dataByte >> (7 − ( bitIdx & 7 ) ) ) & 1
    crc = ( ( ( crc << 1 ) + bitVal ) & 0xFFFF ) ^ ( crcMsb * 0x1021 )
}
crcVal[ cIdx ] = crc
```

NOTE 2 – The same CRC specification is found in Rec. ITU-T H.271.

picture_checksum[cIdx] is the checksum of the colour component cIdx of the decoded picture. The value of picure_checksum[cIdx] shall be equal to the value of checksumVal[cIdx] obtained by performing the following pseudocode process.

```
sum = 0
for( y = 0; y < compHeight[ cIdx ]; y++ )
    for( x = 0; x < compWidth[ cIdx ]; x++ ) {
        xorMask = ( x & 0xFF ) ^ ( y & 0xFF ) ^ ( x >> 8 ) ^ ( y >> 8 )
        sum = ( sum + ( ( component[ cIdx ][ y * compWidth[ cIdx ] + x ] & 0xFF ) ^ xorMask ) )
            & 0xFFFFFFFF
        if( compDepth[ cIdx ] > 8 )
            sum = ( sum + ( ( component[ cIdx ][ y * compWidth[ cIdx ] + x ] >> 8 ) ^ xorMask ) )
                & 0xFFFFFFFF
    }
checksumVal[ cIdx ] = sum
```

D.2.20 Active parameter sets SEI message semantics

The active parameter sets SEI message provides indicates which video parameter set is active for the VCL NAL units of the access unit associated with the SEI message. The SEI message may also provide information on which sequence parameter set is active for the VCL NAL units of the access unit associated with the SEI message, and possibly other information.

When an SEI NAL unit that contains an active parameter sets SEI message and has nuh_reserved_zero_6bits equal to 0 is present, the SEI NAL unit shall precede, in decoding order, the first VCL NAL unit in the access unit.

active_vps_id identifies [Ed. (GJS): How? Clarify what syntax element this is referring to.] the video parameter set that is active for the VCL NAL units of the access unit associated with the SEI message. The value of active_vps_id shall be in the range of 0 to 15, inclusive.

num_sps_ids_minus1 plus 1 specifies the number of sequence parameter sets that are active for the VCL NAL units of the access unit associated with the active parameter sets SEI message. The value of num_sps_ids_minus1 shall be in the range of 0 to 15, inclusive. In bitstreams conforming to this version of this Specification, num_sps_ids_minus1 shall be equal to 0. However, decoders shall allow other values to appear in the num_sps_ids_minus1 syntax.

active_seq_parameter_set_id[i] specifies the sps_seq_parameter_set_id of the i-th sequence parameter set that is active for the VCL NAL units of the access unit associated with the SEI message. [Ed. (GJS): Vague. What does it mean for i equal to 0? What does it mean for i not equal to 0? What does it mean at all?] The value of active_seq_parameter_set_id[i] shall be in the range of 0 to 15, inclusive. [Ed. (GJS): Does that constraint apply to the values for i > 0?]

D.2.21 Decoding unit information SEI message semantics

The decoding unit information SEI message provides CPB removal delay information for the decoding unit associated with the SEI message.

The following applies for the decoding unit information SEI message syntax and semantics:

– The syntax elements sub_pic_cpb_params_present_flag, sub_pic_cpb_params_in_pic_timing_sei_flag, and au_cpb_removal_delay_length_minus1, and the variable CpbDpbDelaysPresentFlag are found in or derived from syntax elements in the hrd_parameters( ) syntax structure that is applicable to at least one of the operation points to which the decoding unit 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 decoding unit information SEI message applies.

The presence of the decoding unit information SEI message in the bitstream is specified as follows.

– If CpbDpbDelaysPresentFlag is equal to 1 and sub_pic_cpb_params_present_flag is equal to 1, one decoding unit information SEI message applicable to the specified operation points may be present in each decoding unit in the coded video sequence.

– Otherwise (CpbDpbDelaysPresentFlag is equal to 0 or sub_pic_cpb_params_present_flag is equal to 0), no access unit in the coded video sequence shall be associated with a decoding unit information SEI message applicable to the specified operation points.

The set of NAL units associated with a decoding unit information SEI message consists, in decoding order, of the SEI NAL unit containing the decoding unit information SEI message and all subsequent NAL units in the access unit up to but not including any subsequent SEI NAL unit containing a decoding unit information SEI message.

decoding_unit_idx specifies the index, starting from 0, to the list of decoding units in the current access unit, of the decoding unit associated with the decoding unit information SEI message. The value of decoding_unit_idx shall be in the range of 0 to PicSizeInCtbsY − 1, inclusive.

A decoding unit identified by a particular value of duIdx includes and only includes all NAL units associated with all decoding unit information SEI messages that have decoding_unit_idx equal to duIdx. Such a decoding unit is also referred to as associated with the decoding unit information SEI messages having decoding_unit_idx equal to duIdx.

For any two decoding units duA and duB in one access unit 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 decoding unit shall not reside, in decoding order, between any two NAL units of another decoding unit.

du_spt_cpb_removal_delay specifies the duration, in units of sub-picture clock ticks, between removal from the CPB of the last decoding unit in decoding order in the current access unit and the decoding unit associated with the decoding unit information SEI message. This value is also used to calculate an earliest possible time of arrival of decoding unit data

into the CPB for the HSS, as specified in Annex C. The syntax element is represented by a fixed length code whose length in bits is given by du_cpb_removal_delay_length_minus1 + 1. When the decoding unit associated with the decoding unit information SEI message is the last decoding unit in the current access unit, the value of du_spt_cpb_removal_delay shall be equal to 0.

D.2.22 Temporal level zero index SEI message semantics

The temporal level zero index SEI message can assist the decoder for detecting when coded pictures with TemporalId equal to 0 are missing.

When a temporal level zero index SEI message is present for the current access unit and the current picture has RapPicFlag equal to 0, a temporal level zero index SEI message shall also be present for the preceding access unit in decoding order with TemporalId equal to 0:

tl0_idx is a temporal level zero index as follows:

– If TemporalId is equal to 0, tl0_idx indicates the temporal level zero index for the current access unit.

– Otherwise, tl0_idx indicates the temporal level zero index of the preceding access unit in decoding order with TemporalId equal to 0.

The variable picForReference is specified as follows:

– If TemporalId is equal to sps_max_sub_layers_minus1 and nal_unit_type is equal to TRAIL_N, TSA_N, STSA_N, RADL_N, RASL_N, RSV_VCL_N10, RSV_VCL_N12 or RSV_VCL_N14, picForReference is equal to 0.

– Otherwise, picForReference is equal to 1.

[Ed. (JB): May want to define this variable earlier and make it global, so that it can also be used for the SOP description SEI message.]

When the bitstream contains a preceding access unit in decoding order that has TemporalId equal to 0 and the preceding access unit that has TemporalId equal to 0 has an associated temporal level 0 index SEI message, the variable prevTL0Idx is set to the value of tl0_idx that is associated with the preceding access unit in decoding order that has TemporalId equal to 0.

The following constraints apply to the value of tl0_idx:

– If the current access unit contains a NAL unit with RapPicFlag equal to 1, tl0_idx shall be equal to 0.

– Otherwise, the following applies:

  – If the TemporalId of the current access unit is equal to 0 and picForReference is equal to 1, tl0_idx shall be equal to (prevTL0Idx + 1) % 65536.

  – Otherwise, tl0_idx shall be equal to prevTL0Idx.

rap_idx is the RAP index for the current access unit. When the bitstream contains a preceding access unit in decoding order with RapPicFlag equal to 1 and the preceding access unit in decoding order with RapPicFlag equal to 1 has an associated temporal level 0 index SEI message, the following constraints apply to the value of rap_idx:

– If the current access unit has RapPicFlag equal to 1, rap_idx shall differ in value from the value of rap_idx of the temporal level zero index SEI message of the preceding access unit in decoding order with RapPicFlag equal to 1.

– Otherwise, rap_idx shall be equal to the value of rap_idx of the temporal level zero index SEI message of the preceding access unit in decoding order with RapPicFlag equal to 1.

D.2.23 Scalable nesting SEI message semantics

The scalable nesting SEI message provides a mechanism to associate SEI messages with bitstream subsets corresponding to various operation points or with specific layers or sub-layers.

A scalable nesting SEI message contains one or more SEI messages. An SEI message contained in a scalable nesting SEI message is referred to as a nested SEI message. An SEI message not contained in a scalable nesting SEI message is referred to as a non-nested SEI message.

A buffering period SEI message and an SEI message of any other type shall not be nested in the same scalable nesting SEI message. A picture timing SEI message and an SEI message of any other type shall not be nested in the same scalable nesting SEI message.

bitstream_subset_flag equal to 0 specifies that the nested SEI messages apply to specific layers or sub-layers. bitstream_subset_flag equal to 1 specifies that the nested SEI messages apply to one or more sub-bitstreams resulting from a sub-bitstream extraction process of subclause 10.1 with inputs specified by the syntax elements of the scalable nesting SEI message as specified below.

When the nested SEI messages are picture buffering SEI messages, picture timing SEI messages or decoding unit information SEI messages, bitstream_subset_flag shall be equal to 1.

Depending on the value of bitstream_subset_flag, the layers or sub-layers, or the operation points to which the nested SEI messages apply are specified by deriving the sets nestingLayedIdSet[i] and the variables maxTemporalId[i] from syntax element values as specified below.

nesting_op_flag equal to 0 specifies that the set nestingLayerIdSet[0] is specified by all_layers_flag and, when present, nesting_layer_id[i] for all i values in the range of 0 to nesting_num_layers_minus1, inclusive, and that the variable maxTemporalId[0] is specified by nesting_no_op_max_temporal_id_plus1. nesting_op_flag equal to 1 specifies that the set nestingLayerIdSet[i] and the variable maxTemporalId[i] are specified by nesting_num_ops_minus1, default_op_flag, nesting_max_temporal_id_plus1[i], when present, and nesting_op_idx[i], when present.

default_op_flag equal to 1 specifies that maxTemporalId[0] is equal to nuh_temporal_id_plus1 of the current SEI NAL unit minus 1 and that nestingLayerIdSet[0] contains all integer values in the range of 0 to nuh_reserved_zero_6bits of the current SEI NAL unit, inclusive.

nesting_num_ops_minus1 plus 1 minus default_op_flag specifies the number of the following nesting_op_idx[i] syntax elements. The value of nesting_num_ops_minus1 shall be in the range of 0 to 1023, inclusive.

If nesting_op_flag is equal to 0, the variable nestingNumOps is set equal to 1; otherwise, the variable nestingNumOps is set equal to nesting_num_ops_minus1 + 1.

nesting_max_temporal_id_plus1[i] is used to specify the variable maxTemporalId[i]. The value of nesting_max_temporal_id_plus1[i] shall be greater than or equal to nuh_temporal_id_plus1 of the current SEI NAL unit. The variable maxTemporalId[i] is set equal to nesting_max_temporal_id_plus1[i] − 1.

nesting_op_idx[i] is used to specify the set nestingLayerIdSet[i]. The value of nesting_ ops_idx[i] shall be in the range of 0 to 1023, inclusive.

The set nestingLayerIdSet[i] is set equal to the OpLayerIdSet of the nesting_op_idx[i]-th operation point set specified by the active video parameter set.

all_layers_flag equal to 0 specifies that the set nestingLayerIdSet[0] is specified by nesting_layer_id[i] for all i values in the range of 0 to nesting_num_layers_minus1, inclusive. all_layers_flag equal to 1 specifies that the set nestingLayerIdSet[0] consists of all values of nuh_reserved_zero_6bits present in the current access unit that are greater than or equal to nuh_reserved_zero_6bits of the current SEI NAL unit.

nesting_no_op_max_temporal_id_plus1 minus 1 specifies the value of maxTemporalId[0] when nesting_op_flag is equal to 0 and all_layers_flag is equal to 0. The value of nesting_no_op_max_temporal_id_plus1 shall not be equal to 0.

nesting_num_layers_minus1 plus 1 specifies the number of the following nesting_layer_id[i] syntax elements. The value of nesting_num_layers_minus1 shall be in the range of 0 to 63, inclusive.

nesting_layer_id[i] specifies the i-th nuh_reserved_zero_6bits value included in the set nestingLayerIdSet[0]. The set nestingLayerIdSet[0] is set to consist of nesting_layer_id[i] for all i values in the range of 0 to nesting_num_layers_minus1, inclusive.

When bitstream_subset_flag is equal to 0, the nested SEI messages apply to the sets of layers or sub-layers subLayerSet[i] for all i values in the range of 0 to nestingNumOps – 1, where the VCL NAL units of the layers or sub-layers in each set subLayerSet[i] are with nuh_reserved_zero_6bits included in the set nestingLayerIdSet[i] and with nuh_temporal_id_plus1 in the range of nuh_temporal_id_plus1 of the current SEI NAL unit to maxTemporalId[i] + 1, inclusive.

When bitstream_subset_flag is equal to 1, the nested SEI messages apply to sub-bitstreams subBitstream[i] for all i values in the range of 0 to nestingNumOps − 1, inclusive, where each sub-bitstream subBitstream[i] is the output of the sub-bitstream extraction process of subclause 10.1 with the bitstream, HighestTid, and targetDecLayerIdSet as inputs.

nesting_zero_bit shall be equal to 0.

D.2.24 Region refresh information SEI message semantics

The region refresh information SEI message indicates whether the slice segments that the current SEI message applies to belong to a refreshed region of the current picture (as defined below).

An access unit that is not a RAP access unit and that contains a recovery point SEI message is referred to as a gradual decoding refresh (GDR) access unit, and its corresponding picture is referred to as a GDR picture. The access unit corresponding to the the indicated recovery point is referred to as the recovery point access unit and the corresponding picture is referred to as the recovery point picture.

Let gdrPicSet be the set of pictures starting from a GDR picture to the recovery point picture, inclusive, in decoding order. When the decoding process is started from a GDR access unit, the refreshed region in each picture of the gdrPicSet is indicated to be the region of the picture that is correct or approximately correct in content, and the refreshed region in the recovery point picture covers the entire picture.

The slice segments to which a region refresh information SEI message applies consists of all slice segments within the access unit that follow the SEI NAL unit containing the region refresh information SEI message and precede the next the SEI NAL unit containing a region refresh information SEI message (if any) in decoding order. These slice segments are referred to as the slice segments associated with the region refresh information SEI message.

Let gdrAuSet be the set of access units corresponding to gdrPicSet. A gdrAuSet and the corresponding gdrPicSet are referred to as being associated with the recovery point SEI message contained in the GDR access unit.

Region refresh information SEI messages shall not be present in an access unit unless the access unit is included in a gdrAuSet associated with a recovery point SEI message. When any access unit that is included in a gdrAuSet contains one or more region refresh information SEI messages, all access units in the gdrAuSet except for the recovery point access unit shall contain one or more region refresh information SEI messages.

refreshed_region_flag equal to 1 indicates that the slice segments associated with the current SEI message belong to the refreshed region in the current picture. refreshed_region_flag equal to 0 indicates that the slice segments associated with the current SEI message may not belong to the refreshed region in the current picture.

When one or more region refresh information SEI messages are present in an access unit and the first slice segment of the access unit in decoding order does not have an associated region refresh information SEI message, the value of refreshed_region_flag for the slice segments that precede the first region refresh information SEI message is inferred to be equal to 0.

When any region refresh SEI message is included in a recovery point access unit, the first slice segment of the access unit in decoding order shall have an associated region refresh SEI message, and the value of refreshed_region_flag shall be equal to 1 in all region refresh SEI messages in the access unit.

When one or more region refresh information SEI messages are present in an access unit, the refreshed region in a picture is specified as the set of CTUs in all slice segments of the access unit that are associated with region refresh information SEI messages that have refreshed_region_flag equal to 1. Other slice segments belong to the non-refreshed region of the picture.

It is a requirement of bitstream conformance that when a dependent slice segment belongs to the refreshed region, the preceding slice segment in decoding order shall also belong to the refreshed region.

Let gdrRefreshedSliceSegmentSet be the set of all slice segments that belong to the refreshed regions in the gdrPicSet. When a gdrAuSet contains one or more region refresh information SEI messages, it is a requirement of bitstream conformance that the following constraints all apply:

– The refreshed region in the first picture included in the corresponding gdrPicSet in decoding order that contains any refreshed region shall contain only coding units that are coded in an intra coding mode.

– For each picture included in the gdrPicSet, the syntax elements in gdrRefreshedSliceSegmentSet shall be constrained such that no samples or motion vector values outside of gdrRefreshedSliceSet are used for inter prediction in the decoding process of any samples within gdrRefreshedSliceSegmentSet.

– For any picture that follows the recovery point picture in output order, the syntax elements in the slice segments of the picture shall be constrained such that no samples or motion vector values outside of gdrRefreshedSliceSet are used for inter prediction in the decoding process of the picture other than those of the recovery point picture or other pictures that follow the recovery point picture in both decoding order and output order.

D.2.25 Reserved SEI message semantics

The reserved SEI message consists of data reserved for future backward-compatible use by ITU-T | ISO/IEC. It is a requirement of bitstream conformance that bitstreams shall not contain reserved SEI messages until and unless the use of such messages has been specified by ITU-T | ISO/IEC. Decoders that encounter reserved SEI messages shall discard their content without effect on the decoding process, except as specified in the future by ITU-T | ISO/IEC.

Annex E

Video usability information

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies syntax and semantics of the VUI parameters of the sequence parameter sets.

VUI parameters are not required for constructing the luma or chroma samples by the decoding process. Conforming decoders are not required to process this information for output order conformance to this Specification (see Annex C for the specification of output order conformance). Some VUI parameters are required to check bitstream conformance and for output timing decoder conformance.

In Annex E, specification for presence of VUI parameters is also satisfied when those parameters (or some subset of them) are conveyed to decoders (or to the HRD) by other means not specified by this Specification. When present in the bitstream, VUI parameters shall follow the syntax and semantics specified in this annex. When the content of VUI parameters is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the VUI parameters is not required to use the same syntax specified in this annex. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

E.1 VUI syntax

E.1.1 VUI parameters syntax

vui_parameters( ) {	**Descriptor**
aspect_ratio_info_present_flag	u(1)
if(aspect_ratio_info_present_flag) {	
aspect_ratio_idc	u(8)
if(aspect_ratio_idc = = Extended_SAR) {	
sar_width	u(16)
sar_height	u(16)
}	
}	
overscan_info_present_flag	u(1)
if(overscan_info_present_flag)	
overscan_appropriate_flag	u(1)
video_signal_type_present_flag	u(1)
if(video_signal_type_present_flag) {	
video_format	u(3)
video_full_range_flag	u(1)
colour_description_present_flag	u(1)
if(colour_description_present_flag) {	
colour_primaries	u(8)
transfer_characteristics	u(8)
matrix_coefficients	u(8)
}	
}	
chroma_loc_info_present_flag	u(1)
if(chroma_loc_info_present_flag) {	
chroma_sample_loc_type_top_field	ue(v)
chroma_sample_loc_type_bottom_field	ue(v)
}	
neutral_chroma_indication_flag	u(1)
field_seq_flag	u(1)
frame_field_info_present_flag	u(1)
default_display_window_flag	u(1)
if (default_display_window_flag) {	
def_disp_win_left_offset	ue(v)
def_disp_win_right_offset	ue(v)
def_disp_win_top_offset	ue(v)
def_disp_win_bottom_offset	ue(v)
}	
hrd_parameters_present_flag	u(1)
if(hrd_parameters_present_flag) [Ed. (GJS): Syntax element naming convention violation.]	
hrd_parameters(1, sps_max_sub_layers_minus1) [Ed. (BB): Syntax element naming convention violation.]	
poc_proportional_to_timing_flag	u(1)
if(poc_proportional_to_timing_flag && timing_info_present_flag)	
num_ticks_poc_diff_one_minus1	ue(v)

bitstream_restriction_flag	u(1)
if(bitstream_restriction_flag) {	
tiles_fixed_structure_flag	u(1)
motion_vectors_over_pic_boundaries_flag	u(1)
restricted_ref_pic_lists_flag	u(1)
min_spatial_segmentation_idc [Ed. (GJS): It seems inconsistent to use a u(8) here rather than ue(v) like what was done for other similar VUI parameters. Also, u(8) limits the degree of parallelism that can be indicated to only 64 threads. A limit more like 0..4095 would seem more appropriate.]	u(8)
max_bytes_per_pic_denom	ue(v)
max_bits_per_mincu_denom	ue(v)
log2_max_mv_length_horizontal	ue(v)
log2_max_mv_length_vertical	ue(v)
}	
}	

E.1.2 HRD parameters syntax

hrd_parameters(commonInfPresentFlag, maxNumSubLayersMinus1) {	**Descriptor**
if(commonInfPresentFlag) {	
timing_info_present_flag	u(1)
if(timing_info_present_flag) {	
num_units_in_tick	u(32)
time_scale	u(32)
}	
nal_hrd_parameters_present_flag	u(1)
vcl_hrd_parameters_present_flag	u(1)
if(nal_hrd_parameters_present_flag | | vcl_hrd_parameters_present_flag){	
sub_pic_cpb_params_present_flag	u(1)
if(sub_pic_cpb_params_present_flag) {	
tick_divisor_minus2	u(8)
du_cpb_removal_delay_length_minus1	u(5)
sub_pic_cpb_params_in_pic_timing_sei_flag	u(1)
}	
bit_rate_scale	u(4)
cpb_size_scale	u(4)
if(sub_pic_cpb_params_present_flag)	
cpb_size_du_scale	u(4)
initial_cpb_removal_delay_length_minus1	u(5)
au_cpb_removal_delay_length_minus1	u(5)
dpb_output_delay_length_minus1	u(5)
}	
}	
for(i = 0; i <= maxNumSubLayersMinus1; 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)
low_delay_hrd_flag[i]	u(1)
cpb_cnt_minus1[i]	ue(v)
if(nal_hrd_parameters_present_flag)	
sub_layer_hrd_parameters(i)	
if(vcl_hrd_parameters_present_flag)	
sub_layer_hrd_parameters(i)	
}	
}	

E.1.3 Sub-layer HRD parameters syntax

sub_layer_hrd_parameters(tId) {	**Descriptor**
for(i = 0; i <= CpbCnt; i++) {	
bit_rate_value_minus1[i]	ue(v)
cpb_size_value_minus1[i]	ue(v)
if(sub_pic_cpb_params_present_flag)	
cpb_size_du_value_minus1[i]	ue(v)
cbr_flag[i]	u(1)
}	
}	

E.2 VUI semantics

E.2.1 VUI parameters semantics

aspect_ratio_info_present_flag equal to 1 specifies that aspect_ratio_idc is present. aspect_ratio_info_present_flag equal to 0 specifies that aspect_ratio_idc is not present.

aspect_ratio_idc specifies the value of the sample aspect ratio of the luma samples. Table E-1 shows the meaning of the code. When aspect_ratio_idc indicates Extended_SAR, the sample aspect ratio is represented by sar_width : sar_height. When the aspect_ratio_idc syntax element is not present, aspect_ratio_idc value is inferred to be equal to 0.

Table E-1 – Interpretation of sample aspect ratio indicator

aspect_ratio_idc	Sample aspect ratio	(informative) Examples of use
0	Unspecified	
1	1:1 ("square")	7680x4320 16:9 frame without horizontal overscan 3840x2160 16:9 frame without horizontal overscan 1280x720 16:9 frame without horizontal overscan 1920x1080 16:9 frame without horizontal overscan (cropped from 1920x1088) 640x480 4:3 frame without horizontal overscan
2	12:11	720x576 4:3 frame with horizontal overscan 352x288 4:3 frame without horizontal overscan
3	10:11	720x480 4:3 frame with horizontal overscan 352x240 4:3 frame without horizontal overscan
4	16:11	720x576 16:9 frame with horizontal overscan 528x576 4:3 frame without horizontal overscan
5	40:33	720x480 16:9 frame with horizontal overscan 528x480 4:3 frame without horizontal overscan
6	24:11	352x576 4:3 frame without horizontal overscan 480x576 16:9 frame with horizontal overscan
7	20:11	352x480 4:3 frame without horizontal overscan 480x480 16:9 frame with horizontal overscan
8	32:11	352x576 16:9 frame without horizontal overscan
9	80:33	352x480 16:9 frame without horizontal overscan
10	18:11	480x576 4:3 frame with horizontal overscan
11	15:11	480x480 4:3 frame with horizontal overscan
12	64:33	528x576 16:9 frame without horizontal overscan
13	160:99	528x480 16:9 frame without horizontal overscan
14	4:3	1440x1080 16:9 frame without horizontal overscan
15	3:2	1280x1080 16:9 frame without horizontal overscan
16	2:1	960x1080 16:9 frame without horizontal overscan
17..254	Reserved	
255	Extended_SAR	

NOTE 1 – For the examples in Table E-1, the term "without horizontal overscan" refers to display processes in which the display area matches the area of the cropped decoded pictures and the term "with horizontal overscan" refers to display processes in which some parts near the left and/or right border of the cropped decoded pictures are not visible in the display area. As an example, the entry "720x576 4:3 frame with horizontal overscan" for aspect_ratio_idc equal to 2 refers to having an area of 704x576 luma samples (which has an aspect ratio of 4:3) of the cropped decoded frame (720x576 luma samples) that is visible in the display area.

sar_width indicates the horizontal size of the sample aspect ratio (in arbitrary units).

sar_height indicates the vertical size of the sample aspect ratio (in the same arbitrary units as sar_width).

sar_width and sar_height shall be relatively prime or equal to 0. When aspect_ratio_idc is equal to 0 or sar_width is equal to 0 or sar_height is equal to 0, the sample aspect ratio is unspecified by this Specification.

overscan_info_present_flag equal to 1 specifies that the overscan_appropriate_flag is present. When overscan_info_present_flag is equal to 0 or is not present, the preferred display method for the video signal is unspecified.

overscan_appropriate_flag equal to 1 indicates that the cropped decoded pictures output are suitable for display using overscan. overscan_appropriate_flag equal to 0 indicates that the cropped decoded pictures output contain visually important information in the entire region out to the edges of the cropping rectangle of the picture, such that the cropped decoded pictures output should not be displayed using overscan. Instead, they should be displayed using either an exact match between the display area and the cropping rectangle, or using underscan. As used in this paragraph, the term "overscan" refers to display processes in which some parts near the borders of the cropped decoded pictures are not visible in the display area. The term "underscan" describes display processes in which the entire cropped decoded

pictures are visible in the display area, but they do not cover the entire display area. For display processes that neither use overscan nor underscan, the display area exactly matches the area of the cropped decoded pictures.

NOTE 2 – For example, overscan_appropriate_flag equal to 1 might be used for entertainment television programming, or for a live view of people in a videoconference, and overscan_appropriate_flag equal to 0 might be used for computer screen capture or security camera content.

video_signal_type_present_flag equal to 1 specifies that video_format, video_full_range_flag and colour_description_present_flag are present. video_signal_type_present_flag equal to 0, specify that video_format, video_full_range_flag and colour_description_present_flag are not present.

video_format indicates the representation of the pictures as specified in Table E-2, before being coded in accordance with this Specification. When the video_format syntax element is not present, video_format value is inferred to be equal to 5.

Table E-2 – Meaning of video_format

video_format	Meaning
0	Component
1	PAL
2	NTSC
3	SECAM
4	MAC
5	Unspecified video format
6	Reserved
7	Reserved

video_full_range_flag indicates the black level and range of the luma and chroma signals as derived from E'_Y, E'_{PB}, and E'_{PR} or E'_R, E'_G, and E'_B real-valued component signals.

When the video_full_range_flag syntax element is not present, the value of video_full_range_flag is inferred to be equal to 0.

colour_description_present_flag equal to 1 specifies that colour_primaries, transfer_characteristics and matrix_coefficients are present. colour_description_present_flag equal to 0 specifies that colour_primaries, transfer_characteristics and matrix_coefficients are not present.

colour_primaries indicates the chromaticity coordinates of the source primaries as specified in Table E-3 in terms of the CIE 1931 definition of x and y as specified by ISO 11664-1.

When the colour_primaries syntax element is not present, the value of colour_primaries is inferred to be equal to 2 (the chromaticity is unspecified or is determined by the application).

Table E-3 – Colour primaries

Value	Primaries			Informative Remark
0	Reserved			For future use by ITU-T | ISO/IEC
1	primary green blue red white D65	x 0.300 0.150 0.640 0.3127	y 0.600 0.060 0.330 0.3290	Rec. ITU-R BT.709-5 Rec. ITU-R BT.1361 conventional colour gamut system and extended colour gamut system IEC 61966-2-1 (sRGB or sYCC) IEC 61966-2-4 Society of Motion Picture and Television Engineers RP 177 (1993) Annex B
2	Unspecified			Image characteristics are unknown or are determined by the application.
3	Reserved			For future use by ITU-T | ISO/IEC
4	primary green blue red white C	x 0.21 0.14 0.67 0.310	y 0.71 0.08 0.33 0.316	Rec. ITU-R BT.470-6 System M (historical) United States National Television System Committee 1953 Recommendation for transmission standards for colour television United States Federal Communications Commission Title 47 Code of Federal Regulations (2003) 73.682 (a) (20)
5	primary green blue red white D65	x 0.29 0.15 0.64 0.3127	y 0.60 0.06 0.33 0.3290	Rec. ITU-R BT.470-6 System B, G (historical) Rec. ITU-R BT.601-6 625 Rec. ITU-R BT.1358 625 Rec. ITU-R BT.1700 625 PAL and 625 SECAM
6	primary green blue red white D65	x 0.310 0.155 0.630 0.3127	y 0.595 0.070 0.340 0.3290	Rec. ITU-R BT.601-6 525 Rec. ITU-R BT.1358 525 Rec. ITU-R BT.1700 NTSC Society of Motion Picture and Television Engineers 170M (2004) (functionally the same as the value 7)
7	primary green blue red white D65	x 0.310 0.155 0.630 0.3127	y 0.595 0.070 0.340 0.3290	Society of Motion Picture and Television Engineers 240M (1999) (functionally the same as the value 6)
8	primary green blue red white C	x 0.243 0.145 0.681 0.310	y 0.692 (Wratten 58) 0.049 (Wratten 47) 0.319 (Wratten 25) 0.316	Generic film (colour filters using Illuminant C)
9	primary green blue red white D65	x 0.170 0.131 0.708 0.3127	y 0.797 0.046 0.292 0.3290	Rec. ITU-R BT.2020
10..255	Reserved			For future use by ITU-T | ISO/IEC

transfer_characteristics indicates the opto-electronic transfer characteristic of the source picture as specified in Table E-4 as a function of a linear optical intensity input Lc with a nominal real-valued range of 0 to 1.

When the transfer_characteristics syntax element is not present, the value of transfer_characteristics is inferred to be equal to 2 (the transfer characteristics are unspecified or are determined by the application).

Table E-4 – Transfer characteristics

Value	Transfer Characteristic		Informative Remark
0	Reserved		For future use by ITU-T | ISO/IEC
1	$V = 1.099 * L_c^{0.45} - 0.099$ $V = 4.500 * L_c$	for $1 >= L_c >= 0.018$ for $0.018 > L_c >= 0$	Rec. ITU-R BT.709-5 Rec. ITU-R BT.1361 conventional colour gamut system (functionally the same as the value 6)
2	Unspecified		Image characteristics are unknown or are determined by the application.
3	Reserved		For future use by ITU-T | ISO/IEC
4	Assumed display gamma 2.2		Rec. ITU-R BT.470-6 System M (historical) United States National Television System Committee 1953 Recommendation for transmission standards for colour television United States Federal Communications Commission Title 47 Code of Federal Regulations (2003) 73.682 (a) (20) Rec. ITU-R BT.1700 (2007 revision) 625 PAL and 625 SECAM
5	Assumed display gamma 2.8		Rec. ITU-R BT.470-6 System B, G (historical)
6	$V = 1.099 * L_c^{0.45} - 0.099$ $V = 4.500 * L_c$	for $1 >= L_c >= 0.018$ for $0.018 > L_c >= 0$	Rec. ITU-R BT.601-6 525 or 625 Rec. ITU-R BT.1358 525 or 625 Rec. ITU-R BT.1700 NTSC Society of Motion Picture and Television Engineers 170M (2004) (functionally the same as the value 1)
7	$V = 1.1115 * L_c^{0.45} - 0.1115$ $V = 4.0 * L_c$	for $1 >= L_c >= 0.0228$ for $0.0228 > L_c >= 0$	Society of Motion Picture and Television Engineers 240M (1999)
8	$V = L_c$	for $1 > L_c >= 0$	Linear transfer characteristics
9	$V = 1.0 + \text{Log10}(L_c) \div 2$ $V = 0.0$	for $1 >= L_c >= 0.01$ for $0.01 > L_c >= 0$	Logarithmic transfer characteristic (100:1 range)
10	$V = 1.0 + \text{Log10}(L_c) \div 2.5$ $V = 0.0$	for $1 >= L_c >= \text{Sqrt}(10) \div 1000$ for $\text{Sqrt}(10) \div 1000 > L_c >= 0$	Logarithmic transfer characteristic (100 * Sqrt(10) : 1 range)
11	$V = 1.099 * L_c^{0.45} - 0.099$ $V = 4.500 * L_c$ $V = -1.099 * (-L_c)^{0.45} + 0.099$	for $L_c >= 0.018$ for $0.018 > L_c > -0.018$ for $-0.018 >= L_c$	IEC 61966-2-4
12	$V = 1.099 * L_c^{0.45} - 0.099$ $V = 4.500 * L_c$ $V = -(1.099 * (-4 * L_c)^{0.45} - 0.099) \div 4$	for $1.33 > L_c >= 0.018$ for $0.018 > L_c >= -0.0045$ for $-0.0045 > L_c >= -0.25$	Rec. ITU-R BT.1361 extended colour gamut system
13	$V = 1.055 * L_c^{(1 \div 2.4)} - 0.055$ $V = 12.92 * L_c$	for $1 >= L_c >= 0.0031308$ for $0.0031308 > L_c >= 0$	IEC 61966-2-1 (sRGB or sYCC)
14	$V = 1.099 * L_c^{0.45} - 0.099$ $V = 4.500 * L_c$	for $1 >= L_c >= 0.018$ for $0.018 > L_c >= 0$	Rec. ITU-R BT.2020 for 10 bit system
15	$V = 1.0993 * L_c^{0.45} - 0.0993$ $V = 4.500 * L_c$	for $1 >= L_c >= 0.0181$ for $0.0181 > L_c >= 0$	Rec. ITU-R BT.2020 for 12 bit system
15..255	Reserved		For future use by ITU-T | ISO/IEC

matrix_coefficients describes the matrix coefficients used in deriving luma and chroma signals from the green, blue, and red primaries, as specified in Table E-5.

matrix_coefficients shall not be equal to 0 unless one or more of the following conditions are true:

– BitDepth_C is equal to BitDepth_Y,

– chroma_format_idc is equal to 3 (4:4:4).

The specification of the use of matrix_coefficients equal to 0 under all other conditions is reserved for future use by ITU-T | ISO/IEC.

matrix_coefficients shall not be equal to 8 unless one of the following conditions is true:

– $BitDepth_C$ is equal to $BitDepth_Y$,

– $BitDepth_C$ is equal to $BitDepth_Y + 1$ and chroma_format_idc is equal to 3 (4:4:4).

The specification of the use of matrix_coefficients equal to 8 under all other conditions is reserved for future use by ITU-T | ISO/IEC.

When the matrix_coefficients syntax element is not present, the value of matrix_coefficients is inferred to be equal to 2 (unspecified).

The interpretation of matrix_coefficients, together with colour_primaries and transfer_characteristics, is specified by the following equations.

E_R, E_G, and E_B are defined as "linear-domain" real-valued signals based on the indicated colour primaries before application of the transfer characteristics function. The application of the transfer characteristics function is denoted by $(x)'$ for an argument x. The signals E'_R, E'_G, and E'_B are determined by application of the transfer characteristics function as follows:

$$E'_R = (E_R)' \tag{E-1}$$

$$E'_G = (E_G)' \tag{E-2}$$

$$E'_B = (E_B)' \tag{E-3}$$

The range of E'_R, E'_G, and E'_B is specified as follows:

– If transfer_characteristics is not equal to 11 or 12, E'_R, E'_G, and E'_B are real numbers with values in the range of 0 to 1.

– Otherwise, (transfer_characteristics is equal to 11 (IEC 61966-2-4) or 12 (Rec. ITU-R BT.1361 extended colour gamut system)), E'_R, E'_G and E'_B are real numbers with a larger range not specified in this Specification.

Nominal white is specified as having E'_R equal to 1, E'_G equal to 1, and E'_B equal to 1.

Nominal black is specified as having E'_R equal to 0, E'_G equal to 0, and E'_B equal to 0.

The interpretation of matrix_coefficients is specified as follows:

– If video_full_range_flag is equal to 0, the following applies:
  – If matrix_coefficients is equal to 1, 4, 5, 6, 7, 9, or 10, the following equations apply:

$$Y = Clip1_Y(Round((1 << (BitDepth_Y - 8)) * (219 * E'_Y + 16))) \tag{E-4}$$

$$Cb = Clip1_C(Round((1 << (BitDepth_C - 8)) * (224 * E'_{PB} + 128))) \tag{E-5}$$

$$Cr = Clip1_C(Round((1 << (BitDepth_C - 8)) * (224 * E'_{PR} + 128))) \tag{E-6}$$

  – Otherwise, if matrix_coefficients is equal to 0 or 8, the following equations apply:

$$R = Clip1_Y((1 << (BitDepth_Y - 8)) * (219 * E'_R + 16)) \tag{E-7}$$

$$G = Clip1_Y((1 << (BitDepth_Y - 8)) * (219 * E'_G + 16)) \tag{E-8}$$

$$B = Clip1_Y((1 << (BitDepth_Y - 8)) * (219 * E'_B + 16)) \tag{E-9}$$

  – Otherwise, if matrix_coefficients is equal to 2, the interpretation of the matrix_coefficients syntax element is unknown or is determined by the application.
  – Otherwise (matrix_coefficients is not equal to 0, 1, 2, 4, 5, 6, 7, 8, 9, or 10), the interpretation of the matrix_coefficients syntax element is reserved for future definition by ITU-T | ISO/IEC.
– Otherwise (video_full_range_flag is equal to 1), the following applies:
  – If matrix_coefficients is equal to 1, 4, 5, 6, or 7, the following equations apply:

$$Y = Clip1_Y(Round(((1 << BitDepth_Y) - 1) * E'_Y)) \tag{E-10}$$

$$Cb = Clip1_C(Round(((1 << BitDepth_C) - 1) * E'_{PB} + (1 << (BitDepth_C - 1)))) \quad (E\text{-}11)$$

$$Cr = Clip1_C(Round(((1 << BitDepth_C) - 1) * E'_{PR} + (1 << (BitDepth_C - 1)))) \quad (E\text{-}12)$$

– Otherwise, if matrix_coefficients is equal to 0 or 8, the following equations apply:

$$R = Clip1_Y(((1 << BitDepth_Y) - 1) * E'_R) \quad (E\text{-}13)$$

$$G = Clip1_Y(((1 << BitDepth_Y) - 1) * E'_G) \quad (E\text{-}14)$$

$$B = Clip1_Y(((1 << BitDepth_Y) - 1) * E'_B) \quad (E\text{-}15)$$

– Otherwise, if matrix_coefficients is equal to 2, the interpretation of the matrix_coefficients syntax element is unknown or is determined by the application.

– Otherwise (matrix_coefficients is not equal to 0, 1, 2, 4, 5, 6, 7, or 8), the interpretation of the matrix_coefficients syntax element is reserved for future definition by ITU-T | ISO/IEC.

The variables E'_Y, E'_{PB}, and E'_{PR} (for matrix_coefficients not equal to 0 or 8) or Y, Cb, and Cr (for matrix_coefficients equal to 0 or 8) are specified as follows:

– If matrix_coefficients is not equal to 0, 8, or 10, the following equations apply:

$$E'_Y = K_R * E'_R + (1 - K_R - K_B) * E'_G + K_B * E'_B \quad (E\text{-}16)$$

$$E'_{PB} = 0.5 * (E'_B - E'_Y) \div (1 - K_B) \quad (E\text{-}17)$$

$$E'_{PR} = 0.5 * (E'_R - E'_Y) \div (1 - K_R) \quad (E\text{-}18)$$

NOTE 3 – E'_Y is a real number with the value 0 associated with nominal black and the value 1 associated with nominal white. E'_{PB} and E'_{PR} are real numbers with the value 0 associated with both nominal black and nominal white. When transfer_characteristics is not equal to 11 or 12, E'_Y is a real number with values in the range of 0 to 1. When transfer_characteristics is not equal to 11 or 12, E'_{PB} and E'_{PR} are real numbers with values in the range of −0.5 to 0.5. When transfer_characteristics is equal to 11 (IEC 61966-2-4), or 12 (ITU-R BT.1361 extended colour gamut system), E'_Y, E'_{PB} and E'_{PR} are real numbers with a larger range not specified in this Specification.

– Otherwise, if matrix_coefficients is equal to 0, the following equations apply:

$$Y = Round(G) \quad (E\text{-}19)$$

$$Cb = Round(B) \quad (E\text{-}20)$$

$$Cr = Round(R) \quad (E\text{-}21)$$

– Otherwise, if matrix_coefficients is equal to 8, the following applies:

  – If $BitDepth_C$ is equal to $BitDepth_Y$, the following equations apply:

$$Y = Round(0.5 * G + 0.25 * (R + B)) \quad (E\text{-}22)$$

$$Cb = Round(0.5 * G - 0.25 * (R + B)) + (1 << (BitDepth_C - 1)) \quad (E\text{-}23)$$

$$Cr = Round(0.5 * (R - B)) + (1 << (BitDepth_C - 1)) \quad (E\text{-}24)$$

NOTE 4 – For purposes of the YCgCo nomenclature used in Table E-5, Cb and Cr of Equations E-23 and E-24 may be referred to as Cg and Co, respectively. The inverse conversion for the above three equations should be computed as:

$$t = Y - (Cb - (1 << (BitDepth_C - 1))) \quad (E\text{-}25)$$

$$G = Clip1_Y(Y + (Cb - (1 << (BitDepth_C - 1)))) \quad (E\text{-}26)$$

$$B = Clip1_Y(t - (Cr - (1 << (BitDepth_C - 1)))) \quad (E\text{-}27)$$

$$R = Clip1_Y(t + (Cr - (1 << (BitDepth_C - 1)))) \quad (E\text{-}28)$$

  – Otherwise ($BitDepth_C$ is not equal to $BitDepth_Y$), the following equations apply:

$$Cr = Round(R) - Round(B) + (1 << (BitDepth_C - 1)) \quad (E\text{-}29)$$

$$t = Round(B) + ((Cr - (1 << (BitDepth_C - 1))) >> 1) \quad (E\text{-}30)$$

$$Cb = Round(G) - t + (1 << (BitDepth_C - 1)) \quad \text{(E-31)}$$

$$Y = t + ((Cb - (1 << (BitDepth_C - 1))) >> 1) \quad \text{(E-32)}$$

NOTE 5 – For purposes of the YCgCo nomenclature used in Table E-5, Cb and Cr of Equations E-31 and E-29 may be referred to as Cg and Co, respectively. The inverse conversion for the above four equations should be computed as.

$$t = Y - ((Cb - (1 << (BitDepth_C - 1))) >> 1) \quad \text{(E-33)}$$

$$G = Clip1_Y(t + (Cb - (1 << (BitDepth_C - 1)))) \quad \text{(E-34)}$$

$$B = Clip1_Y(t - ((Cr - (1 << (BitDepth_C - 1))) >> 1)) \quad \text{(E-35)}$$

$$R = Clip1_Y(B + (Cr - (1 << (BitDepth_C - 1)))) \quad \text{(E-36)}$$

– Otherwise (matrix_coefficients is equal to 10), the following equations apply:

$$E_Y = K_R * E_R + (1 - K_R - K_B) * E_G + K_B * E_B \quad \text{(E-37)}$$

$$E'_Y = (E_Y)' \quad \text{(E-38)}$$

NOTE 6 – In this case, E_Y is defined from the "linear-domain" signals for E_R, E_G, and E_B, prior to application of the transfer characteristics function, which is then applied to produce the signal E'_Y. E_Y and E'_Y are analogue with the value 0 associated with nominal black and the value 1 associated with nominal white.

$$E'_{PB} = (E'_B - E'_Y) \div 1.9404 \quad \text{for } -0.9702 <= E'_B - E'_Y <= 0 \quad \text{(E-39)}$$

$$E'_{PB} = (E'_B - E'_Y) \div 1.5816 \quad \text{for } 0 < E'_B - E'_Y <= 0.7908 \quad \text{(E-40)}$$

$$E'_{PR} = (E'_R - E'_Y) \div 1.7184 \quad \text{for } -0.8592 <= E'_R - E'_Y <= 0 \quad \text{(E-41)}$$

$$E'_{PR} = (E'_R - E'_Y) \div 0.9936 \quad \text{for } 0 < E'_R - E'_Y <= 0.4968 \quad \text{(E-42)}$$

Table E-5 – Matrix coefficients

Value	Matrix	Informative remark
0	GBR	Typically referred to as RGB; see Equations E-19 to E-21 IEC 61966-2-1 (sRGB)
1	$K_R = 0.2126$; $K_B = 0.0722$	ITU-R Rec. BT.709-5 ITU-R Rec. BT.1361 conventional colour gamut system and extended colour gamut system IEC 61966-2-1 (sYCC) IEC 61966-2-4 $xvYCC_{709}$ Society of Motion Picture and Television Engineers RP 177 (1993) Annex B
2	Unspecified	Image characteristics are unknown or are determined by the application.
3	Reserved	For future use by ITU-T | ISO/IEC
4	$K_R = 0.30$; $K_B = 0.11$	United States Federal Communications Commission Title 47 Code of Federal Regulations (2003) 73.682 (a) (20)
5	$K_R = 0.299$; $K_B = 0.114$	ITU-R Rec. BT.470-6 System B, G (historical) ITU-R Rec. BT.601-6 625 ITU-R Rec. BT.1358 625 ITU-R Rec. BT.1700 625 PAL and 625 SECAM IEC 61966-2-4 $xvYCC_{601}$ (functionally the same as the value 6)
6	$K_R = 0.299$; $K_B = 0.114$	ITU-R Rec. BT.601-6 525 ITU-R Rec. BT.1358 525 ITU-R Rec. BT.1700 NTSC Society of Motion Picture and Television Engineers 170M (2004) (functionally the same as the value 5)
7	$K_R = 0.212$; $K_B = 0.087$	Society of Motion Picture and Television Engineers 240M (1999)
8	YCgCo	See Equations E-22 to E-36
9	$K_R = 0.2627$; $K_B = 0.0593$	Rec. ITU-R BT.2020 non-constant luminance system; see Equations E-16 to E-18
10	$K_R = 0.2627$; $K_B = 0.0593$	Rec. ITU-R BT.2020 constant luminance system see Equations E-37 to E-42
11..255	Reserved	For future use by ITU-T | ISO/IEC

chroma_loc_info_present_flag equal to 1 specifies that chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field are present. chroma_loc_info_present_flag equal to 0 specifies that chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field are not present.

When chroma_format_idc is not equal to 1, chroma_loc_info_present_flag should be equal to 0.

chroma_sample_loc_type_top_field and **chroma_sample_loc_type_bottom_field** specify the location of chroma samples as follows:

– If chroma_format_idc is equal to 1 (4:2:0 chroma format), chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field specify the location of chroma samples for the top field and the bottom field, respectively, as shown in Figure E-1.

– Otherwise (chroma_format_idc is not equal to 1), the values of the syntax elements chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field shall be ignored. When chroma_format_idc is equal to 2 (4:2:2 chroma format) or 3 (4:4:4 chroma format), the location of chroma samples is specified in subclause 6.2. When chroma_format_idc is equal to 0, there is no chroma sample array.

The value of chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field shall be in the range of 0 to 5, inclusive. When the chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field are not present, the values of chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field is inferred to be equal to 0.

NOTE 7 – When coding progressive source material, chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field should have the same value.

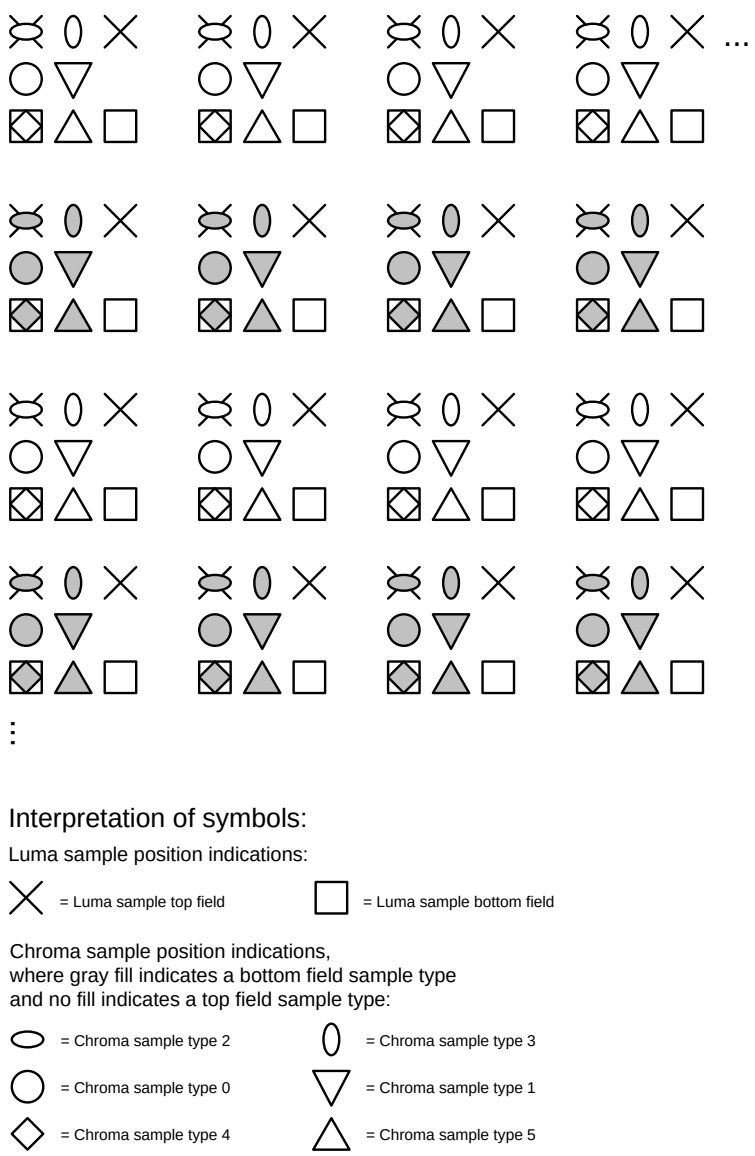


Figure E-1 – Location of chroma samples for top and bottom fields for chroma_format_idc equal to 1 (4:2:0 chroma format) as a function of chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field

neutral_chroma_indication_flag equal to 1 indicates that the value of all decoded chroma samples is equal to 1 << ($BitDepth_C$ − 1). neutral_chroma_indication_flag equal to 0 provides no indication of decoded chroma sample values. When neutral_chroma_indication_flag is equal to 1, it is a requirement of bitstream conformance that the value of all decoded chroma samples produced by the decoding process shall be equal to 1 << ($BitDepth_C$ − 1). When neutral_chroma_indication_flag is not present, it is inferred to be equal to 0.

NOTE 8 – When neutral_chroma_indication_flag is equal to 1, it is not necessary for the decoder to apply the specified decoding process in order to determine the value of the decoded chroma samples.

field_seq_flag equal to 1 indicates that the coded video sequence conveys pictures that represent fields, and specifies that a picture timing SEI message shall be present in every access unit of the current coded video sequence. field_seq_flag equal to 0 indicates that the coded video sequence conveys pictures that represent frames and that a picture timing SEI message may or may not be present in any access unit of the current coded video sequence. When field_seq_flag is not present, it is inferred to be equal to 0.

NOTE 9 – The specified decoding process does not treat access units conveying 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, access units containing 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).

frame_field_info_present _flag equal to 1 specifies that picture timing SEI messages are present for every picture and include the pic_struct, progressive_source_idc, and duplicate_flag syntax elements. frame_field_info_present_flag equal to 0 specifies that the pic_struct syntax element is not present in picture timing SEI messages. When field_seq_flag is equal to 1, frame_field_info_present_flag shall be equal to 1. When frame_field_info_present_flag is not present, its value is inferred to be equal to 0.

default_display_window_flag equal to 1 indicates that the default display window parameters follow next in the VUI. default_display_window_flag equal to 0 indicates that the default display window parameters are not present. The default display window parameters identify the area within the conformance rectangle that is suggested to be displayed in the absence of any alternative indication (provided within the bitstream or by external means not specified in this Specification) of preferred display characteristics.

def_disp_win_left_offset, **def_disp_win_right_offset**, **def_disp_win_top_offset**, and **def_disp_win_bottom_offset** specify the samples of the pictures in the coded video sequence that are within the default display window, in terms of a rectangular region specified in picture coordinates for display. When default_display_window_flag is equal to 0, the values of def_disp_win_left_offset, def_disp_win_right_offset, def_disp_win_top_offset, and def_disp_win_bottom_offset are inferred to be equal to 0.

The following variables are derived from the default display window parameters.

leftOffset = conf_win_left_offset + def_disp_win_left_offset (E-43)
rightOffset = conf_win_right_offset + def_disp_win_right_offset (E-44)
topOffset = conf_win_top_offset + def_disp_win_top_offset (E-45)
bottomOffset = conf_win_bottom_offset + def_disp_win_bottom_offset (E-46)

The default display window contains the luma samples with horizontal picture coordinates from SubWidthC * leftOffset to pic_width_in_luma_samples − (SubWidthC * rightOffset + 1) and vertical picture coordinates from SubHeightC * topOffset to pic_height_in_luma_samples − (SubHeightC * bottomOffset + 1), inclusive. It is a requirement of bitstream conformance that the value of leftOffset shall be in the range of 0 to (pic_width_in_luma_samples / SubWidthC) − (rightOffset + 1), inclusive; and the value of topOffset shall be in the range of 0 to (pic_height_in_luma_samples / SubHeightC) − (bottomOffset + 1), inclusive.

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.

hrd_parameters_present_flag equal to 1 specifies that the syntax structure hrd_parameters( ) is present in the vui_parameters( ) syntax structue. hrd_parameters_present_flag equal to 0 specifies that the syntax structure hrd_parameters( ) is not present in the vui_parameters( ) syntax structue.

poc_proportional_to_timing_flag equal to 1 indicates that the picture order count value for each picture in the coded video sequence that is not the first picture in the coded video sequence, in decoding order, is proportional to the output time of the picture relative to the output time of the first picture in the coded video sequence. poc_proportional_to_timing_flag equal to 0 indicates that the picture order count value for each picture in the coded video sequence that is not the first picture in the coded video sequence, in decoding order, may or may not be proportional to the output time of the picture relative to the output time of the first picture in the coded video sequence.

num_ticks_poc_diff_one_minus1 plus 1 specifies the number of clock ticks [Ed. (GJS): Clarify "clock tick".] corresponding to a difference of picture order count values equal to 1.

bitstream_restriction_flag equal to 1, specifies that the following coded video sequence bitstream restriction parameters are present. bitstream_restriction_flag equal to 0, specifies that the following coded video sequence bitstream restriction parameters are not present.

tiles_fixed_structure_flag equal to 1 indicates that each picture parameter set that is active in the coded video sequence has the same value of the syntax elements num_tile_columns_minus1, num_tile_rows_minus1, uniform_spacing_flag, column_width_minus1[i], row_height_minus1[i] and loop_filter_across_tiles_enabled_flag, when present. tiles_fixed_structure_flag equal to 0 indicates that tiles syntax elements in different picture parameter sets may or may not have the same value. When the tiles_fixed_structure_flag syntax element is not present, it is inferred to be equal to 0.

NOTE 10 – The signalling of tiles_fixed_structure_flag equal to 1 is a guarantee to a decoder that each picture in the coded video sequence has the same number of tiles distributed in the same way which might be useful for workload allocation in the case of multi-threaded decoding.

motion_vectors_over_pic_boundaries_flag equal to 0 indicates that no sample outside the picture boundaries and no sample at a fractional sample position for which the sample value is derived using one or more samples outside the picture boundaries is used for inter prediction of any sample. motion_vectors_over_pic_boundaries_flag equal to 1 indicates that one or more samples outside picture boundaries may be used in inter prediction. When the

motion_vectors_over_pic_boundaries_flag syntax element is not present, motion_vectors_over_pic_boundaries_flag value is inferred to be equal to 1.

restricted_ref_pic_lists_flag equal to 1 indicates that all P and B slices (if present) that belong to the same picture have an identical reference picture list 0, and that all B slices (if present) that belong to the same picture have an identical reference picture list 1.

min_spatial_segmentation_idc, when not equal to 0, establishes a bound on the maximum possible size of distinct coded spatial segmentation regions in the pictures of the coded video sequence. When min_spatial_segmentation_idc is not present, it is inferred to be equal to 0.

The variable minSpatialSegmentation is derived from min_spatial_segmentation_idc as follows.

$$minSpatialSegmentationTimes4 = min\_spatial\_segmentation\_idc + 4 \quad \text{(E-47)}$$

A slice is said to contain a specific luma sample when the coding block that contains the luma sample is contained in the slice. Correspondingly, a tile is said to contain a specific luma sample when the coding block that contains the luma sample is contained in the tile.

Depending on the value of min_spatial_segmentation_idc, the following applies:

– If min_spatial_segmentation_idc is equal to 0, no limit on the maximum size of spatial segments is indicated.

– Otherwise (min_spatial_segmentation_idc is not equal to 0), it is a requirement of bitstream conformance that exactly one of the following conditions shall be true:

  – In each picture parameter set that is activated within the coded video sequence, tiles_enabled_flag is equal to 0 and entropy_coding_sync_enabled_flag is equal to 0, and there is no slice in the coded video sequence that contains more than (4 * PicSizeInSamplesY) / minSpatialSegmentationTimes4 luma samples.

  – In each picture parameter set that is activated within the coded video sequence, tiles_enabled_flag is equal to 1 and entropy_coding_sync_enabled_flag is equal to 0, and there is no tile in the coded video sequence that contains more than (4 * PicSizeInSamplesY) / minSpatialSegmentationTimes4 luma samples.

  – In each picture parameter set that is activated within the coded video sequence, tiles_enabled_flag is equal to 0 and entropy_coding_sync_enabled_flag is equal to 1, and the syntax elements pic_width_in_luma_samples, pic_height_in_luma_samples and the variable CtbSizeY obey the following constraint:

$$(2 * pic\_height\_in\_luma\_samples + pic\_width\_in\_luma\_samples) * CtbSizeY \le (4 * PicSizeInSamplesY) / minSpatialSegmentationTimes4 \quad \text{(E-48)}$$

NOTE 11 – The syntax element min_spatial_segmentation_idc can be used by a decoder to calculate the maximum number of luma samples to be processed by one processing thread, making the assumption that the decoder maximally utilizes the parallel decoding information. However, it is important to be aware that there may be some inter-dependencies between the different threads – e.g. due to entropy coding synchronization or deblocking filtering across tile or slice boundaries. To aid decoders in planning the decoding workload distribution, encoders are encouraged to set the value of min_spatial_segmentation_idc to the highest possible value for which one of the above three conditions is true. For example, for the case when tiles_enabled_flag is equal to 0 and entropy_coding_sync_enabled_flag is equal to 1, encoders should set min_spatial_segmentation_idc equal to 4 * PicSizeInSamplesY / ((2 * pic_height_in_luma_samples + pic_width_in_luma_samples) * CtbSizeY) − 4.

max_bytes_per_pic_denom indicates a number of bytes not exceeded by the sum of the sizes of the VCL NAL units associated with any coded picture in the coded video sequence.

The number of bytes that represent a picture in the NAL unit stream is specified for this purpose as the total number of bytes of VCL NAL unit data (i.e. the total of the NumBytesInNALunit variables for the VCL NAL units) for the picture. The value of max_bytes_per_pic_denom shall be in the range of 0 to 16, inclusive.

Depending on max_bytes_per_pic_denom the following applies:

– If max_bytes_per_pic_denom is equal to 0, no limits are indicated.

– Otherwise (max_bytes_per_pic_denom is not equal to 0), it is a requirement of bitstream conformance that no coded picture shall be represented in the coded video sequence by more than the following number of bytes.

$$(PicSizeInMinCbsY * RawMinCuBits) \div (8 * max\_bytes\_per\_pic\_denom) \quad \text{(E-49)}$$

When the max_bytes_per_pic_denom syntax element is not present, the value of max_bytes_per_pic_denom is inferred to be equal to 2.

max_bits_per_mincu_denom indicates an upper bound for the number of coded bits of coding_unit( ) data for any coding block in any picture of the coded video sequence. The value of max_bits_per_mincu_denom shall be in the range of 0 to 16, inclusive. [Ed. Improve syntax element name.]

Depending on max_bits_per_mincu_denom, the following applies:

– If max_bits_per_mincu_denom is equal to 0, no limit is specified by this syntax element.

– Otherwise (max_bits_per_mincu_denom is not equal to 0), it is a requirement of bitstream conformance that no coded coding_unit( ) shall be represented in the bitstream by more than the following number of bits.

$$(128 + \text{RawMinCuBits}) \div \text{max\_bits\_per\_mincu\_denom} * (2 << (\text{log2CbSize} - \text{Log2MinCbSizeY})) \quad \text{(E-50)}$$

where log2CbSize is the value of log2CbSize for the given coding block and the number of bits of coding_unit( ) data for the same coding block is given by the number of times read_bits(1) is called in subclauses 9.2.3.2.2 and 9.2.3.2.3.

When the max_bits_per_mincu_denom is not present, the value of max_bits_per_mincu_denom is inferred to be equal to 1.

log2_max_mv_length_horizontal and **log2_max_mv_length_vertical** indicate the maximum absolute value of a decoded horizontal and vertical motion vector component, respectively, in ¼ luma sample units, for all pictures in the coded video sequence. A value of n asserts that no value of a motion vector component is outside the range from -2^n to $2^n - 1$, inclusive, in units of ¼ luma sample displacement. The value of log2_max_mv_length_horizontal shall be in the range of 0 to 16, inclusive. The value of log2_max_mv_length_vertical shall be in the range of 0 to 15, inclusive. When log2_max_mv_length_horizontal is not present, the values of log2_max_mv_length_horizontal and log2_max_mv_length_vertical is inferred to be equal to 15.

NOTE 12 – The maximum absolute value of a decoded vertical or horizontal motion vector component is also constrained by profile, tier and level limits as specified in Annex A.

E.2.2 HRD parameters semantics

The hrd_parameters( ) syntax structure provides HRD parameters used in the HRD operations for an operation point set. When the hrd_parameters( ) syntax structure is included in a video parameter set, the applicable operation point set to which the hrd_parameters( ) syntax structure applies is specified by the corresponding hrd_op_set_idx[i] syntax element in the video parameter set. When the hrd_parameters( ) syntax structure is included in a sequence parameter set, the operation point set to which the hrd_parameters( ) syntax structure applies is the operation point set that has OpLayerIdSet containing all nuh_reserved_zero_6bits values present in the coded video sequence.

For interpretation of the following semantics, the bitstream (or a part thereof) refers to the bitstream subset (or a part thereof) associated with the operation point set to which the hrd_parameters( ) syntax structure applies.

timing_info_present_flag equal to 1 specifies that num_units_in_tick and time_scale are present in the hrd_parameters( ) syntax structure. timing_info_present_flag equal to 0 specifies that num_units_in_tick and time_scale are not present in the hrd_parameters( ) syntax structure. When not present, the value of timing_info_present_flag is inferred to be 0. [Ed. (GJS): I'm worried about this statement. What if it is present in the VPS but not in the VUI, or vice-versa? In that case, which value of timing_info_present_flag is used to determine the presence or absence of num_ticks_poc_diff_one_minus1?]

num_units_in_tick is the number of time units of a clock operating at the frequency time_scale Hz that corresponds to one increment (called a clock tick) of a clock tick counter. num_units_in_tick shall be greater than 0. A clock tick is the minimum interval of time that can be represented in the coded data when sub_pic_cpb_params_present_flag is equal to 0. 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. See Equation C-1.

time_scale is the number of time units that pass in one second. For example, a time coordinate system that measures time using a 27 MHz clock has a time_scale of 27 000 000. time_scale shall be greater than 0.

nal_hrd_parameters_present_flag equal to 1 specifies that NAL HRD parameters (pertaining to Type II bitstream conformance) are present in the hrd_parameters( ) syntax structure. nal_hrd_parameters_present_flag equal to 0 specifies that NAL HRD parameters are not present in the hrd_parameters( ) syntax structure.

NOTE 1 – When nal_hrd_parameters_present_flag is equal to 0, the conformance of the bitstream cannot be verified without provision of the NAL HRD parameters and all buffering period and picture timing SEI messages, by some means not specified in this 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:

  – nal_hrd_parameters_present_flag is present in the bitstream and is equal to 1,

  - The need for presence of buffering periods for NAL HRD operation to be present in the bitstream in buffering period SEI messages is determined by the application, by some means not specified in this Specification.

- Otherwise, the value of NalHrdBpPresentFlag is set equal to 0.

vcl_hrd_parameters_present_flag equal to 1 specifies that VCL HRD parameters (pertaining to all bitstream conformance) are present in the hrd_parameters( ) syntax structure. vcl_hrd_parameters_present_flag equal to 0 specifies that VCL HRD parameters are not present in the hrd_parameters( ) syntax structure.

NOTE 2 – When vcl_hrd_parameters_present_flag is equal to 0, the conformance of the bitstream cannot be verified without provision of the VCL HRD parameters and all buffering period and picture timing SEI messages, by some means not specified in this 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:
  - vcl_hrd_parameters_present_flag is present in the bitstream and is equal to 1,
  - The need for presence of buffering periods for VCL HRD operation to be present in the bitstream in buffering period SEI messages is determined by the application, by some means not specified in this 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:
  - nal_hrd_parameters_present_flag is present in the bitstream and is equal to 1,
  - vcl_hrd_parameters_present_flag is present in the bitstream and is equal to 1,
  - The need for presence of CPB and DPB output delays to be present in the bitstream in picture timing SEI messages is determined by the application, by some means not specified in this Specification.
- Otherwise, the value of CpbDpbDelaysPresentFlag is set equal to 0.

sub_pic_cpb_params_present_flag equal to 1 specifies that sub-picture level CPB removal delay parameters are present and the CPB may operate at access unit level or sub-picture level. sub_pic_cpb_params_present_flag equal to 0 specifies that sub-picture level CPB removal delay parameters are not present and the CPB operates at access unit level. When sub_pic_cpb_params_present_flag is not present, its value is inferred to be equal to 0.

tick_divisor_minus2 is used to specify the sub-picture clock tick. A sub-picture clock tick is the minimum interval of time that can be represented in the coded data when sub_pic_cpb_params_present_flag is equal to 1.

du_cpb_removal_delay_length_minus1 plus 1 specifies the length, in bits, of the du_cpb_removal_delay_minus1[i] and du_common_cpb_removal_delay_minus1 syntax elements of the picture timing SEI message and the du_spt_cpb_removal_delay syntax element in the decoding unit information SEI message.

sub_pic_cpb_params_in_pic_timing_sei_flag equal to 1 specifies that sub-picture level CPB removal delay parameters are present in picture timing SEI messages and no decoding unit information SEI message is available (in the coded video sequence or provided through external means not specified in this Specification). sub_pic_cpb_params_in_pic_timing_sei_flag equal to 0 specifies that sub-picture level CPB removal delay parameters are present in decoding unit information SEI messages and picture timing SEI messages do not include sub-picture level CPB removal delay parameters. When the sub_pic_cpb_params_in_pic_timing_sei_flag syntax element is not present, it is inferred to be equal to 0.

bit_rate_scale (together with bit_rate_value_minus1[i]) specifies the maximum input bit rate of the i-th CPB.

cpb_size_scale (together with cpb_size_value_minus1[i]) specifies the CPB size of the i-th CPB when the CPB operates at the access unit level.

cpb_size_du_scale (together with cpb_size_du_value_minus1[i]) specifies the CPB size of the i-th CPB when the CPB operates at sub-picture level.

initial_cpb_removal_delay_length_minus1 plus 1 specifies the length, in bits, of the initial_cpb_removal_delay[i] and initial_cpb_removal_offset[i] syntax elements of the buffering period SEI message. When the initial_cpb_removal_delay_length_minus1 syntax element is not present, it is inferred to be equal to 23.

au_cpb_removal_delay_length_minus1 plus 1 specifies the length, in bits, of the au_cpb_removal_delay_minus1 syntax element in the picture timing SEI message. When the au_cpb_removal_delay_length_minus1 syntax element is not present, it is inferred to be equal to 23.

dpb_output_delay_length_minus1 plus 1 specifies the length, in bits, of the pic_dpb_output_delay syntax element in the picture timing SEI message. When the dpb_output_delay_length_minus1 syntax element is not present, it is inferred to be equal to 23.

fixed_pic_rate_general_flag[i] equal to 1 indicates that, when HighestTid 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 HighestTid 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 HighestTid 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.

For each picture n where n indicates the n-th picture (in output order) that is output and picture n is not the last picture in the bitstream (in output order) that is output, the value of $\Delta t_{e,dpb}(n)$ is specified by

$$\Delta t_{e,dpb}(n) = \Delta t_{o,dpb}(n) \div \text{DeltaToDivisor} \quad \text{(E-51)}$$

where $\Delta t_{o,dpb}(n)$ is specified in Equation C-16 and DeltaToDivisor is specified by Table E-6 based on the value of frame_field_info_present_flag and pic_struct for the coded video sequence containing picture n. Entries marked "-" in Table E-6 indicate a lack of dependence of DeltaToDivisor on the corresponding syntax element.

When HighestTid is equal to i and fixed_pic_rate_general_flag[i] is equal to 1 for a coded video sequence containing picture n, the value computed for $\Delta t_{e,dpb}(n)$ shall be equal to t_c * (elemental_duration_in_tcs_minus1[i] + 1), wherein t_c is as specified in Equation C-1 (using the value of t_c for the coded video sequence containing picture n) when one of the following conditions is true for the following picture n_n that is specified for use in Equation C-16:

– picture n_n is in the same coded video sequence as picture n.

– picture n_n is in a different coded video sequence and fixed_pic_rate_general_flag[i] is equal to 1 in the coded video sequence containing picture n_n, the value of num_units_in_tick ÷ time_scale is the same for both coded video sequences, and the value of elemental_duration_in_tc_minus1[i] is the same for both coded video sequences.

When HighestTid is equal to i and fixed_pic_rate_within_cvs_flag[i] is equal to 1 for a coded video sequence containing picture n, the value computed for $\Delta t_{e,dpb}(n)$ shall be equal to t_c * (elemental_duration_in_tcs_minus1[i] + 1), wherein t_c is as specified in Equation C-1 (using the value of t_c for the coded video sequence containing picture n) when the following picture n_n that is specified for use in Equation C-16 is in the same coded video sequence as picture n.

Table E-6 – Divisor for computation of $\Delta t_{e,dpb}(n)$

frame_field_info_present_flag	pic_struct	DeltaToDivisor
0	-	1
1	1	1
1	2	1
1	0	2
1	3	2
1	4	2
1	5	3
1	6	3
1	7	2
1	8	3
1	9	1
1	10	1
1	11	1
1	12	1

low_delay_hrd_flag[i] specifies the HRD operational mode, when HighestTid is equal to i, as specified in Annex C. When fixed_pic_rate_within_cvs_flag[i] is equal to 1, low_delay_hrd_flag[i] shall 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 access unit are permitted. It is expected, but not required, that such "big pictures" occur only occasionally.

cpb_cnt_minus1[i] plus 1 specifies the number of alternative CPB specifications in the bitstream of the coded video sequence when HighestTid is equal to i. The value of cpb_cnt_minus1[i] shall be in the range of 0 to 31, inclusive. When low_delay_hrd_flag[i] is equal to 1, cpb_cnt_minus1[i] shall be equal to 0. When cpb_cnt_minus1[i] is not present, it is inferred to be equal to 0.

E.2.3 Sub-layer HRD parameters semantics

The variable CpbCnt is set equal to cpb_cnt_minus1[tId].

bit_rate_value_minus1[i] (together with bit_rate_scale) specifies the maximum input bit rate for the i th CPB. bit_rate_value_minus1[i] shall be in the range of 0 to $2^{32} - 2$, inclusive. For any i > 0, bit_rate_value_minus1[i] shall be greater than bit_rate_value_minus1[i − 1]. The bit rate in bits per second is given by

$$BitRate[i] = (bit\_rate\_value\_minus1[i] + 1) * 2^{(6 + bit\_rate\_scale)} \quad \text{(E-52)}$$

When the bit_rate_value_minus1[i] syntax element is not present, the value of BitRate[i] 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 subclause A.4.

cpb_size_value_minus1[i] is used together with cpb_size_scale to specify the i-th CPB size when the CPB operates at the access unit level. cpb_size_value_minus1[i] shall be in the range of 0 to $2^{32} - 2$, inclusive. For any i greater than 0, cpb_size_value_minus1[i] shall be less than or equal to cpb_size_value_minus1[i − 1].

When SubPicCpbFlag is equal to 0, the CPB size in bits is given by

$$CpbSize[i] = (cpb\_size\_value\_minus1[i] + 1) * 2^{(4 + cpb\_size\_scale)} \quad \text{(E-53)}$$

When SubPicCpbFlag is equal to 0 and the cpb_size_value_minus1[i] syntax element is not present, the value of CpbSize[i] 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 subclause A.4.

cpb_size_du_value_minus1[i] is used together with cpb_size_du_scale to specify the i-th CPB size when the CPB operates at sub-picture level. cpb_size_du_value_minus1[i] shall be in the range of 0 to $2^{32} - 2$, inclusive. For any i greater than 0, cpb_size_du_value_minus1[i] shall be less than or equal to cpb_size_du_value_minus1[i − 1].

When SubPicCpbFlag is equal to 1, the CPB size in bits is given by

$$CpbSize[i] = (cpb\_size\_du\_value\_minus1[i] + 1) * 2^{(4 + cpb\_size\_du\_scale)} \quad \text{(E-54)}$$

When SubPicCpbFlag is equal to 1 and the cpb_size_du_value_minus1[i] syntax element is not present, the value of CpbSize[i] 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 subclause A.4.

cbr_flag[i] equal to 0 specifies that to decode this bitstream by the HRD using the i-th CPB specification, the hypothetical stream delivery scheduler (HSS) operates in an intermittent bit rate mode. cbr_flag[i] equal to 1 specifies that the HSS operates in a constant bit rate (CBR) mode. When the cbr_flag[i] syntax element is not present, the value of cbr_flag is inferred to be equal to 0.

Bibliography

[1] IEC 61966-2-1, *Multimedia systems and equipment – Colour measurement and management – Part 2-1: Colour management*.

[2] IEC 61966-2-4, *Multimedia systems – Colour measurement*

[3] Rec. ITU-R BT.1358, *Studio parameters of 625 and 525 line progressive television systems*

[4] Rec. ITU-R BT.1361, *Worldwide unified colorimetry and related characteristics of future television and imaging systems*

[5] Rec. ITU-R BT.1700, *Characteristics of composite video signals for conventional analogue television systems*

[6] Rec. ITU R BT.601-6, *Studio encoding parameters of digital television for standard 4:3 and wide screen 16:9 aspect ratios*

[7] Rec. ITU-R BT.709-5, *Parameter values for the HDTV standards for production and international programme exchange*

[8] Rec. ITU-R BT.470-6, *Conventional Television Systems*

[9] Rec. ITU-R BT.2020, *Parameter values for ultra-high definition television systems for production and international programme exchange*

[10] Rec. ITU-T H.271, *Video back-channel messages for conveyance of status information and requests from a video receiver to a video sender*

[11] Society of Motion Picture and Television Engineers 170M (2004), *Television – Composite Analog Video Signal – NTSC for Studio Applications*

[12] Society of Motion Picture and Television Engineers 240M (1999), *Television – Signal Parameters – 1125-Line High-Definition Production*

[13] Society of Motion Picture and Television Engineers RP 177 (1993), *Derivation of Basic Television Color Equations*

[14] United States Federal Communications Commission (2003), *Title 47 Code of Federal Regulations 73.682 (a) (20)*

[15] United States National Television System Committee (1953), *Recommendation for transmission standards for colour television*