



(51) International Patent Classification:

H04N 19/70 (2014.01)

(21) International Application Number:

PCT/JP2021/010119

(22) International Filing Date:

12 March 2021 (12.03.2021)

(25) Filing Language:

English

(26) Publication Language:

English

(30) Priority Data:

62/991,519

18 March 2020 (18.03.2020)

US

(71) Applicant: SHARP KABUSHIKI KAISHA [JP/JP]; 1, Takumi-cho, Sakai-ku, Sakai City, Osaka, 5908522 (JP).

(72) Inventor: DESHPANDE, Sachin G..

(74) Agent: HARAKENZO WORLD PATENT & TRADE-MARK; Daiwa Minamimorimachi Building, 2-6, Tenjin-bashi 2-chome Kita, Kita-ku, Osaka-shi, Osaka, 5300041 (JP).

(81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, IT, JO, JP, KE, KG, KH, KN,

KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

- as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))
- as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))

Published:

- with international search report (Art. 21(3))

(54) Title: SYSTEMS AND METHODS FOR APPLYING DEBLOCKING FILTERS IN VIDEO CODING

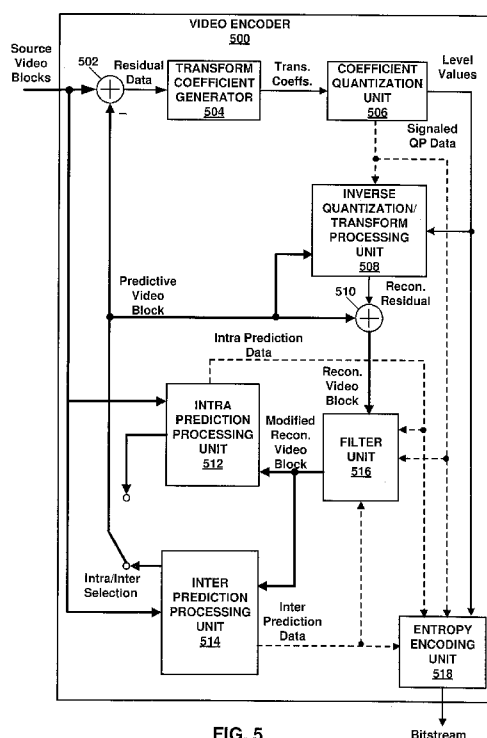


FIG. 5

(57) Abstract: This disclosure relates to video coding and more particularly to techniques for applying deblocking filters for coded video. In one example, a method of decoding chroma component parameter information for decoding video data comprises: receiving a picture parameter set (PPS) including syntax elements corresponding to a picture; parsing a flag in the PPS indicating whether chroma quantization parameter syntax elements are included in the PPS and whether chroma deblocking parameter syntax elements are included in the PPS; conditionally parsing the chroma quantization parameter syntax elements in the PPS based on the value of the flag; conditionally parsing the chroma deblocking parameter syntax elements in the PPS based on the value of the flag; deriving a quantization parameter based on the chroma quantization parameter syntax elements; and performing a deblocking filter based on the chroma deblocking parameter syntax elements.

Description

Title of Invention: SYSTEMS AND METHODS FOR APPLYING DEBLOCKING FILTERS IN VIDEO CODING

Technical Field

[0001] This disclosure relates to video coding and more particularly to techniques for applying deblocking filters for coded video.

Background Art

[0002] Digital video capabilities can be incorporated into a wide range of devices, including digital televisions, laptop or desktop computers, tablet computers, digital recording devices, digital media players, video gaming devices, cellular telephones, including so-called smartphones, medical imaging devices, and the like. Digital video may be coded according to a video coding standard. Video coding standards define the format of a compliant bitstream encapsulating coded video data. A compliant bitstream is a data structure that may be received and decoded by a video decoding device to generate reconstructed video data. Video coding standards may incorporate video compression techniques. Examples of video coding standards include ISO/IEC MPEG-4 Visual and ITU-T H.264 (also known as ISO/IEC MPEG-4 AVC) and High-Efficiency Video Coding (HEVC). HEVC is described in High Efficiency Video Coding (HEVC), Rec. ITU-T H.265, December 2016, which is incorporated by reference, and referred to herein as ITU-T H.265. Extensions and improvements for ITU-T H.265 are currently being considered for the development of next generation video coding standards. For example, the ITU-T Video Coding Experts Group (VCEG) and ISO/IEC (Moving Picture Experts Group (MPEG) (collectively referred to as the Joint Video Exploration Team (JVET)) are working to standardized video coding technology with a compression capability that significantly exceeds that of ITU-T H.265. The Joint Exploration Model 7 (JEM 7), Algorithm Description of Joint Exploration Test Model 7 (JEM 7), ISO/IEC JTC1/SC29/WG11 Document: JVET-G1001, July 2017, Torino, IT, which is incorporated by reference herein, describes the coding features that were under coordinated test model study by the JVET as potentially enhancing video coding technology beyond the capabilities of ITU-T H.265. It should be noted that the coding features of JEM 7 are implemented in JEM reference software. As used herein, the term JEM may collectively refer to algorithms included in JEM 7 and implementations of JEM reference software. Further, in response to a “Joint Call for Proposals on Video Compression with Capabilities beyond HEVC,” jointly issued by VCEG and MPEG, multiple descriptions of video coding tools were proposed by various groups at the 10th Meeting of ISO/IEC JTC1/SC29/WG11 16-20 April 2018, San Diego, CA. From the

multiple descriptions of video coding tools, a resulting initial draft text of a video coding specification is described in “Versatile Video Coding (Draft 1),” 10th Meeting of ISO/IEC JTC1/SC29/WG11 16-20 April 2018, San Diego, CA, document JVET-J1001-v2, which is incorporated by reference herein, and referred to as JVET-J1001. The current development of a next generation video coding standard by the VCEG and MPEG is referred to as the Versatile Video Coding (VVC) project. “Versatile Video Coding (Draft 8),” 17th Meeting of ISO/IEC JTC1/SC29/WG11 7-17 January 2020, Brussel, BE, document JVET-Q2001-vE, which is incorporated by reference herein, and referred to as JVET-Q2001, represents the current iteration of the draft text of a video coding specification corresponding to the VVC project.

[0003] Video compression techniques enable data requirements for storing and transmitting video data to be reduced. Video compression techniques may reduce data requirements by exploiting the inherent redundancies in a video sequence. Video compression techniques may sub-divide a video sequence into successively smaller portions (i.e., groups of pictures within a video sequence, a picture within a group of pictures, regions within a picture, sub-regions within regions, etc.). Intra prediction coding techniques (e.g., spatial prediction techniques within a picture) and inter prediction techniques (i.e., inter-picture techniques (temporal)) may be used to generate difference values between a unit of video data to be coded and a reference unit of video data. The difference values may be referred to as residual data. Residual data may be coded as quantized transform coefficients. Syntax elements may relate residual data and a reference coding unit (e.g., intra-prediction mode indices and motion information). Residual data and syntax elements may be entropy coded. Entropy encoded residual data and syntax elements may be included in data structures forming a compliant bitstream.

Summary of Invention

[0004] In one example, a method of decoding chroma component parameter information for decoding video data, the method comprising: receiving a picture parameter set including syntax elements corresponding to a picture; parsing a flag in the picture parameter set indicating whether chroma quantization parameter syntax elements are included in the picture parameter set and whether chroma deblocking parameter syntax elements are included in the picture parameter set; conditionally parsing the chroma quantization parameter syntax elements in the picture parameter set based on the value of the flag; conditionally parsing the chroma deblocking parameter syntax elements in the picture parameter set based on the value of the flag; deriving a quantization parameter based on the chroma quantization parameter syntax elements; and performing a deblocking filter based on the chroma deblocking parameter syntax

elements.

- [0005] In one example, a device comprising one or more processors configured to: receive a picture parameter set including syntax elements corresponding to a picture; parse a flag in the picture parameter set indicating whether chroma quantization parameter syntax elements are included in the picture parameter set and whether chroma deblocking parameter syntax elements are included in the picture parameter set; conditionally parse the chroma quantization parameter syntax elements in the picture parameter set based on the value of the flag; conditionally parse the chroma deblocking parameter syntax elements in the picture parameter set based on the value of the flag; derive a quantization parameter based on the chroma quantization parameter syntax elements; and perform a deblocking filter based on the chroma deblocking parameter syntax elements.

Brief Description of Drawings

- [0006] [fig.1]FIG. 1 is a block diagram illustrating an example of a system that may be configured to encode and decode video data according to one or more techniques of this disclosure.
- [fig.2]FIG. 2 is a conceptual diagram illustrating coded video data and corresponding data structures according to one or more techniques of this disclosure.
- [fig.3]FIG. 3 is a conceptual diagram illustrating a data structure encapsulating coded video data and corresponding metadata according to one or more techniques of this disclosure.
- [fig.4]FIG. 4 is a conceptual drawing illustrating an example of components that may be included in an implementation of a system that may be configured to encode and decode video data according to one or more techniques of this disclosure.
- [fig.5]FIG. 5 is a block diagram illustrating an example of a video encoder that may be configured to encode video data according to one or more techniques of this disclosure.
- [fig.6]FIG. 6 is a block diagram illustrating an example of a video decoder that may be configured to decode video data according to one or more techniques of this disclosure.

Description of Embodiments

- [0007] In general, this disclosure describes various techniques for coding video data. In particular, this disclosure describes techniques applying deblocking filters. It should be noted that although techniques of this disclosure are described with respect to ITU-T H.264, ITU-T H.265, JEM, and JVET-Q2001, the techniques of this disclosure are generally applicable to video coding. For example, the coding techniques described herein may be incorporated into video coding systems, (including video coding systems based on future video coding standards) including video block structures, intra prediction techniques, inter prediction techniques, transform techniques, filtering

techniques, and/or entropy coding techniques other than those included in ITU-T H.265, JEM, and JVET-Q2001. Thus, reference to ITU-T H.264, ITU-T H.265, JEM, and/or JVET-Q2001 is for descriptive purposes and should not be construed to limit the scope of the techniques described herein. Further, it should be noted that incorporation by reference of documents herein is for descriptive purposes and should not be construed to limit or create ambiguity with respect to terms used herein. For example, in the case where an incorporated reference provides a different definition of a term than another incorporated reference and/or as the term is used herein, the term should be interpreted in a manner that broadly includes each respective definition and/or in a manner that includes each of the particular definitions in the alternative.

- [0008] In one example, a device comprises one or more processors configured to signal a syntax element in a picture parameter set indicating whether chroma deblocking parameter syntax elements are included in the picture parameter set and conditionally signal the chroma deblocking parameter syntax elements in the picture parameter set based on the value of the syntax element indicating whether chroma deblocking parameter syntax elements are included in the picture parameter set.
- [0009] In one example, a non-transitory computer-readable storage medium comprises instructions stored thereon that, when executed, cause one or more processors of a device to signal a syntax element in a picture parameter set indicating whether chroma deblocking parameter syntax elements are included in the picture parameter set and conditionally signal the chroma deblocking parameter syntax elements in the picture parameter set based on the value of the syntax element indicating whether chroma deblocking parameter syntax elements are included in the picture parameter set.
- [0010] In one example, an apparatus comprises means for signaling a syntax element in a picture parameter set indicating whether chroma deblocking parameter syntax elements are included in the picture parameter set and means for conditionally signaling the chroma deblocking parameter syntax elements in the picture parameter set based on the value of the syntax element indicating whether chroma deblocking parameter syntax elements are included in the picture parameter set.
- [0011] In one example, a device comprises one or more processors configured to parse a syntax element in a picture parameter set indicating whether chroma deblocking parameter syntax elements are included in the picture parameter set and conditionally parse the chroma deblocking parameter syntax elements in the picture parameter set based on the value of the syntax element indicating whether chroma deblocking parameter syntax elements are included in the picture parameter set.
- [0012] In one example, a non-transitory computer-readable storage medium comprises instructions stored thereon that, when executed, cause one or more processors of a device to parse a syntax element in a picture parameter set indicating whether chroma de-

blocking parameter syntax elements are included in the picture parameter set and conditionally parse the chroma deblocking parameter syntax elements in the picture parameter set based on the value of the syntax element indicating whether chroma deblocking parameter syntax elements are included in the picture parameter set.

[0013] In one example, an apparatus comprises means for parsing a syntax element in a picture parameter set indicating whether chroma deblocking parameter syntax elements are included in the picture parameter set and means for conditionally parsing the chroma deblocking parameter syntax elements in the picture parameter set based on the value of the syntax element indicating whether chroma deblocking parameter syntax elements are included in the picture parameter set..

[0014] The details of one or more examples are set forth in the accompanying drawings and the description below. Other features, objects, and advantages will be apparent from the description and drawings, and from the claims.

[0015] Video content includes video sequences comprised of a series of frames (or pictures). A series of frames may also be referred to as a group of pictures (GOP). Each video frame or picture may be divided into one or more regions. Regions may be defined according to a base unit (e.g., a video block) and sets of rules defining a region. For example, a rule defining a region may be that a region must be an integer number of video blocks arranged in a rectangle. Further, video blocks in a region may be ordered according to a scan pattern (e.g., a raster scan). As used herein, the term video block may generally refer to an area of a picture or may more specifically refer to the largest array of sample values that may be predictively coded, sub-divisions thereof, and/or corresponding structures. Further, the term current video block may refer to an area of a picture being encoded or decoded. A video block may be defined as an array of sample values. It should be noted that in some cases pixel values may be described as including sample values for respective components of video data, which may also be referred to as color components, (e.g., luma (Y) and chroma (Cb and Cr) components or red, green, and blue components). It should be noted that in some cases, the terms pixel value and sample value are used interchangeably. Further, in some cases, a pixel or sample may be referred to as a pel. A video sampling format, which may also be referred to as a chroma format, may define the number of chroma samples included in a video block with respect to the number of luma samples included in a video block. For example, for the 4:2:0 sampling format, the sampling rate for the luma component is twice that of the chroma components for both the horizontal and vertical directions.

[0016] A video encoder may perform predictive encoding on video blocks and sub-divisions thereof. Video blocks and sub-divisions thereof may be referred to as nodes. ITU-T H.264 specifies a macroblock including 16x16 luma samples. That is, in ITU-T H.264, a picture is segmented into macroblocks. ITU-T H.265 specifies an analogous Coding

Tree Unit (CTU) structure (which may be referred to as a largest coding unit (LCU)). In ITU-T H.265, pictures are segmented into CTUs. In ITU-T H.265, for a picture, a CTU size may be set as including 16x16, 32x32, or 64x64 luma samples. In ITU-T H.265, a CTU is composed of respective Coding Tree Blocks (CTB) for each component of video data (e.g., luma (Y) and chroma (Cb and Cr)). It should be noted that video having one luma component and the two corresponding chroma components may be described as having two channels, i.e., a luma channel and a chroma channel. Further, in ITU-T H.265, a CTU may be partitioned according to a quadtree (QT) partitioning structure, which results in the CTBs of the CTU being partitioned into Coding Blocks (CB). That is, in ITU-T H.265, a CTU may be partitioned into quadtree leaf nodes. According to ITU-T H.265, one luma CB together with two corresponding chroma CBs and associated syntax elements are referred to as a coding unit (CU). In ITU-T H.265, a minimum allowed size of a CB may be signaled. In ITU-T H.265, the smallest minimum allowed size of a luma CB is 8x8 luma samples. In ITU-T H.265, the decision to code a picture area using intra prediction or inter prediction is made at the CU level.

[0017] In ITU-T H.265, a CU is associated with a prediction unit structure having its root at the CU. In ITU-T H.265, prediction unit structures allow luma and chroma CBs to be split for purposes of generating corresponding reference samples. That is, in ITU-T H.265, luma and chroma CBs may be split into respective luma and chroma prediction blocks (PBs), where a PB includes a block of sample values for which the same prediction is applied. In ITU-T H.265, a CB may be partitioned into 1, 2, or 4 PBs. ITU-T H.265 supports PB sizes from 64x64 samples down to 4x4 samples. In ITU-T H.265, square PBs are supported for intra prediction, where a CB may form the PB or the CB may be split into four square PBs. In ITU-T H.265, in addition to the square PBs, rectangular PBs are supported for inter prediction, where a CB may be halved vertically or horizontally to form PBs. Further, it should be noted that in ITU-T H.265, for inter prediction, four asymmetric PB partitions are supported, where the CB is partitioned into two PBs at one quarter of the height (at the top or the bottom) or width (at the left or the right) of the CB. Intra prediction data (e.g., intra prediction mode syntax elements) or inter prediction data (e.g., motion data syntax elements) corresponding to a PB is used to produce reference and/or predicted sample values for the PB.

[0018] JEM specifies a CTU having a maximum size of 256x256 luma samples. JEM specifies a quadtree plus binary tree (QTBT) block structure. In JEM, the QTBT structure enables quadtree leaf nodes to be further partitioned by a binary tree (BT) structure. That is, in JEM, the binary tree structure enables quadtree leaf nodes to be recursively divided vertically or horizontally. In JVET-Q2001, CTUs are partitioned according to a quadtree plus multi-type tree (QTMT or QT+MTT) structure. The QTMT

in JVET-Q2001 is similar to the QTBT in JEM. However, in JVET-Q2001, in addition to indicating binary splits, the multi-type tree may indicate so-called ternary (or triple tree (TT)) splits. A ternary split divides a block vertically or horizontally into three blocks. In the case of a vertical TT split, a block is divided at one quarter of its width from the left edge and at one quarter its width from the right edge and in the case of a horizontal TT split a block is at one quarter of its height from the top edge and at one quarter of its height from the bottom edge.

[0019] As described above, each video frame or picture may be divided into one or more regions. For example, according to ITU-T H.265, each video frame or picture may be partitioned to include one or more slices and further partitioned to include one or more tiles, where each slice includes a sequence of CTUs (e.g., in raster scan order) and where a tile is a sequence of CTUs corresponding to a rectangular area of a picture. It should be noted that a slice, in ITU-T H.265, 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). A slice segment, like a slice, is a sequence of CTUs. Thus, in some cases, the terms slice and slice segment may be used interchangeably to indicate a sequence of CTUs arranged in a raster scan order. Further, it should be noted that in ITU-T H.265, a tile may consist of CTUs contained in more than one slice and a slice may consist of CTUs contained in more than one tile. However, ITU-T H.265 provides that one or both of the following conditions shall be fulfilled: (1) All CTUs in a slice belong to the same tile; and (2) All CTUs in a tile belong to the same slice.

[0020] With respect to JVET-Q2001, slices are required to consist of an integer number of complete tiles or an integer number of consecutive complete CTU rows within a tile, instead of only being required to consist of an integer number of CTUs. It should be noted that in JVET-Q2001, the slice design does not include slice segments (i.e., no independent/dependent slice segments). Thus, in JVET-Q2001, a picture may include a single tile, where the single tile is contained within a single slice or a picture may include multiple tiles where the multiple tiles (or CTU rows thereof) may be contained within one or more slices. In JVET-Q2001, the partitioning of a picture into tiles is specified by specifying respective heights for tile rows and respective widths for tile columns. Thus, in JVET-Q2001 a tile is a rectangular region of CTUs within a particular tile row and a particular tile column position. Further, it should be noted that JVET-Q2001 provides where a picture may be partitioned into subpictures, where a subpicture is a rectangular region of a CTUs within a picture. The top-left CTU of a subpicture may be located at any CTU position within a picture with subpictures being constrained to include one or more slices. Thus, unlike a tile, a subpicture is not necessarily limited to a particular row and column position. It should be noted that sub-

pictures may be useful for encapsulating regions of interest within a picture and a sub-bitstream extraction process may be used to only decode and display a particular region of interest. That is, as described in further detail below, a bitstream of coded video data includes a sequence of network abstraction layer (NAL) units, where a NAL unit encapsulates coded video data, (i.e., video data corresponding to a slice of picture) or a NAL unit encapsulates metadata used for decoding video data (e.g., a parameter set) and a sub-bitstream extraction process forms a new bitstream by removing one or more NAL units from a bitstream.

[0021] FIG. 2 is a conceptual diagram illustrating an example of a picture within a group of pictures partitioned according to tiles, slices, and subpictures. It should be noted that the techniques described herein may be applicable to tiles, slices, subpictures, subdivisions thereof and/or equivalent structures thereto. That is, the techniques described herein may be generally applicable regardless of how a picture is partitioned into regions. For example, in some cases, the techniques described herein may be applicable in cases where a tile may be partitioned into so-called bricks, where a brick is a rectangular region of CTU rows within a particular tile. Further, for example, in some cases, the techniques described herein may be applicable in cases where one or more tiles may be included in so-called tile groups, where a tile group includes an integer number of adjacent tiles. In one example, a tile group may be called a slice. In the example illustrated in FIG. 2, Pic₃ is illustrated as including 16 tiles (i.e., Tile₀ to Tile₁₅) and three slices (i.e., Slice₀ to Slice₂). In the example illustrated in FIG. 2, Slice₀ includes four tiles (i.e., Tile₀ to Tile₃), Slice₁ includes eight tiles (i.e., Tile₄ to Tile₁₁), and Slice₂ includes four tiles (i.e., Tile₁₂ to Tile₁₅). Further, as illustrated in the example of FIG. 2, Pic₃ is illustrated as including two subpictures (i.e., Subpicture₀ and Subpicture₁), where Subpicture₀ includes Slice₀ and Slice₁ and where Subpicture₁ includes Slice₂. As described above, subpictures may be useful for encapsulating regions of interest within a picture and a sub-bitstream extraction process may be used in order to selectively decode (and display) a region interest. For example, referring to FIG. 2, Subpicture₀ may correspond to an action portion of a sporting event presentation (e.g., a view of the field) and Subpicture₁ may correspond to a scrolling banner displayed during the sporting event presentation. By using organizing a picture into subpictures in this manner, a viewer may be able to disable the display of the scrolling banner. That is, through a sub-bitstream extraction process Slice₂ NAL unit may be removed from a bitstream (and thus not decoded) and Slice₀ NAL unit and Slice₁ NAL unit may be decoded and displayed. The encapsulation of slices of a picture into respective NAL unit data structures and sub-bitstream extraction are described in further detail below.

[0022] For intra prediction coding, an intra prediction mode may specify the location of

reference samples within a picture. In ITU-T H.265, defined possible intra prediction modes include a planar (i.e., surface fitting) prediction mode, a DC (i.e., flat overall averaging) prediction mode, and 33 angular prediction modes (predMode: 2-34). In JEM, defined possible intra-prediction modes include a planar prediction mode, a DC prediction mode, and 65 angular prediction modes. It should be noted that planar and DC prediction modes may be referred to as non-directional prediction modes and that angular prediction modes may be referred to as directional prediction modes. It should be noted that the techniques described herein may be generally applicable regardless of the number of defined possible prediction modes.

[0023] For inter prediction coding, a reference picture is determined and a motion vector (MV) identifies samples in the reference picture that are used to generate a prediction for a current video block. For example, a current video block may be predicted using reference sample values located in one or more previously coded picture(s) and a motion vector is used to indicate the location of the reference block relative to the current video block. A motion vector may describe, for example, a horizontal displacement component of the motion vector (i.e., MV_x), a vertical displacement component of the motion vector (i.e., MV_y), and a resolution for the motion vector (e.g., one-quarter pixel precision, one-half pixel precision, one-pixel precision, two-pixel precision, four-pixel precision). Previously decoded pictures, which may include pictures output before or after a current picture, may be organized into one or more to reference pictures lists and identified using a reference picture index value. Further, in inter prediction coding, uni-prediction refers to generating a prediction using sample values from a single reference picture and bi-prediction refers to generating a prediction using respective sample values from two reference pictures. That is, in uni-prediction, a single reference picture and corresponding motion vector are used to generate a prediction for a current video block and in bi-prediction, a first reference picture and corresponding first motion vector and a second reference picture and corresponding second motion vector are used to generate a prediction for a current video block. In bi-prediction, respective sample values are combined (e.g., added, rounded, and clipped, or averaged according to weights) to generate a prediction. Pictures and regions thereof may be classified based on which types of prediction modes may be utilized for encoding video blocks thereof. That is, for regions having a B type (e.g., a B slice), bi-prediction, uni-prediction, and intra prediction modes may be utilized, for regions having a P type (e.g., a P slice), uni-prediction, and intra prediction modes may be utilized, and for regions having an I type (e.g., an I slice), only intra prediction modes may be utilized. As described above, reference pictures are identified through reference indices. For example, for a P slice, there may be a single reference picture list, RefPicList0 and for a B slice, there may be a second independent reference picture

list, RefPicList1, in addition to RefPicList0. It should be noted that for uni-prediction in a B slice, one of RefPicList0 or RefPicList1 may be used to generate a prediction. Further, it should be noted that during the decoding process, at the onset of decoding a picture, reference picture list(s) are generated from previously decoded pictures stored in a decoded picture buffer (DPB).

[0024] Further, a coding standard may support various modes of motion vector prediction. Motion vector prediction enables the value of a motion vector for a current video block to be derived based on another motion vector. For example, a set of candidate blocks having associated motion information may be derived from spatial neighboring blocks and temporal neighboring blocks to the current video block. Further, generated (or default) motion information may be used for motion vector prediction. Examples of motion vector prediction include advanced motion vector prediction (AMVP), temporal motion vector prediction (TMVP), so-called “merge” mode, and “skip” and “direct” motion inference. Further, other examples of motion vector prediction include advanced temporal motion vector prediction (ATMVP) and Spatial-temporal motion vector prediction (STMVP). For motion vector prediction, both a video encoder and video decoder perform the same process to derive a set of candidates. Thus, for a current video block, the same set of candidates is generated during encoding and decoding.

[0025] As described above, for inter prediction coding, reference samples in a previously coded picture are used for coding video blocks in a current picture. Previously coded pictures which are available for use as reference when coding a current picture are referred as reference pictures. It should be noted that the decoding order does not necessarily correspond with the picture output order, i.e., the temporal order of pictures in a video sequence. In ITU-T H.265, when a picture is decoded it is stored to a decoded picture buffer (DPB) (which may be referred to as frame buffer, a reference buffer, a reference picture buffer, or the like). In ITU-T H.265, pictures stored to the DPB are removed from the DPB when they been output and are no longer needed for coding subsequent pictures. In ITU-T H.265, a determination of whether pictures should be removed from the DPB is invoked once per picture, after decoding a slice header, i.e., at the onset of decoding a picture. For example, referring to FIG. 2, Pic₂ is illustrated as referencing Pic₁. Similarly, Pic₃ is illustrated as referencing Pic₀. With respect to FIG. 2, assuming the picture number corresponds to the decoding order, the DPB would be populated as follows: after decoding Pic₀, the DPB would include {Pic₀}; at the onset of decoding Pic₁, the DPB would include {Pic₀}; after decoding Pic₁, the DPB would include {Pic₀, Pic₁}; at the onset of decoding Pic₂, the DPB would include {Pic₀, Pic₁}. Pic₂ would then be decoded with reference to Pic₁ and after decoding Pic₂, the DPB would include {Pic₀, Pic₁, Pic₂}. At the onset of decoding Pic₃, pictures Pic₀

and Pic_1 would be marked for removal from the DPB, as they are not needed for decoding Pic_3 (or any subsequent pictures, not shown) and assuming Pic_1 and Pic_2 have been output, the DPB would be updated to include $\{Pic_0\}$. Pic_3 would then be decoded by referencing Pic_0 . The process of marking pictures for removal from a DPB may be referred to as reference picture set (RPS) management.

[0026] As described above, intra prediction data or inter prediction data is used to produce reference sample values for a block of sample values. The difference between sample values included in a current PB, or another type of picture area structure, and associated reference samples (e.g., those generated using a prediction) may be referred to as residual data. Residual data may include respective arrays of difference values corresponding to each component of video data. Residual data may be in the pixel domain. A transform, such as, a discrete cosine transform (DCT), a discrete sine transform (DST), an integer transform, a wavelet transform, or a conceptually similar transform, may be applied to an array of difference values to generate transform coefficients. It should be noted that in ITU-T H.265 and JVET-Q2001, a CU is associated with a transform tree structure having its root at the CU level. The transform tree is partitioned into one or more transform units (TUs). That is, an array of difference values may be partitioned for purposes of generating transform coefficients (e.g., four 8x8 transforms may be applied to a 16x16 array of residual values). For each component of video data, such sub-divisions of difference values may be referred to as Transform Blocks (TBs). It should be noted that in some cases, a core transform and a subsequent secondary transforms may be applied (in the video encoder) to generate transform coefficients. For a video decoder, the order of transforms is reversed.

[0027] A quantization process may be performed on transform coefficients or residual sample values directly (e.g., in the case, of palette coding quantization). Quantization approximates transform coefficients by amplitudes restricted to a set of specified values. Quantization essentially scales transform coefficients in order to vary the amount of data required to represent a group of transform coefficients. Quantization may include division of transform coefficients (or values resulting from the addition of an offset value to transform coefficients) by a quantization scaling factor and any associated rounding functions (e.g., rounding to the nearest integer). Quantized transform coefficients may be referred to as coefficient level values. Inverse quantization (or “dequantization”) may include multiplication of coefficient level values by the quantization scaling factor, and any reciprocal rounding or offset addition operations. It should be noted that as used herein the term quantization process in some instances may refer to division by a scaling factor to generate level values and multiplication by a scaling factor to recover transform coefficients in some instances. That is, a quantization process may refer to quantization in some cases and inverse quantization in

some cases. Further, it should be noted that although in some of the examples below quantization processes are described with respect to arithmetic operations associated with decimal notation, such descriptions are for illustrative purposes and should not be construed as limiting. For example, the techniques described herein may be implemented in a device using binary operations and the like. For example, multiplication and division operations described herein may be implemented using bit shifting operations and the like.

[0028] Quantized transform coefficients and syntax elements (e.g., syntax elements indicating a coding structure for a video block) may be entropy coded according to an entropy coding technique. An entropy coding process includes coding values of syntax elements using lossless data compression algorithms. Examples of entropy coding techniques include content adaptive variable length coding (CAVLC), context adaptive binary arithmetic coding (CABAC), probability interval partitioning entropy coding (PIPE), and the like. Entropy encoded quantized transform coefficients and corresponding entropy encoded syntax elements may form a compliant bitstream that can be used to reproduce video data at a video decoder. An entropy coding process, for example, CABAC, may include performing a binarization on syntax elements. Binarization refers to the process of converting a value of a syntax element into a series of one or more bits. These bits may be referred to as “bins.” Binarization may include one or a combination of the following coding techniques: fixed length coding, unary coding, truncated unary coding, truncated Rice coding, Golomb coding, k-th order exponential Golomb coding, and Golomb-Rice coding. For example, binarization may include representing the integer value of 5 for a syntax element as 00000101 using an 8-bit fixed length binarization technique or representing the integer value of 5 as 11110 using a unary coding binarization technique. As used herein each of the terms fixed length coding, unary coding, truncated unary coding, truncated Rice coding, Golomb coding, k-th order exponential Golomb coding, and Golomb-Rice coding may refer to general implementations of these techniques and/or more specific implementations of these coding techniques. For example, a Golomb-Rice coding implementation may be specifically defined according to a video coding standard. In the example of CABAC, for a particular bin, a context provides a most probable state (MPS) value for the bin (i.e., an MPS for a bin is one of 0 or 1) and a probability value of the bin being the MPS or the least probably state (LPS). For example, a context may indicate, that the MPS of a bin is 0 and the probability of the bin being 1 is 0.3. It should be noted that a context may be determined based on values of previously coded bins including bins in the current syntax element and previously coded syntax elements. For example, values of syntax elements associated with neighboring video blocks may be used to determine a context for a current bin.

- [0029] As described above, the sample values of a reconstructed block may differ from the sample values of the current video block that is encoded. Further, it should be noted that in some cases, coding video data on a block-by-block basis may result in artifacts (e.g., so-called blocking artifacts, banding artifacts, etc.) For example, blocking artifacts may cause coding block boundaries of reconstructed video data to be visually perceptible to a user. In this manner, reconstructed sample values may be modified to minimize the difference between the sample values of the current video block that is encoded and the reconstructed block and/or minimize artifacts introduced by the video coding process. Such modifications may generally be referred to as filtering. It should be noted that filtering may occur as part of an in-loop filtering process or a post-loop filtering process. For an in-loop filtering process, the resulting sample values of a filtering process may be used for predictive video blocks (e.g., stored to a reference frame buffer for subsequent encoding at video encoder and subsequent decoding at a video decoder). For a post-loop filtering process the resulting sample values of a filtering process are merely output as part of the decoding process (e.g., not used for subsequent coding). For example, in the case of a video decoder, for an in-loop filtering process, the sample values resulting from filtering the reconstructed block would be used for subsequent decoding (e.g., stored to a reference buffer) and would be output (e.g., to a display). For a post-loop filtering process, the reconstructed block would be used for subsequent decoding and the sample values resulting from filtering the reconstructed block would be output.
- [0030] Deblocking (or de-blocking), deblock filtering, or applying a deblocking filter refers to the process of smoothing the boundaries of neighboring reconstructed video blocks (i.e., making boundaries less perceptible to a viewer). Smoothing the boundaries of neighboring reconstructed video blocks may include modifying sample values included in rows or columns adjacent to a boundary. ITU-T H.265 provides where a deblocking filter is applied to reconstructed sample values as part of an in-loop filtering process. ITU-T H.265 includes two types deblocking filters that may be used for modifying luma samples: a Strong Filter which modifies sample values in the three adjacent rows or columns to a boundary and a Weak Filter which modifies sample values in the immediately adjacent row or column to a boundary and conditionally modifies sample values in the second row or column from the boundary. Further, ITU-T H.265 includes one type of filter that may be used for modifying chroma samples: Normal Filter.
- [0031] In addition to applying a deblocking filter as part of an in-loop filtering process, ITU-T H.265 provides where Sample Adaptive Offset (SAO) filtering may be applied in the in-loop filtering process. In ITU-T H.265, SAO is a process that modifies the deblocked sample values in a region by conditionally adding an offset value. ITU-T H.265 provides two types of SAO filters that may be applied to a CTB: band offset or

edge offset. For each of band offset and edge offset, four offset values are included in a bitstream. For band offset, the offset which is applied depends on the amplitude of a sample value (e.g., amplitudes are mapped to bands which are mapped to the four signaled offsets). For edge offset, the offset which is applied depends on a CTB having one of a horizontal, vertical, first diagonal, or second diagonal edge classification (e.g., classifications are mapped to the four signaled offsets).

[0032] Another type of filtering process includes the so-called adaptive loop filter (ALF). An ALF with block-based adaption is specified in JEM. In JEM, the ALF is applied after the SAO filter. It should be noted that an ALF may be applied to reconstructed samples independently of other filtering techniques. The process for applying the ALF specified in JEM at a video encoder may be summarized as follows: (1) each 2x2 block of the luma component for a reconstructed picture is classified according to a classification index; (2) sets of filter coefficients are derived for each classification index; (3) filtering decisions are determined for the luma component; (4) a filtering decision is determined for the chroma components; and (5) filter parameters (e.g., coefficients and decisions) are signaled. JVET-Q2001 specifies deblocking, SAO, and ALF filters which can be described as being generally based on the deblocking, SAO, and ALF filters provided in ITU-T H.265 and JEM.

[0033] With respect to the equations used herein, the following arithmetic operators may be used:

+ Addition

− Subtraction

* 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.

Further, the following mathematical functions may be used:

$\text{Log2}(x)$ the base-2 logarithm of x ;

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

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

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

With respect to the example syntax used herein, the following definitions of logical operators may be applied:

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

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

$!$ Boolean logical "not"

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

Further, the following relational operators may be applied:

$>$	Greater than
$\geq$	Greater than or equal to
$<$	Less than
$\leq$	Less than or equal to
$==$	Equal to
$!=$	Not equal to

Further, it should be noted that in the syntax descriptors used herein, the following descriptors may be applied:

- ae(v): context-adaptive arithmetic entropy-coded syntax element.
- b(8): byte having any pattern of bit string (8 bits). The parsing process for this descriptor is specified by the return value of the function read_bits(8).
- f(n): fixed-pattern bit string using n bits written (from left to right) with the left bit first. The parsing process for this descriptor is specified by the return value of the function read_bits(n).
- i(n): signed integer using n bits. When n is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function read_bits(n) interpreted as a two's complement integer representation with most significant bit written first.
- se(v): signed integer 0-th order Exp-Golomb-coded syntax element with the left bit first.
- 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.

As described above, video content includes video sequences comprised of a series of pictures and each picture may be divided into one or more regions. In JVET-Q2001, a coded representation of a picture comprises VCL NAL units of a particular layer within an AU and contains all CTUs of the picture. For example, referring again to FIG. 2, the coded representation of Pic₃ is encapsulated in three coded slice NAL units (i.e., Slice₀ NAL unit, Slice₁ NAL unit, and Slice₂ NAL unit). It should be noted that the term video coding layer (VCL) NAL unit is used as a collective term for coded slice NAL units, i.e., VCL NAL is a collective term which includes all types of slice NAL units. As described above, and in further detail below, a NAL unit may encapsulate metadata used for decoding video data. A NAL unit encapsulating metadata used for decoding a video sequence is generally referred to as a non-VCL NAL unit. Thus, in JVET-Q2001, a NAL unit may be a VCL NAL unit or a non-VCL NAL unit. It should be noted that a VCL NAL unit includes slice header data, which provides information used for decoding the particular slice. Thus, in JVET-Q2001, information used for decoding video data, which may be referred to as metadata in some cases, is not limited to being included in non-VCL NAL units. JVET-Q2001 provides where a picture unit (PU) is 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 and where an access unit (AU) is a set of PUs that belong to different layers and contain coded pictures associated with the same time for output from the DPB. JVET-Q2001 further provides where a layer is a set of VCL NAL units that all have a particular value of a layer identifier and the associated non-VCL NAL units. Further, in JVET-Q2001, a PU consists of zero or one PH NAL units, one coded picture, which comprises of one or more VCL NAL units, and zero or more other non-VCL NAL units. Further, in JVET-Q2001, a coded video sequence (CVS) is a sequence of AUs that consists, in decoding order, of a CVSS AU, followed by zero or more AUs that are not CVSS AUs, including all subsequent AUs up to but not including any subsequent AU that is a CVSS AU, where a coded video sequence start (CVSS) AU is an AU in which there is a PU for each layer in the CVS and the coded picture in each present picture unit is a coded layer video sequence start (CLVSS) picture. In JVET-Q2001, a coded layer video sequence (CLVS) is a sequence of PUs within the same layer that consists, in decoding order, of a CLVSS PU, followed by zero or more PUs that are not CLVSS PUs, including all subsequent PUs up to but not including any subsequent PU that is a CLVSS PU. This is, in JVET-Q2001, a bitstream may be described as including a sequence of AUs forming one or more CVSs.

[0034] Multi-layer video coding enables a video presentation to be decoded/displayed as a presentation corresponding to a base layer of video data and decoded/displayed one or more additional presentations corresponding to enhancement layers of video data. For example, a base layer may enable a video presentation having a basic level of quality (e.g., a High Definition rendering and/or a 30 Hz frame rate) to be presented and an enhancement layer may enable a video presentation having an enhanced level of quality (e.g., an Ultra High Definition rendering and/or a 60 Hz frame rate) to be presented. An enhancement layer may be coded by referencing a base layer. That is, for example, a picture in an enhancement layer may be coded (e.g., using inter-layer prediction techniques) by referencing one or more pictures (including scaled versions thereof) in a base layer. It should be noted that layers may also be coded independent of each other. In this case, there may not be inter-layer prediction between two layers. Each NAL unit may include an identifier indicating a layer of video data the NAL unit is associated with. As described above, a sub-bitstream extraction process may be used to only decode and display a particular region of interest of a picture. Further, a sub-bitstream extraction process may be used to only decode and display a particular layer of video. Sub-bitstream extraction may refer to a process where a device receiving a compliant or conforming bitstream forms a new compliant or conforming bitstream by discarding and/or modifying data in the received bitstream. For example, sub-bitstream extraction may be used to form a new compliant or conforming bitstream corresponding to a

particular representation of video (e.g., a high quality representation).

[0035] In JVET-Q2001, each of a video sequence, a GOP, a picture, a slice, and CTU may be associated with metadata that describes video coding properties and some types of metadata are encapsulated in non-VCL NAL units. JVET-Q2001 defines parameter sets that may be used to describe video data and/or video coding properties. In particular, JVET-Q2001 includes the following four types of parameter sets: video parameter set (VPS), sequence parameter set (SPS), picture parameter set (PPS), and adaptation parameter set (APS), where a SPS applies to zero or more entire CVSs, a PPS applies to zero or more entire coded pictures, a APS applies to zero or more slices, and a VPS may be optionally referenced by a SPS. A PPS applies to an individual coded picture that refers to it. In JVET-Q2001, parameter sets may be encapsulated as a non-VCL NAL unit and/or may be signaled as a message. JVET-Q2001 also includes a picture header (PH) which is encapsulated as a non-VCL NAL unit. In JVET-Q2001, a picture header applies to all slices of a coded picture. JVET-Q2001 further enables decoding capability information (DCI) and supplemental enhancement information (SEI) messages to be signaled. In JVET-Q2001, DCI and SEI messages assist in processes related to decoding, display or other purposes, however, DCI and SEI messages may not be required for constructing the luma or chroma samples according to a decoding process. In JVET-Q2001, DCI and SEI messages may be signaled in a bitstream using non-VCL NAL units. Further, DCI and SEI messages may be conveyed by some mechanism other than by being present in the bitstream (i.e., signaled out-of-band).

[0036] FIG. 3 illustrates an example of a bitstream including multiple CVSs, where a CVS includes AUs, and AUs include picture units. The example illustrated in FIG. 3 corresponds to an example of encapsulating the slice NAL units illustrated in the example of FIG. 2 in a bitstream. In the example illustrated in FIG. 3, the corresponding picture unit for Pic₃ includes the three VCL NAL coded slice NAL units, i.e., Slice₀ NAL unit, Slice₁ NAL unit, and Slice₂ NAL unit and two non-VCL NAL units, i.e., a PPS NAL Unit and a PH NAL unit. It should be noted that in FIG. 3, HEADER is a NAL unit header (i.e., not to be confused with a slice header). Further, it should be noted that in FIG. 3, other non-VCL NAL units, which are not illustrated may be included in the CVSs, e.g., SPS NAL units, VPS NAL units, SEI message NAL units, etc. Further, it should be noted that in other examples, a PPS NAL Unit used for decoding Pic₃ may be included elsewhere in the bitstream, e.g., in the picture unit corresponding to Pic₀ or may be provided by an external mechanism. As described in further detail below, in JVET-Q2001, a PH syntax structure may be present in the slice header of a VCL NAL unit or in a PH NAL unit of the current PU.

[0037]

JVET-Q2001 defines NAL unit header semantics that specify the type of Raw Byte Sequence Payload (RBSP) data structure included in the NAL unit. Table 1 illustrates the syntax of the NAL unit header provided in JVET-Q2001.

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

Table 1

JVET-Q2001 provides the following definitions for the respective syntax elements illustrated in Table 1.

forbidden_zero_bit shall be equal to 0.

nuh_reserved_zero_bit shall be equal to '0'. The value 1 of **nuh_reserved_zero_bit** may be specified in the future by ITU-T | ISO/IEC. Decoders shall ignore (i.e. remove from the bitstream and discard) NAL units with **nuh_reserved_zero_bit** equal to '1'.

nuh_layer_id specifies the identifier of the layer to which a VCL NAL unit belongs or the identifier of a layer to which a non-VCL NAL unit applies. The value of **nuh_layer_id** shall be in the range of 0 to 55, inclusive. Other values for **nuh_layer_id** are reserved for future use by ITU-T | ISO/IEC.

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

The value of **nuh_layer_id** for AUD, PH, EOS, and FD NAL units is constrained as follows:

- If **nal_unit_type** is equal to AUD_NUT, **nuh_layer_id** shall be equal to **vps_layer_id[0]**.
- Otherwise, when **nal_unit_type** is equal to PH_NUT, EOS_NUT, or FD_NUT, **nuh_layer_id** shall be equal to the **nuh_layer_id** of associated VCL NAL unit.

NOTE – The value of **nuh_layer_id** of DCI, VPS, and EOB NAL units is not constrained.

The value of **nal_unit_type** shall be the same for all pictures of a CVSS AU.

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

The value of **nuh_temporal_id_plus1** shall not be equal to 0.

The variable **TemporalId** is derived as follows:

$$\text{TemporalId} = \text{nuh_temporal_id_plus1} - 1$$

When `nal_unit_type` is in the range of `IDR_W_RADL` to `RSV_IRAP_12`, inclusive, `TemporalId` shall be equal to 0.

When `nal_unit_type` is equal to `STSA_NUT` and `vps_independent_layer_flag[ GeneralLayerIdx[ nuh_layer_id ] ]` is equal to 1, `TemporalId` shall not be equal to 0.

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

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

- If `nal_unit_type` is equal to `DCI_NUT`, `VPS_NUT`, or `SPS_NUT`, `TemporalId` shall be equal to 0 and the `TemporalId` of the AU containing the NAL unit shall be equal to 0.
- Otherwise, if `nal_unit_type` is equal to `PH_NUT`, `TemporalId` shall be equal to the `TemporalId` of the PU containing the NAL unit.
- Otherwise, if `nal_unit_type` is equal to `EOS_NUT` or `EOB_NUT`, `TemporalId` shall be equal to 0.
- Otherwise, if `nal_unit_type` is equal to `AUD_NUT`, `FD_NUT`, `PREFIX_SEI_NUT`, or `SUFFIX_SEI_NUT`, `TemporalId` shall be equal to the `TemporalId` of the AU containing the NAL unit.
- Otherwise, when `nal_unit_type` is equal to `PPS_NUT`, `PREFIX_APS_NUT`, or `SUFFIX_APS_NUT`, `TemporalId` shall be greater than or equal to the `TemporalId` of the PU containing the NAL unit.

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

`nal_unit_type` specifies the NAL unit type, i.e., the type of RBSP data structure contained in the NAL unit as specified in Table 2.

NAL units that have `nal_unit_type` in the range of `UNSPEC28`..`UNSPEC31`, inclusive, for which semantics are not specified, shall not affect the decoding process specified in this Specification.

NOTE – NAL unit types in the range of UNSPEC28..UNSPEC31 may be used as determined by the application. No decoding process for these values of `nal_unit_type` is specified in this Specification. Since different applications might use these NAL unit types for different purposes, particular care must be exercised in the design of encoders that generate NAL units with these `nal_unit_type` values, and in the design of decoders that interpret the content of NAL units with these `nal_unit_type` values. This Specification does not define any management for these values. These `nal_unit_type` values might only be suitable for use in contexts in which "collisions" of usage (i.e., different definitions of the meaning of the NAL unit content for the same `nal_unit_type` value) are unimportant, or not possible, or are managed – e.g., defined or managed in the controlling application or transport specification, or by controlling the environment in which bitstreams are distributed.

For purposes other than determining the amount of data in the decoding units of the bitstream, decoders shall ignore (remove from the bitstream and discard) the contents of all NAL units that use reserved values of `nal_unit_type`.

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

nal_unit_type	Name of nal_unit_type	Content of NAL unit and RBSP syntax structure	NAL unit type class
0	TRAIL_NUT	Coded slice of a trailing picture slice_layer_rbsp()	VCL
1	STSA_NUT	Coded slice of an STSA picture slice_layer_rbsp()	VCL
2	RADL_NUT	Coded slice of a RADL picture slice_layer_rbsp()	VCL
3	RASL_NUT	Coded slice of a RASL picture slice_layer_rbsp()	VCL
4.6	RSV_VCL_4.. RSV_VCL_6	Reserved non-IRAP VCL NAL unit types	VCL
7 8	IDR_W_RADL IDR_N_LP	Coded slice of an IDR picture slice_layer_rbsp()	VCL
9	CRA_NUT	Coded slice of a CRA picture silce_layer_rbsp()	VCL
10	GDR_NUT	Coded slice of a GDR picture slice_layer_rbsp()	VCL
11 12	RSV_IRAP_11 RSV_IRAP_12	Reserved IRAP VCL NAL unit types	VCL
13	DCI_NUT	Decoding capability information decoding_capability_information_rbsp()	non-VCL
14	VPS_NUT	Video parameter set video_parameter_set_rbsp()	non-VCL
15	SPS_NUT	Sequence parameter set seq_parameter_set_rbsp()	non-VCL
16	PPS_NUT	Picture parameter set pic_parameter_set_rbsp()	non-VCL

17	PREFIX_APS_NUT	Adaptation parameter set	non-VCL
18	SUFFIX_APS_NUT	adaptation_parameter_set_rbsp()	
19	PH_NUT	Picture header picture_header_rbsp()	non-VCL
20	AUD_NUT	AU delimiter access_unit_delimiter_rbsp()	non-VCL
21	EOS_NUT	End of sequence end_of_seq_rbsp()	non-VCL
22	EOB_NUT	End of bitstream end_of_bitstream_rbsp()	non-VCL
23	PREFIX_SEI_NUT	Supplemental enhancement information	non-VCL
24	SUFFIX_SEI_NUT	sei_rbsp()	
25	FD_NUT	Filler data filler_data_rbsp()	non-VCL
26	RSV_NVCL_26	Reserved non-VCL NAL unit types	non-VCL
27	RSV_NVCL_27		
28..31	UNSPEC_28.. UNSPEC_31	Unspecified non-VCL NAL unit types	non-VCL

Table 2

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

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

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

- If `mixed_nalu_types_in_pic_flag` is equal to 0, the value of `nal_unit_type` shall be the same for all coded slice NAL units of a picture. A picture or a PU is referred to as having the same NAL unit type as the coded slice NAL units of the picture or PU.
- Otherwise (`mixed_nalu_types_in_pic_flag` is equal to 1), the VCL NAL units of one or more subpictures of the picture all have a particular value of `nal_unit_type` equal to `STSA_NUT`, `RADL_NUT`, `RASL_NUT`, `IDR_W_RADL`, `IDR_N_LP`, or `CRA_NUT`, while the other VCL NAL units in the picture all have a different particular value of `nal_unit_type` equal to `TRAIL_NUT`, `RADL_NUT`, or `RASL_NUT`.

For a single-layer bitstream, the following constraints apply:

- Each picture, other than the first picture in the bitstream in decoding order, is considered to be associated with the previous IRAP picture in decoding order.
- When a picture is a leading picture of an IRAP picture, it shall be a RADL or RASL picture.
- When a picture is a trailing picture of an IRAP picture, it shall not be a RADL or RASL picture.
- 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 an IDR picture having `nal_unit_type` equal to `IDR_N_LP`.

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

- Any picture that precedes an IRAP picture in decoding order shall precede the IRAP picture in output order and shall precede any RADL picture associated with the IRAP picture in output order.
- Any RASL picture associated with a CRA picture shall precede any RADL picture associated with the CRA picture in output order.
- Any RASL picture associated with a CRA picture shall follow, in output order, any IRAP picture that precedes the CRA picture in decoding order.
- If `field_seq_flag` is equal to 0 and the current picture is a leading picture associated with an IRAP picture, it shall precede, in decoding order, all non-leading pictures that are associated with the same IRAP picture. Otherwise, let `picA` and `picB` be the first and the last leading pictures, in decoding order, associated with an IRAP picture, respectively, there shall be at most one non-leading picture preceding `picA` in decoding order, and there shall be no non-leading picture between `picA` and `picB` in decoding order.

It should be noted that generally, an Intra Random Access Point (IRAP) picture is a picture that does not refer to any pictures other than itself for prediction in its decoding process. In JVET-Q2001, an IRAP picture may be a clean random access (CRA) picture or an instantaneous decoder refresh (IDR) picture. In JVET-Q2001, the first picture in the bitstream in decoding order must be an IRAP or a gradual decoding refresh (GDR) picture. JVET-Q2001 describes the concept of a leading picture, which is a picture that precedes the associated IRAP picture in output order. JVET-Q2001 further describes the concept of a trailing picture which is a non-IRAP picture that follows the associated IRAP picture in output order. Trailing pictures associated with an IRAP picture also follow the IRAP picture in decoding order. For IDR pictures, there are no trailing pictures that require reference to a picture decoded prior to the IDR picture. JVET-Q2001 provides where a CRA picture may have leading pictures that follow the CRA picture in decoding order and contain inter picture prediction references to pictures decoded prior to the CRA picture. Thus, when the CRA picture is used as a random access point these leading pictures may not be decodable and are identified as random access skipped leading (RASL) pictures. The other type of picture that can follow an IRAP picture in decoding order and precede it in output order is the random access decodable leading (RADL) picture, which cannot contain references to any pictures that precede the IRAP picture in decoding order. A GDR picture, is a picture for which each VCL NAL unit has nal_unit_type equal to GDR_NUT. If the current picture is a GDR picture that is associated with a picture header which signals a syntax element recovery_poc_cnt and there is a picture picA that follows the current GDR picture in decoding order in the CLVS and that has PicOrderCntVal equal to the PicOrderCntVal of the current GDR picture plus the value of recovery_poc_cnt, the picture picA is referred to as the recovery point picture.

[0038] As provided in Table 2, a NAL unit may include a picture parameter set syntax structure. Table 3 illustrates the syntax of the picture parameter set syntax structure provided in JVET-Q2001.

[0039]

pic_parameter_set_rbsp() {	Descriptor
pps_pic_parameter_set_id	ue(v)
pps_seq_parameter_set_id	u(4)
mixed_nalu_types_in_pic_flag	u(1)
pic_width_in_luma_samples	ue(v)
pic_height_in_luma_samples	ue(v)
pps_conformance_window_flag	u(1)
if(pps_conformance_window_flag) {	
pps_conf_win_left_offset	ue(v)
pps_conf_win_right_offset	ue(v)
pps_conf_win_top_offset	ue(v)
pps_conf_win_bottom_offset	ue(v)
}	
scaling_window_explicit_signalling_flag	u(1)
if(scaling_window_explicit_signalling_flag) {	
scaling_win_left_offset	ue(v)
scaling_win_right_offset	ue(v)
scaling_win_top_offset	ue(v)
scaling_win_bottom_offset	ue(v)
}	
output_flag_present_flag	u(1)
subpic_id_mapping_in_pps_flag	u(1)
if(subpic_id_mapping_in_pps_flag) {	
pps_num_subpics_minus1	ue(v)
pps_subpic_id_len_minus1	ue(v)
for(i = 0; i <= pps_num_subpic_minus1; i++)	
pps_subpic_id[i]	u(v)
}	
no_pic_partition_flag	u(1)
if(!no_pic_partition_flag) {	

pps_log2_ctu_size_minus5	u(2)
num_exp_tile_columns_minus1	ue(v)
num_exp_tile_rows_minus1	ue(v)
for(i = 0; i <= num_exp_tile_columns_minus1; i++)	
tile_column_width_minus1[i]	ue(v)
for(i = 0; i <= num_exp_tile_rows_minus1; i++)	
tile_row_height_minus1[i]	ue(v)
if(NumTilesInPic > 1)	
rect_slice_flag	u(1)
if(rect_slice_flag)	
single_slice_per_subpic_flag	u(1)
if(rect_slice_flag && !single_slice_per_subpic_flag) {	
num_slices_in_pic_minus1	ue(v)
if(num_slices_in_pic_minus1 > 0)	
tile_idx_delta_present_flag	u(1)
for(i = 0; i < num_slices_in_pic_minus1; i++) {	
if(NumTileColumns > 1)	
slice_width_in_tiles_minus1[i]	ue(v)
if(NumTileRows > 1 && (tile_idx_delta_present_flag SliceTopLeftTileIdx[i] % NumTileColumns == 0))	
slice_height_in_tiles_minus1[i]	ue(v)
if(slice_width_in_tiles_minus1[i] == 0 && slice_height_in_tiles_minus1[i] == 0 && RowHeight[SliceTopLeftTileIdx[i] / NumTileColumns] > 1) {	
num_exp_slices_in_tile[i]	ue(v)
for(j = 0; j < num_exp_slices_in_tile[i]; j++)	
exp_slice_height_in_ctus_minus1[i][j]	ue(v)
i += NumSlicesInTile[i] - 1	
}	
if(tile_idx_delta_present_flag && i < num_slices_in_pic_minus1)	
tile_idx_delta[i]	se(v)
}	
}	

loop_filter_across_tiles_enabled_flag	u(1)
loop_filter_across_slices_enabled_flag	u(1)
}	
cabac_init_present_flag	u(1)
for(i = 0; i < 2; i++)	
num_ref_idx_default_active_minus1[i]	ue(v)
rpl1_idx_present_flag	u(1)
init_qp_minus26	se(v)
cu_qp_delta_enabled_flag	u(1)
pps_chroma_tool_offsets_present_flag	u(1)
if(pps_chroma_tool_offsets_present_flag) {	
pps_cb_qp_offset	se(v)
pps_cr_qp_offset	se(v)
pps_joint_cbr_qp_offset_present_flag	u(1)
if(pps_joint_cbr_qp_offset_present_flag)	
pps_joint_cbr_qp_offset_value	se(v)
pps_slice_chroma_qp_offsets_present_flag	u(1)
pps_cu_chroma_qp_offset_list_enabled_flag	u(1)
}	
if(pps_cu_chroma_qp_offset_list_enabled_flag) {	
chroma_qp_offset_list_len_minus1	ue(v)
for(i = 0; i <= chroma_qp_offset_list_len_minus1; i++) {	
cb_qp_offset_list[i]	se(v)
cr_qp_offset_list[i]	se(v)
if(pps_joint_cbr_qp_offset_present_flag)	
joint_cbr_qp_offset_list[i]	se(v)
}	
}	
pps_weighted_pred_flag	u(1)
pps_weighted_bipred_flag	u(1)
deblocking_filter_control_present_flag	u(1)
if(deblocking_filter_control_present_flag) {	

deblocking_filter_override_enabled_flag	u(1)
pps_deblocking_filter_disabled_flag	u(1)
if(!pps_deblocking_filter_disabled_flag) {	
pps_beta_offset_div2	se(v)
pps_tc_offset_div2	se(v)
pps_cb_beta_offset_div2	se(v)
pps_cb_tc_offset_div2	se(v)
pps_cr_beta_offset_div2	se(v)
pps_cr_tc_offset_div2	se(v)
}	
}	
rpl_info_in_ph_flag	u(1)
if(deblocking_filter_override_enabled_flag)	
dbf_info_in_ph_flag	u(1)
sao_info_in_ph_flag	u(1)
alf_info_in_ph_flag	u(1)
if((pps_weighted_pred_flag pps_weighted_bipred_flag) && rpl_info_in_ph_flag)	
wp_info_in_ph_flag	u(1)
qp_delta_info_in_ph_flag	u(1)
pps_ref_wraparound_enabled_flag	u(1)
if(pps_ref_wraparound_enabled_flag)	
pps_ref_wraparound_offset	ue(v)
picture_header_extension_present_flag	u(1)
slice_header_extension_present_flag	u(1)
pps_extension_flag	u(1)
if(pps_extension_flag)	
while(more_rbsp_data())	
pps_extension_data_flag	u(1)
rbsp_trailing_bits()	
}	

Table 3

With respect to Table 3, JVET-Q2001 provides the following semantics:

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

All PPS NAL units with a particular value of `pps_pic_parameter_set_id` within a PU shall have the same content.

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

PPS NAL units, regardless of the `nuh_layer_id` values, share the same value space of `pps_pic_parameter_set_id`.

Let `ppsLayerId` be the value of the `nuh_layer_id` of a particular PPS NAL unit, and `vclLayerId` be the value of the `nuh_layer_id` of a particular VCL NAL unit. The particular VCL NAL unit shall not refer to the particular PPS NAL unit unless `ppsLayerId` is less than or equal to `vclLayerId` and the layer with `nuh_layer_id` equal to `ppsLayerId` is included in at least one OLS that includes the layer with `nuh_layer_id` equal to `vclLayerId`.

`pps_seq_parameter_set_id` specifies the value of `sps_seq_parameter_set_id` for the SPS. The value of `pps_seq_parameter_set_id` shall be in the range of 0 to 15, inclusive. The value of `pps_seq_parameter_set_id` shall be the same in all PPSs that are referred to by coded pictures in a CLVS.

`mixed_nalu_types_in_pic_flag` equal to 1 specifies that each picture referring to the PPS has more than one VCL NAL unit and the VCL NAL units do not have the same value of `nalu_unit_type`. **`mixed_nalu_types_in_pic_flag`** equal to 0 specifies that each picture referring to the PPS has one or more VCL NAL units and the VCL NAL units of each picture referring to the PPS have the same value of `nalu_unit_type`.

When `no_mixed_nalu_types_in_pic_constraint_flag` is equal to 1, the value of `mixed_nalu_types_in_pic_flag` shall be equal to 0.

For each slice with a `nal_unit_type` value `nalUnitTypeA` in the range of `IDR_W_RADL` to `CRA_NUT`, inclusive, in a picture `picA` that also contains one or more slices with another value of `nal_unit_type` (i.e., the value of `mixed_nalu_types_in_pic_flag` for the picture `picA` is equal to 1), the following applies:

- The slice shall belong to a subpicture `subpicA` for which the value of the corresponding `subpic_treated_as_pic_flag[ i ]` is equal to 1.
- The slice shall not belong to a subpicture of `picA` containing VCL NAL units with `nal_unit_type` not equal to `nalUnitTypeA`.
- If `nalUnitTypeA` is equal to `CRA`, for all the following PUs following the current picture in the CLVS in decoding order and in output order, neither `RefPicList[ 0 ]` nor `RefPicList[ 1 ]` of a slice in `subpicA` in those PUs shall include any picture preceding `picA` in decoding order in an active entry.
- Otherwise (i.e., `nalUnitTypeA` is equal to `IDR_W_RADL` or `IDR_N_LP`), for all the PUs in the CLVS following the current picture in decoding order, neither `RefPicList[ 0 ]` nor `RefPicList[ 1 ]` of a slice in `subpicA` in those PUs shall include any picture preceding `picA` in decoding order in an active entry.

NOTE – `mixed_nalu_types_in_pic_flag` equal to 1 indicates that pictures referring to the PPS contain slices with different NAL unit types, e.g., coded pictures originating from a subpicture bitstream merging operation for which encoders have to ensure matching bitstream structure and further alignment of parameters of the original bitstreams. One example of such alignments is as follows: When the value of `sps_idr_rpl_present_flag` is equal to 0 and `mixed_nalu_types_in_pic_flag` is equal to 1, a picture referring to the PPS cannot have slices with `nal_unit_type` equal to `IDR_W_RADL` or `IDR_N_LP`.

`pic_width_in_luma_samples` specifies the width of each decoded picture referring to the PPS in units of luma samples. `pic_width_in_luma_samples` shall not be equal to 0, shall be an integer multiple of $\text{Max}(8, \text{MinCbSizeY})$, and shall be less than or equal to `pic_width_max_in_luma_samples`.

When `res_change_in_clvs_allowed_flag` equal to 0, the value of `pic_width_in_luma_samples` shall be equal to `pic_width_max_in_luma_samples`.

When `sps_ref_wraparound_enabled_flag` is equal to 1, the value of $(\text{CtbSizeY} / \text{MinCbSizeY} + 1)$ shall be less than or equal to the value of $(\text{pic_width_in_luma_samples} / \text{MinCbSizeY} - 1)$.

pic_height_in_luma_samples specifies the height of each decoded picture referring to the PPS in units of luma samples. **pic_height_in_luma_samples** shall not be equal to 0 and shall be an integer multiple of $\text{Max}(8, \text{MinCbSizeY})$, and shall be less than or equal to **pic_height_max_in_luma_samples**.

When **res_change_in_clvs_allowed_flag** equal to 0, the value of **pic_height_in_luma_samples** shall be equal to **pic_height_max_in_luma_samples**.

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

$$\text{PicWidthInCtbsY} = \text{Ceil}(\text{pic_width_in_luma_samples} \div \text{CtbSizeY})$$

$$\text{PicHeightInCtbsY} = \text{Ceil}(\text{pic_height_in_luma_samples} \div \text{CtbSizeY})$$

$$\text{PicSizeInCtbsY} = \text{PicWidthInCtbsY} * \text{PicHeightInCtbsY}$$

$$\text{PicWidthInMinCbsY} = \text{pic_width_in_luma_samples} / \text{MinCbSizeY}$$

$$\text{PicHeightInMinCbsY} = \text{pic_height_in_luma_samples} / \text{MinCbSizeY}$$

$$\text{PicSizeInMinCbsY} = \text{PicWidthInMinCbsY} * \text{PicHeightInMinCbsY}$$

$$\text{PicSizeInSamplesY} = \text{pic_width_in_luma_samples} * \text{pic_height_in_luma_samples}$$

$$\text{PicWidthInSamplesC} = \text{pic_width_in_luma_samples} / \text{SubWidthC}$$

$$\text{PicHeightInSamplesC} = \text{pic_height_in_luma_samples} / \text{SubHeightC}$$

pps_conformance_window_flag equal to 1 indicates that the conformance cropping window offset parameters follow next in the PPS. **pps_conformance_window_flag** equal to 0 indicates that the conformance cropping window offset parameters are not present in the PPS.

pps_conf_win_left_offset, **pps_conf_win_right_offset**, **pps_conf_win_top_offset**, and **pps_conf_win_bottom_offset** specify the samples of the pictures in the CLVS that are output from the decoding process, in terms of a rectangular region specified in picture coordinates for output. When **pps_conformance_window_flag** is equal to 0, the values of **pps_conf_win_left_offset**, **pps_conf_win_right_offset**, **pps_conf_win_top_offset**, and **pps_conf_win_bottom_offset** are inferred to be equal to 0.

The conformance cropping window contains the luma samples with horizontal picture coordinates from $\text{SubWidthC} * \text{pps_conf_win_left_offset}$ to $\text{pic_width_in_luma_samples} - (\text{SubWidthC} * \text{pps_conf_win_right_offset} + 1)$ and vertical picture coordinates from $\text{SubHeightC} * \text{pps_conf_win_top_offset}$ to $\text{pic_height_in_luma_samples} - (\text{SubHeightC} * \text{pps_conf_win_bottom_offset} + 1)$, inclusive.

The value of $\text{SubWidthC} * (\text{pps_conf_win_left_offset} + \text{pps_conf_win_right_offset})$ shall be less than $\text{pic_width_in_luma_samples}$, and the value of $\text{SubHeightC} * (\text{pps_conf_win_top_offset} + \text{pps_conf_win_bottom_offset})$ shall be less than $\text{pic_height_in_luma_samples}$.

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

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

Let ppsA and ppsB be any two PPSs referring to the same SPS. It is a requirement of bitstream conformance that, when ppsA and ppsB have the same the values of $\text{pic_width_in_luma_samples}$ and $\text{pic_height_in_luma_samples}$, respectively, ppsA and ppsB shall have the same values of $\text{pps_conf_win_left_offset}$, $\text{pps_conf_win_right_offset}$, $\text{pps_conf_win_top_offset}$, and $\text{pps_conf_win_bottom_offset}$, respectively.

When $\text{pic_width_in_luma_samples}$ is equal to $\text{pic_width_max_in_luma_samples}$ and $\text{pic_height_in_luma_samples}$ is equal to $\text{pic_height_max_in_luma_samples}$, it is a requirement of bitstream conformance that $\text{pps_conf_win_left_offset}$, $\text{pps_conf_win_right_offset}$, $\text{pps_conf_win_top_offset}$, and $\text{pps_conf_win_bottom_offset}$, are equal to $\text{sps_conf_win_left_offset}$, $\text{sps_conf_win_right_offset}$, $\text{sps_conf_win_top_offset}$, and $\text{sps_conf_win_bottom_offset}$, respectively.

scaling_window_explicit_signalling_flag equal to 1 specifies that the scaling window offset parameters are present in the PPS. **scaling_window_explicit_signalling_flag** equal to 0 specifies that the scaling window offset parameters are not present in the PPS. When **res_change_in_clvs_allowed_flag** is equal to 0, the value of **scaling_window_explicit_signalling_flag** shall be equal to 0.

scaling_win_left_offset, **scaling_win_right_offset**, **scaling_win_top_offset**, and **scaling_win_bottom_offset** specify the offsets that are applied to the picture size for scaling ratio calculation. When not present, the values of **scaling_win_left_offset**, **scaling_win_right_offset**, **scaling_win_top_offset**, and **scaling_win_bottom_offset** are inferred to be equal to **pps_conf_win_left_offset**, **pps_conf_win_right_offset**, **pps_conf_win_top_offset**, and **pps_conf_win_bottom_offset**, respectively.

The value of $\text{SubWidthC} * (\text{scaling_win_left_offset} + \text{scaling_win_right_offset})$ shall be less than **pic_width_in_luma_samples**, and the value of $\text{SubHeightC} * (\text{scaling_win_top_offset} + \text{scaling_win_bottom_offset})$ shall be less than **pic_height_in_luma_samples**.

The variables **PicOutputWidthL** and **PicOutputHeightL** are derived as follows:

$$\text{PicOutputWidthL} = \text{pic_width_in_luma_samples} - \text{SubWidthC} * (\text{scaling_win_right_offset} + \text{scaling_win_left_offset})$$

$$\text{PicOutputHeightL} = \text{pic_height_in_luma_samples} -$$

Let **refPicOutputWidthL** and **refPicOutputHeightL** be the **PicOutputWidthL** and **PicOutputHeightL**, respectively, of a reference picture of a current picture referring to this PPS. Is a requirement of bitstream conformance that all of the following conditions are satisfied:

- **PicOutputWidthL** * 2 shall be greater than or equal to **refPicWidthInLumaSamples**.
- **PicOutputHeightL** * 2 shall be greater than or equal to **refPicHeightInLumaSamples**.
- **PicOutputWidthL** shall be less than or equal to **refPicWidthInLumaSamples** * 8.
- **PicOutputHeightL** shall be less than or equal to **refPicHeightInLumaSamples** * 8.
- **PicOutputWidthL** * **pic_width_max_in_luma_samples** shall be greater than or equal to **refPicOutputWidthL** * (**pic_width_in_luma_samples** – **Max(8, MinCbSizeY)**).
- **PicOutputHeightL** * **pic_height_max_in_luma_samples** shall be greater than or equal to **refPicOutputHeightL** * (**pic_height_in_luma_samples** – **Max(8, MinCbSizeY)**).

output_flag_present_flag equal to 1 indicates that the **pic_output_flag** syntax element is present in PHs referring to the PPS. **output_flag_present_flag** equal to 0 indicates that the **pic_output_flag** syntax element is not present in PHs referring to the PPS.

subpic_id_mapping_in_pps_flag equal to 1 specifies that the subpicture ID mapping is signalled in the PPS. **subpic_id_mapping_in_pps_flag** equal to 0 specifies that the subpicture ID mapping is not signalled in the PPS. If **subpic_id_mapping_explicitly_signalled_flag** is 0 or **subpic_id_mapping_in_sps_flag** is equal to 1, the value of **subpic_id_mapping_in_pps_flag** shall be equal to 0. Otherwise (**subpic_id_mapping_explicitly_signalled_flag** is equal to 1 and **subpic_id_mapping_in_sps_flag** is equal to 0), the value of **subpic_id_mapping_in_pps_flag** shall be equal to 1.

pps_num_subpics_minus1 shall be equal to **sps_num_subpics_minus1**.

pps_subpic_id_len_minus1 shall be equal to **sps_subpic_id_len_minus1**.

pps_subpic_id[i] specifies the subpicture ID of the *i*-th subpicture. The length of the **pps_subpic_id[i]** syntax element is **pps_subpic_id_len_minus1** + 1 bits.

The variable **SubpicIdVal[i]**, for each value of *i* in the range of 0 to **sps_num_subpics_minus1**, inclusive, is derived as follows:

```
for( i = 0; i <= sps_num_subpics_minus1; i++)
    if( subpic_id_mapping_explicitly_signalled_flag )
        SubpicIdVal[ i ] = subpic_id_mapping_in_pps_flag ? pps_subpic_id[ i ] : sps_subpic_id[ i ]
    else
        SubpicIdVal[ i ] = i
```

(80)

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

- For any two different values of *i* and *j* in the range of 0 to **sps_num_subpics_minus1**, inclusive, **SubpicIdVal[i]** shall not be equal to **SubpicIdVal[j]**.
- When the current picture is not the first picture of the CLVS, for each value of *i* in the range of 0 to **sps_num_subpics_minus1**, inclusive, if the value of **SubpicIdVal[i]** is not equal to the value of **SubpicIdVal[i]** of the previous picture in decoding order in the same layer, the **nal_unit_type** for all coded slice NAL units of the subpicture in the current picture with subpicture index *i* shall be equal to a particular value in the range of **IDR_W_RADL** to **CRA_NUT**, inclusive.

no_pic_partition_flag equal to 1 specifies that no picture partitioning is applied to each picture referring to the PPS. **no_pic_partition_flag** equal to 0 specifies each picture referring to the PPS may be partitioned into more than one tile or slice.

It is a requirement of bitstream conformance that the value of **no_pic_partition_flag** shall be the same for all PPSs that are referred to by coded pictures within a CLVS.

It is a requirement of bitstream conformance that the value of **no_pic_partition_flag** shall not be equal to 1 when the value of **sps_num_subpics_minus1** + 1 is greater than 1.

pps_log2_ctu_size_minus5 plus 5 specifies the luma coding tree block size of each CTU. **pps_log2_ctu_size_minus5** shall be equal to **sps_log2_ctu_size_minus5**.

num_exp_tile_columns_minus1 plus 1 specifies the number of explicitly provided tile column widths. The value of **num_exp_tile_columns_minus1** shall be in the range of 0 to **PicWidthInCtbsY** - 1, inclusive. When **no_pic_partition_flag** is equal to 1, the value of **num_exp_tile_columns_minus1** is inferred to be equal to 0.

num_exp_tile_rows_minus1 plus 1 specifies the number of explicitly provided tile row heights. The value of **num_exp_tile_rows_minus1** shall be in the range of 0 to **PicHeightInCtbsY** - 1, inclusive. When **no_pic_partition_flag** is equal to 1, the value of **num_exp_tile_rows_minus1** is inferred to be equal to 0.

tile_column_width_minus1[i] plus 1 specifies the width of the i-th tile column in units of CTBs for i in the range of 0 to **num_exp_tile_columns_minus1** - 1, inclusive. **tile_column_width_minus1[num_exp_tile_columns_minus1]** is used to derive the width of the tile columns with index greater than or equal to **num_exp_tile_columns_minus1** as specified. The value of **tile_column_width_minus1[i]** shall be in the range of 0 to **PicWidthInCtbsY** - 1, inclusive. When not present, the value of **tile_column_width_minus1[0]** is inferred to be equal to **PicWidthInCtbsY** - 1.

tile_row_height_minus1[i] plus 1 specifies the height of the i-th tile row in units of CTBs for i in the range of 0 to **num_exp_tile_rows_minus1** - 1, inclusive. **tile_row_height_minus1[num_exp_tile_rows_minus1]** is used to derive the height of the tile rows with index greater than or equal to **num_exp_tile_rows_minus1** as specified. The value of **tile_row_height_minus1[i]** shall be in the range of 0 to **PicHeightInCtbsY** - 1, inclusive. When not present, the value of **tile_row_height_minus1[0]** is inferred to be equal to **PicHeightInCtbsY** - 1.

rect_slice_flag equal to 0 specifies that tiles within each slice are in raster scan order and the slice information is not signalled in PPS. **rect_slice_flag** equal to 1 specifies that tiles within each slice cover a rectangular region of the picture and the slice information is signalled in the PPS. When not present, **rect_slice_flag** is inferred to be equal to 1. When **subpic_info_present_flag** is equal to 1, the value of **rect_slice_flag** shall be equal to 1.

single_slice_per_subpic_flag equal to 1 specifies that each subpicture consists of one and only one rectangular slice. **single_slice_per_subpic_flag** equal to 0 specifies that each subpicture may consist of one or more rectangular slices. When not present, the value of **single_slice_per_subpic_flag** is inferred to be equal to 0.

num_slices_in_pic_minus1 plus 1 specifies the number of rectangular slices in each picture referring to the PPS. The value of **num_slices_in_pic_minus1** shall be in the range of 0 to **MaxSlicesPerPicture** - 1, inclusive, where **MaxSlicesPerPicture** is specified in Annex A. When **no_pic_partition_flag** is equal to 1, the value of **num_slices_in_pic_minus1** is inferred to be equal to 0. When **single_slice_per_subpic_flag** is equal to 1, the value of **num_slices_in_pic_minus1** is inferred to be equal to **sps_num_subpics_minus1**.

tile_idx_delta_present_flag equal to 0 specifies that **tile_idx_delta[i]** syntax elements are not present in the PPS and all pictures referring to the PPS are partitioned into rectangular slice rows and rectangular slice columns in slice raster order. **tile_idx_delta_present_flag** equal to 1 specifies that **tile_idx_delta[i]** syntax elements may be present in the PPS and all rectangular slices in pictures referring to the PPS are specified in the order indicated by the values of the **tile_idx_delta[i]** in increasing values of *i*. When not present, the value of **tile_idx_delta_present_flag** is inferred to be equal to 0.

slice_width_in_tiles_minus1[i] plus 1 specifies the width of the *i*-th rectangular slice in units of tile columns. The value of **slice_width_in_tiles_minus1[i]** shall be in the range of 0 to **NumTileColumns** - 1, inclusive.

When *i* is less than **num_slices_in_pic_minus1** and **NumTileColumns** is equal to 1, the value of **slice_width_in_tiles_minus1[i]** is inferred to be equal to 0.

slice_height_in_tiles_minus1[i] plus 1 specifies the height of the *i*-th rectangular slice in units of tile rows when **num_exp_slices_in_tile[i]** is equal to 0. The value of **slice_height_in_tiles_minus1[i]** shall be in the range of 0 to **NumTileRows** - 1, inclusive.

When *i* is less than **num_slices_in_pic_minus1** and **slice_height_in_tiles_minus1[i]** is not present, it is inferred to be equal to **NumTileRows** - 1. When *i* is equal to **num_slices_in_pic_minus1**, **slice_height_in_tiles_minus1[i]** is inferred to be equal to **slice_height_in_tiles_minus1[i - 1]**.

num_exp_slices_in_tile[i] specifies the number of explicitly provided slice heights for the slices in the tile containing the *i*-th slice (i.e., the tile with tile index equal to **SliceTopLeftTileIdx[i]**). The value of **num_exp_slices_in_tile[i]** shall be in the range of 0 to **RowHeight[SliceTopLeftTileIdx[i] / NumTileColumns] - 1**, inclusive. When not present, the value of **num_exp_slices_in_tile[i]** is inferred to be equal to 0.

NOTE – If **num_exp_slices_in_tile[i]** is equal to 0, the tile containing the *i*-th slice is not split into multiple slices. Otherwise (**num_exp_slices_in_tile[i]** is greater than 0), the tile containing the *i*-th slice may or may not be split into multiple slices.

exp_slice_height_in_ctus_minus1[i][j] plus 1 specifies the height of the *j*-th rectangular slice in the tile containing the *i*-th slice in units of CTU rows. The value of **exp_slice_height_in_ctus_minus1[i][j]** shall be in the range of 0 to **RowHeight[SliceTopLeftTileIdx[i] / NumTileColumns] - 1**, inclusive.

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

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

loop_filter_across_slices_enabled_flag equal to 1 specifies that in-loop filtering operations may be performed across slice boundaries in pictures referring to the PPS. **loop_filter_across_slices_enabled_flag** equal to 0 specifies that in-loop filtering operations are not performed across slice boundaries in pictures referring to the PPS. The in-loop filtering operations include the deblocking filter, sample adaptive offset filter, and adaptive loop filter operations. When not present, the value of **loop_filter_across_slices_enabled_flag** is inferred to be equal to 0.

cabac_init_present_flag equal to 1 specifies that **cabac_init_flag** is present in slice headers referring to the PPS. **cabac_init_present_flag** equal to 0 specifies that **cabac_init_flag** is not present in slice headers referring to the PPS.

num_ref_idx_default_active_minus1[i] plus 1, when *i* is equal to 0, specifies the inferred value of the variable **NumRefIdxActive[0]** for P or B slices with **num_ref_idx_active_override_flag** equal to 0, and, when *i* is equal to 1, specifies the inferred value of **NumRefIdxActive[1]** for B slices with **num_ref_idx_active_override_flag** equal to 0. The value of **num_ref_idx_default_active_minus1[i]** shall be in the range of 0 to 14, inclusive.

rpl1_idx_present_flag equal to 0 specifies that **rpl_sps_flag[1]** and **rpl_idx[1]** are not present in the PH syntax structures or the slice headers for pictures referring to the PPS. **rpl1_idx_present_flag** equal to 1 specifies that **rpl_sps_flag[1]** and **rpl_idx[1]** may be present in the PH syntax structures or the slice headers for pictures referring to the PPS.

init_qp_minus26 plus 26 specifies the initial value of **SliceQpY** for each slice referring to the PPS. The initial value of **SliceQpY** is modified at the picture level when a non-zero value of **ph_qp_delta** is decoded or at the slice level when a non-zero value of **slice_qp_delta** is decoded. The value of **init_qp_minus26** shall be in the range of $-(26 + QpBdOffset)$ to +37, inclusive.

cu_qp_delta_enabled_flag equal to 1 specifies that the **ph_cu_qp_delta_subdiv_intra_slice** and **ph_cu_qp_delta_subdiv_inter_slice** syntax elements are present in PHs referring to the PPS and **cu_qp_delta_abs** may be present in the transform unit syntax. **cu_qp_delta_enabled_flag** equal to 0 specifies that the **ph_cu_qp_delta_subdiv_intra_slice** and **ph_cu_qp_delta_subdiv_inter_slice** syntax elements are not present in PHs referring to the PPS and **cu_qp_delta_abs** is not present in the transform unit syntax.

pps_chroma_tool_offsets_present_flag equal to 1 specifies that chroma tool offsets related syntax elements are present in the PPS RBSP syntax structure. **pps_chroma_tool_offsets_present_flag** equal to 0 specifies that chroma tool offsets related syntax elements are not present in the PPS RBSP syntax structure. When **ChromaArrayType** is equal to 0, the value of **pps_chroma_tool_offsets_present_flag** shall be equal to 0.

pps_cb_qp_offset and **pps_cr_qp_offset** specify the offsets to the luma quantization parameter Qp'_Y used for deriving Qp'_{Cb} and Qp'_{Cr} , respectively. The values of **pps_cb_qp_offset** and **pps_cr_qp_offset** shall be in the range of -12 to +12, inclusive. When **ChromaArrayType** is equal to 0, **pps_cb_qp_offset** and **pps_cr_qp_offset** are not used in the decoding process and decoders shall ignore their value. When not present, the values of **pps_cb_qp_offset** and **pps_cr_qp_offset** are inferred to be equal to 0.

pps_joint_cbr_qp_offset_present_flag equal to 1 specifies that **pps_joint_cbr_qp_offset_value** and **joint_cbr_qp_offset_list[i]** are present in the PPS RBSP syntax structure. **pps_joint_cbr_qp_offset_present_flag** equal to 0 specifies that **pps_joint_cbr_qp_offset_value** and **joint_cbr_qp_offset_list[i]** are not present in the PPS RBSP syntax structure. When **ChromaArrayType** is equal to 0 or **sps_joint_cbr_enabled_flag** is equal to 0, the value of **pps_joint_cbr_qp_offset_present_flag** shall be equal to 0. When not present, the value of **pps_joint_cbr_qp_offset_present_flag** is inferred to be equal to 0.

pps_joint_cbr_qp_offset_value specifies the offset to the luma quantization parameter Qp'_Y used for deriving Qp'_{CbCr} . The value of **pps_joint_cbr_qp_offset_value** shall be in the range of -12 to +12, inclusive. When **ChromaArrayType** is equal to 0 or **sps_joint_cbr_enabled_flag** is equal to 0, **pps_joint_cbr_qp_offset_value** is not used in the decoding process and decoders shall ignore its value. When **pps_joint_cbr_qp_offset_present_flag** is equal to 0, **pps_joint_cbr_qp_offset_value** is not present and is inferred to be equal to 0.

pps_slice_chroma_qp_offsets_present_flag equal to 1 specifies that the **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 specifies that the **slice_cb_qp_offset** and **slice_cr_qp_offset** syntax elements are not present in the associated slice headers. When not present, the value of **pps_slice_chroma_qp_offsets_present_flag** is inferred to be equal to 0.

pps_cu_chroma_qp_offset_list_enabled_flag equal to 1 specifies that the **ph_cu_chroma_qp_offset_subdiv_intra_slice** and **ph_cu_chroma_qp_offset_subdiv_inter_slice** syntax elements are present in PHs referring to the PPS and **cu_chroma_qp_offset_flag** may be present in the transform unit syntax and the palette coding syntax. **pps_cu_chroma_qp_offset_list_enabled_flag** equal to 0 specifies that the **ph_cu_chroma_qp_offset_subdiv_intra_slice** and **ph_cu_chroma_qp_offset_subdiv_inter_slice** syntax elements are not present in PHs referring to the PPS and the **cu_chroma_qp_offset_flag** is not present in the transform unit syntax and the palette coding syntax. When not present, the value of **pps_cu_chroma_qp_offset_list_enabled_flag** is inferred to be equal to 0.

chroma_qp_offset_list_len_minus1 plus 1 specifies the number of **cb_qp_offset_list[i]**, **cr_qp_offset_list[i]**, and **joint_cbr_qp_offset_list[i]**, syntax elements that are present in the PPS RBSP syntax structure. The value of **chroma_qp_offset_list_len_minus1** shall be in the range of 0 to 5, inclusive.

cb_qp_offset_list[i], **cr_qp_offset_list[i]**, and **joint_cbr_qp_offset_list[i]**, specify offsets used in the derivation of Qp'_{cb} , Qp'_{cr} , and Qp'_{cbr} , respectively. The values of **cb_qp_offset_list[i]**, **cr_qp_offset_list[i]**, and **joint_cbr_qp_offset_list[i]** shall be in the range of -12 to +12, inclusive. When **pps_joint_cbr_qp_offset_present_flag** is equal to 0, **joint_cbr_qp_offset_list[i]** is not present and it is inferred to be equal to 0.

pps_weighted_pred_flag equal to 0 specifies that weighted prediction is not applied to P slices referring to the PPS. **pps_weighted_pred_flag** equal to 1 specifies that weighted prediction is applied to P slices referring to the PPS. When **sps_weighted_pred_flag** is equal to 0, the value of **pps_weighted_pred_flag** shall be equal to 0.

pps_weighted_bipred_flag equal to 0 specifies that explicit weighted prediction is not applied to B slices referring to the PPS. **pps_weighted_bipred_flag** equal to 1 specifies that explicit weighted prediction is applied to B slices referring to the PPS. When **sps_weighted_bipred_flag** is equal to 0, the value of **pps_weighted_bipred_flag** shall be equal to 0.

deblocking_filter_control_present_flag equal to 1 specifies the presence of deblocking filter control syntax elements in the PPS. **deblocking_filter_control_present_flag** equal to 0 specifies the absence of deblocking filter control syntax elements in the PPS.

deblocking_filter_override_enabled_flag equal to 1 specifies the presence of **ph_deblocking_filter_override_flag** in the PHs referring to the PPS or **slice_deblocking_filter_override_flag** in the slice headers referring to the PPS. **deblocking_filter_override_enabled_flag** equal to 0 specifies the absence of **ph_deblocking_filter_override_flag** in PHs referring to the PPS or **slice_deblocking_filter_override_flag** in slice headers referring to the PPS. When not present, the value of **deblocking_filter_override_enabled_flag** is inferred to be equal to 0.

pps_deblocking_filter_disabled_flag equal to 1 specifies that the operation of deblocking filter is not applied for slices referring to the PPS in which **slice_deblocking_filter_disabled_flag** is not present. **pps_deblocking_filter_disabled_flag** equal to 0 specifies that the operation of the deblocking filter is applied for slices referring to the PPS in which **slice_deblocking_filter_disabled_flag** is not present. When not present, the value of **pps_deblocking_filter_disabled_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 to the luma component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of **pps_beta_offset_div2** and **pps_tc_offset_div2** shall both be in the range of -12 to 12, inclusive. When not present, the values of **pps_beta_offset_div2** and **pps_tc_offset_div2** are both inferred to be equal to 0.

pps_cb_beta_offset_div2 and **pps_cb_tc_offset_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of **pps_cb_beta_offset_div2** and **pps_cb_tc_offset_div2** shall both be in the range of -12 to 12, inclusive. When not present, the values of **pps_cb_beta_offset_div2** and **pps_cb_tc_offset_div2** are both inferred to be equal to 0.

pps_cr_beta_offset_div2 and **pps_cr_tc_offset_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of **pps_cr_beta_offset_div2** and **pps_cr_tc_offset_div2** shall both be in the range of -12 to 12, inclusive. When not present, the values of **pps_cr_beta_offset_div2** and **pps_cr_tc_offset_div2** are both inferred to be equal to 0.

rpl_info_in_ph_flag equal to 1 specifies that reference picture list information is present in the PH syntax structure

and not present in slice headers referring to the PPS that do not contain a PH syntax structure. **rpl_info_in_ph_flag** equal to 0 specifies that reference picture list information is not present in the PH syntax structure and may be present in slice headers referring to the PPS that do not contain a PH syntax structure.

dbf_info_in_ph_flag equal to 1 specifies that deblocking filter information is present in the PH syntax structure and not present in slice headers referring to the PPS that do not contain a PH syntax structure. **dbf_info_in_ph_flag** equal to 0 specifies that deblocking filter information is not present in the PH syntax structure and may be present in slice headers referring to the PPS that do not contain a PH syntax structure. When not present, the value of **dbf_info_in_ph_flag** is inferred to be equal to 0.

sao_info_in_ph_flag equal to 1 specifies that SAO filter information is present in the PH syntax structure and not present in slice headers referring to the PPS that do not contain a PH syntax structure. **sao_info_in_ph_flag** equal to 0 specifies that SAO filter information is not present in the PH syntax structure and may be present in slice headers referring to the PPS that do not contain a PH syntax structure.

alf_info_in_ph_flag equal to 1 specifies that ALF information is present in the PH syntax structure and not present in slice headers referring to the PPS that do not contain a PH syntax structure. **alf_info_in_ph_flag** equal to 0 specifies that ALF information is not present in the PH syntax structure and may be present in slice headers referring to the PPS that do not contain a PH syntax structure.

wp_info_in_ph_flag equal to 1 specifies that weighted prediction information may be present in the PH syntax structure and not present in slice headers referring to the PPS that do not contain a PH syntax structure. **wp_info_in_ph_flag** equal to 0 specifies that weighted prediction information is not present in the PH syntax structure and may be present in slice headers referring to the PPS that do not contain a PH syntax structure. When not present, the value of **wp_info_in_ph_flag** is inferred to be equal to 0.

qp_delta_info_in_ph_flag equal to 1 specifies that QP delta information is present in the PH syntax structure and not present in slice headers referring to the PPS that do not contain a PH syntax structure. **qp_delta_info_in_ph_flag** equal to 0 specifies that QP delta information is not present in the PH syntax structure and may be present in slice headers referring to the PPS that do not contain a PH syntax structure.

pps_ref_wraparound_enabled_flag equal to 1 specifies that horizontal wrap-around motion compensation is applied in inter prediction. **pps_ref_wraparound_enabled_flag** equal to 0 specifies that horizontal wrap-around motion compensation is not applied. When the value of $\text{CtbSizeY} / \text{MinCbSizeY} + 1$ is greater than $\text{pic_width_in_luma_samples} / \text{MinCbSizeY} - 1$, the value of **pps_ref_wraparound_enabled_flag** shall be equal to 0. When **sps_ref_wraparound_enabled_flag** is equal to 0, the value of **pps_ref_wraparound_enabled_flag** shall be equal to 0.

pps_ref_wraparound_offset plus $(\text{CtbSizeY} / \text{MinCbSizeY}) + 2$ specifies the offset used for computing the horizontal wrap-around position in units of **MinCbSizeY** luma samples. The value of **pps_ref_wraparound_offset** shall be in the range of 0 to $(\text{pic_width_in_luma_samples} / \text{MinCbSizeY}) - (\text{CtbSizeY} / \text{MinCbSizeY}) - 2$, inclusive. The variable **PpsRefWraparoundOffset** is set equal to **pps_ref_wraparound_offset** + $(\text{CtbSizeY} / \text{MinCbSizeY}) + 2$.

picture_header_extension_present_flag equal to 0 specifies that no PH extension syntax elements are present in PHs referring to the PPS. **picture_header_extension_present_flag** equal to 1 specifies that PH extension syntax elements are present in PHs referring to the PPS. **picture_header_extension_present_flag** shall be equal to 0 in bitstreams conforming to this version of this Specification.

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

pps_extension_flag equal to 0 specifies that no **pps_extension_data_flag** syntax elements are present in the PPS RBSP syntax structure. **pps_extension_flag** equal to 1 specifies that there are **pps_extension_data_flag** syntax elements present in the PPS RBSP syntax structure.

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

As provided in Table 2, a NAL unit may include a **picture_header_rbsp()** which includes a picture header syntax structure, **picture_header_structure()**. Table 4 illustrates the syntax of the picture header syntax structure provided in JVET-Q2001.

[0040]

picture_header_structure() {	Descriptor
gdr_or_irap_pic_flag	u(1)
if(gdr_or_irap_pic_flag)	
gdr_pic_flag	u(1)
ph_inter_slice_allowed_flag	u(1)
if(ph_inter_slice_allowed_flag)	
ph_intra_slice_allowed_flag	u(1)
non_reference_picture_flag	u(1)
ph_pic_parameter_set_id	ue(v)
ph_pic_order_cnt_lsb	u(v)
if(gdr_or_irap_pic_flag)	
no_output_of_prior_pics_flag	u(1)
if(gdr_pic_flag)	
recovery_poc_cnt	ue(v)
for(i = 0; i < NumExtraPhBits; i++)	
ph_extra_bit[i]	u(1)
if(sps_poc_msb_flag) {	
ph_poc_msb_present_flag	u(1)
if(ph_poc_msb_present_flag)	
poc_msb_val	u(v)
}	
if(sps_alf_enabled_flag && alf_info_in_ph_flag) {	
ph_alf_enabled_flag	u(1)
if(ph_alf_enabled_flag) {	
ph_num_alf_aps_ids_luma	u(3)
for(i = 0; i < ph_num_alf_aps_ids_luma; i++)	
ph_alf_aps_id_luma[i]	u(3)
if(ChromaArrayType != 0)	
ph_alf_chroma_idc	u(2)
if(ph_alf_chroma_idc > 0)	

ph_alf_aps_id_chroma	u(3)
if(sps_ccalf_enabled_flag) {	
ph_cc_alf_cb_enabled_flag	u(1)
if(ph_cc_alf_cb_enabled_flag)	
ph_cc_alf_cb_aps_id	u(3)
ph_cc_alf_cr_enabled_flag	u(1)
if(ph_cc_alf_cr_enabled_flag)	
ph_cc_alf_cr_aps_id	u(3)
}	
}	
}	
if(sps_lmcs_enabled_flag) {	
ph_lmcs_enabled_flag	u(1)
if(ph_lmcs_enabled_flag) {	
ph_lmcs_aps_id	u(2)
if(ChromaArrayType != 0)	
ph_chroma_residual_scale_flag	u(1)
}	
}	
if(sps_explicit_scaling_list_enabled_flag) {	
ph_explicit_scaling_list_enabled_flag	u(1)
if(ph_explicit_scaling_list_enabled_flag)	
ph_scaling_list_aps_id	u(3)
}	
if(sps_virtual_boundaries_enabled_flag && !sps_virtual_boundaries_present_flag) {	
ph_virtual_boundaries_present_flag	u(1)
if(ph_virtual_boundaries_present_flag) {	
ph_num_ver_virtual_boundaries	u(2)
for(i = 0; i < ph_num_ver_virtual_boundaries; i++)	
ph_virtual_boundaries_pos_x[i]	u(13)
ph_num_hor_virtual_boundaries	u(2)
for(i = 0; i < ph_num_hor_virtual_boundaries; i++)	

ph_virtual_boundaries_pos_y[i]	u(13)
}	
}	
if(output_flag_present_flag)	
pic_output_flag	u(1)
if(rpl_info_in_ph_flag)	
ref_pic_lists()	
if(partition_constraints_override_enabled_flag)	
partition_constraints_override_flag	u(1)
if(ph_intra_slice_allowed_flag) {	
if(partition_constraints_override_flag) {	
ph_log2_diff_min_qt_min_cb_intra_slice_luma	ue(v)
ph_max_mtt_hierarchy_depth_intra_slice_luma	ue(v)
if(ph_max_mtt_hierarchy_depth_intra_slice_luma != 0) {	
ph_log2_diff_max_bt_min_qt_intra_slice_luma	ue(v)
ph_log2_diff_max_tt_min_qt_intra_slice_luma	ue(v)
}	
if(qtbtt_dual_tree_intra_flag) {	
ph_log2_diff_min_qt_min_cb_intra_slice_chroma	ue(v)
ph_max_mtt_hierarchy_depth_intra_slice_chroma	ue(v)
if(ph_max_mtt_hierarchy_depth_intra_slice_chroma != 0) {	
ph_log2_diff_max_bt_min_qt_intra_slice_chroma	ue(v)
ph_log2_diff_max_tt_min_qt_intra_slice_chroma	ue(v)
}	
}	
}	
if(cu_qp_delta_enabled_flag)	
ph_cu_qp_delta_subdiv_intra_slice	ue(v)
if(pps_cu_chroma_qp_offset_list_enabled_flag)	
ph_cu_chroma_qp_offset_subdiv_intra_slice	ue(v)
}	
if(ph_inter_slice_allowed_flag) {	

if(partition_constraints_override_flag) {	
ph_log2_diff_min_qt_min_cb_inter_slice	ue(v)
ph_max_mtt_hierarchy_depth_inter_slice	ue(v)
if(ph_max_mtt_hierarchy_depth_inter_slice != 0) {	
ph_log2_diff_max_bt_min_qt_inter_slice	ue(v)
ph_log2_diff_max_tt_min_qt_inter_slice	ue(v)
}	
}	
if(cu_qp_delta_enabled_flag)	
ph_cu_qp_delta_subdiv_inter_slice	ue(v)
if(pps_cu_chroma_qp_offset_list_enabled_flag)	
ph_cu_chroma_qp_offset_subdiv_inter_slice	ue(v)
if(sps_temporal_mvp_enabled_flag) {	
ph_temporal_mvp_enabled_flag	u(1)
if(ph_temporal_mvp_enabled_flag && rpl_info_in_ph_flag) {	
ph_collocated_from_l0_flag	u(1)
if((ph_collocated_from_l0_flag && num_ref_entries[0][RplIdx[0]] > 1) (!ph_collocated_from_l0_flag && num_ref_entries[1][RplIdx[1]] > 1))	
ph_collocated_ref_idx	ue(v)
}	
}	
mvd_l1_zero_flag	u(1)
if(sps_fpel_mmvd_enabled_flag)	
ph_fpel_mmvd_enabled_flag	u(1)
if(sps_bdof_pic_present_flag)	
ph_disable_bdof_flag	u(1)
if(sps_dmvr_pic_present_flag)	
ph_disable_dmvr_flag	u(1)
if(sps_prof_pic_present_flag)	
ph_disable_prof_flag	u(1)
if((pps_weighted_pred_flag pps_weighted_bipred_flag) &&	

wp_info_in_ph_flag)	
pred_weight_table()	
}	
if(qp_delta_info_in_ph_flag)	
ph_qp_delta	se(v)
if(sps_joint_cbr_enabled_flag)	
ph_joint_cbr_sign_flag	u(1)
if(sps_sao_enabled_flag && sao_info_in_ph_flag) {	
ph_sao_luma_enabled_flag	u(1)
if(ChromaArrayType != 0)	
ph_sao_chroma_enabled_flag	u(1)
}	
if(sps_dep_quant_enabled_flag)	
ph_dep_quant_enabled_flag	u(1)
if(sps_sign_data_hiding_enabled_flag && !ph_dep_quant_enabled_flag)	
pic_sign_data_hiding_enabled_flag	u(1)
if(deblocking_filter_override_enabled_flag && dbf_info_in_ph_flag) {	
ph_deblocking_filter_override_flag	u(1)
if(ph_deblocking_filter_override_flag) {	
ph_deblocking_filter_disabled_flag	u(1)
if(!ph_deblocking_filter_disabled_flag) {	
ph_beta_offset_div2	se(v)
ph_tc_offset_div2	se(v)
ph_cb_beta_offset_div2	se(v)
ph_cb_tc_offset_div2	se(v)
ph_cr_beta_offset_div2	se(v)
ph_cr_tc_offset_div2	se(v)
}	
}	
}	
if(picture_header_extension_present_flag) {	
ph_extension_length	ue(v)

for(i = 0; i < ph_extension_length; i++)	
ph_extension_data_byte[i]	u(8)
}	
}	

Table 4

With respect to Table 4, JVET-Q2001 provides the following semantics:

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

gdr_or_irap_pic_flag equal to 1 specifies that the current picture is a GDR or IRAP picture. **gdr_or_irap_pic_flag** equal to 0 specifies that the current picture may or may not be a GDR or IRAP picture.

gdr_pic_flag equal to 1 specifies the picture associated with the PH is a GDR picture. **gdr_pic_flag** equal to 0 specifies that the picture associated with the PH is not a GDR picture. When not present, the value of **gdr_pic_flag** is inferred to be equal to 0. When **gdr_enabled_flag** is equal to 0, the value of **gdr_pic_flag** shall be equal to 0.

NOTE – When **gdr_or_irap_pic_flag** is equal to 1 and **gdr_pic_flag** is equal to 0, the picture associated with the PH is an IRAP picture.

ph_inter_slice_allowed_flag equal to 0 specifies that all coded slices of the picture have **slice_type** equal to 2. **ph_inter_slice_allowed_flag** equal to 1 specifies that there may or may not be one or more coded slices in the picture that have **slice_type** equal to 0 or 1.

ph_intra_slice_allowed_flag equal to 0 specifies that all coded slices of the picture have **slice_type** equal to 0 or 1. **ph_intra_slice_allowed_flag** equal to 1 specifies that there may or may not be one or more coded slices in the picture that have **slice_type** equal to 2. When not present, the value of **ph_intra_slice_allowed_flag** is inferred to be equal to 1.

NOTE – For bitstreams that are supposed to work subpicture based bitstream merging without the need of changing PH NAL units, the encoder is expected to set the values of both **ph_inter_slice_allowed_flag** and **ph_intra_slice_allowed_flag** equal to 1.

non_reference_picture_flag equal to 1 specifies the picture associated with the PH is never used as a reference picture. **non_reference_picture_flag** equal to 0 specifies the picture associated with the PH may or may not be used as a reference picture.

ph_pic_parameter_set_id specifies the value of **pps_pic_parameter_set_id** for the PPS in use. The value of **ph_pic_parameter_set_id** shall be in the range of 0 to 63, inclusive. It is a requirement of bitstream conformance that the value of **TemporalId** of the PH shall be greater than or equal to the value of **TemporalId** of the PPS that has **pps_pic_parameter_set_id** equal to **ph_pic_parameter_set_id**.

ph_pic_order_cnt_lsb specifies the picture order count modulo **MaxPicOrderCntLsb** for the current picture. The length of the **ph_pic_order_cnt_lsb** syntax element is $\log_2 \text{max_pic_order_cnt_lsb_minus4} + 4$ bits. The value of the **ph_pic_order_cnt_lsb** shall be in the range of 0 to **MaxPicOrderCntLsb** - 1, inclusive.

no_output_of_prior_pics_flag affects the output of previously-decoded pictures in the DPB after the decoding of a CLVSS picture that is not the first picture in the bitstream as specified.

recovery_poc_cnt specifies the recovery point of decoded pictures in output order. If the current picture is a GDR picture that is associated with the PH, and there is a picture **picA** that follows the current GDR picture in decoding order in the CLVS that has **PicOrderCntVal** equal to the **PicOrderCntVal** of the current GDR picture plus the value of **recovery_poc_cnt**, the picture **picA** is referred to as the recovery point picture. Otherwise, the first picture in output order that has **PicOrderCntVal** greater than the **PicOrderCntVal** of the current picture plus the value of **recovery_poc_cnt** is referred to as the recovery point picture. The recovery point picture shall not precede the current GDR picture in decoding order. The value of **recovery_poc_cnt** shall be in the range of 0 to **MaxPicOrderCntLsb** - 1, inclusive.

When the current picture is a GDR picture, the variable **RpPicOrderCntVal** is derived as follows:

$$\text{RpPicOrderCntVal} = \text{PicOrderCntVal} + \text{recovery_poc_cnt}$$

NOTE – When **gdr_enabled_flag** is equal to 1 and **PicOrderCntVal** of the current picture is greater than or equal to **RpPicOrderCntVal** of the associated GDR picture, the current and subsequent decoded pictures in output order are exact match to the corresponding pictures produced by starting the decoding process from the previous IRAP picture, when present, preceding the associated GDR picture in decoding order.

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

ph_poc_msb_present_flag equal to 1 specifies that the syntax element **poc_msb_val** is present in the PH. **ph_poc_msb_present_flag** equal to 0 specifies that the syntax element **poc_msb_val** is not present in the PH. When **vps_independent_layer_flag[GeneralLayerIdx[nuh_layer_id]]** is equal to 0 and there is a picture in the current AU in a reference layer of the current layer, the value of **ph_poc_msb_present_flag** shall be equal to 0.

poc_msb_val specifies the POC MSB value of the current picture. The length of the syntax element poc_msb_val is poc_msb_len_minus1 + 1 bits.

ph_alf_enabled_flag equal to 1 specifies that adaptive loop filter is enabled for all slices associated with the PH and may be applied to Y, Cb, or Cr colour component in the slices. ph_alf_enabled_flag equal to 0 specifies that adaptive loop filter may be disabled for one, or more, or all slices associated with the PH. When not present, ph_alf_enabled_flag is inferred to be equal to 0.

ph_num_alf_aps_ids_luma specifies the number of ALF APSs that the slices associated with the PH refers to.

ph_alf_aps_id_luma[i] specifies the adaptation_parameter_set_id of the i-th ALF APS that the luma component of the slices associated with the PH refers to.

The value of alf_luma_filter_signal_flag of the APS NAL unit having aps_params_type equal to ALF_APS and adaptation_parameter_set_id equal to ph_alf_aps_id_luma[i] shall be equal to 1.

The TemporalId of the APS NAL unit having aps_params_type equal to ALF_APS and adaptation_parameter_set_id equal to ph_alf_aps_id_luma[i] shall be less than or equal to the TemporalId of the picture associated with the PH.

ph_alf_chroma_idc equal to 0 specifies that the adaptive loop filter is not applied to Cb and Cr colour components. ph_alf_chroma_idc equal to 1 indicates that the adaptive loop filter is applied to the Cb colour component. ph_alf_chroma_idc equal to 2 indicates that the adaptive loop filter is applied to the Cr colour component. ph_alf_chroma_idc equal to 3 indicates that the adaptive loop filter is applied to Cb and Cr colour components. When ph_alf_chroma_idc is not present, it is inferred to be equal to 0.

ph_alf_aps_id_chroma specifies the adaptation_parameter_set_id of the ALF APS that the chroma component of the slices associated with the PH refers to.

The value of alf_chroma_filter_signal_flag of the APS NAL unit having aps_params_type equal to ALF_APS and adaptation_parameter_set_id equal to ph_alf_aps_id_chroma shall be equal to 1.

The TemporalId of the APS NAL unit having aps_params_type equal to ALF_APS and adaptation_parameter_set_id equal to ph_alf_aps_id_chroma shall be less than or equal to the TemporalId of the picture associated with the PH.

ph_cc_alf_cb_enabled_flag equal to 1 specifies that cross-component filter for Cb colour component is enabled for all slices associated with the PH and may be applied to Cb colour component in the slices. **ph_cc_alf_cb_enabled_flag** equal to 0 specifies that cross-component filter for Cb colour component may be disabled for one, or more, or all slices associated with the PH. When not present, **ph_cc_alf_cb_enabled_flag** is inferred to be equal to 0.

ph_cc_alf_cb_aps_id specifies the **adaptation_parameter_set_id** of the ALF APS that the Cb colour component of the slices associated with the PH refers to.

The value of **alf_cc_cb_filter_signal_flag** of the APS NAL unit having **aps_params_type** equal to **ALF_APS** and **adaptation_parameter_set_id** equal to **ph_cc_alf_cb_aps_id** shall be equal to 1.

The **TemporalId** of the APS NAL unit having **aps_params_type** equal to **ALF_APS** and **adaptation_parameter_set_id** equal to **ph_cc_alf_cb_aps_id** shall be less than or equal to the **TemporalId** of the picture associated with the PH.

ph_cc_alf_cr_enabled_flag equal to 1 specifies that cross-component filter for Cr colour component is enabled for all slices associated with the PH and may be applied to Cr colour component in the slices. **ph_cc_alf_cr_enabled_flag** equal to 0 specifies that cross-component filter for Cr colour component may be disabled for one, or more, or all slices associated with the PH. When not present, **ph_cc_alf_cr_enabled_flag** is inferred to be equal to 0.

ph_cc_alf_cr_aps_id specifies the **adaptation_parameter_set_id** of the ALF APS that the Cr colour component of the slices associated with the PH refers to.

The value of **alf_cc_cr_filter_signal_flag** of the APS NAL unit having **aps_params_type** equal to **ALF_APS** and **adaptation_parameter_set_id** equal to **ph_cc_alf_cr_aps_id** shall be equal to 1.

The **TemporalId** of the APS NAL unit having **aps_params_type** equal to **ALF_APS** and **adaptation_parameter_set_id** equal to **ph_cc_alf_cr_aps_id** shall be less than or equal to the **TemporalId** of the picture associated with the PH.

ph_lmcs_enabled_flag equal to 1 specifies that luma mapping with chroma scaling is enabled for all slices associated with the PH. **ph_lmcs_enabled_flag** equal to 0 specifies that luma mapping with chroma scaling may be disabled for one, or more, or all slices associated with the PH. When not present, the value of **ph_lmcs_enabled_flag** is inferred to be equal to 0.

ph_lmcs_aps_id specifies the **adaptation_parameter_set_id** of the LMCS APS that the slices associated with the PH refers to. The **TemporalId** of the APS NAL unit having **aps_params_type** equal to **LMCS_APS** and **adaptation_parameter_set_id** equal to **ph_lmcs_aps_id** shall be less than or equal to the **TemporalId** of the picture associated with PH.

ph_chroma_residual_scale_flag equal to 1 specifies that chroma residual scaling is enabled for the all slices associated with the PH. **ph_chroma_residual_scale_flag** equal to 0 specifies that chroma residual scaling may be disabled for one, or more, or all slices associated with the PH. When **ph_chroma_residual_scale_flag** is not present, it is inferred to be equal to 0.

ph_explicit_scaling_list_enabled_flag equal to 1 specifies that the use of the explicit scaling list, which is signalled in the referenced scaling list APS (i.e., the APS with **aps_params_type** equal to SCALING_APS and **adaptation_parameter_set_id** equal to **ph_explicit_scaling_list_aps_id**), in the scaling process for transform coefficients when decoding a slice is enabled for the picture. **ph_explicit_scaling_list_enabled_flag** equal to 0 specifies that the use of an explicit scaling list in the scaling process for transform coefficients when decoding a slice is disabled for the picture. When not present, the value of **ph_explicit_scaling_list_enabled_flag** is inferred to be equal to 0.

ph_scaling_list_aps_id specifies the **adaptation_parameter_set_id** of the scaling list APS. The **TemporalId** of the APS NAL unit having **aps_params_type** equal to SCALING_APS and **adaptation_parameter_set_id** equal to **ph_scaling_list_aps_id** shall be less than or equal to the **TemporalId** of the picture associated with PH.

ph_virtual_boundaries_present_flag equal to 1 specifies that information of virtual boundaries is signalled in the PH. **ph_virtual_boundaries_present_flag** equal to 0 specifies that information of virtual boundaries is not signalled in the PH. When there is one or more than one virtual boundaries signalled in the PH, the in-loop filtering operations are disabled across the virtual boundaries in the picture. The in-loop filtering operations include the deblocking filter, sample adaptive offset filter, and adaptive loop filter operations. When not present, the value of **ph_virtual_boundaries_present_flag** is inferred to be equal to 0.

It is a requirement of bitstream conformance that, when **subpic_info_present_flag** is equal to 1, the value of **ph_virtual_boundaries_present_flag** shall be equal to 0.

The variable **VirtualBoundariesPresentFlag** is derived as follows:

```
VirtualBoundariesPresentFlag = 0
if( sps_virtual_boundaries_enabled_flag )
    VirtualBoundariesPresentFlag = sps_virtual_boundaries_present_flag ||
    ph_virtual_boundaries_present_flag
```

ph_num_ver_virtual_boundaries specifies the number of **ph_virtual_boundaries_pos_x[i]** syntax elements that are present in the PH. When **ph_num_ver_virtual_boundaries** is not present, it is inferred to be equal to 0.

The variable **NumVerVirtualBoundaries** is derived as follows:

```

NumVerVirtualBoundaries = 0
if( sps_virtual_boundaries_enabled_flag )
    NumVerVirtualBoundaries = sps_virtual_boundaries_present_flag ?
        sps_num_ver_virtual_boundaries : ph_num_ver_virtual_boundaries

```

ph_virtual_boundaries_pos_x[i] specifies the location of the i-th vertical virtual boundary in units of luma samples divided by 8. The value of **ph_virtual_boundaries_pos_x[i]** shall be in the range of 1 to $\text{Ceil}(\text{pic_width_in_luma_samples} \div 8) - 1$, inclusive.

The list **VirtualBoundariesPosX[i]** for i ranging from 0 to **NumVerVirtualBoundaries** - 1, inclusive, in units of luma samples, specifying the locations of the vertical virtual boundaries, is derived as follows:

```

for( i = 0; i < NumVerVirtualBoundaries; i++)
    VirtualBoundariesPosX[ i ] = ( sps_virtual_boundaries_present_flag ?
        sps_virtual_boundaries_pos_x[ i ] : ph_virtual_boundaries_pos_x[ i ] ) * 8

```

The distance between any two vertical virtual boundaries shall be greater than or equal to **CtbSizeY** luma samples.

ph_num_hor_virtual_boundaries specifies the number of **ph_virtual_boundaries_pos_y[i]** syntax elements that are present in the PH. When **ph_num_hor_virtual_boundaries** is not present, it is inferred to be equal to 0.

The parameter **NumHorVirtualBoundaries** is derived as follows:

```

NumHorVirtualBoundaries = 0
if( sps_virtual_boundaries_enabled_flag )
    NumHorVirtualBoundaries = sps_virtual_boundaries_present_flag ?
        sps_num_hor_virtual_boundaries : ph_num_hor_virtual_boundaries

```

When **sps_virtual_boundaries_enabled_flag** is equal to 1 and **ph_virtual_boundaries_present_flag** is equal to 1, the sum of **ph_num_ver_virtual_boundaries** and **ph_num_hor_virtual_boundaries** shall be greater than 0.

ph_virtual_boundaries_pos_y[i] specifies the location of the i-th horizontal virtual boundary in units of luma samples divided by 8. The value of **ph_virtual_boundaries_pos_y[i]** shall be in the range of 1 to $\text{Ceil}(\text{pic_height_in_luma_samples} \div 8) - 1$, inclusive.

The list **VirtualBoundariesPosY[i]** for i ranging from 0 to **NumHorVirtualBoundaries** - 1, inclusive, in units of luma samples, specifying the locations of the horizontal virtual boundaries, is derived as follows:

```
for( i = 0; i < NumHorVirtualBoundaries; i++)
    VirtualBoundariesPosY[ i ] = ( sps_virtual_boundaries_present_flag ?
        sps_virtual_boundaries_pos_y[ i ] : ph_virtual_boundaries_pos_y[ i ] ) * 8
```

The distance between any two horizontal virtual boundaries shall be greater than or equal to **CtbSizeY** luma samples.

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.

partition_constraints_override_flag equal to 1 specifies that partition constraint parameters are present in the PH. **partition_constraints_override_flag** equal to 0 specifies that partition constraint parameters are not present in the PH. When not present, the value of **partition_constraints_override_flag** is inferred to be equal to 0.

ph_log2_diff_min_qt_min_cb_intra_slice_luma specifies the difference between the base 2 logarithm of the minimum size in luma samples of a luma leaf block resulting from quadtree splitting of a CTU and the base 2 logarithm of the minimum coding block size in luma samples for luma CUs in the slices with **slice_type** equal to 2 (I) associated with the PH. The value of **ph_log2_diff_min_qt_min_cb_intra_slice_luma** shall be in the range of 0 to $\text{CtbLog2SizeY} - \text{MinCbLog2SizeY}$, inclusive. When not present, the value of **ph_log2_diff_min_qt_min_cb_luma** is inferred to be equal to **sps_log2_diff_min_qt_min_cb_intra_slice_luma**.

ph_max_mtt_hierarchy_depth_intra_slice_luma specifies the maximum hierarchy depth for coding units resulting from multi-type tree splitting of a quadtree leaf in slices with slice_type equal to 2 (I) associated with the PH. The value of **ph_max_mtt_hierarchy_depth_intra_slice_luma** shall be in the range of 0 to $2 * (\text{CtbLog2SizeY} - \text{MinCbLog2SizeY})$, inclusive. When not present, the value of **ph_max_mtt_hierarchy_depth_intra_slice_luma** is inferred to be equal to **sps_max_mtt_hierarchy_depth_intra_slice_luma**.

ph_log2_diff_max_bt_min_qt_intra_slice_luma specifies the difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a luma coding block that can be split using a binary split and the minimum size (width or height) in luma samples of a luma leaf block resulting from quadtree splitting of a CTU in slices with slice_type equal to 2 (I) associated with the PH. The value of **ph_log2_diff_max_bt_min_qt_intra_slice_luma** shall be in the range of 0 to $\text{CtbLog2SizeY} - \text{MinQtLog2SizeIntraY}$, inclusive. When not present, the value of **ph_log2_diff_max_bt_min_qt_intra_slice_luma** is inferred to be equal to **sps_log2_diff_max_bt_min_qt_intra_slice_luma**.

ph_log2_diff_max_tt_min_qt_intra_slice_luma specifies the difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a luma coding block that can be split using a ternary split and the minimum size (width or height) in luma samples of a luma leaf block resulting from quadtree splitting of a CTU in slices with slice_type equal to 2 (I) associated with the PH. The value of **ph_log2_diff_max_tt_min_qt_intra_slice_luma** shall be in the range of 0 to $\text{CtbLog2SizeY} - \text{MinQtLog2SizeIntraY}$, inclusive. When not present, the value of **ph_log2_diff_max_tt_min_qt_intra_slice_luma** is inferred to be equal to **sps_log2_diff_max_tt_min_qt_intra_slice_luma**.

ph_log2_diff_min_qt_min_cb_intra_slice_chroma specifies the difference between the base 2 logarithm of the minimum size in luma samples of a chroma leaf block resulting from quadtree splitting of a chroma CTU with treeType equal to DUAL_TREE_CHROMA and the base 2 logarithm of the minimum coding block size in luma samples for chroma CUs with treeType equal to DUAL_TREE_CHROMA in slices with slice_type equal to 2 (I) associated with the PH. The value of **ph_log2_diff_min_qt_min_cb_intra_slice_chroma** shall be in the range of 0 to $\text{CtbLog2SizeY} - \text{MinCbLog2SizeY}$, inclusive. When not present, the value of **ph_log2_diff_min_qt_min_cb_intra_slice_chroma** is inferred to be equal to **sps_log2_diff_min_qt_min_cb_intra_slice_chroma**.

ph_max_mtt_hierarchy_depth_intra_slice_chroma specifies the maximum hierarchy depth for chroma coding units resulting from multi-type tree splitting of a chroma quadtree leaf with treeType equal to DUAL_TREE_CHROMA in slices with slice_type equal to 2 (I) associated with the PH. The value of ph_max_mtt_hierarchy_depth_intra_slice_chroma shall be in the range of 0 to $2 * (CtbLog2SizeY - MinCbLog2SizeY)$, inclusive. When not present, the value of ph_max_mtt_hierarchy_depth_intra_slice_chroma is inferred to be equal to sps_max_mtt_hierarchy_depth_intra_slice_chroma.

ph_log2_diff_max_bt_min_qt_intra_slice_chroma specifies the difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a chroma coding block that can be split using a binary split and the minimum size (width or height) in luma samples of a chroma leaf block resulting from quadtree splitting of a chroma CTU with treeType equal to DUAL_TREE_CHROMA in slices with slice_type equal to 2 (I) associated with the PH. The value of ph_log2_diff_max_bt_min_qt_intra_slice_chroma shall be in the range of 0 to $CtbLog2SizeY - MinQtLog2SizeIntraC$, inclusive. When not present, the value of ph_log2_diff_max_bt_min_qt_intra_slice_chroma is inferred to be equal to sps_log2_diff_max_bt_min_qt_intra_slice_chroma.

ph_log2_diff_max_tt_min_qt_intra_slice_chroma specifies the difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a chroma coding block that can be split using a ternary split and the minimum size (width or height) in luma samples of a chroma leaf block resulting from quadtree splitting of a chroma CTU with treeType equal to DUAL_TREE_CHROMA in slices with slice_type equal to 2 (I) associated with the PH. The value of ph_log2_diff_max_tt_min_qt_intra_slice_chroma shall be in the range of 0 to $CtbLog2SizeY - MinQtLog2SizeIntraC$, inclusive. When not present, the value of ph_log2_diff_max_tt_min_qt_intra_slice_chroma is inferred to be equal to sps_log2_diff_max_tt_min_qt_intra_slice_chroma.

ph_cu_qp_delta_subdiv_intra_slice specifies the maximum cbSubdiv value of coding units in intra slice that convey cu_qp_delta_abs and cu_qp_delta_sign_flag. The value of ph_cu_qp_delta_subdiv_intra_slice shall be in the range of 0 to $2 * (CtbLog2SizeY - MinQtLog2SizeIntraY + ph_max_mtt_hierarchy_depth_intra_slice_luma)$, inclusive.

When not present, the value of ph_cu_qp_delta_subdiv_intra_slice is inferred to be equal to 0.

ph_cu_chroma_qp_offset_subdiv_intra_slice specifies the maximum cbSubdiv value of coding units in intra slice that convey cu_chroma_qp_offset_flag. The value of ph_cu_chroma_qp_offset_subdiv_intra_slice shall be in the range of 0 to $2 * (CtbLog2SizeY - MinQtLog2SizeIntraY + ph_max_mtt_hierarchy_depth_intra_slice_luma)$, inclusive.

When not present, the value of ph_cu_chroma_qp_offset_subdiv_intra_slice is inferred to be equal to 0.

ph_log2_diff_min_qt_min_cb_inter_slice specifies the difference between the base 2 logarithm of the minimum size in luma samples of a luma leaf block resulting from quadtree splitting of a CTU and the base 2 logarithm of the minimum luma coding block size in luma samples for luma CUs in the slices with slice_type equal to 0 (B) or 1 (P) associated with the PH. The value of ph_log2_diff_min_qt_min_cb_inter_slice shall be in the range of 0 to $CtbLog2SizeY - MinCbLog2SizeY$, inclusive. When not present, the value of ph_log2_diff_min_qt_min_cb_luma is inferred to be equal to sps_log2_diff_min_qt_min_cb_inter_slice.

ph_max_mtt_hierarchy_depth_inter_slice specifies the maximum hierarchy depth for coding units resulting from multi-type tree splitting of a quadtree leaf in slices with slice_type equal to 0 (B) or 1 (P) associated with the PH. The value of ph_max_mtt_hierarchy_depth_inter_slice shall be in the range of 0 to $2 * (CtbLog2SizeY - MinCbLog2SizeY)$, inclusive. When not present, the value of ph_max_mtt_hierarchy_depth_inter_slice is inferred to be equal to sps_max_mtt_hierarchy_depth_inter_slice.

ph_log2_diff_max_bt_min_qt_inter_slice specifies the difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a luma coding block that can be split using a binary split and the minimum size (width or height) in luma samples of a luma leaf block resulting from quadtree splitting of a CTU in the slices with slice_type equal to 0 (B) or 1 (P) associated with the PH. The value of ph_log2_diff_max_bt_min_qt_inter_slice shall be in the range of 0 to $CtbLog2SizeY - MinQtLog2SizeInterY$, inclusive. When not present, the value of ph_log2_diff_max_bt_min_qt_inter_slice is inferred to be equal to sps_log2_diff_max_bt_min_qt_inter_slice.

ph_log2_diff_max_tt_min_qt_inter_slice specifies the difference between the base 2 logarithm of the maximum size (width or height) in luma samples of a luma coding block that can be split using a ternary split and the minimum size (width or height) in luma samples of a luma leaf block resulting from quadtree splitting of a CTU in slices with slice_type equal to 0 (B) or 1 (P) associated with the PH. The value of **ph_log2_diff_max_tt_min_qt_inter_slice** shall be in the range of 0 to $\text{CtbLog2SizeY} - \text{MinQtLog2SizeInterY}$, inclusive. When not present, the value of **ph_log2_diff_max_tt_min_qt_inter_slice** is inferred to be equal to **sps_log2_diff_max_tt_min_qt_inter_slice**.

ph_cu_qp_delta_subdiv_inter_slice specifies the maximum cbSubdiv value of coding units that in inter slice convey **cu_qp_delta_abs** and **cu_qp_delta_sign_flag**. The value of **ph_cu_qp_delta_subdiv_inter_slice** shall be in the range of 0 to $2 * (\text{CtbLog2SizeY} - \text{MinQtLog2SizeInterY} + \text{ph_max_mtt_hierarchy_depth_inter_slice})$, inclusive.

When not present, the value of **ph_cu_qp_delta_subdiv_inter_slice** is inferred to be equal to 0.

ph_cu_chroma_qp_offset_subdiv_inter_slice specifies the maximum cbSubdiv value of coding units in inter slice that convey **cu_chroma_qp_offset_flag**. The value of **ph_cu_chroma_qp_offset_subdiv_inter_slice** shall be in the range of 0 to $2 * (\text{CtbLog2SizeY} - \text{MinQtLog2SizeInterY} + \text{ph_max_mtt_hierarchy_depth_inter_slice})$, inclusive.

When not present, the value of **ph_cu_chroma_qp_offset_subdiv_inter_slice** is inferred to be equal to 0.

ph_temporal_mvp_enabled_flag specifies whether temporal motion vector predictors can be used for inter prediction for slices associated with the PH. If **ph_temporal_mvp_enabled_flag** is equal to 0, the syntax elements of the slices associated with the PH shall be constrained such that no temporal motion vector predictor is used in decoding of the slices. Otherwise (**ph_temporal_mvp_enabled_flag** is equal to 1), temporal motion vector predictors may be used in decoding of the slices associated with the PH. When not present, the value of **ph_temporal_mvp_enabled_flag** is inferred to be equal to 0. When no reference picture in the DPB has the same spatial resolution as the current picture, the value of **ph_temporal_mvp_enabled_flag** shall be equal to 0.

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

```

if( sps_affine_enabled_flag )
    MaxNumSubblockMergeCand = 5 - five_minus_max_num_subblock_merge_cand
else
    MaxNumSubblockMergeCand = sps_sbtmvp_enabled_flag && ph_temporal_mvp_enabled_flag

```

The value of **MaxNumSubblockMergeCand** shall be in the range of 0 to 5, inclusive.

ph_collocated_from_l0_flag equal to 1 specifies that the collocated picture used for temporal motion vector prediction is derived from reference picture list 0. **ph_collocated_from_l0_flag** equal to 0 specifies that the collocated picture used for temporal motion vector prediction is derived from reference picture list 1.

ph_collocated_ref_idx specifies the reference index of the collocated picture used for temporal motion vector prediction.

When **ph_collocated_from_l0_flag** is equal to 1, **ph_collocated_ref_idx** refers to an entry in reference picture list 0, and the value of **ph_collocated_ref_idx** shall be in the range of 0 to **num_ref_entries[0][RplIdx[0]] - 1**, inclusive.

When **ph_collocated_from_l0_flag** is equal to 0, **ph_collocated_ref_idx** refers to an entry in reference picture list 1, and the value of **ph_collocated_ref_idx** shall be in the range of 0 to **num_ref_entries[1][RplIdx[1]] - 1**, inclusive.

When not present, the value of **ph_collocated_ref_idx** is inferred to be equal to 0.

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]` and `MvdCpL1[x0][y0][cpIdx][compIdx]` are set equal to 0 for `compIdx = 0..1` and `cpIdx = 0..2`. **mvd_l1_zero_flag** equal to 0 indicates that the `mvd_coding(x0, y0, 1)` syntax structure is parsed.

ph_fpel_mmvd_enabled_flag equal to 1 specifies that merge mode with motion vector difference uses integer sample precision in the slices associated with the PH. **ph_fpel_mmvd_enabled_flag** equal to 0 specifies that merge mode with motion vector difference can use fractional sample precision in the slices associated with the PH. When not present, the value of **ph_fpel_mmvd_enabled_flag** is inferred to be 0.

ph_disable_bdof_flag equal to 1 specifies that bi-directional optical flow inter prediction based inter bi-prediction is disabled in the slices associated with the PH. **ph_disable_bdof_flag** equal to 0 specifies that bi-directional optical flow inter prediction based inter bi-prediction may or may not be enabled in the slices associated with the PH.

When **ph_disable_bdof_flag** is not present, the following applies:

- If **sps_bdof_enabled_flag** is equal to 1, the value of **ph_disable_bdof_flag** is inferred to be equal to 0.
- Otherwise (**sps_bdof_enabled_flag** is equal to 0), the value of **ph_disable_bdof_flag** is inferred to be equal to 1.

ph_disable_dmvr_flag equal to 1 specifies that decoder motion vector refinement based inter bi-prediction is disabled in the slices associated with the PH. **ph_disable_dmvr_flag** equal to 0 specifies that decoder motion vector refinement based inter bi-prediction may or may not be enabled in the slices associated with the PH.

When **ph_disable_dmvr_flag** is not present, the following applies:

- If **sps_dmvr_enabled_flag** is equal to 1, the value of **ph_disable_dmvr_flag** is inferred to be equal to 0.
- Otherwise (**sps_dmvr_enabled_flag** is equal to 0), the value of **ph_disable_dmvr_flag** is inferred to be equal to 1.

ph_disable_prof_flag equal to 1 specifies that prediction refinement with optical flow is disabled in the slices associated with the PH. **ph_disable_prof_flag** equal to 0 specifies that prediction refinement with optical flow may or may not be enabled in the slices associated with the PH. When **ph_disable_prof_flag** is not present, the following applies:

- If **sps_affine_prof_enabled_flag** is equal to 1, the value of **ph_disable_prof_flag** is inferred to be equal to 0.
- Otherwise (**sps_affine_prof_enabled_flag** is equal to 0), the value of **ph_disable_prof_flag** is inferred to be equal to 1.

ph_qp_delta specifies the initial value of Q_{PY} to be used for the coding blocks in the picture until modified by the value of **CuQpDeltaVal** in the coding unit layer.

When **qp_delta_info_in_ph_flag** is equal to 1, the initial value of the Q_{PY} quantization parameter for all slices of the picture, $SliceQ_{PY}$, is derived as follows:

$$SliceQ_{PY} = 26 + init_qp_minus26 + ph_qp_delta$$

The value of $SliceQ_{PY}$ shall be in the range of $-Q_{pBdOffset}$ to +63, inclusive.

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

ph_sao_luma_enabled_flag equal to 1 specifies that SAO is enabled for the luma component in all slices associated with the PH; **ph_sao_luma_enabled_flag** equal to 0 specifies that SAO for the luma component may be disabled for one, or more, or all slices associated with the PH. When **ph_sao_luma_enabled_flag** is not present, it is inferred to be equal to 0.

ph_sao_chroma_enabled_flag equal to 1 specifies that SAO is enabled for the chroma component in all slices associated with the PH; **ph_sao_chroma_enabled_flag** equal to 0 specifies that SAO for chroma component may be disabled for one, or more, or all slices associated with the PH. When **ph_sao_chroma_enabled_flag** is not present, it is inferred to be equal to 0.

ph_dep_quant_enabled_flag equal to 0 specifies that dependent quantization is disabled for the current picture. **ph_dep_quant_enabled_flag** equal to 1 specifies that dependent quantization is enabled for the current picture. When **ph_dep_quant_enabled_flag** is not present, it is inferred to be equal to 0.

pic_sign_data_hiding_enabled_flag equal to 0 specifies that sign bit hiding is disabled for the current picture. **pic_sign_data_hiding_enabled_flag** equal to 1 specifies that sign bit hiding is enabled for the current picture. When **pic_sign_data_hiding_enabled_flag** is not present, it is inferred to be equal to 0.

ph_deblocking_filter_override_flag equal to 1 specifies that deblocking parameters are present in the PH. **ph_deblocking_filter_override_flag** equal to 0 specifies that deblocking parameters are not present in the PH. When not present, the value of **ph_deblocking_filter_override_flag** is inferred to be equal to 0.

ph_deblocking_filter_disabled_flag equal to 1 specifies that the operation of the deblocking filter is not applied for the slices associated with the PH. **ph_deblocking_filter_disabled_flag** equal to 0 specifies that the operation of the deblocking filter is applied for the slices associated with the PH. When **ph_deblocking_filter_disabled_flag** is not present, it is inferred to be equal to **pps_deblocking_filter_disabled_flag**.

ph_beta_offset_div2 and **ph_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the luma component for the slices associated with the PH. The values of **ph_beta_offset_div2** and **ph_tc_offset_div2** shall both be in the range of -12 to 12, inclusive. When not present, the values of **ph_beta_offset_div2** and **ph_tc_offset_div2** are inferred to be equal to **pps_beta_offset_div2** and **pps_tc_offset_div2**, respectively.

ph_cb_beta_offset_div2 and **ph_cb_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for the slices associated with the PH. The values of **ph_cb_beta_offset_div2** and **ph_cb_tc_offset_div2** shall both be in the range of -12 to 12, inclusive. When not present, the values of **ph_cb_beta_offset_div2** and **ph_cb_tc_offset_div2** are inferred to be equal to **pps_cb_beta_offset_div2** and **pps_cb_tc_offset_div2**, respectively.

ph_cr_beta_offset_div2 and **ph_cr_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for the slices associated with the PH. The values of **ph_cr_beta_offset_div2** and **ph_cr_tc_offset_div2** shall both be in the range of -12 to 12 , inclusive. When not present, the values of **ph_cr_beta_offset_div2** and **ph_cr_tc_offset_div2** are inferred to be equal to **pps_cr_beta_offset_div2** and **pps_cr_tc_offset_div2**, respectively.

ph_extension_length specifies the length of the PH extension data in bytes, not including the bits used for signalling **ph_extension_length** itself. The value of **ph_extension_length** shall be in the range of 0 to 256 , inclusive. When not present, the value of **ph_extension_length** is inferred to be equal to 0 .

ph_extension_data_byte may have any value. Decoders conforming to this version of this Specification shall ignore the value of **ph_extension_data_byte**. Its value does not affect decoder conformance to profiles specified in this version of specification.

As provided in Table 2, a NAL unit may include a **slice_layer_rbsp()** syntax structure. Table 5 illustrates the syntax structure of the **slice_layer_rbsp()** and Table 6 illustrates the syntax structure of the **slice_header()** in JVET-Q2001.

slice_layer_rbsp() {	Descriptor
slice_header()	
slice_data()	
rsbp_slice_trailing_bits()	
}	

Table 5

slice_header() {	Descriptor
picture_header_in_slice_header_flag	u(1)
if(picture_header_in_slice_header_flag)	
picture_header_structure()	
if(subpic_info_present_flag)	
slice_subpic_id	u(v)
if((rect_slice_flag && NumSlicesInSubpic[CurrSubpicIdx] > 1) (!rect_slice_flag && NumTilesInPic > 1))	
slice_address	u(v)
for(i = 0; i < NumExtraShBits; i++)	
sh_extra_bit[i]	u(1)
if(!rect_slice_flag && NumTilesInPic > 1)	
num_tiles_in_slice_minus1	ue(v)
if(ph_inter_slice_allowed_flag)	
slice_type	ue(v)
if(sps_alf_enabled_flag && !alf_info_in_ph_flag) {	
slice_alf_enabled_flag	u(1)
if(slice_alf_enabled_flag) {	
slice_num_alf_aps_ids_luma	u(3)
for(i = 0; i < slice_num_alf_aps_ids_luma; i++)	
slice_alf_aps_id_luma[i]	u(3)
if(ChromaArrayType != 0)	
slice_alf_chroma_idc	u(2)
if(slice_alf_chroma_idc)	
slice_alf_aps_id_chroma	u(3)
if(sps_ccalf_enabled_flag) {	
slice_cc_alf_cb_enabled_flag	u(1)
if(slice_cc_alf_cb_enabled_flag)	
slice_cc_alf_cb_aps_id	u(3)
slice_cc_alf_cr_enabled_flag	u(1)
if(slice_cc_alf_cr_enabled_flag)	

slice_cc_alf_cr_aps_id	u(3)
}	
}	
}	
if(separate_colour_plane_flag == 1)	
colour_plane_id	u(2)
if(!rpl_info_in_ph_flag && ((nal_unit_type != IDR_W_RADL && nal_unit_type != IDR_N_LP) sps_idr_rpl_present_flag))	
ref_pic_lists()	
if((rpl_info_in_ph_flag ((nal_unit_type != IDR_W_RADL && nal_unit_type != IDR_N_LP) sps_idr_rpl_present_flag)) && ((slice_type != I && num_ref_entries[0][RplIdx[0]] > 1) (slice_type == B && num_ref_entries[1][RplIdx[1]] > 1))) {	
num_ref_idx_active_override_flag	u(1)
if(num_ref_idx_active_override_flag)	
for(i = 0; i < (slice_type == B ? 2 : 1); i++)	
if(num_ref_entries[i][RplIdx[i]] > 1)	
num_ref_idx_active_minus1[i]	ue(v)
}	
if(slice_type != I) {	
if(cabac_init_present_flag)	
cabac_init_flag	u(1)
if(ph_temporal_mvp_enabled_flag && !rpl_info_in_ph_flag) {	
if(slice_type == B)	
slice_collocated_from_l0_flag	u(1)
if((slice_collocated_from_l0_flag && NumRefIdxActive[0] > 1) (!slice_collocated_from_l0_flag && NumRefIdxActive[1] > 1))	
slice_collocated_ref_idx	ue(v)
}	
if(!wp_info_in_ph_flag && ((pps_weighted_pred_flag && slice_type == P) (pps_weighted_bipred_flag && slice_type == B)))	

pred_weight_table()	
}	
if(!qp_delta_info_in_ph_flag)	
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(sps_joint_cbr_enabled_flag)	
slice_joint_cbr_qp_offset	se(v)
}	
if(pps_cu_chroma_qp_offset_list_enabled_flag)	
cu_chroma_qp_offset_enabled_flag	u(1)
if(sps_sao_enabled_flag && !sao_info_in_ph_flag) {	
slice_sao_luma_flag	u(1)
if(ChromaArrayType != 0)	
slice_sao_chroma_flag	u(1)
}	
if(deblocking_filter_override_enabled_flag && !dbf_info_in_ph_flag)	
slice_deblocking_filter_override_flag	u(1)
if(slice_deblocking_filter_override_flag) {	
slice_deblocking_filter_disabled_flag	u(1)
if(!slice_deblocking_filter_disabled_flag) {	
slice_beta_offset_div2	se(v)
slice_tc_offset_div2	se(v)
slice_cb_beta_offset_div2	se(v)
slice_cb_tc_offset_div2	se(v)
slice_cr_beta_offset_div2	se(v)
slice_cr_tc_offset_div2	se(v)
}	
}	
slice_ts_residual_coding_disabled_flag	u(1)
if(ph_lmcs_enabled_flag)	

slice_lmcs_enabled_flag	u(1)
if(ph_explicit_scaling_list_enabled_flag)	
slice_explicit_scaling_list_used_flag	u(1)
if(NumEntryPoints > 0) {	
offset_len_minus1	ue(v)
for(i = 0; i < NumEntryPoints; i++)	
entry_point_offset_minus1[i]	u(v)
}	
if(slice_header_extension_present_flag) {	
slice_header_extension_length	ue(v)
for(i = 0; i < slice_header_extension_length; i++)	
slice_header_extension_data_byte[i]	u(8)
}	
byte_alignment()	
}	

Table 6

With respect to Table 6, JVET-Q2001 provides the following semantics:

The variable `CuQpDeltaVal`, specifying the difference between a luma quantization parameter for the coding unit containing `cu_qp_delta_abs` and its prediction, is set equal to 0. The variables `CuQpOffsetCb`, `CuQpOffsetCr`, and `CuQpOffsetCbCr`, specifying values to be used when determining the respective values of the Qp'_{Cb} , Qp'_{Cr} , and Qp'_{CbCr} quantization parameters for the coding unit containing `cu_chroma_qp_offset_flag`, are all set equal to 0.

picture_header_in_slice_header_flag equal to 1 specifies that the PH syntax structure is present in the slice header. **picture_header_in_slice_header_flag** equal to 0 specifies that the PH syntax structure is not present in the slice header.

It is a requirement of bitstream conformance that the value of **picture_header_in_slice_header_flag** shall be the same in all coded slices in a CLVS.

When **picture_header_in_slice_header_flag** is equal to 1 for a coded slice, it is a requirement of bitstream conformance that no VCL NAL unit with `nal_unit_type` equal to PH_NUT shall be present in the CLVS.

When **picture_header_in_slice_header_flag** is equal to 0, all coded slices in the current picture shall have **picture_header_in_slice_header_flag** is equal to 0, and the current PU shall have a PH NAL unit.

slice_subpic_id specifies the subpicture ID of the subpicture that contains the slice. If **slice_subpic_id** is present, the value of the variable `CurrSubpicIdx` is derived to be such that `SubpicIdVal[ CurrSubpicIdx ]` is equal to **slice_subpic_id**. Otherwise (**slice_subpic_id** is not present), `CurrSubpicIdx` is derived to be equal to 0. The length of **slice_subpic_id** is `sps_subpic_id_len_minus1 + 1` bits.

slice_address specifies the slice address of the slice. When not present, the value of **slice_address** is inferred to be equal to 0.

If **rect_slice_flag** is equal to 0, the following applies:

- The slice address is the raster scan tile index of the first tile in the slice.
- The length of **slice_address** is $\text{Ceil}(\text{Log}_2(\text{NumTilesInPic}))$ bits.
- The value of **slice_address** shall be in the range of 0 to $\text{NumTilesInPic} - 1$, inclusive.

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

- The slice address is the subpicture-level slice index of the slice.
- The length of slice_address is $\text{Ceil}(\text{Log2}(\text{NumSlicesInSubpic}[\text{CurrSubpicIdx}]))$ bits.
- The value of slice_address shall be in the range of 0 to $\text{NumSlicesInSubpic}[\text{CurrSubpicIdx}] - 1$, inclusive.

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

- If rect_slice_flag is equal to 0 or subpic_info_present_flag is equal to 0, the value of slice_address shall not be equal to the value of slice_address of any other coded slice NAL unit of the same coded picture.
- Otherwise, the pair of slice_subpic_id and slice_address values shall not be equal to the pair of slice_subpic_id and slice_address values of any other coded slice NAL unit of the same coded picture.
- The shapes of the slices of a picture shall be such that each CTU, when decoded, shall have its entire left boundary and entire top boundary consisting of a picture boundary or consisting of boundaries of previously decoded CTU(s).

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

num_tiles_in_slice_minus1 plus 1, when present, specifies the number of tiles in the slice. The value of **num_tiles_in_slice_minus1** shall be in the range of 0 to **NumTilesInPic** - 1, inclusive. The variable **NumCtusInCurrSlice**, which specifies the number of CTUs in the current slice, and the list **CtbAddrInCurrSlice[i]**, for **i** ranging from 0 to **NumCtusInCurrSlice** - 1, inclusive, specifying the picture raster scan address of the **i**-th CTB within the slice, are derived as follows:

```

if( rect_slice_flag ) {
    picLevelSliceIdx = slice_address
    for( j = 0; j < CurrSubpicIdx; j++ )
        picLevelSliceIdx += NumSlicesInSubpic[ j ]
    NumCtusInCurrSlice = NumCtusInSlice[ picLevelSliceIdx ]
    for( i = 0; i < NumCtusInCurrSlice; i++ )
        CtbAddrInCurrSlice[ i ] = CtbAddrInSlice[ picLevelSliceIdx ][ i ]
} else {
    NumCtusInCurrSlice = 0
    for( tileIdx = slice_address; tileIdx <= slice_address + num_tiles_in_slice_minus1;
        tileIdx++ ) {
        tileX = tileIdx % NumTileColumns
        tileY = tileIdx / NumTileColumns
        for( ctbY = tileRowBd[ tileY ]; ctbY < tileRowBd[ tileY + 1 ]; ctbY++ ) {
            for( ctbX = tileColBd[ tileX ]; ctbX < tileColBd[ tileX + 1 ]; ctbX++ ) {
                CtbAddrInCurrSlice[ NumCtusInCurrSlice ] = ctbY * PicWidthInCtb + ctbX
                NumCtusInCurrSlice++
            }
        }
    }
}

```

The variables SubpicLeftBoundaryPos, SubpicTopBoundaryPos, SubpicRightBoundaryPos, and SubpicBotBoundaryPos are derived as follows:

```

if( subpic_treated_as_pic_flag[ CurrSubpicIdx ] ) {
    SubpicLeftBoundaryPos = subpic_ctu_top_left_x[ CurrSubpicIdx ] * CtbSizeY
    SubpicRightBoundaryPos = Min( pic_width_max_in_luma_samples - 1,
        ( subpic_ctu_top_left_x[ CurrSubpicIdx ] +
          subpic_width_minus1[ CurrSubpicIdx ] + 1 ) * CtbSizeY - 1 )
    SubpicTopBoundaryPos = subpic_ctu_top_left_y[ CurrSubpicIdx ] * CtbSizeY
    SubpicBotBoundaryPos = Min( pic_height_max_in_luma_samples - 1,
        ( subpic_ctu_top_left_y[ CurrSubpicIdx ] +
          subpic_height_minus1[ CurrSubpicIdx ] + 1 ) * CtbSizeY - 1 )
}

```

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

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

Table 7

When not present, the value of slice_type is inferred to be equal to 2.

When ph_intra_slice_allowed_flag is equal to 0, the value of slice_type shall be equal to 0 or 1.

When nal_unit_type is in the range of IDR_W_RADL to CRA_NUT, inclusive, and vps_independent_layer_flag[GeneralLayerIdx[nuh_layer_id]] is equal to 1, slice_type shall be equal to 2.

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

- If slice_type equal to 2 (I), the following applies:

$$\begin{aligned}\text{MinQtLog2SizeY} &= \text{MinCbLog2SizeY} + \text{ph_log2_diff_min_qt_min_cb_intra_slice_luma} \\ \text{MinQtLog2SizeC} &= \text{MinCbLog2SizeY} + \text{ph_log2_diff_min_qt_min_cb_intra_slice_chroma} \\ \text{MaxBtSizeY} &= 1 \ll (\text{MinQtLog2SizeY} + \text{ph_log2_diff_max_bt_min_qt_intra_slice_luma}) \\ \text{MaxBtSizeC} &= 1 \ll (\text{MinQtLog2SizeC} + \text{ph_log2_diff_max_bt_min_qt_intra_slice_chroma}) \\ \text{MaxTtSizeY} &= 1 \ll (\text{MinQtLog2SizeY} + \text{ph_log2_diff_max_tt_min_qt_intra_slice_luma}) \\ \text{MaxTtSizeC} &= 1 \ll (\text{MinQtLog2SizeC} + \text{ph_log2_diff_max_tt_min_qt_intra_slice_chroma}) \\ \text{MaxMttDepthY} &= \text{ph_max_mtt_hierarchy_depth_intra_slice_luma} \\ \text{MaxMttDepthC} &= \text{ph_max_mtt_hierarchy_depth_intra_slice_chroma} \\ \text{CuQpDeltaSubdiv} &= \text{ph_cu_qp_delta_subdiv_intra_slice} \\ \text{CuChromaQpOffsetSubdiv} &= \text{ph_cu_chroma_qp_offset_subdiv_intra_slice}\end{aligned}$$

- Otherwise (slice_type equal to 0 (B) or 1 (P)), the following applies:

$$\begin{aligned}\text{MinQtLog2SizeY} &= \text{MinCbLog2SizeY} + \text{ph_log2_diff_min_qt_min_cb_inter_slice} \\ \text{MinQtLog2SizeC} &= \text{MinCbLog2SizeY} + \text{ph_log2_diff_min_qt_min_cb_inter_slice} \\ \text{MaxBtSizeY} &= 1 \ll (\text{MinQtLog2SizeY} + \text{ph_log2_diff_max_bt_min_qt_inter_slice}) \\ \text{MaxBtSizeC} &= 1 \ll (\text{MinQtLog2SizeC} + \text{ph_log2_diff_max_bt_min_qt_inter_slice}) \\ \text{MaxTtSizeY} &= 1 \ll (\text{MinQtLog2SizeY} + \text{ph_log2_diff_max_tt_min_qt_inter_slice}) \\ \text{MaxTtSizeC} &= 1 \ll (\text{MinQtLog2SizeC} + \text{ph_log2_diff_max_tt_min_qt_inter_slice}) \\ \text{MaxMttDepthY} &= \text{ph_max_mtt_hierarchy_depth_inter_slice} \\ \text{MaxMttDepthC} &= \text{ph_max_mtt_hierarchy_depth_inter_slice} \\ \text{CuQpDeltaSubdiv} &= \text{ph_cu_qp_delta_subdiv_inter_slice} \\ \text{CuChromaQpOffsetSubdiv} &= \text{ph_cu_chroma_qp_offset_subdiv_inter_slice}\end{aligned}$$

- The following applies:

$$\text{MinQtSizeY} = 1 \ll \text{MinQtLog2SizeY}$$

$$\text{MinQtSizeC} = 1 \ll \text{MinQtLog2SizeC}$$

$$\text{MinBtSizeY} = 1 \ll \text{MinCbLog2SizeY}$$

$$\text{MinTtSizeY} = 1 \ll \text{MinCbLog2SizeY}$$

slice_alf_enabled_flag equal to 1 specifies that adaptive loop filter is enabled and may be applied to Y, Cb, or Cr colour component in a slice. **slice_alf_enabled_flag** equal to 0 specifies that adaptive loop filter is disabled for all colour components in a slice. When not present, the value of **slice_alf_enabled_flag** is inferred to be equal to **ph_alf_enabled_flag**.

slice_num_alf_aps_ids_luma specifies the number of ALF APSs that the slice refers to. When **slice_alf_enabled_flag** is equal to 1 and **slice_num_alf_aps_ids_luma** is not present, the value of **slice_num_alf_aps_ids_luma** is inferred to be equal to the value of **ph_num_alf_aps_ids_luma**.

slice_alf_aps_id_luma[i] specifies the **adaptation_parameter_set_id** of the i-th ALF APS that the luma component of the slice refers to. The **TemporalId** of the APS NAL unit having **aps_params_type** equal to **ALF_APS** and **adaptation_parameter_set_id** equal to **slice_alf_aps_id_luma[i]** shall be less than or equal to the **TemporalId** of the coded slice NAL unit. When **slice_alf_enabled_flag** is equal to 1 and **slice_alf_aps_id_luma[i]** is not present, the value of **slice_alf_aps_id_luma[i]** is inferred to be equal to the value of **ph_alf_aps_id_luma[i]**. The value of **alf_luma_filter_signal_flag** of the APS NAL unit having **aps_params_type** equal to **ALF_APS** and **adaptation_parameter_set_id** equal to **slice_alf_aps_id_luma[i]** shall be equal to 1.

slice_alf_chroma_idc equal to 0 specifies that the adaptive loop filter is not applied to Cb and Cr colour components. **slice_alf_chroma_idc** equal to 1 indicates that the adaptive loop filter is applied to the Cb colour component. **slice_alf_chroma_idc** equal to 2 indicates that the adaptive loop filter is applied to the Cr colour component. **slice_alf_chroma_idc** equal to 3 indicates that the adaptive loop filter is applied to Cb and Cr colour components. When **slice_alf_chroma_idc** is not present, it is inferred to be equal to **ph_alf_chroma_idc**.

slice_alf_aps_id_chroma specifies the `adaptation_parameter_set_id` of the ALF APS that the chroma component of the slice refers to. The `TemporalId` of the APS NAL unit having `aps_params_type` equal to `ALF_APS` and `adaptation_parameter_set_id` equal to `slice_alf_aps_id_chroma` shall be less than or equal to the `TemporalId` of the coded slice NAL unit. When `slice_alf_enabled_flag` is equal to 1 and `slice_alf_aps_id_chroma` is not present, the value of `slice_alf_aps_id_chroma` is inferred to be equal to the value of `ph_alf_aps_id_chroma`. The value of `alf_chroma_filter_signal_flag` of the APS NAL unit having `aps_params_type` equal to `ALF_APS` and `adaptation_parameter_set_id` equal to `slice_alf_aps_id_chroma` shall be equal to 1.

slice_cc_alf_cb_enabled_flag equal to 0 specifies that the cross-component filter is not applied to the Cb colour component. `slice_cc_alf_cb_enabled_flag` equal to 1 indicates that the cross-component filter is enabled and may be applied to the Cb colour component. When `slice_cc_alf_cb_enabled_flag` is not present, it is inferred to be equal to `ph_cc_alf_cb_enabled_flag`.

slice_cc_alf_cb_aps_id specifies the `adaptation_parameter_set_id` that the Cb colour component of the slice refers to.

The `TemporalId` of the APS NAL unit having `aps_params_type` equal to `ALF_APS` and `adaptation_parameter_set_id` equal to `slice_cc_alf_cb_aps_id` shall be less than or equal to the `TemporalId` of the coded slice NAL unit. When `slice_cc_alf_cb_enabled_flag` is equal to 1 and `slice_cc_alf_cb_aps_id` is not present, the value of `slice_cc_alf_cb_aps_id` is inferred to be equal to the value of `ph_cc_alf_cb_aps_id`.

The value of `alf_cc_cb_filter_signal_flag` of the APS NAL unit having `aps_params_type` equal to `ALF_APS` and `adaptation_parameter_set_id` equal to `slice_cc_alf_cb_aps_id` shall be equal to 1.

slice_cc_alf_cr_enabled_flag equal to 0 specifies that the cross-component filter is not applied to the Cr colour component. `slice_cc_alf_cr_enabled_flag` equal to 1 indicates that the cross-component adaptive loop filter is enabled and may be applied to the Cr colour component. When `slice_cc_alf_cr_enabled_flag` is not present, it is inferred to be equal to `ph_cc_alf_cr_enabled_flag`.

slice_cc_alf_cr_aps_id specifies the **adaptation_parameter_set_id** that the Cr colour component of the slice refers to. The **TemporalId** of the APS NAL unit having **aps_params_type** equal to **ALF_APS** and **adaptation_parameter_set_id** equal to **slice_cc_alf_cr_aps_id** shall be less than or equal to the **TemporalId** of the coded slice NAL unit. When **slice_cc_alf_cr_enabled_flag** is equal to 1 and **slice_cc_alf_cr_aps_id** is not present, the value of **slice_cc_alf_cr_aps_id** is inferred to be equal to the value of **ph_cc_alf_cr_aps_id**.

The value of **alf_cc_cr_filter_signal_flag** of the APS NAL unit having **aps_params_type** equal to **ALF_APS** and **adaptation_parameter_set_id** equal to **slice_cc_alf_cr_aps_id** shall be equal to 1.

colour_plane_id identifies the colour plane associated with the current slice 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** values 0, 1 and 2 correspond to the Y, Cb and Cr planes, respectively. The value 3 of **colour_plane_id** is reserved for future use by ITU-T | ISO/IEC.

NOTE – There is no dependency between the decoding processes of different colour planes of one picture.

num_ref_idx_active_override_flag equal to 1 specifies that the syntax element **num_ref_idx_active_minus1[0]** is present for P and B slices and the syntax element **num_ref_idx_active_minus1[1]** is present for B slices. **num_ref_idx_active_override_flag** equal to 0 specifies that the syntax elements **num_ref_idx_active_minus1[0]** and **num_ref_idx_active_minus1[1]** are not present. When not present, the value of **num_ref_idx_active_override_flag** is inferred to be equal to 1.

num_ref_idx_active_minus1[i] is used for the derivation of the variable **NumRefIdxActive[i]** as specified by the equation below. The value of **num_ref_idx_active_minus1[i]** shall be in the range of 0 to 14, inclusive.

For *i* equal to 0 or 1, when the current slice is a B slice, **num_ref_idx_active_override_flag** is equal to 1, and **num_ref_idx_active_minus1[i]** is not present, **num_ref_idx_active_minus1[i]** is inferred to be equal to 0.

When the current slice is a P slice, **num_ref_idx_active_override_flag** is equal to 1, and **num_ref_idx_active_minus1[0]** is not present, **num_ref_idx_active_minus1[0]** is inferred to be equal to 0.

The variable NumRefIdxActive[i] is derived as follows:

```

for( i = 0; i < 2; i++ ) {
    if( slice_type == B || ( slice_type == P && i == 0 ) ) {
        if( num_ref_idx_active_override_flag )
            NumRefIdxActive[ i ] = num_ref_idx_active_minus1[ i ] + 1
        else {
            if( num_ref_entries[ i ][ RplIdx[ i ] ] >= num_ref_idx_default_active_minus1[ i ] + 1 )
                NumRefIdxActive[ i ] = num_ref_idx_default_active_minus1[ i ] + 1
            else
                NumRefIdxActive[ i ] = num_ref_entries[ i ][ RplIdx[ i ] ]
        }
    } else /* slice_type == I || ( slice_type == P && i == 1 ) */
        NumRefIdxActive[ i ] = 0
}

```

The value of NumRefIdxActive[i] - 1 specifies the maximum reference index for reference picture list i that may be used to decode the slice. When the value of NumRefIdxActive[i] is equal to 0, no reference index for reference picture list i may be used to decode the slice.

When the current slice is a P slice, the value of NumRefIdxActive[0] shall be greater than 0.

When the current slice is a B slice, both NumRefIdxActive[0] and NumRefIdxActive[1] shall be greater than 0.

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 equal to 0.

slice_collocated_from_l0_flag equal to 1 specifies that the collocated picture used for temporal motion vector prediction is derived from reference picture list 0. **slice_collocated_from_l0_flag** equal to 0 specifies that the collocated picture used for temporal motion vector prediction is derived from reference picture list 1.

When **slice_type** is equal to B or P, **ph_temporal_mvp_enabled_flag** is equal to 1, and **slice_collocated_from_l0_flag** is not present, the following applies:

- If **rpl_info_in_ph_flag** is equal to 1, **slice_collocated_from_l0_flag** is inferred to be equal to **ph_collocated_from_l0_flag**.
- Otherwise (**rpl_info_in_ph_flag** is equal to 0 and **slice_type** is equal to P), the value of **slice_collocated_from_l0_flag** is inferred to be equal to 1.

slice_collocated_ref_idx specifies the reference index of the collocated picture used for temporal motion vector prediction.

When **slice_type** is equal to P or when **slice_type** is equal to B and **slice_collocated_from_l0_flag** is equal to 1, **slice_collocated_ref_idx** refers to an entry in reference picture list 0, and the value of **slice_collocated_ref_idx** shall be in the range of 0 to $\text{NumRefIdxActive}[0] - 1$, inclusive.

When **slice_type** is equal to B and **slice_collocated_from_l0_flag** is equal to 0, **slice_collocated_ref_idx** refers to an entry in reference picture list 1, and the value of **slice_collocated_ref_idx** shall be in the range of 0 to $\text{NumRefIdxActive}[1] - 1$, inclusive.

When **slice_collocated_ref_idx** is not present, the following applies:

- If **rpl_info_in_ph_flag** is equal to 1, the value of **slice_collocated_ref_idx** is inferred to be equal to **ph_collocated_ref_idx**.
- Otherwise (**rpl_info_in_ph_flag** is equal to 0), the value of **slice_collocated_ref_idx** is inferred to be equal to 0.

It is a requirement of bitstream conformance that the picture referred to by **slice_collocated_ref_idx** shall be the same for all slices of a coded picture.

It is a requirement of bitstream conformance that the values of **pic_width_in_luma_samples** and **pic_height_in_luma_samples** of the reference picture referred to by **slice_collocated_ref_idx** shall be equal to the values of **pic_width_in_luma_samples** and **pic_height_in_luma_samples**, respectively, of the current picture, and $\text{RprConstraintsActive}[\text{slice_collocated_from_l0_flag} ? 0 : 1][\text{slice_collocated_ref_idx}]$ shall be equal to 0.

slice_qp_delta specifies the initial value of Q_{PY} to be used for the coding blocks in the slice until modified by the value of $CuQpDeltaVal$ in the coding unit layer.

When $qp_delta_info_in_ph_flag$ is equal to 0, the initial value of the Q_{PY} quantization parameter for the slice, $SliceQ_{PY}$, is derived as follows:

$$SliceQ_{PY} = 26 + init_qp_minus26 + slice_qp_delta$$

The value of $SliceQ_{PY}$ shall be in the range of $-QpBdOffset$ to +63, inclusive.

When either of the following conditions is true, the value of $NumRefIdxActive[0]$ shall be less than or equal to the value of $NumWeightsL0$:

- The value of $wp_info_in_ph_flag$ is equal to 1, $pps_weighted_pred_flag$ is equal to 1, and $slice_type$ is equal to P.
- The value of $wp_info_in_ph_flag$ is equal to 1, $pps_weighted_bipred_flag$ is equal to 1, and $slice_type$ is equal to B.

When $wp_info_in_ph_flag$ is equal to 1, $pps_weighted_bipred_flag$ is equal to 1, and $slice_type$ is equal to B, the value of $NumRefIdxActive[1]$ shall be less than or equal to the value of $NumWeightsL1$.

When either of the following conditions is true, for each value of i in the range of 0 to $NumRefIdxActive[0] - 1$, inclusive, the values of $luma_weight_l0_flag[i]$ and $chroma_weight_l0_flag[i]$ are both inferred to be equal to 0:

- The value of $wp_info_in_ph_flag$ is equal to 1, $pps_weighted_pred_flag$ is equal to 0, and $slice_type$ is equal to P.
- The value of $wp_info_in_ph_flag$ is equal to 1, $pps_weighted_bipred_flag$ is equal to 0, and $slice_type$ is equal to B.

slice_cb_qp_offset specifies a difference to be added to the value of $pps_cb_qp_offset$ when determining the value of the $Q_{P/CB}$ 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.

slice_joint_cbr_qp_offset specifies a difference to be added to the value of `pps_joint_cbr_qp_offset_value` when determining the value of the Qp'_{CbCr} . The value of `slice_joint_cbr_qp_offset` shall be in the range of -12 to $+12$, inclusive. When `slice_joint_cbr_qp_offset` is not present, it is inferred to be equal to 0. The value of `pps_joint_cbr_qp_offset_value` + `slice_joint_cbr_qp_offset` shall be in the range of -12 to $+12$, inclusive.

cu_chroma_qp_offset_enabled_flag equal to 1 specifies that the `cu_chroma_qp_offset_flag` may be present in the transform unit and palette coding syntax. `cu_chroma_qp_offset_enabled_flag` equal to 0 specifies that the `cu_chroma_qp_offset_flag` is not present in the transform unit or palette coding syntax. When not present, the value of `cu_chroma_qp_offset_enabled_flag` is inferred to be equal to 0.

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 `ph_sao_luma_enabled_flag`.

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 `ph_sao_chroma_enabled_flag`.

slice_deblocking_filter_override_flag equal to 1 specifies that deblocking parameters are present in the slice header. `slice_deblocking_filter_override_flag` equal to 0 specifies that deblocking parameters are not present in the slice header. When not present, the value of `slice_deblocking_filter_override_flag` is inferred to be equal to `ph_deblocking_filter_override_flag`.

slice_deblocking_filter_disabled_flag equal to 1 specifies that the operation of the deblocking filter is not applied for the current slice. **slice_deblocking_filter_disabled_flag** equal to 0 specifies that the operation of the deblocking filter is applied for the current slice. When **slice_deblocking_filter_disabled_flag** is not present, it is inferred to be equal to **ph_deblocking_filter_disabled_flag**.

slice_beta_offset_div2 and **slice_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the luma component for the current slice. The values of **slice_beta_offset_div2** and **slice_tc_offset_div2** shall both be in the range of -12 to 12, inclusive. When not present, the values of **slice_beta_offset_div2** and **slice_tc_offset_div2** are inferred to be equal to **ph_beta_offset_div2** and **ph_tc_offset_div2**, respectively.

slice_cb_beta_offset_div2 and **slice_cb_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for the current slice. The values of **slice_cb_beta_offset_div2** and **slice_cb_tc_offset_div2** shall both be in the range of -12 to 12, inclusive. When not present, the values of **slice_cb_beta_offset_div2** and **slice_cb_tc_offset_div2** are inferred to be equal to **ph_cb_beta_offset_div2** and **ph_cb_tc_offset_div2**, respectively.

slice_cr_beta_offset_div2 and **slice_cr_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for the current slice. The values of **slice_cr_beta_offset_div2** and **slice_cr_tc_offset_div2** shall both be in the range of -12 to 12, inclusive. When not present, the values of **slice_cr_beta_offset_div2** and **slice_cr_tc_offset_div2** are inferred to be equal to **ph_cr_beta_offset_div2** and **ph_cr_tc_offset_div2**, respectively.

slice_ts_residual_coding_disabled_flag equal to 1 specifies that the **residual_coding()** syntax structure is used to parse the residual samples of a transform skip block for the current slice. **slice_ts_residual_coding_disabled_flag** equal to 0 specifies that the **residual_ts_coding()** syntax structure is used to parse the residual samples of a transform skip block for the current slice. When **slice_ts_residual_coding_disabled_flag** is not present, it is inferred to be equal to 0.

slice_lmcs_enabled_flag equal to 1 specifies that luma mapping with chroma scaling is enabled for the current slice. **slice_lmcs_enabled_flag** equal to 0 specifies that luma mapping with chroma scaling is not enabled for the current slice. When **slice_lmcs_enabled_flag** is not present, it is inferred to be equal to 0.

slice_explicit_scaling_list_used_flag equal to 1 specifies that the explicit scaling list, which is signalled in the referenced scaling list APS with **aps_params_type** equal to **SCALING_APS** and **adaptation_parameter_set_id** equal to **ph_scaling_list_aps_id**, is used in the scaling process for transform coefficients when decoding of the current slice. **slice_explicit_scaling_list_used_flag** equal to 0 specifies that no explicit scaling list is not used in the scaling process for transform coefficients when decoding of the current slice. When not present, the value of **slice_explicit_scaling_list_used_flag** is inferred to be equal to 0.

The variable **NumEntryPoints**, which specifies the number of entry points in the current slice, is derived as follows:

```

NumEntryPoints = 0
for( i = 1; i < NumCtusInCurrSlice; i++ ) {
    ctbAddrX = CtbAddrInCurrSlice[ i ] % PicWidthInCtbsY
    ctbAddrY = CtbAddrInCurrSlice[ i ] / PicWidthInCtbsY

    (144)  prevCtbAddrX = CtbAddrInCurrSlice[ i - 1 ] % PicWidthInCtbsY
    prevCtbAddrY = CtbAddrInCurrSlice[ i - 1 ] / PicWidthInCtbsY
    if( CtbToTileRowBd[ ctbAddrY ] != CtbToTileRowBd[ prevCtbAddrY ] ||
        CtbToTileColBd[ ctbAddrX ] != CtbToTileColBd[ prevCtbAddrX ] ||
        ( ctbAddrY != prevCtbAddrY && sps_wpp_entry_point_offsets_present_flag ) )
        NumEntryPoints++
}

```

offset_len_minus1 plus 1 specifies the length, in bits, of the **entry_point_offset_minus1[i]** syntax elements. The value of **offset_len_minus1** shall be in the range of 0 to 31, inclusive.

entry_point_offset_minus1[*i*] plus 1 specifies the *i*-th entry point offset in bytes, and is represented by **offset_len_minus1** plus 1 bits. The slice data that follow the slice header consists of **NumEntryPoints** + 1 subsets, with subset index values ranging from 0 to **NumEntryPoints**, inclusive. The first byte of the slice data is considered byte 0. When present, emulation prevention bytes that appear in the slice data portion of the coded slice NAL unit are counted as part of the slice data for purposes of subset identification. Subset 0 consists of bytes 0 to **entry_point_offset_minus1**[0], inclusive, of the coded slice data, subset *k*, with *k* in the range of 1 to **NumEntryPoints** - 1, inclusive, consists of bytes **firstByte**[*k*] to **lastByte**[*k*], inclusive, of the coded slice data with **firstByte**[*k*] and **lastByte**[*k*] defined as:

$$\text{firstByte}[k] = \sum_{n=1}^k (\text{entry_point_offset_minus1}[n-1] + 1)$$

$$\text{lastByte}[k] = \text{firstByte}[k] + \text{entry_point_offset_minus1}[k]$$

The last subset (with subset index equal to **NumEntryPoints**) consists of the remaining bytes of the coded slice data.

When **sps_entropy_coding_sync_enabled_flag** is equal to 0 and the slice contains one or more complete tiles, each subset shall consist of all coded bits of all CTUs in the slice that are within the same tile, and the number of subsets (i.e., the value of **NumEntryPoints** + 1) shall be equal to the number of tiles in the slice.

When **sps_entropy_coding_sync_enabled_flag** is equal to 0 and the slice contains a subset of CTU rows from a single tile, the **NumEntryPoints** shall be 0, and the number of subsets shall be 1. The subset shall consist of all coded bits of all CTUs in the slice.

When **sps_entropy_coding_sync_enabled_flag** is equal to 1, each subset *k* with *k* in the range of 0 to **NumEntryPoints**, inclusive, shall consist of all coded bits of all CTUs in a CTU row within a tile, and the number of subsets (i.e., the value of **NumEntryPoints** + 1) shall be equal to the total number of tile-specific CTU rows in the slice.

slice_header_extension_length specifies the length of the slice header extension data in bytes, not including the bits used for signalling **slice_header_extension_length** itself. The value of **slice_header_extension_length** shall be in the range of 0 to 256, inclusive. When not present, the value of **slice_header_extension_length** is inferred to be equal to 0.

slice_header_extension_data_byte[*i*] may have any value. Decoders conforming to this version of this Specification shall ignore the values of all the **slice_header_extension_data_byte**[*i*] syntax elements. Its value does not affect decoder conformance to profiles specified in this version of specification.

As provided above, in JVET-Q2001, a PPS syntax structure may include **dbf_info_in_ph_flag** which indicates whether deblocking filter information is present in the PH syntax structure or in slice headers referring to the PPS. The placement of syntax element **dbf_info_in_ph_flag** in the PPS syntax structure as provided in JVET-Q2001 may be less than ideal. Further, the PPS syntax structure in JVET-Q2001 includes syntax elements indicating deblocking parameters. The signaling of the syntax elements indicating deblocking parameters in JVET-Q2001 is less than ideal.

FIG. 1 is a block diagram illustrating an example of a system that may be configured to code (i.e., encode and/or decode) video data according to one or more techniques of

this disclosure. System 100 represents an example of a system that may encapsulate video data according to one or more techniques of this disclosure. As illustrated in FIG. 1, system 100 includes source device 102, communications medium 110, and destination device 120. In the example illustrated in FIG. 1, source device 102 may include any device configured to encode video data and transmit encoded video data to communications medium 110. Destination device 120 may include any device configured to receive encoded video data via communications medium 110 and to decode encoded video data. Source device 102 and/or destination device 120 may include computing devices equipped for wired and/or wireless communications and may include, for example, set top boxes, digital video recorders, televisions, desktop, laptop or tablet computers, gaming consoles, medical imaging devices, and mobile devices, including, for example, smartphones, cellular telephones, personal gaming devices.

[0041] Communications medium 110 may include any combination of wireless and wired communication media, and/or storage devices. Communications medium 110 may include coaxial cables, fiber optic cables, twisted pair cables, wireless transmitters and receivers, routers, switches, repeaters, base stations, or any other equipment that may be useful to facilitate communications between various devices and sites. Communications medium 110 may include one or more networks. For example, communications medium 110 may include a network configured to enable access to the World Wide Web, for example, the Internet. A network may operate according to a combination of one or more telecommunication protocols. Telecommunications protocols may include proprietary aspects and/or may include standardized telecommunication protocols. Examples of standardized telecommunications protocols include Digital Video Broadcasting (DVB) standards, Advanced Television Systems Committee (ATSC) standards, Integrated Services Digital Broadcasting (ISDB) standards, Data Over Cable Service Interface Specification (DOCSIS) standards, Global System Mobile Communications (GSM) standards, code division multiple access (CDMA) standards, 3rd Generation Partnership Project (3GPP) standards, European Telecommunications Standards Institute (ETSI) standards, Internet Protocol (IP) standards, Wireless Application Protocol (WAP) standards, and Institute of Electrical and Electronics Engineers (IEEE) standards.

[0042] Storage devices may include any type of device or storage medium capable of storing data. A storage medium may include a tangible or non-transitory computer-readable media. A computer readable medium may include optical discs, flash memory, magnetic memory, or any other suitable digital storage media. In some examples, a memory device or portions thereof may be described as non-volatile memory and in other examples portions of memory devices may be described as volatile memory.

Examples of volatile memories may include random access memories (RAM), dynamic random access memories (DRAM), and static random access memories (SRAM). Examples of non-volatile memories may include magnetic hard discs, optical discs, floppy discs, flash memories, or forms of electrically programmable memories (EPROM) or electrically erasable and programmable (EEPROM) memories. Storage device(s) may include memory cards (e.g., a Secure Digital (SD) memory card), internal/external hard disk drives, and/or internal/external solid state drives. Data may be stored on a storage device according to a defined file format.

[0043] FIG. 4 is a conceptual drawing illustrating an example of components that may be included in an implementation of system 100. In the example implementation illustrated in FIG. 4, system 100 includes one or more computing devices 402A-402N, television service network 404, television service provider site 406, wide area network 408, local area network 410, and one or more content provider sites 412A-412N. The implementation illustrated in FIG. 4 represents an example of a system that may be configured to allow digital media content, such as, for example, a movie, a live sporting event, etc., and data and applications and media presentations associated therewith to be distributed to and accessed by a plurality of computing devices, such as computing devices 402A-402N. In the example illustrated in FIG. 4, computing devices 402A-402N may include any device configured to receive data from one or more of television service network 404, wide area network 408, and/or local area network 410. For example, computing devices 402A-402N may be equipped for wired and/or wireless communications and may be configured to receive services through one or more data channels and may include televisions, including so-called smart televisions, set top boxes, and digital video recorders. Further, computing devices 402A-402N may include desktop, laptop, or tablet computers, gaming consoles, mobile devices, including, for example, “smart” phones, cellular telephones, and personal gaming devices.

[0044] Television service network 404 is an example of a network configured to enable digital media content, which may include television services, to be distributed. For example, television service network 404 may include public over-the-air television networks, public or subscription-based satellite television service provider networks, and public or subscription-based cable television provider networks and/or over the top or Internet service providers. It should be noted that although in some examples television service network 404 may primarily be used to enable television services to be provided, television service network 404 may also enable other types of data and services to be provided according to any combination of the telecommunication protocols described herein. Further, it should be noted that in some examples, television service network 404 may enable two-way communications between

television service provider site 406 and one or more of computing devices 402A-402N. Television service network 404 may comprise any combination of wireless and/or wired communication media. Television service network 404 may include coaxial cables, fiber optic cables, twisted pair cables, wireless transmitters and receivers, routers, switches, repeaters, base stations, or any other equipment that may be useful to facilitate communications between various devices and sites. Television service network 404 may operate according to a combination of one or more telecommunication protocols. Telecommunications protocols may include proprietary aspects and/or may include standardized telecommunication protocols. Examples of standardized telecommunications protocols include DVB standards, ATSC standards, ISDB standards, DTMB standards, DMB standards, Data Over Cable Service Interface Specification (DOCSIS) standards, HbbTV standards, W3C standards, and UPnP standards.

[0045] Referring again to FIG. 4, television service provider site 406 may be configured to distribute television service via television service network 404. For example, television service provider site 406 may include one or more broadcast stations, a cable television provider, or a satellite television provider, or an Internet-based television provider. For example, television service provider site 406 may be configured to receive a transmission including television programming through a satellite uplink/downlink. Further, as illustrated in FIG. 4, television service provider site 406 may be in communication with wide area network 408 and may be configured to receive data from content provider sites 412A-412N. It should be noted that in some examples, television service provider site 406 may include a television studio and content may originate therefrom.

[0046] Wide area network 408 may include a packet based network and operate according to a combination of one or more telecommunication protocols. Telecommunications protocols may include proprietary aspects and/or may include standardized telecommunication protocols. Examples of standardized telecommunications protocols include Global System Mobile Communications (GSM) standards, code division multiple access (CDMA) standards, 3rd Generation Partnership Project (3GPP) standards, European Telecommunications Standards Institute (ETSI) standards, European standards (EN), IP standards, Wireless Application Protocol (WAP) standards, and Institute of Electrical and Electronics Engineers (IEEE) standards, such as, for example, one or more of the IEEE 802 standards (e.g., Wi-Fi). Wide area network 408 may comprise any combination of wireless and/or wired communication media. Wide area network 408 may include coaxial cables, fiber optic cables, twisted pair cables, Ethernet cables, wireless transmitters and receivers, routers, switches, repeaters, base stations, or any other equipment that may be useful to facilitate communications

between various devices and sites. In one example, wide area network 408 may include the Internet. Local area network 410 may include a packet based network and operate according to a combination of one or more telecommunication protocols. Local area network 410 may be distinguished from wide area network 408 based on levels of access and/or physical infrastructure. For example, local area network 410 may include a secure home network.

[0047] Referring again to FIG. 4, content provider sites 412A-412N represent examples of sites that may provide multimedia content to television service provider site 406 and/or computing devices 402A-402N. For example, a content provider site may include a studio having one or more studio content servers configured to provide multimedia files and/or streams to television service provider site 406. In one example, content provider sites 412A-412N may be configured to provide multimedia content using the IP suite. For example, a content provider site may be configured to provide multimedia content to a receiver device according to Real Time Streaming Protocol (RTSP), HTTP, or the like. Further, content provider sites 412A-412N may be configured to provide data, including hypertext based content, and the like, to one or more of receiver devices computing devices 402A-402N and/or television service provider site 406 through wide area network 408. Content provider sites 412A-412N may include one or more web servers. Data provided by data provider site 412A-412N may be defined according to data formats.

[0048] Referring again to FIG. 1, source device 102 includes video source 104, video encoder 106, data encapsulator 107, and interface 108. Video source 104 may include any device configured to capture and/or store video data. For example, video source 104 may include a video camera and a storage device operably coupled thereto. Video encoder 106 may include any device configured to receive video data and generate a compliant bitstream representing the video data. A compliant bitstream may refer to a bitstream that a video decoder can receive and reproduce video data therefrom. Aspects of a compliant bitstream may be defined according to a video coding standard. When generating a compliant bitstream video encoder 106 may compress video data. Compression may be lossy (discernible or indiscernible to a viewer) or lossless. FIG. 5 is a block diagram illustrating an example of video encoder 500 that may implement the techniques for encoding video data described herein. It should be noted that although example video encoder 500 is illustrated as having distinct functional blocks, such an illustration is for descriptive purposes and does not limit video encoder 500 and/or sub-components thereof to a particular hardware or software architecture. Functions of video encoder 500 may be realized using any combination of hardware, firmware, and/or software implementations.

[0049] Video encoder 500 may perform intra prediction coding and inter prediction coding

of picture areas, and, as such, may be referred to as a hybrid video encoder. In the example illustrated in FIG. 5, video encoder 500 receives source video blocks. In some examples, source video blocks may include areas of picture that has been divided according to a coding structure. For example, source video data may include macroblocks, CTUs, CBs, sub-divisions thereof, and/or another equivalent coding unit. In some examples, video encoder 500 may be configured to perform additional sub-divisions of source video blocks. It should be noted that the techniques described herein are generally applicable to video coding, regardless of how source video data is partitioned prior to and/or during encoding. In the example illustrated in FIG. 5, video encoder 500 includes summer 502, transform coefficient generator 504, coefficient quantization unit 506, inverse quantization and transform coefficient processing unit 508, summer 510, intra prediction processing unit 512, inter prediction processing unit 514, filter unit 516, and entropy encoding unit 518. As illustrated in FIG. 5, video encoder 500 receives source video blocks and outputs a bitstream.

[0050] In the example illustrated in FIG. 5, video encoder 500 may generate residual data by subtracting a predictive video block from a source video block. The selection of a predictive video block is described in detail below. Summer 502 represents a component configured to perform this subtraction operation. In one example, the subtraction of video blocks occurs in the pixel domain. Transform coefficient generator 504 applies a transform, such as a discrete cosine transform (DCT), a discrete sine transform (DST), or a conceptually similar transform, to the residual block or sub-divisions thereof (e.g., four 8 x 8 transforms may be applied to a 16 x 16 array of residual values) to produce a set of residual transform coefficients. Transform coefficient generator 504 may be configured to perform any and all combinations of the transforms included in the family of discrete trigonometric transforms, including approximations thereof. Transform coefficient generator 504 may output transform coefficients to coefficient quantization unit 506. Coefficient quantization unit 506 may be configured to perform quantization of the transform coefficients. The quantization process may reduce the bit depth associated with some or all of the coefficients. The degree of quantization may alter the rate-distortion (i.e., bit-rate vs. quality of video) of encoded video data. The degree of quantization may be modified by adjusting a quantization parameter (QP). A quantization parameter may be determined based on slice level values and/or CU level values (e.g., CU delta QP values). QP data may include any data used to determine a QP for quantizing a particular set of transform coefficients. As illustrated in FIG. 5, quantized transform coefficients (which may be referred to as level values) are output to inverse quantization and transform coefficient processing unit 508. Inverse quantization and transform coefficient processing unit 508 may be configured to apply an inverse quantization and an inverse transformation to

generate reconstructed residual data. As illustrated in FIG. 5, at summer 510, reconstructed residual data may be added to a predictive video block. In this manner, an encoded video block may be reconstructed and the resulting reconstructed video block may be used to evaluate the encoding quality for a given prediction, transformation, and/or quantization. Video encoder 500 may be configured to perform multiple coding passes (e.g., perform encoding while varying one or more of a prediction, transformation parameters, and quantization parameters). The rate-distortion of a bitstream or other system parameters may be optimized based on evaluation of reconstructed video blocks. Further, reconstructed video blocks may be stored and used as reference for predicting subsequent blocks.

- [0051] Referring again to FIG. 5, intra prediction processing unit 512 may be configured to select an intra prediction mode for a video block to be coded. Intra prediction processing unit 512 may be configured to evaluate a frame and determine an intra prediction mode to use to encode a current block. As described above, possible intra prediction modes may include planar prediction modes, DC prediction modes, and angular prediction modes. Further, it should be noted that in some examples, a prediction mode for a chroma component may be inferred from a prediction mode for a luma prediction mode. Intra prediction processing unit 512 may select an intra prediction mode after performing one or more coding passes. Further, in one example, intra prediction processing unit 512 may select a prediction mode based on a rate-distortion analysis. As illustrated in FIG. 5, intra prediction processing unit 512 outputs intra prediction data (e.g., syntax elements) to entropy encoding unit 518 and transform coefficient generator 504. As described above, a transform performed on residual data may be mode dependent (e.g., a secondary transform matrix may be determined based on a prediction mode).
- [0052] Referring again to FIG. 5, inter prediction processing unit 514 may be configured to perform inter prediction coding for a current video block. Inter prediction processing unit 514 may be configured to receive source video blocks and calculate a motion vector for PUs of a video block. A motion vector may indicate the displacement of a prediction unit of a video block within a current video frame relative to a predictive block within a reference frame. Inter prediction coding may use one or more reference pictures. Further, motion prediction may be uni-predictive (use one motion vector) or bi-predictive (use two motion vectors). Inter prediction processing unit 514 may be configured to select a predictive block by calculating a pixel difference determined by, for example, sum of absolute difference (SAD), sum of square difference (SSD), or other difference metrics. As described above, a motion vector may be determined and specified according to motion vector prediction. Inter prediction processing unit 514 may be configured to perform motion vector prediction, as described above. Inter

prediction processing unit 514 may be configured to generate a predictive block using the motion prediction data. For example, inter prediction processing unit 514 may locate a predictive video block within a frame buffer (not shown in FIG. 5). It should be noted that inter prediction processing unit 514 may further be configured to apply one or more interpolation filters to a reconstructed residual block to calculate sub-integer pixel values for use in motion estimation. Inter prediction processing unit 514 may output motion prediction data for a calculated motion vector to entropy encoding unit 518.

[0053] Referring again to FIG. 5, filter unit 516 receives reconstructed video blocks and coding parameters and outputs modified reconstructed video data. Filter unit 516 may be configured to perform deblocking, SAO, and ALF filtering, as described above. It should be noted that as illustrated in FIG. 5, intra prediction processing unit 512 and inter prediction processing unit 514 may receive modified reconstructed video block via filter unit 216. Entropy encoding unit 518 receives quantized transform coefficients and predictive syntax data (i.e., intra prediction data and motion prediction data). It should be noted that in some examples, coefficient quantization unit 506 may perform a scan of a matrix including quantized transform coefficients before the coefficients are output to entropy encoding unit 518. In other examples, entropy encoding unit 518 may perform a scan. Entropy encoding unit 518 may be configured to perform entropy encoding according to one or more of the techniques described herein. In this manner, video encoder 500 represents an example of a device configured to generate encoded video data according to one or more techniques of this disclosure.

[0054] Referring again to FIG. 1, data encapsulator 107 may receive encoded video data and generate a compliant bitstream, e.g., a sequence of NAL units according to a defined data structure. A device receiving a compliant bitstream can reproduce video data therefrom. Further, as described above, sub-bitstream extraction may refer to a process where a device receiving a ITU-T H.265 compliant bitstream forms a new ITU-T H.265 compliant bitstream by discarding and/or modifying data in the received bitstream. It should be noted that the term conforming bitstream may be used in place of the term compliant bitstream. In one example, data encapsulator 107 may be configured to generate syntax according to one or more techniques described herein. It should be noted that data encapsulator 107 need not necessarily be located in the same physical device as video encoder 106. For example, functions described as being performed by video encoder 106 and data encapsulator 107 may be distributed among devices illustrated in FIG. 4.

[0055]

As described above, the signaling provided in JVET-Q2001 for signaling deblocking parameter in a PPS is less than ideal. In one example, according to the techniques herein the signalling location of syntax element **dbf_info_in_ph_flag** may be moved to locate it together with the other deblocking control parameters signalling. This can also avoid a conditional signalling (if) check (when **deblocking_filter_control_present_flag** is equal to 0). Table 8, illustrates an example of the relevant syntax of a **pic_parameter_set_rbsp()** according to the techniques herein.

pic_parameter_set_rbsp() {	Descriptor
pps_pic_parameter_set_id	ue(v)
...	u(4)
deblocking_filter_control_present_flag	u(1)
if(deblocking_filter_control_present_flag) {	
deblocking_filter_override_enabled_flag	u(1)
if(deblocking_filter_override_enabled_flag)	
dbf_info_in_ph_flag	u(1)
pps_deblocking_filter_disabled_flag	u(1)
if(!pps_deblocking_filter_disabled_flag) {	
pps_beta_offset_div2	se(v)
pps_tc_offset_div2	se(v)
pps_cb_beta_offset_div2	se(v)
pps_cb_tc_offset_div2	se(v)
pps_cr_beta_offset_div2	se(v)
pps_cr_tc_offset_div2	se(v)
}	
}	
rpl_info_in_ph_flag	u(1)
...	
}	

Table 8

In another example, the syntax may instead be as provided in Table 9:

pic_parameter_set_rbsp() {	Descriptor
pps_pic_parameter_set_id	ue(v)
...	u(4)
deblocking_filter_control_present_flag	u(1)
if(deblocking_filter_control_present_flag) {	
deblocking_filter_override_enabled_flag	u(1)
pps_deblocking_filter_disabled_flag	u(1)
if(!pps_deblocking_filter_disabled_flag) {	
pps_beta_offset_div2	se(v)
pps_tc_offset_div2	se(v)
pps_cb_beta_offset_div2	se(v)
pps_cb_tc_offset_div2	se(v)
pps_cr_beta_offset_div2	se(v)
pps_cr_tc_offset_div2	se(v)
}	
}	
if(deblocking_filter_override_enabled_flag)	
dbf_info_in_ph_flag	u(1)
}	
rpl_info_in_ph_flag	u(1)
...	
}	

Table 9

In another example, the syntax may instead be as provided in Table 10:

pic_parameter_set_rbsp() {	Descriptor
pps_pic_parameter_set_id	ue(v)
...	u(4)
deblocking_filter_control_present_flag	u(1)
if(deblocking_filter_control_present_flag) {	
deblocking_filter_override_enabled_flag	u(1)
pps_deblocking_filter_disabled_flag	u(1)
if(deblocking_filter_override_enabled_flag)	
dbf_info_in_ph_flag	u(1)
if(!pps_deblocking_filter_disabled_flag) {	
pps_beta_offset_div2	se(v)
pps_tc_offset_div2	se(v)
pps_cb_beta_offset_div2	se(v)
pps_cb_tc_offset_div2	se(v)
pps_cr_beta_offset_div2	se(v)
pps_cr_tc_offset_div2	se(v)
}	
}	
rpl_info_in_ph_flag	u(1)
...	
}	

Table 10

With respect to Tables 8-10, the semantics may be based on the semantics provided above, with in one example, the semantics of syntax element **dbf_info_in_ph_flag** based on the following:

dbf_info_in_ph_flag equal to 1 specifies that deblocking filter information is present in the PH syntax structure and not present in slice headers referring to the PPS that do not contain a PH syntax structure. **dbf_info_in_ph_flag** equal to 0 specifies that deblocking filter information is not present in the PH syntax structure and may be present in slice headers referring to the PPS that do not contain a PH syntax structure. When not present, the value of **dbf_info_in_ph_flag** is inferred to be equal to 0.

It should be noted that in another variant example, the conditional signaling of syntax element **dbf_info_in_ph_flag** may be moved to inside the **if(deblocking_filter_control_present_flag)** control block.

For example this may be done at a location different than the locations shown in Table 8-10.

In yet another example, the location of syntax element **dbf_info_in_ph_flag** may be moved to locate it near the other deblocking control parameters signalling as shown in Table 10A. In the example of Table 10, it is located right after the control block of **if(deblocking_filter_control_present_flag)**, but before any non deblocking related syntax elements.

pic_parameter_set_rbsp() {	Descriptor
pps_pic_parameter_set_id	ue(v)
...	u(4)
deblocking_filter_control_present_flag	u(1)
if(deblocking_filter_control_present_flag) {	
deblocking_filter_override_enabled_flag	u(1)
pps_deblocking_filter_disabled_flag	u(1)
if(!pps_deblocking_filter_disabled_flag) {	
pps_beta_offset_div2	se(v)
pps_tc_offset_div2	se(v)
pps_cb_beta_offset_div2	se(v)
pps_cb_tc_offset_div2	se(v)
pps_cr_beta_offset_div2	se(v)
pps_cr_tc_offset_div2	se(v)
}	
}	
if(deblocking_filter_override_enabled_flag)	
dbf_info_in_ph_flag	u(1)
rpl_info_in_ph_flag	u(1)
...	
}	

Table 10A

In one example, according to the techniques herein, the presence of chroma related deblocking filter control parameters, **pps_cb_beta_offset_div2**, **pps_cb_tc_offset_div2**, **pps_cr_beta_offset_div2**, and **pps_cr_tc_offset_div2**, in a PPS may be based on a flag. When not signaled the value for those syntax elements may be inferred. Table 11 illustrates an example of the relevant syntax of a **pic_parameter_set_rbsp()** according to the techniques herein.

pic_parameter_set_rbsp() {	Descriptor
pps_pic_parameter_set_id	ue(v)
...	u(4)
deblocking_filter_control_present_flag	u(1)
if(deblocking_filter_control_present_flag) {	
deblocking_filter_override_enabled_flag	u(1)
if(!deblocking_filter_override_enabled_flag) {	
pps_deblocking_filter_disabled_flag	u(1)
if(!pps_deblocking_filter_disabled_flag) {	
pps_beta_offset_div2	se(v)
pps_tc_offset_div2	se(v)
pps_chroma_dbf_params_present_flag	u(1)
if(pps_chroma_dbf_params_present_flag) {	
pps_cb_beta_offset_div2	se(v)
pps_cb_tc_offset_div2	se(v)
pps_cr_beta_offset_div2	se(v)
pps_cr_tc_offset_div2	se(v)
}	
}	
}	
}	
rpl_info_in_ph_flag	u(1)
...	
}	

Table 11

With respect to Table 11, the semantics may be based on the semantics provided above, with in one example,

the semantics of syntax element **pps_chroma_dbf_params_flag** based on the following:

pps_chroma_dbf_params_present_flag equal to 1 specifies that syntax elements **pps_cb_beta_offset_div2**, **pps_cb_tc_offset_div2**, **pps_cr_beta_offset_div2**, **pps_cr_tc_offset_div2** are present in the PPS RBSP syntax structure. **pps_chroma_dbf_params_present_flag** equal to 0 specifies that syntax elements **pps_cb_beta_offset_div2**, **pps_cb_tc_offset_div2**, **pps_cr_beta_offset_div2**, **pps_cr_tc_offset_div2** are not present in the PPS RBSP syntax structure. When **ChromaArrayType** is equal to 0, the value of **pps_chroma_dbf_params_present_flag** shall be equal to 0.

In another example, a general flag which specifies if chroma related parameters are signaled in PPS may be signaled. Table 12 illustrates an example of the relevant syntax of a `pic_parameter_set_rbsp()` according to the techniques herein.

<code>Pic_parameter_set_rbsp() {</code>	Descriptor
<code>pps_pic_parameter_set_id</code>	<code>ue(v)</code>
<code>pps_seq_parameter_set_id</code>	<code>u(4)</code>
<code>pps_chroma_params_present_flag</code>	<code>u(1)</code>
<code>...</code>	<code>u(4)</code>
<code>deblocking_filter_control_present_flag</code>	<code>u(1)</code>
<code>if(deblocking_filter_control_present_flag) {</code>	
<code>deblocking_filter_override_enabled_flag</code>	<code>u(1)</code>
<code>if(!deblocking_filter_override_enabled_flag) {</code>	
<code>pps_deblocking_filter_disabled_flag</code>	<code>u(1)</code>
<code>if(!pps_deblocking_filter_disabled_flag) {</code>	
<code>pps_beta_offset_div2</code>	<code>se(v)</code>
<code>pps_tc_offset_div2</code>	<code>se(v)</code>
<code>if(pps_chroma_params_present_flag) {</code>	
<code>pps_cb_beta_offset_div2</code>	<code>se(v)</code>
<code>pps_cb_tc_offset_div2</code>	<code>se(v)</code>
<code>pps_cr_beta_offset_div2</code>	<code>se(v)</code>
<code>pps_cr_tc_offset_div2</code>	<code>se(v)</code>
<code>}</code>	
<code>}</code>	
<code>}</code>	
<code>}</code>	
<code>rpl_info_in_ph_flag</code>	<code>u(1)</code>
<code>...</code>	
<code>}</code>	

Table 12

With respect to Table 12, the semantics may be based on the semantics provided above, with in one example, the semantics of syntax element **pps_chroma_params_present_flag** based on the following:

pps_chroma_params_present_flag equal to 1 specifies that chroma related PPS syntax elements (including **pps_cb_beta_offset_div2**, **pps_cb_tc_offset_div2**, **pps_cr_beta_offset_div2**, **pps_cr_tc_offset_div2**) are present in the PPS RBSP syntax structure. **pps_chroma_params_present_flag** equal to 0 specifies that chroma related PPS syntax elements (including **pps_cb_beta_offset_div2**, **pps_cb_tc_offset_div2**, **pps_cr_beta_offset_div2**, **pps_cr_tc_offset_div2**) are not present in the PPS RBSP syntax structure. When ChromaArrayType is equal to 0, the value of **pps_chroma_params_present_flag** shall be equal to 0.

In this case, additionally, according to the techniques herein, (1) Either the flag **pps_chroma_tool_offsets_present_flag** may be conditionally signalled only when **pps_chroma_params_present_flag** is equal to 1 and its value inferred to be equal to 0 when not present OR (2) The flag **pps_chroma_tool_offsets_present_flag** may be removed and replaced everywhere by the flag **pps_chroma_params_present_flag**. Table 13 illustrates an example, where the flag **pps_chroma_tool_offsets_present_flag** is removed and replaced everywhere by the flag **pps_chroma_params_present_flag**. Additionally, in this case, the flag **pps_chroma_params_present_flag** is used to determine if syntax elements **pps_cb_beta_offset_div2**, **pps_cb_tc_offset_div2**, **pps_cr_beta_offset_div2**, **pps_cr_tc_offset_div2** are included (when the flag is equal to 1) or not (when the flag is equal to 0).

pic_parameter_set_rbsp() {	Descriptor
pps_pic_parameter_set_id	ue(v)
pps_seq_parameter_set_id	u(4)
pps_chroma_params_present_flag	u(1)
...	se(v)
cu_qp_delta_enabled_flag	u(1)
if(pps_chroma_params_present_flag) {	
pps_cb_qp_offset	se(v)
pps_cr_qp_offset	se(v)
pps_joint_cbr_qp_offset_present_flag	u(1)
if(pps_joint_cbr_qp_offset_present_flag)	
pps_joint_cbr_qp_offset_value	se(v)
pps_slice_chroma_qp_offsets_present_flag	u(1)
pps_cu_chroma_qp_offset_list_enabled_flag	u(1)
}	
if(pps_cu_chroma_qp_offset_list_enabled_flag) {	
chroma_qp_offset_list_len_minus1	ue(v)
for(i = 0; i <= chroma_qp_offset_list_len_minus1; i++) {	
cb_qp_offset_list[i]	se(v)
cr_qp_offset_list[i]	se(v)
if(pps_joint_cbr_qp_offset_present_flag)	
joint_cbr_qp_offset_list[i]	se(v)
}	
}	
pps_weighted_pred_flag	u(1)
pps_weighted_bipred_flag	u(1)
deblocking_filter_control_present_flag	u(1)
if(deblocking_filter_control_present_flag) {	
deblocking_filter_override_enabled_flag	u(1)
pps_deblocking_filter_disabled_flag	u(1)

if(!pps_deblocking_filter_disabled_flag) {	
pps_beta_offset_div2	se(v)
pps_tc_offset_div2	se(v)
if(pps_chroma_params_present_flag) {	
pps_cb_beta_offset_div2	se(v)
pps_cb_tc_offset_div2	se(v)
pps_cr_beta_offset_div2	se(v)
pps_cr_tc_offset_div2	se(v)
}	
}	
}	
rpl_info_in_ph_flag	u(1)
if(deblocking_filter_override_enabled_flag)	
dbf_info_in_ph_flag	u(1)
sao_info_in_ph_flag	u(1)

Table 13

With respect to Table 13, the semantics may be based on the semantics provided above, with in one example, the semantics of syntax element **pps_chroma_params_present_flag** based on the following:

pps_chroma_params_present_flag equal to 1 specifies that chroma related PPS syntax elements (including pps_cb_qp_offset, pps_cr_qp_offset, pps_jointcbr_qp_offset_present_flag, pps_joint_cbr_qp_offset_value , pps_slice_chroma_qp_offsets_present_flag , pps_cu_chroma_qp_offset_list_enabled_flag, pps_cb_beta_offset_div2, pps_cb_tc_offset_div2, pps_cr_beta_offset_div2, pps_cr_tc_offset_div2) are present in the PPS RBSP syntax structure. pps_chroma_params_present_flag equal to 0 specifies that chroma related PPS syntax elements (including pps_cb_qp_offset, pps_cr_qp_offset, pps_jointcbr_qp_offset_present_flag, pps_joint_cbr_qp_offset_value , pps_slice_chroma_qp_offsets_present_flag , pps_cu_chroma_qp_offset_list_enabled_flag, pps_cb_beta_offset_div2, pps_cb_tc_offset_div2, pps_cr_beta_offset_div2, pps_cr_tc_offset_div2) are not present in the PPS RBSP syntax structure. When ChromaArrayType is equal to 0, the value of pps_chroma_params_present_flag shall be equal to 0.

In a variant the location of **pps_chroma_params_present_flag** inside a PPS may be different than one shown above. For example, it may be signalled at the current location of **pps_chroma_tool_offsets_present_flag**.

In another example, a **pps_separate_colour_plane_flag** may be signaled in a PPS and used to control the presence of **pps_cb_beta_offset_div2**, **pps_cb_tc_offset_div2**, **pps_cr_beta_offset_div2**, and **pps_cr_tc_offset_div2** parameters. Table 14 illustrates an example of the relevant syntax of a **pic_parameter_set_rbsp()** according to the techniques herein.

pic_parameter_set_rbsp() {	Descriptor
pps_pic_parameter_set_id	ue(v)
pps_separate_colour_plane_flag	u(1)
...	u(4)
deblocking_filter_control_present_flag	u(1)
if(deblocking_filter_control_present_flag) {	
deblocking_filter_override_enabled_flag	u(1)
if(!deblocking_filter_override_enabled_flag) {	
pps_deblocking_filter_disabled_flag	u(1)
if(!pps_deblocking_filter_disabled_flag) {	
pps_beta_offset_div2	se(v)
pps_tc_offset_div2	se(v)
if(pps_separate_colour_plane_flag!=1) {	
pps_cb_beta_offset_div2	se(v)
pps_cb_tc_offset_div2	se(v)
pps_cr_beta_offset_div2	se(v)
pps_cr_tc_offset_div2	se(v)
}	
}	
}	
}	
rpl_info_in_ph_flag	u(1)
...	
}	

Table 14

With respect to Table 14, the semantics may be based on the semantics provided above, with in one example, the semantics of syntax element **pps_separate_colour_plane_flag** and **separate_colour_plane_flag** based on the following:

pps_separate_colour_plane_flag has the same semantics meaning as the syntax element **separate_colour_plane_flag**. The value of **pps_separate_colour_plane_flag** in PPS and the value of **separate_colour_plane_flag** in SPS referred by each slice shall be the same.

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) and uses the monochrome coding syntax. In this case, each colour plane is associated with a specific **colour_plane_id** value.

NOTE – 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.

chroma_format_idc specifies the chroma sampling relative to the luma sampling as provided in Table 15:

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

Table 15

In another example, the chromaArrayType value may be signalled as an indicator in PPS and then if this value is equal to 0 then only **pps_cb_beta_offset_div2**, **pps_cb_tc_offset_div2**, **pps_cr_beta_offset_div2**, **pps_cr_tc_offset_div2** may be signaled in PPS. Table 16 illustrates an example of the relevant syntax of a `pic_parameter_set_rbsp()` according to the techniques herein.

<code>pic_parameter_set_rbsp()</code> {	Descriptor
pps_pic_parameter_set_id	ue(v)
chroma_array_type	u(2)
...	u(4)
deblocking_filter_control_present_flag	u(1)
if(deblocking_filter_control_present_flag) {	
deblocking_filter_override_enabled_flag	u(1)
if(!deblocking_filter_override_enabled_flag) {	
pps_deblocking_filter_disabled_flag	u(1)
if(!pps_deblocking_filter_disabled_flag) {	
pps_beta_offset_div2	se(v)
pps_tc_offset_div2	se(v)
if(chroma_array_type != 0) {	
pps_cb_beta_offset_div2	se(v)
pps_cb_tc_offset_div2	se(v)
pps_cr_beta_offset_div2	se(v)
pps_cr_tc_offset_div2	se(v)
}	
}	
}	
}	
rpl_info_in_ph_flag	u(1)
...	
}	

Table 16

With respect to Table 16, the semantics may be based on the semantics provided above, with in one example,

the semantics of syntax element **chroma_array_type** based on the following:

chroma_array_type has the same semantics meaning as the derived variable ChromaArrayType. The value of chroma_array_type in PPS and the value of ChromaArrayType derived from SPS referred by each slice shall be the same.

In a variant some other flag or indicator may be signaled in a parameter set (e.g., DPS, VPS, or some other parameter set) or a picture header or a slice header to indicate that chroma related syntax elements are signaled and/or chroma deblocking related syntax elements are signaled and this flag may be used to gate the presence of chroma deblocking syntax elements.

[0056]

In another example, one or more of the syntax elements **pps_cb_beta_offset_div2**, **pps_cb_tc_offset_div2**, **pps_cr_beta_offset_div2**, and **pps_cr_tc_offset_div2** may be signaled only when ChromaArrayType is not equal to 0. In another example, semantics of the syntax elements in a PPS may be modified as follows:

pps_cb_beta_offset_div2 and **pps_cb_tc_offset_div2** specify the default deblocking parameter offsets for β and tc (divided by 2) that are applied to the Cb component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of **pps_cb_beta_offset_div2** and **pps_cb_tc_offset_div2** shall both be in the range of -12 to 12, inclusive. When ChromaArrayType is equal to 0, **pps_cb_beta_offset_div2** and **pps_cb_tc_offset_div2** are not used in the decoding process and decoders shall ignore their value. When not present, the values of **pps_cb_beta_offset_div2** and **pps_cb_tc_offset_div2** are both inferred to be equal to 0.

In another example it may be a conformance requirement that:

When ChromaArrayType is equal to 0, **pps_cb_beta_offset_div2** and **pps_cb_tc_offset_div2** shall be equal to 0.

`pps_cr_beta_offset_div2` and `pps_cr_tc_offset_div2` specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of `pps_cr_beta_offset_div2` and `pps_cr_tc_offset_div2` shall both be in the range of -12 to 12 , inclusive. When `ChromaArrayType` is equal to 0, `pps_cr_beta_offset_div2` and `pps_cr_tc_offset_div2` are not used in the decoding process and decoders shall ignore their value. When not present, the values of `pps_cr_beta_offset_div2` and `pps_cr_tc_offset_div2` are both inferred to be equal to 0.

In another example it may be a conformance requirement that: When `ChromaArrayType` is equal to 0, `pps_cr_beta_offset_div2` and `pps_cr_tc_offset_div2` shall be equal to 0.

Additionally, the picture header signaling of chroma related deblocking control parameters may be modified as shown in Table 17. In this case **ph_cb_beta_offset_div2**, **ph_cb_tc_offset_div2**, **ph_cr_beta_offset_div2**, **ph_cr_tc_offset_div2** are only signaled when ChromaArrayType is not equal to 0.

picture_header_structure() {	Descriptor
gdr_or_irap_pic_flag	u(1)
...	
if(deblocking_filter_override_enabled_flag && dbf_info_in_ph_flag) {	
ph_deblocking_filter_override_flag	u(1)
if(ph_deblocking_filter_override_flag) {	
ph_deblocking_filter_disabled_flag	u(1)
if(!ph_deblocking_filter_disabled_flag) {	
ph_beta_offset_div2	se(v)
ph_tc_offset_div2	se(v)
if(ChromaArrayType != 0) {	
ph_cb_beta_offset_div2	se(v)
ph_cb_tc_offset_div2	se(v)
ph_cr_beta_offset_div2	se(v)
ph_cr_tc_offset_div2	se(v)
}	
}	
}	
}	
...	
}	

Table 17

With respect to Table 17, the semantics may be based on the semantics provided above and on the following:

ph_beta_offset_div2 and **ph_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the luma component for the slices associated with the PH. The values of **ph_beta_offset_div2** and **ph_tc_offset_div2** shall both be in the range of -12 to 12, inclusive. When not present, the values of **ph_beta_offset_div2** and **ph_tc_offset_div2** are inferred to be equal to **pps_beta_offset_div2** and **pps_tc_offset_div2**, respectively.

ph_cb_beta_offset_div2 and **ph_cb_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for the slices associated with the PH. The values of **ph_cb_beta_offset_div2** and **ph_cb_tc_offset_div2** shall both be in the range of -12 to 12, inclusive. When not present, the values of **ph_cb_beta_offset_div2** and **ph_cb_tc_offset_div2** are inferred to be equal to **pps_cb_beta_offset_div2** and **pps_cb_tc_offset_div2**, respectively.

ph_cr_beta_offset_div2 and **ph_cr_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for the slices associated with the PH. The values of **ph_cr_beta_offset_div2** and **ph_cr_tc_offset_div2** shall both be in the range of -12 to 12, inclusive. When not present, the values of **ph_cr_beta_offset_div2** and **ph_cr_tc_offset_div2** are inferred to be equal to **pps_cr_beta_offset_div2** and **pps_cr_tc_offset_div2**, respectively.

Additionally, the picture header signaling of chroma related deblocking control parameters may be modified as shown in Table 18. In this case, **slice_cb_beta_offset_div2**, **slice_cb_tc_offset_div2**, **slice_cr_beta_offset_div2**, **slice_cr_tc_offset_div2** are only signaled when ChromaArrayType is not equal to 0.

slice_header() {	Descriptor
picture_header_in_slice_header_flag	u(1)
...	
if(deblocking_filter_override_enabled_flag && !dbf_info_in_ph_flag)	
slice_deblocking_filter_override_flag	u(1)
if(slice_deblocking_filter_override_flag) {	
slice_deblocking_filter_disabled_flag	u(1)
if(!slice_deblocking_filter_disabled_flag) {	
slice_beta_offset_div2	se(v)
slice_tc_offset_div2	se(v)
if(ChromaArrayType != 0) {	
slice_cb_beta_offset_div2	se(v)
slice_cb_tc_offset_div2	se(v)
slice_cr_beta_offset_div2	se(v)
slice_cr_tc_offset_div2	se(v)
}	
}	
}	
...	
}	

Table 18

With respect to Table 18, the semantics may be based on the semantics provided above and on the following:

slice_beta_offset_div2 and **slice_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the luma component for the current slice. The values of **slice_beta_offset_div2** and **slice_tc_offset_div2** shall both be in the range of -12 to 12 , inclusive. When not present, the values of **slice_beta_offset_div2** and **slice_tc_offset_div2** are inferred to be equal to **ph_beta_offset_div2** and **ph_tc_offset_div2**, respectively.

slice_cb_beta_offset_div2 and **slice_cb_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for the current slice. The values of **slice_cb_beta_offset_div2** and **slice_cb_tc_offset_div2** shall both be in the range of -12 to 12 , inclusive. When not present, the values of **slice_cb_beta_offset_div2** and **slice_cb_tc_offset_div2** are inferred to be equal to **ph_cb_beta_offset_div2** and **ph_cb_tc_offset_div2**, respectively.

slice_cr_beta_offset_div2 and **slice_cr_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for the current slice. The values of **slice_cr_beta_offset_div2** and **slice_cr_tc_offset_div2** shall both be in the range of -12 to 12 , inclusive. When not present, the values of **slice_cr_beta_offset_div2** and **slice_cr_tc_offset_div2** are inferred to be equal to **ph_cr_beta_offset_div2** and **ph_cr_tc_offset_div2**, respectively.

In this manner, source device 102 represents an example of a device configured to signal a syntax element in a picture parameter set indicating whether chroma deblocking parameter syntax elements are included in the picture parameter set and conditionally signal the chroma deblocking parameter syntax elements in the picture parameter set based on the value of the syntax element indicating whether chroma deblocking parameter syntax elements are included in the picture parameter set.

[0057] Referring again to FIG. 1, interface 108 may include any device configured to receive data generated by data encapsulator 107 and transmit and/or store the data to a communications medium. Interface 108 may include a network interface card, such as an Ethernet card, and may include an optical transceiver, a radio frequency transceiver, or any other type of device that can send and/or receive information. Further, interface 108 may include a computer system interface that may enable a file to be stored on a storage device. For example, interface 108 may include a chipset supporting Peripheral Component Interconnect (PCI) and Peripheral Component Interconnect Express (PCIe) bus protocols, proprietary bus protocols, Universal Serial Bus (USB) protocols, I²C, or any other logical and physical structure that may be used to interconnect peer devices.

[0058] Referring again to FIG. 1, destination device 120 includes interface 122, data decapsulator 123, video decoder 124, and display 126. Interface 122 may include any device configured to receive data from a communications medium. Interface 122 may include a network interface card, such as an Ethernet card, and may include an optical

transceiver, a radio frequency transceiver, or any other type of device that can receive and/or send information. Further, interface 122 may include a computer system interface enabling a compliant video bitstream to be retrieved from a storage device. For example, interface 122 may include a chipset supporting PCI and PCIe bus protocols, proprietary bus protocols, USB protocols, I²C, or any other logical and physical structure that may be used to interconnect peer devices. Data decapsulator 123 may be configured to receive and parse any of the example syntax structures described herein.

[0059] Video decoder 124 may include any device configured to receive a bitstream (e.g., a sub-bitstream extraction) and/or acceptable variations thereof and reproduce video data therefrom. Display 126 may include any device configured to display video data. Display 126 may comprise one of a variety of display devices such as a liquid crystal display (LCD), a plasma display, an organic light emitting diode (OLED) display, or another type of display. Display 126 may include a High Definition display or an Ultra High Definition display. It should be noted that although in the example illustrated in FIG. 1, video decoder 124 is described as outputting data to display 126, video decoder 124 may be configured to output video data to various types of devices and/or sub-components thereof. For example, video decoder 124 may be configured to output video data to any communication medium, as described herein.

[0060] FIG. 6 is a block diagram illustrating an example of a video decoder that may be configured to decode video data according to one or more techniques of this disclosure (e.g., the decoding process for reference-picture list construction described above). In one example, video decoder 600 may be configured to decode transform data and reconstruct residual data from transform coefficients based on decoded transform data. Video decoder 600 may be configured to perform intra prediction decoding and inter prediction decoding and, as such, may be referred to as a hybrid decoder. Video decoder 600 may be configured to parse any combination of the syntax elements described above in Tables 1-18. Video decoder 600 may decode a picture based on or according to the processes described above, and further based on parsed values in Tables 1-18.

[0061] In the example illustrated in FIG. 6, video decoder 600 includes an entropy decoding unit 602, inverse quantization unit and transform coefficient processing unit 604, intra prediction processing unit 606, inter prediction processing unit 608, summer 610, post filter unit 612, and reference buffer 614. Video decoder 600 may be configured to decode video data in a manner consistent with a video coding system. It should be noted that although example video decoder 600 is illustrated as having distinct functional blocks, such an illustration is for descriptive purposes and does not limit video decoder 600 and/or sub-components thereof to a particular hardware or software

architecture. Functions of video decoder 600 may be realized using any combination of hardware, firmware, and/or software implementations.

[0062] As illustrated in FIG. 6, entropy decoding unit 602 receives an entropy encoded bitstream. Entropy decoding unit 602 may be configured to decode syntax elements and quantized coefficients from the bitstream according to a process reciprocal to an entropy encoding process. Entropy decoding unit 602 may be configured to perform entropy decoding according any of the entropy coding techniques described above. Entropy decoding unit 602 may determine values for syntax elements in an encoded bitstream in a manner consistent with a video coding standard. As illustrated in FIG. 6, entropy decoding unit 602 may determine a quantization parameter, quantized coefficient values, transform data, and prediction data from a bitstream. In the example, illustrated in FIG. 6, inverse quantization unit and transform coefficient processing unit 604 receives a quantization parameter, quantized coefficient values, transform data, and prediction data from entropy decoding unit 602 and outputs reconstructed residual data.

[0063] Referring again to FIG. 6, reconstructed residual data may be provided to summer 610. Summer 610 may add reconstructed residual data to a predictive video block and generate reconstructed video data. A predictive video block may be determined according to a predictive video technique (i.e., intra prediction and inter frame prediction). Intra prediction processing unit 606 may be configured to receive intra prediction syntax elements and retrieve a predictive video block from reference buffer 614. Reference buffer 614 may include a memory device configured to store one or more frames of video data. Intra prediction syntax elements may identify an intra prediction mode, such as the intra prediction modes described above. Inter prediction processing unit 608 may receive inter prediction syntax elements and generate motion vectors to identify a prediction block in one or more reference frames stored in reference buffer 616. Inter prediction processing unit 608 may produce motion compensated blocks, possibly performing interpolation based on interpolation filters. Identifiers for interpolation filters to be used for motion estimation with sub-pixel precision may be included in the syntax elements. Inter prediction processing unit 608 may use interpolation filters to calculate interpolated values for sub-integer pixels of a reference block. Post filter unit 614 may be configured to perform filtering on reconstructed video data. For example, post filter unit 614 may be configured to perform deblocking and/or Sample Adaptive Offset (SAO) filtering, e.g., based on parameters specified in a bitstream. Further, it should be noted that in some examples, post filter unit 614 may be configured to perform proprietary discretionary filtering (e.g., visual enhancements, such as, mosquito noise reduction). As illustrated in FIG. 6, a reconstructed video block may be output by video decoder 600. In this manner, video

decoder 600 represents an example of a device configured to parse a syntax element in a picture parameter set indicating whether chroma deblocking parameter syntax elements are included in the picture parameter set and conditionally parse the chroma deblocking parameter syntax elements in the picture parameter set based on the value of the syntax element indicating whether chroma deblocking parameter syntax elements are included in the picture parameter set.

[0064] In one or more examples, the functions described may be implemented in hardware, software, firmware, or any combination thereof. If implemented in software, the functions may be stored on or transmitted over as one or more instructions or code on a computer-readable medium and executed by a hardware-based processing unit. Computer-readable media may include computer-readable storage media, which corresponds to a tangible medium such as data storage media, or communication media including any medium that facilitates transfer of a computer program from one place to another, e.g., according to a communication protocol. In this manner, computer-readable media generally may correspond to (1) tangible computer-readable storage media which is non-transitory or (2) a communication medium such as a signal or carrier wave. Data storage media may be any available media that can be accessed by one or more computers or one or more processors to retrieve instructions, code and/or data structures for implementation of the techniques described in this disclosure. A computer program product may include a computer-readable medium.

[0065] By way of example, and not limitation, such computer-readable storage media can comprise RAM, ROM, EEPROM, CD-ROM or other optical disk storage, magnetic disk storage, or other magnetic storage devices, flash memory, or any other medium that can be used to store desired program code in the form of instructions or data structures and that can be accessed by a computer. Also, any connection is properly termed a computer-readable medium. For example, if instructions are transmitted from a website, server, or other remote source using a coaxial cable, fiber optic cable, twisted pair, digital subscriber line (DSL), or wireless technologies such as infrared, radio, and microwave, then the coaxial cable, fiber optic cable, twisted pair, DSL, or wireless technologies such as infrared, radio, and microwave are included in the definition of medium. It should be understood, however, that computer-readable storage media and data storage media do not include connections, carrier waves, signals, or other transitory media, but are instead directed to non-transitory, tangible storage media. Disk and disc, as used herein, includes compact disc (CD), laser disc, optical disc, digital versatile disc (DVD), floppy disk and Blu-ray disc where disks usually reproduce data magnetically, while discs reproduce data optically with lasers. Combinations of the above should also be included within the scope of computer-readable media.

[0066] Instructions may be executed by one or more processors, such as one or more digital signal processors (DSPs), general purpose microprocessors, application specific integrated circuits (ASICs), field programmable logic arrays (FPGAs), or other equivalent integrated or discrete logic circuitry. Accordingly, the term “processor,” as used herein may refer to any of the foregoing structure or any other structure suitable for implementation of the techniques described herein. In addition, in some aspects, the functionality described herein may be provided within dedicated hardware and/or software modules configured for encoding and decoding, or incorporated in a combined codec. Also, the techniques could be fully implemented in one or more circuits or logic elements.

[0067] The techniques of this disclosure may be implemented in a wide variety of devices or apparatuses, including a wireless handset, an integrated circuit (IC) or a set of ICs (e.g., a chip set). Various components, modules, or units are described in this disclosure to emphasize functional aspects of devices configured to perform the disclosed techniques, but do not necessarily require realization by different hardware units. Rather, as described above, various units may be combined in a codec hardware unit or provided by a collection of interoperative hardware units, including one or more processors as described above, in conjunction with suitable software and/or firmware.

[0068] Moreover, each functional block or various features of the base station device and the terminal device used in each of the aforementioned embodiments may be implemented or executed by a circuitry, which is typically an integrated circuit or a plurality of integrated circuits. The circuitry designed to execute the functions described in the present specification may comprise a general-purpose processor, a digital signal processor (DSP), an application specific or general application integrated circuit (ASIC), a field programmable gate array (FPGA), or other programmable logic devices, discrete gates or transistor logic, or a discrete hardware component, or a combination thereof. The general-purpose processor may be a microprocessor, or alternatively, the processor may be a conventional processor, a controller, a microcontroller or a state machine. The general-purpose processor or each circuit described above may be configured by a digital circuit or may be configured by an analogue circuit. Further, when a technology of making into an integrated circuit superseding integrated circuits at the present time appears due to advancement of a semiconductor technology, the integrated circuit by this technology is also able to be used.

[0069] Various examples have been described. These and other examples are within the scope of the following claims.

[0070] <Summary>

In one example, a method of signaling deblocking parameter information, the method comprising: signaling a syntax element in a picture parameter set indicating whether

chroma deblocking parameter syntax elements are included in the picture parameter set; and conditionally signaling the chroma deblocking parameter syntax elements in the picture parameter set based on the value of the syntax element indicating whether chroma deblocking parameter syntax elements are included in the picture parameter set.

- [0071] In one example, a method of decoding deblocking parameter information for decoding video data, the method comprising: parsing a syntax element in a picture parameter set indicating whether chroma deblocking parameter syntax elements are included in the picture parameter set; and conditionally parsing the chroma deblocking parameter syntax elements in the picture parameter set based on the value of the syntax element indicating whether chroma deblocking parameter syntax elements are included in the picture parameter set.
- [0072] In one example, a device comprising one or more processors configured to perform any and all combinations of the steps.
- [0073] In one example, the device, wherein the device includes a video encoder.
- [0074] In one example, the device, wherein the device includes a video decoder.
- [0075] In one example, a system comprising: the device includes a video encoder; and the device includes a video decoder.
- [0076] In one example, an apparatus comprising means for performing any and all combinations of the steps.
- [0077] In one example, a non-transitory computer-readable storage medium comprising instructions stored thereon that, when executed, cause one or more processors of a device to perform any and all combinations of the steps.
- [0078] In one example, a method of decoding chroma component parameter information for decoding video data, the method comprising: receiving a picture parameter set including syntax elements corresponding to a picture; parsing a flag in the picture parameter set indicating whether chroma quantization parameter syntax elements are included in the picture parameter set and whether chroma deblocking parameter syntax elements are included in the picture parameter set; conditionally parsing the chroma quantization parameter syntax elements in the picture parameter set based on the value of the flag; conditionally parsing the chroma deblocking parameter syntax elements in the picture parameter set based on the value of the flag; deriving a quantization parameter based on the chroma quantization parameter syntax elements; and performing a deblocking filter based on the chroma deblocking parameter syntax elements.
- [0079] In one example, a device comprising one or more processors configured to: receive a picture parameter set including syntax elements corresponding to a picture; parse a flag in the picture parameter set indicating whether chroma quantization parameter syntax

elements are included in the picture parameter set and whether chroma deblocking parameter syntax elements are included in the picture parameter set; conditionally parse the chroma quantization parameter syntax elements in the picture parameter set based on the value of the flag; conditionally parse the chroma deblocking parameter syntax elements in the picture parameter set based on the value of the flag; derive a quantization parameter based on the chroma quantization parameter syntax elements; and perform a deblocking filter based on the chroma deblocking parameter syntax elements.

[0080] In one example, the device, wherein the device includes a video decoder.

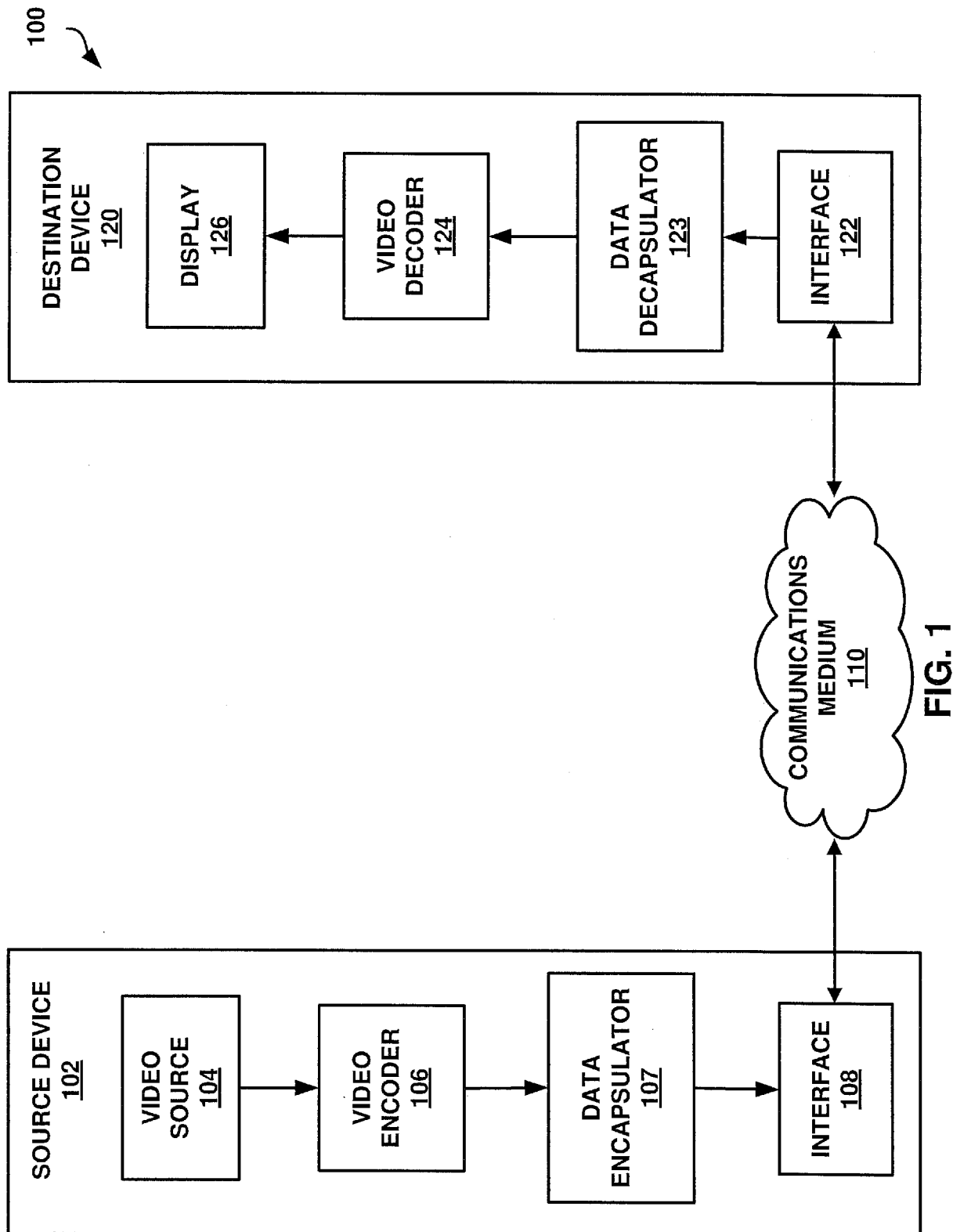
[0081] <Cross Reference>

This Nonprovisional application claims priority under 35 U.S.C. § 119 on provisional Application No. 62/991,519 on March 18, 2020, , the entire contents of which are hereby incorporated by reference.

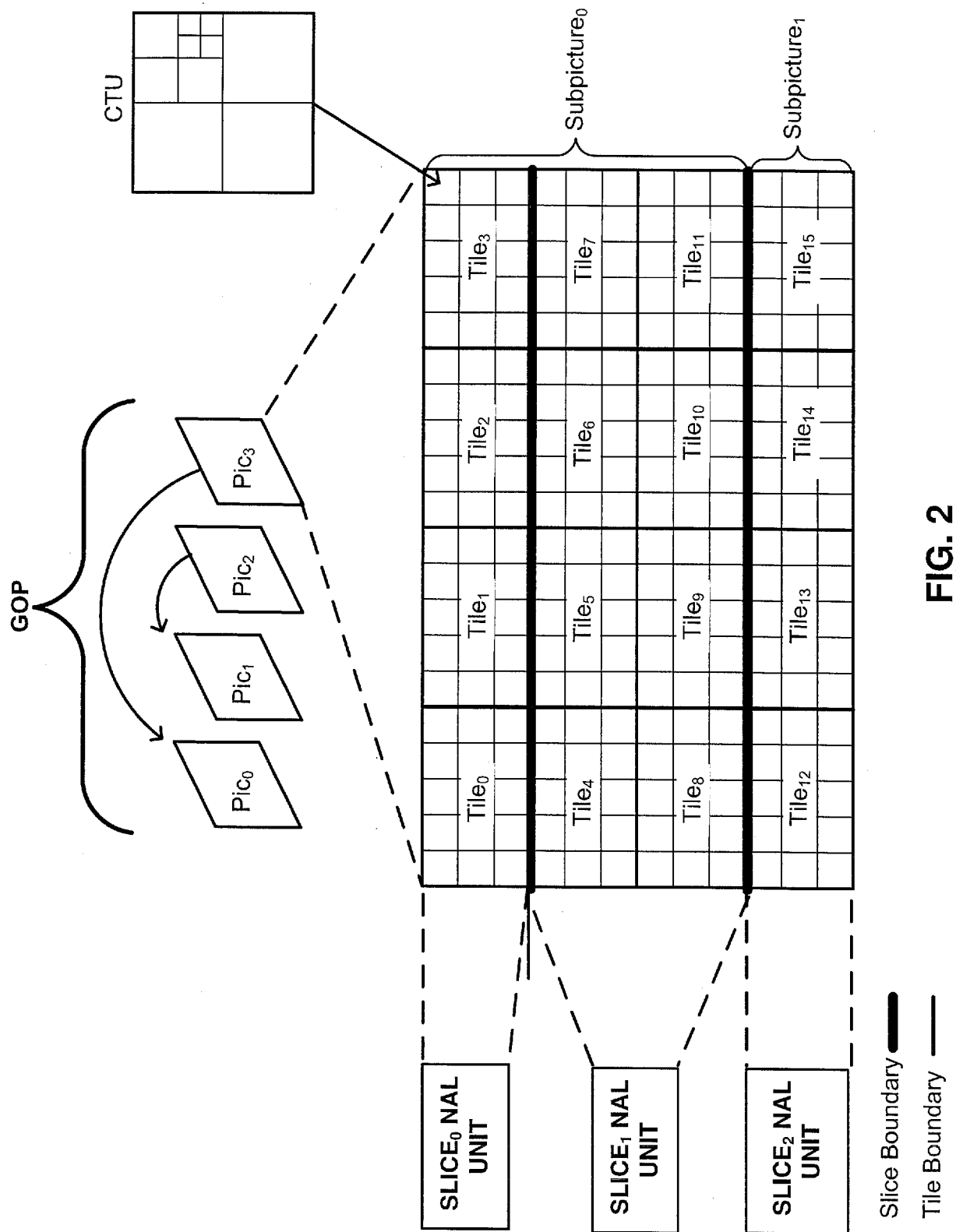
Claims

- [Claim 1] A method of decoding chroma component parameter information for decoding video data, the method comprising:
receiving a picture parameter set including syntax elements corresponding to a picture;
parsing a flag in the picture parameter set indicating whether chroma quantization parameter syntax elements are included in the picture parameter set and whether chroma deblocking parameter syntax elements are included in the picture parameter set;
conditionally parsing the chroma quantization parameter syntax elements in the picture parameter set based on the value of the flag;
conditionally parsing the chroma deblocking parameter syntax elements in the picture parameter set based on the value of the flag;
deriving a quantization parameter based on the chroma quantization parameter syntax elements; and
performing a deblocking filter based on the chroma deblocking parameter syntax elements.
- [Claim 2] A device comprising one or more processors configured to:
receive a picture parameter set including syntax elements corresponding to a picture;
parse a flag in the picture parameter set indicating whether chroma quantization parameter syntax elements are included in the picture parameter set and whether chroma deblocking parameter syntax elements are included in the picture parameter set;
conditionally parse the chroma quantization parameter syntax elements in the picture parameter set based on the value of the flag;
conditionally parse the chroma deblocking parameter syntax elements in the picture parameter set based on the value of the flag;
derive a quantization parameter based on the chroma quantization parameter syntax elements; and
perform a deblocking filter based on the chroma deblocking parameter syntax elements.
- [Claim 3] The device of claim 2, wherein the device includes a video decoder.

[Fig. 1]



[Fig. 2]

**FIG. 2**

[Fig. 3]

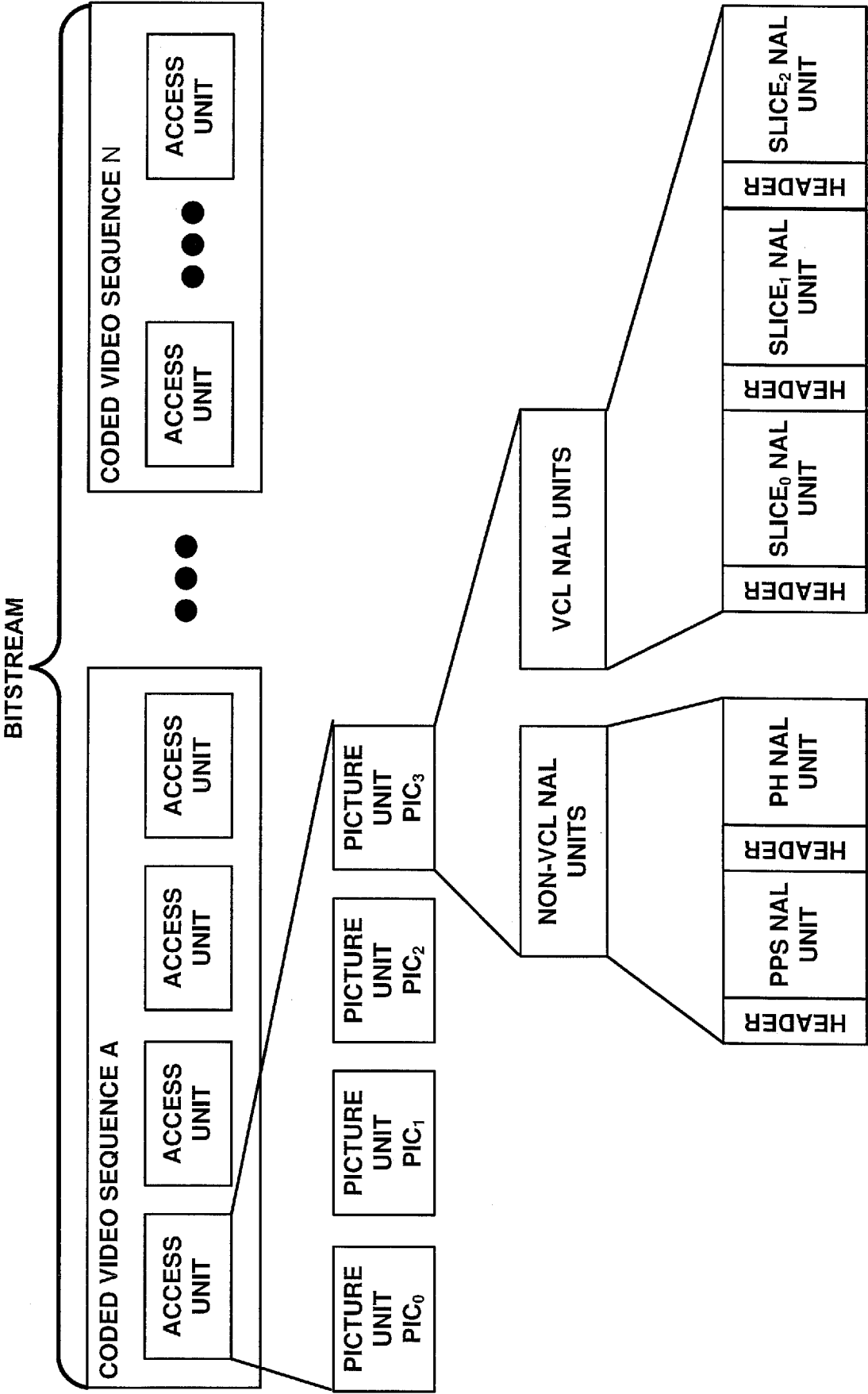


FIG. 3

[Fig. 4]

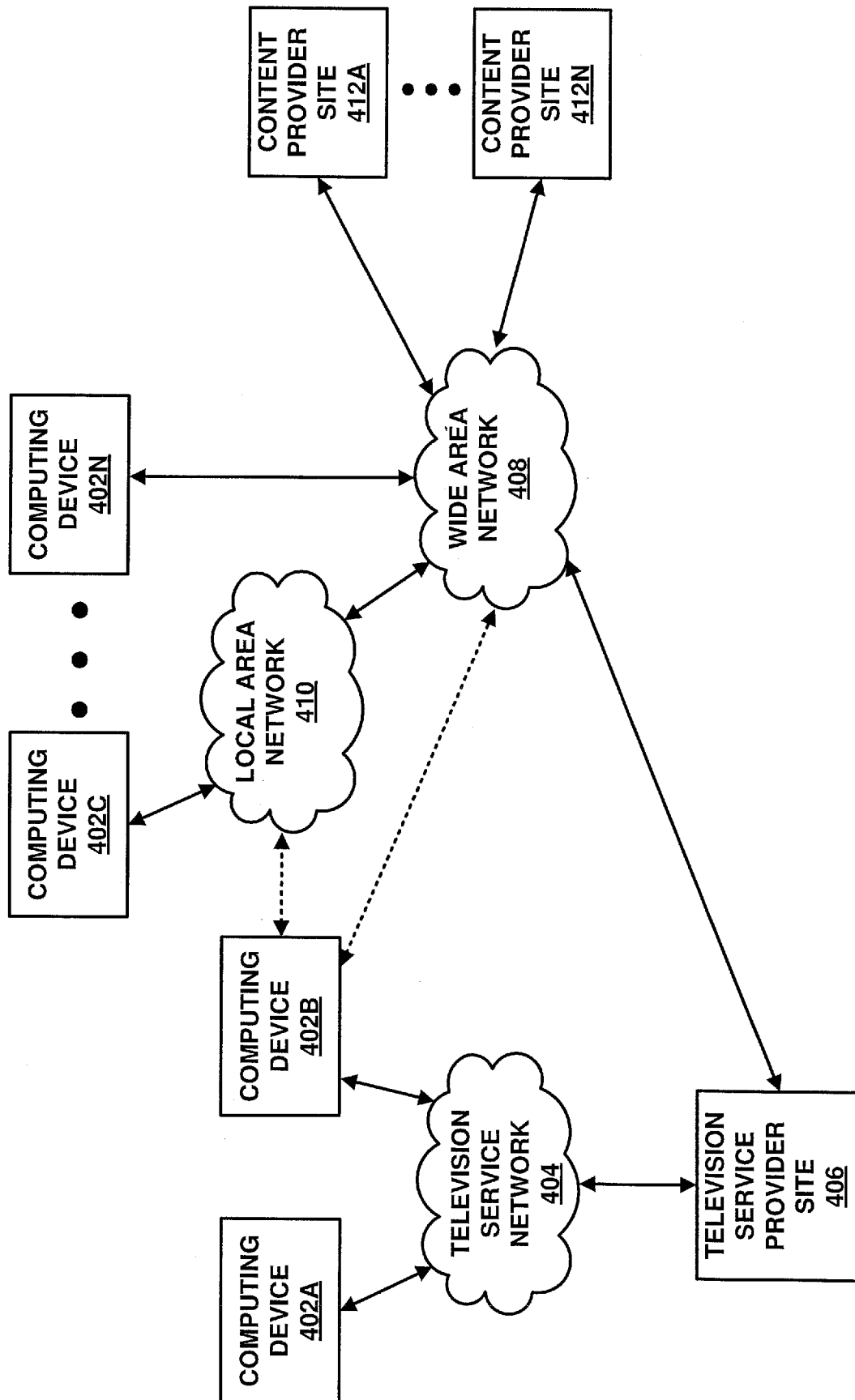
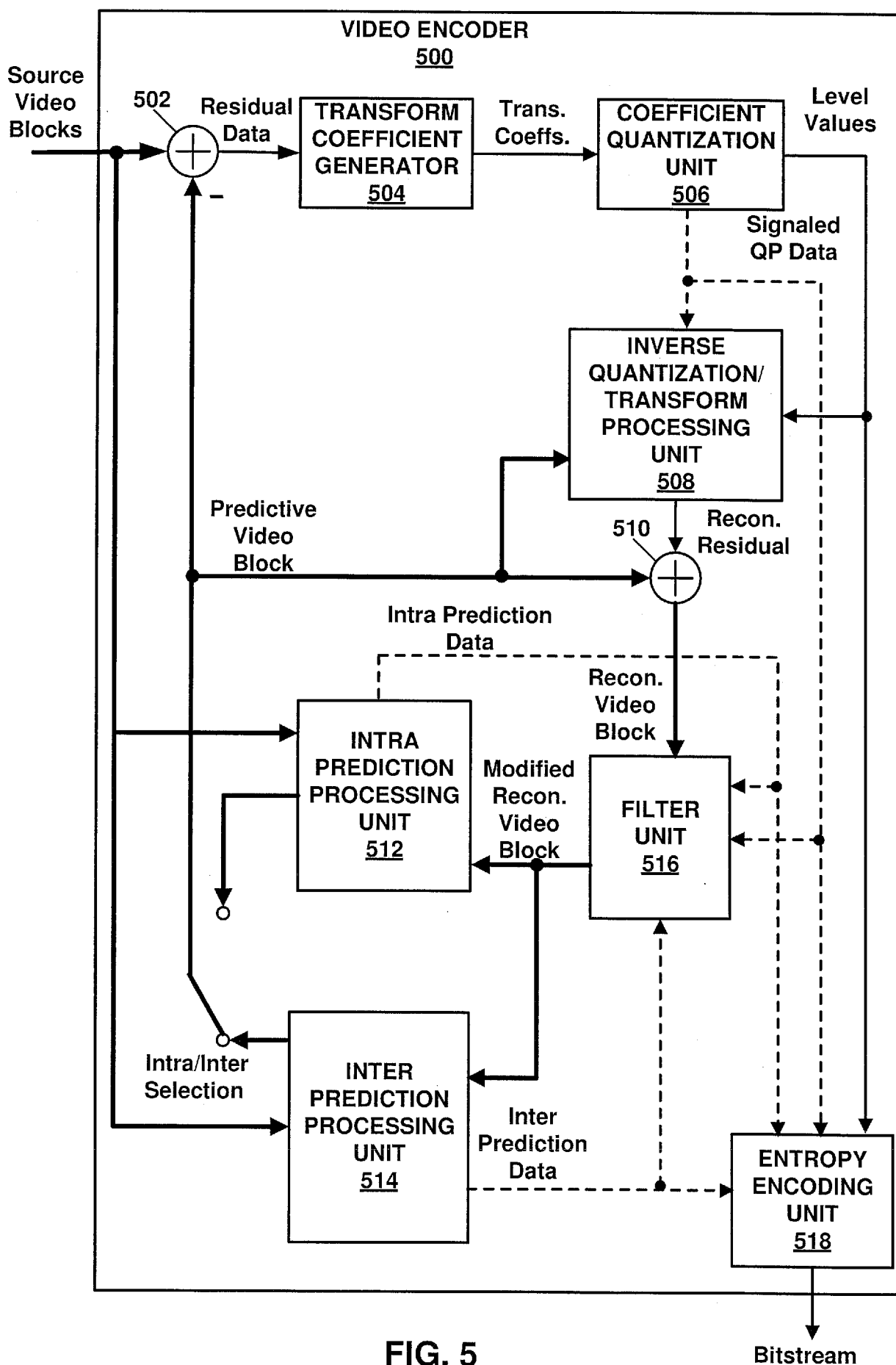


FIG. 4

FIG. 5



[Fig. 6]

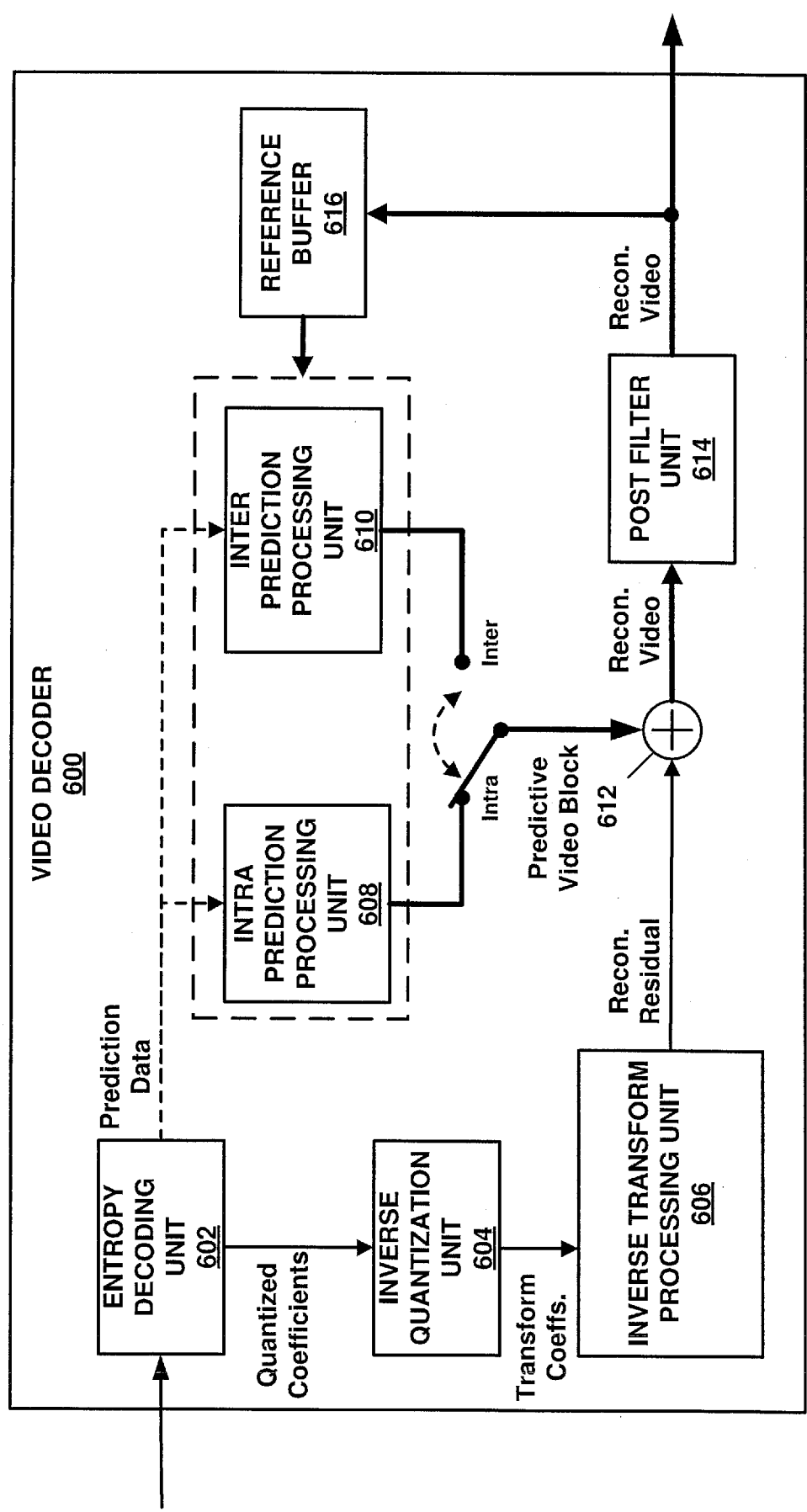


FIG. 6

INTERNATIONAL SEARCH REPORT

International application No.

PCT/JP2021/010119

A. CLASSIFICATION OF SUBJECT MATTER**H04N 19/70**(2014.01)i

FI: H04N19/70

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

H04N19/00-19/98

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Published examined utility model applications of Japan 1922-1996
 Published unexamined utility model applications of Japan 1971-2021
 Registered utility model specifications of Japan 1996-2021
 Published registered utility model applications of Japan 1994-2021

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	Jizheng Xu et al., AHG9: On deblocking control parameters, Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, JVET-Q0121-v1, 17th Meeting: Brussels, BE, 2020.01, pp.1-7 2 Proposed methods	1-3
A	Benjamin Bross et al., Versatile Video Coding (Draft 8), Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, JVET-Q2001 (version 15), 17th Meeting: Brussels, BE, 2010.03.12, pp.44-50,111-118 7.3.2.4, 7.3.2.7, 7.4.3.4	1-3
P, X	Sachin Deshpande, et al., On Deblocking Control, Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, JVET-R0106-v1, 18th Meeting: by teleconference, 2020.04, pp.1-7 2.2 Proposal 2	1-3
P, X	Anand Meher Kotra at al., AHG9: Modified signalling of Chroma deblocking control parameters, Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, JVET-R0206-v1, 18th Meeting: by teleconference, 2020.04, pp.1-6 2 Proposed method	1-3



Further documents are listed in the continuation of Box C.



See patent family annex.

* Special categories of cited documents:

“A” document defining the general state of the art which is not considered to be of particular relevance
 “E” earlier application or patent but published on or after the international filing date
 “L” document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
 “O” document referring to an oral disclosure, use, exhibition or other means
 “P” document published prior to the international filing date but later than the priority date claimed

“T” later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
 “X” document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
 “Y” document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
 “&” document member of the same patent family

Date of the actual completion of the international search

28 April 2021

Date of mailing of the international search report

18 May 2021

Name and mailing address of the ISA/JP

Japan Patent Office
 3-4-3, Kasumigaseki, Chiyoda-ku, Tokyo
 100-8915, Japan

Authorized officer

IWAI Kenji 5C 9465

Telephone No. +81-3-3581-1101 Ext. 3541

INTERNATIONAL SEARCH REPORT

International application No.

PCT/JP2021/010119

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
P, A	Hai Wei Sun et al., AHG9: On signaling of deblocking parameters for coding monochrome pictures, Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, JVET-R0078, 18th Meeting: by teleconference, 2020.04, pp.1-6 2 Proposed changes	1-3