SMPTE STANDARD

VC-4 Layered Video Extension Bitstream Format and Decoding Process

Table of Contents

Approved
March 7, 2011

Table of Figures

Table of Tables

Foreword

SMPTE (the Society of Motion Picture and Television Engineers) is an internationally-recognized standards developing organization. Headquartered and incorporated in the United States of America, SMPTE has members in over 80 countries on six continents. SMPTE's Engineering Documents, including Standards, Recommended Practices, and Engineering Guidelines, are prepared by SMPTE's Technology Committees. Participation in these Committees is open to all with a bona fide interest in their work. SMPTE cooperates closely with other standards-developing organizations, including ISO, IEC and ITU.

SMPTE Engineering Documents are drafted in accordance with the rules given in Part XIII of its Administrative Practices.

SMPTE ST 2058-1 was prepared by Technology Committee on Essence, 10E.

Introduction

This section is entirely informative and does not form an integral part of this Engineering Document.

The purpose of this document is to enable layered video applications for the video formats in the existing SMPTE specifications through independently decodable video layers carrying residual information. The layered video extension framework primarily targets applications in the high video quality range, including the fidelity ranges of up to 16-bit per sample as well as 4:2:2 or 4:4:4 color difference fidelity. The residual layer syntax and decoding process as well as the process of reconstructing the high quality and high fidelity video pictures are included.

The elementary stream definition is not dependent on any transport mapping, and is independent of the base layer coding technology.

Patent Notice

SMPTE draws attention to the fact that it is claimed that compliance with this Standard may involve the use of one or more patents or other intellectual property rights (collectively, "IPR"). The Society takes no position concerning the evidence, validity, or scope of this IPR.

Each holder of claimed IPR has assured the Society that it is willing to License all IPR it owns, and any third party IPR it has the right to sublicense, that is essential to the implementation of this Standard to those (Members and non-Members alike) desiring to implement this Standard under reasonable terms and conditions, demonstrably free of discrimination. Each holder of claimed IPR has filed a statement to such effect with SMPTE. Information may be obtained from the Director, Standards & Engineering at SMPTE Headquarters.

Attention is also drawn to the possibility that elements of this Standard may be subject to IPR other than those identified above. The Society shall not be responsible for identifying any or all such IPR.

1 Scope

This document describes the bitstream syntax and decoding process for the SMPTE Layered Video Extension (LVE). Bit-streams and decoders conforming to one or more of the profiles specified in this specification are completely defined in this specification. The picture decoding and reconstruction process outputs 8 to 16-bit value per video component sample in 4:2:0, 4:2:2 or 4:4:4 sampling grid.

2 Conformance Notation

Normative text is text that describes elements of the design that are indispensable or contains the conformance language keywords: "shall", "should", or "may". Informative text is text that is potentially helpful to the user, but not indispensable, and can be removed, changed, or added editorially without affecting interoperability. Informative text does not contain any conformance keywords.

All text in this document is, by default, normative, except: the Introduction, any section explicitly labeled as "Informative" or individual paragraphs that start with "Note:"

The keywords "shall" and "shall not" indicate requirements strictly to be followed in order to conform to the document and from which no deviation is permitted.

The keywords, "should" and "should not" indicate that, among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited.

The keywords "may" and "need not" indicate courses of action permissible within the limits of the document.

The keyword “reserved” indicates a provision that is not defined at this time, shall not be used, and may be defined in the future. The keyword “forbidden” indicates “reserved” and in addition indicates that the provision will never be defined in the future.

Unless otherwise specified, the order of precedence of the types of normative information in this document shall be as follows: Normative prose shall be the authoritative definition; Tables shall be next; followed by formal languages; then figures; and then any other language forms.

3 Normative References

Note: All references in this document to other SMPTE documents use the current numbering style (e.g. SMPTE ST 421:2006) although, during a transitional phase, the document as published (printed or PDF) may bear an older designation (such as SMPTE 421M-2006). Documents with the same root number (e.g. 421) and publication year (e.g. 2006) are functionally identical.

The following standards contain provisions which, through reference in this text, constitute provisions of this recommended practice. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this recommended practice are encouraged to investigate the possibility of applying the most recent edition of the standards indicated below.

SMPTE ST 421:2006, VC-1 Compressed Video Bitstream Format and Decoding Process

Amendment 1-2007 to SMPTE ST 421:2006

ISO/IEC 14496-10|ITU-T H.264 (03/10), Advanced Video Coding for Generic Audiovisual Services

ISO/IEC 14882:2003, “Programming Languages — C++

4 Overview (Informative)

The bitstream specified by the Layered Video Extension carries residual signals that when integrated with the video content from the base layer picture can provide enhanced quality as well as enhanced fidelity in bit depth, resolution and color difference. Syntax elements specifying the control parameters for the picture reconstruction or integration process are also carried in the same bitstreams. The 8-bit 4:2:0 pictures can be decoded, for example, from SMPTE ST 421 compliant bitstreams as well as content decoded from bitstreams specified by other video coding standards, such as ITU-T H.262 (MPEG-2) or ITU-T H.264 (MPEG-4 AVC), etc. The 8-bit 4:2:0 pictures are referred to as base layer pictures in this document. The residual bitstreams can be decoded independently from the base layer picture decoding process. The reconstructed output pictures can be 4:2:0, 4:2:2, or 4:4:4 color format. If 4:4:4 pictures are referred to as base layer the residual bitstreams are contrained to have 4:4:4 color format and the reconstructed output picture is therefore 4:4:4 color format.

Compared to the decoding process defined in many existing video coding standards, the complexity of the residual bitstream decoding is significantly reduced. In the residual layer bitstream, each frame is either coded in a progressive mode or in an interlaced mode. Each picture can be coded as I picture or P picture. There is no in-loop deblocking process or lapped transform required in the residual decoding process.

A set of new coding tools is introduced in order to achieve high coding efficiency for residual signals. That includes:

- an adaptive VLC coding process (see Section 9.3)
- multiple reference pictures at the picture level
- residual intra prediction in the macroblock level, and
- redefined skip modes in the macroblock level as well as in the picture level.

In addition, a set of picture format conversion operations is also specified as part of the high-quality or high-fidelity picture reconstruction process. Further details of the bitstream syntax and decoding process are specified in the normative provisions of this document.

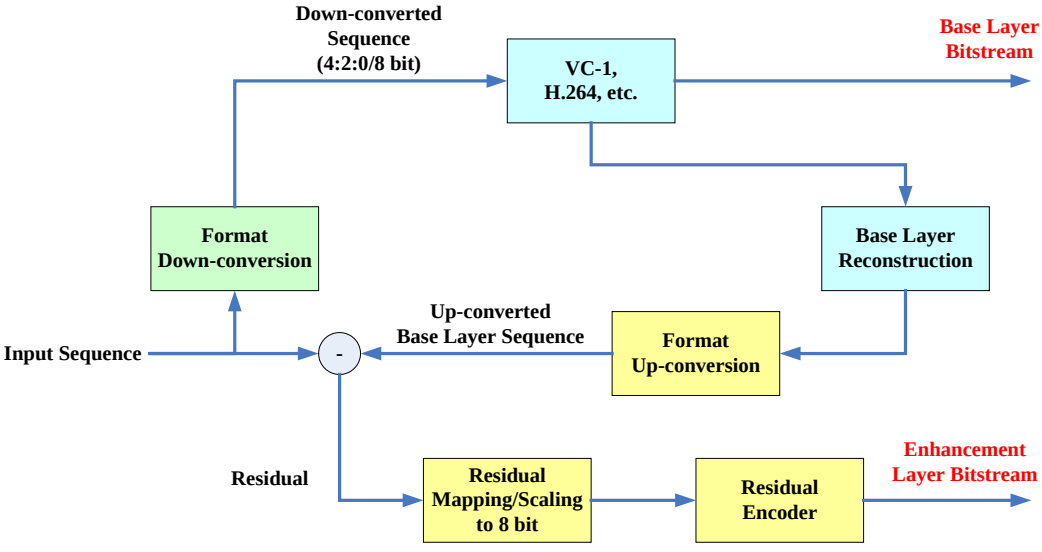

Figure 1 – Encoding process block diagram

An overview of the encoding process is shown in Figure 1 which depicts the major steps in the overall encoding process.

When supported in the system layer, multiple residual streams can be applied to enable more scalable video layers.

When supported in the system layer, the residual stream can also be applied to carry the other view information such as the case of a stereoscopic video sequence.

5 Notation

The pseudo-code used in this document conforms to ISO/IEC 14882:2003 (C++).

5.1 Arithmetic Operators

+	Addition.
–	Subtraction (as a binary operator) or negation (as a unary operator).
++	Increment.
--	Decrement.
+=	a += b is defined as a = a + b
–=	a –= b is defined as a = a – b
*	Multiplication.
/	Integer division with truncation towards zero.
÷	Real division
//	Rest of the line is a comment.
| |	Absolute value.
abs()	Absolute value. | x | = x , when x > 0 | x | = 0, when x == 0 | x | = –x, when x < 0
%	x%a is defined as the modulus operator for x>= 0, a>0. x%a = -((-x%a)) for x< 0, a>0. x%a is defined only for positive values of a.
Sign( )	Sign. Sign(x) = 1, when x >= 0 Sign(x) = –1, when x < 0
CLIP ( )	CLIP(n) = 255 if n > 255, CLIP(n) = 0 if n < 0, CLIP(n) = n otherwise
max	Maximum of the arguments.
min	Minimum of the arguments.
√	Square root.
log2	Logarithm to base-2.

5.2 Logical Operators

||	Logical OR.
&&	Logical AND.
!	Logical NOT
x ? y : z	If x is TRUE or not equal to 0, evaluates to the value of y; otherwise, evaluates to the values of z

5.3 Relational Operators

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

5.4 Bitwise Operators

The following operators are used in the 2's complement number system.

&	AND.
|	OR.
^	XOR.
>>	Shift right with sign extension.
<<	Shift left with zero fill.

5.5 Assignment

=	Assignment operator.

5.6 Mnemonics

The following mnemonics are defined to describe the different data types used in the coded bitstream.

uimsbf	Unsigned integer, most-significant bit first.
vlclbf	Variable length code, left bit first, where "left" refers to the order in which the VLC codes are written. All syntaxes using vlclbf codes define their own VLC tables.
avlclbf	Adaptive variable-length code, left bit first, shall be as defined in section 9.3
mevlclbf	Modified Exponential Golomb code, left bit first, shall be as defined in section 9.1.4.2 (See Figure 12)
evlclbf	Exponential Golomb code, left bit first, shall be as defined in subclause 9.1.1 of Recommendation ITU-T H.264 (See Figure 6, Figure 24, and Figure 55)

5.7 Bitstream Parsing Operation

get_bits(n)	Reads n bits from the bitstream and returns the value. When n is zero, the return value is zero.
ALIGNtoBYTE()	Reads the bitstream until the current position in the bitstream is on a byte boundary.
TerminateBit()	Reads the terminating bit which is one bit '1'. It indicates the end of slice bitstream.

5.8 Function Definitions for Conditional Branches in Bitstream Tables

This section defines the functions for conditional branches in the bitstream tables: Table 11, Table 37, Table 38, Table 64 and Table 67.

is_adaptive_color_difference_upconversion()	Returns TRUE if COLOR_DIFFERENCE_UPCONVERSION == 10b. Otherwise, it returns FALSE. (See Section 7.1.26, Section 8.1.27, Section 8.1.28, Section 10.1.30 and Section 10.1.31)
is_beginning_of_slice()	Returns TRUE if the current macroblock is the start macroblock in the current slice. Otherwise, it returns FALSE. (See Section 8.3.1)
is_previousMB_NotSkip()	Returns TRUE if the previous macroblock was not skipped in the current slice. Otherwise, it returns FALSE. (See Section 8.3.1)

is_not_skipMB()	Returns TRUE if the current macroblock is not skipped in the current slice. Otherwise, it returns FALSE. (See Section 8.3.1)
is_UpperMB_Inter()	Returns TRUE if the upper macroblock of the current macroblock is coded in inter mode. Otherwise, it returns FALSE. (See Table 16 and Table 67)
is_LeftMB_Inter()	Returns TRUE if the left macroblock of the current macroblock is coded in inter mode. Otherwise, it returns FALSE. (See Table 16 and Table 67)
ExistMV()	Returns TRUE if the current macroblock is coded with one or more motion vectors. Otherwise, it returns FALSE. (See Table 16 and Table 67)
is_UpperMB_Skip()	Returns TRUE if the upper macroblock of the current macroblock is coded as skip mode. Otherwise, it returns FALSE. (See Table 16 and Table 67)
is_LeftMB_Skip()	Returns TRUE if the left macroblock of the current macroblock is coded as skip mode. Otherwise, it returns FALSE. (See Table 16 and Table 67)
ExistCodedCoeff()	Returns TRUE if the current macroblock has at least one coded coefficient. Otherwise, it returns FALSE.
Exist_Y_Channel()	Returns TRUE if the Y channel is not skipped by CHANNEL_SKIP_MODE. Otherwise, it returns FALSE. (See Section 8.1.26 and Section 10.1.29)
Exist_UV_Channel()	Returns TRUE if all color difference channels (Cb and Cr) are not skipped by CHANNEL_SKIP_MODE. Otherwise, it returns FALSE. (See Section 8.1.26 and Section 10.1.29)
Exist_Y_Coeff()	Returns TRUE if the current four luma blocks have at least one coded coefficient. Otherwise, it returns FALSE.
Exist_Y0Y1_Coeff()	Returns TRUE if at least one is significant among coefficients of the upper luma blocks in the current macroblock. Otherwise, it returns FALSE. (See Section 8.3.13 and Section 10.3.14)
Exist_Y2Y3_Coeff()	Returns TRUE if at least one is significant among coefficients of the lower luma blocks in the current macroblock. Otherwise, it returns FALSE. (See Section 8.3.13 and Section 10.3.14)
Exist_UV_Coeff()	Returns TRUE if the current color difference blocks have at least one coded coefficient. Otherwise, it returns FALSE.
Exist_U_Channel()	Returns TRUE if the U color difference channel is not skipped by CHANNEL_SKIP_MODE. Otherwise, it returns FALSE. (See Section 8.1.26 and Section 10.1.29)
Exist_V_Channel()	Returns TRUE if the V color difference channel is not skipped by CHANNEL_SKIP_MODE. Otherwise, it returns FALSE. (See Section 8.1.26 and Section 10.1.29)
is422()	Returns TRUE if the color difference format is 4:2:2. Otherwise, it returns FALSE.
Exist_U_Coeff()	Returns TRUE if at least one is significant among coefficients of the Cb color difference blocks in the current macroblock. Otherwise, it returns FALSE. (See Section 8.3.16 and Section 10.3.17)

Exist_V_Coeff()	Returns TRUE if at least one is significant among coefficients of the Cr color difference blocks in the current macroblock. Otherwise, it returns FALSE. (See Section 8.3.16 and Section 10.3.17).
is444()	Returns TRUE if the color difference format is 4:4:4. Otherwise, it returns FALSE.
Exist_U0U1_Coeff()	Returns TRUE if at least one is significant among coefficients of the upper Cb color difference blocks in the current macroblock. Otherwise, it returns FALSE. (See Section 8.3.19 and Section 10.3.20)
Exist_U2U3_Coeff()	Returns TRUE if at least one is significant among coefficients of the lower Cb color difference blocks in the current macroblock. Otherwise, it returns FALSE. (See Section 8.3.19 and Section 10.3.20)
Exist_V0V1_Coeff()	Returns TRUE if at least one is significant among coefficients of the upper Cr color difference blocks in the current macroblock. Otherwise, it returns FALSE. (See Section 8.3.21 and Section 10.3.22)
Exist_V2V3_Coeff()	Returns TRUE if at least one is significant among coefficients of the lower Cr color difference blocks in the current macroblock. Otherwise, it returns FALSE. (See Section 8.3.21 and Section 10.3.22)
i_th_component_of_Layer2_Significant()	Returns TRUE if at least one is significant among 16 AC coefficients corresponding to L2[i] in the current block. Otherwise, it returns FALSE. (See Section 8.4.5 and Section 9.1.4.1).
Last_12_Coeffs_of_L2AC16_CST[i]_Significant()	Returns TRUE if at least one is significant among last 12 AC coefficients corresponding to L2[i] in the current block. Otherwise, it returns FALSE. (See Section 8.4.10 and Section 9.1.4.1).
Last_8_Coeffs_of_L2AC12_CST[i]_Significant()	Returns TRUE if at least one is significant among last eight AC coefficients corresponding to L2[i] in the current block. Otherwise, it returns FALSE. (See Section 8.4.11 and Section 9.1.4.1).
i_th_component_of_Layer1_Significant()	Returns TRUE if at least one is significant among four AC coefficients corresponding to L1[i] in the current block. Otherwise, it returns FALSE. (See Section 8.4.4, Section 8.4.7, Section 8.4.8, Section 8.4.9, Section 8.4.10, Section 8.4.11, Section 8.4.12 and Section 9.1.4.1).
Last_3_Coeffs_of_L1AC4_CST[i]_Significant()	Returns TRUE if at least one is significant among last three AC coefficients corresponding to L1[i] in the current block. Otherwise, it returns FALSE. (See Section 8.4.13 and Section 9.1.4.1).
Last_2_Coeffs_of_L1AC4_CST[i]_Significant()	Returns TRUE if at least one is significant among last two AC coefficients corresponding to L1[i] in the current block. Otherwise, it returns FALSE. (See Section 8.4.14 and Section 9.1.4.1).
isLeftNeighbor( )	Returns TRUE if the left macroblock is available. Otherwise, it returns FALSE. (See Section 9.1.3.3)
isTopNeighbor( )	Returns TRUE if the upper macroblock is available. Otherwise, it returns FALSE. (See Section 9.1.3.3)
isAvailableMVSCALE()	Returns TRUE if the MV scaling operation is possible. Otherwise, it returns FALSE. (See Section 10.3.12 and Section 11.2.3.1).

5.9 Definition of Terminology

1-D: an abbreviation of one-dimensional.

2-D: an abbreviation of two-dimensional.

4-tap bi-cubic filter: A filter whose length is 4 and used to interpolate data points with an extension of the cubic interpolation.

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

adaptive Huffman coding: a variable length coding method which changes a Huffman coding table based on the probability. A non-adaptive Huffman coding uses a fixed Huffman coding table which is designed by the pre-determined probabilities of all symbols. However, an adaptive Huffman coding uses a flexible Huffman coding table which is updated by real occurrences of all symbols. This coding method requires no initial knowledge of symbol probability.

auxiliary reference fields: a field picture which is the opposite field of the main reference field indicated by the header of the current field picture. The main reference field and the auxiliary reference field are a top and bottom pair within a single frame.

base layer picture: A picture that is decoded from the base layer stream, such as SMPTE ST 421 (VC-1), and is used in reconstructing a new picture when integrating with the residual picture decoded from the residual layer stream. The synchronization of the base layer picture and other residual layer streams is expected to be signaled in the system layer.

B field picture: A B picture coded with interlaced field coding mode

B frame picture: A B picture coded with progressive or interlaced frame coding mode.

B picture; bidirectionally predictive-coded picture: A picture coded using motion compensated prediction from past and/or future reference progressive/interlace frames. A B picture can contain macroblocks that are inter-coded and macroblocks that are intra-coded

bit depth up-conversion: The process which converts the bit depth of samples of the lower layer picture to the bit depth of samples of the upper layer picture.

bit-shifting: The process which is bitwise operation and shifts all of bits expressing a certain value. This is one method among the bit-depth up-conversion defined in this document. The bit-shifting process in the format up-conversion shift all bits to the right and the vacant bit positions are filled with zeros.

backward-P picture: A picture coded using motion compensated prediction from future reference progressive/interlaced frame. This picture is a P picture with a future reference picture. A P picture can contain macroblocks that are inter-coded and macroblocks that are intra-coded.

bit depth of residual: The bit depth of residuals which are differences between the current layer picture and the up-converted lower layer picture and are treated as output of the residual decoder. The depth of residuals is eight bits. Therefore, the residuals are mapped to the range from -128 to 127 and shifted to the range from 0 to 255 before the residual encoding. In an encoder, these residuals are dealt with in after these residuals are converted to the range from 0 to 255. In a decoder, the reconstructed residual from 0 to 255 are converted to the original range.

bitstream: An ordered series of bits that forms the coded representation of the data.

bitrate: The rate at which the coded bitstream is delivered to the input of a decoder.

block: An 8-row by 8-column matrix of samples, or 64 transform coefficients.

bottom field: One of two fields that comprise a frame. Each line of a bottom field is spatially located immediately below the corresponding line of the top field.

boundary pixel: A pixel on the picture boundary.

coded order: The order in which the pictures are transmitted and decoded.

color difference: The Cb, Cr signals resulting from the matrix equations defined in the image source documents.

color difference up-conversion: The process which converts the color difference format of the lower layer picture to the color difference format of the upper layer picture..

compressed residual picture: A picture coded with differences and residuals between the picture of up-converted lower layer and the picture of the upper layer. This compressed residual picture is carried via an enhancement layer or upper layer.

current layer: The layer which is decoded or represented currently.

current field: The field picture which is decoded or represented currently

current macroblock: The macroblock which is decoded or represented currently.

current picture: The picture which is decoded or represented currently.

current reference field: The field picture which is a reference field picture for decoding the current field picture and this reference field picture is indicated by the header of the current field picture.

DC coefficient: The transform coefficient for which the frequency is zero in both dimensions.

dequantization: The process of rescaling the quantized transform coefficients after their representation in the bitstream has been decoded and before they are presented to the inverse transform.

display order: The order in which the decoded pictures are displayed. Normally this is the same order in which they were presented at the input of the encoder.

enhancement layer: A residual layer that has at least one lower layer. The lowest layer is a base layer. The number of enhancement layer should be greater than or equal to one and each enhancement layer is a residual layer.

enhancement layer picture: A picture that carries a compressed residual picture. This layer picture is used in reconstructing a new picture when integrating with other lower residual layer pictures and a base layer, such as SMPTE ST 421 (VC-1). The synchronization of the base layer picture and other residual layer streams is expected to be signaled in the system layer.

encoder: An embodiment of an encoding process.

entry point picture: A picture at which the reference frame list or reference field list shalle be made empty. The subsequent coded pictures do not use the previously coded pictures as the reference picture before this entry point picture. It also means the current entry point picture does not use any reference pictures for motion compensation.

field: For an interlaced video signal, a "field" is the assembly of alternate lines of a frame. Therefore an interlaced frame is composed of two fields, a top field and a bottom field.

format up-conversion process: The process which converts the picture format of the lower layer to the picture format of the upper layer.

forward-P picture: A picture coded using motion compensated prediction from past reference progressive/interlaced frame. This picture is a P picture with a past reference picture. A P picture can contain macroblocks that are inter-coded and macroblocks that are intra-coded.

frame: A frame contains lines of spatial information of a video signal. For progressive video, these lines contain samples starting from one time instant and continuing through successive lines to the bottom of the frame. For interlaced video, a frame consists of two fields, a top field and a bottom field. One of these fields will commence one field period later than the other

header: A block of data in the coded bitstream containing the representation of a number of data elements pertaining to the coded data that follow the header in the bitstream.

high-fidelity picture reconstruction: A process of integrating the decoded residual signals and the up-converted version of a picture in 8-bit 4:2:0 format to reconstruct a new picture.

hypothetical reference decoder (HRD): Hypothetical reference decoder is an alternative term for video buffering verifier.

I field picture: An I picture coded with interlaced field coding mode.

I frame picture: An I picture coded with progressive or interlaced frame coding mode.

I picture; intra-coded picture: A picture coded using information only from itself.

inter-coded block; inter-block: A block that has been coded using information both from itself, and from blocks and pictures occurring at other times.

inter coding: Coding of a macroblock or picture that uses information both from itself and from macroblocks and pictures occurring at other times.

interlace: The property of frames where alternating lines of the frame represent different instances in time.

interlaced field coding: The coding mode used when the two fields of an interlaced frame are coded separately. The pictures coded using this mode are called interlaced field pictures, or field pictures.

interlaced frame: A frame is composed of two fields, a top field and a bottom field. Even lines of the interlaced frame is corresponding to the top field and odd lines is corresponding to bottom field.

interlaced frame coding: The coding mode used when the two fields of an interlaced frame are coded together. The pictures coded using this mode are called interlaced frame pictures, or frame pictures.

interpolation: The process used to generate subpixel values when the motion vectors are not integers.

intra prediction: A prediction derived from the decoded samples of the current frame (field).

layer picture: A collective term for a layer field or a layer frame.

lower layer: A layer that is referred to by the current layer.

luma (Y'): is the value resulting from a weighted sum of 3 nonlinear (gamma pre-corrected) R,G,B components..

macroblock (MB): The four 8 by 8 blocks of luma data and the two or four corresponding 8 by 8 blocks of color difference data coming from a 16 by 16 section of the luma component of the picture.

macroblock location: The two-dimensional coordinates of a macroblock in a picture denoted by (x, y). For the top left macroblock of the picture (x, y) is equal to (0, 0). x is incremented by 1 for each macroblock column from left to right. y is incremented by 1 for each macroblock row from top to bottom.

main reference field: A reference field picture which is indicated by the header of the current field picure.

motion compensation: The use of motion vectors to improve the efficiency of the prediction of sample values. The prediction uses motion vectors to provide offsets into the past reference frames or reference fields containing previously decoded sample values that are used to form the prediction error.

motion estimation: The process of estimating motion vectors during the encoding process.

motion vector (MV): A two-dimensional vector used for mostion compensation that provides an offset from the coordinate postion in the current frame picture or field picture to the coordindates in a reference frame or reference field.

non-skipped macroblock: A macroblock which is not skipped, and for which data is coded.

pel: an alternate term for pixel.

pel unit: a unit of distance or length between any two pixels. When distance bwteen two adjacent pixels is represented, the distance is 1 in the full-pel unit or the integer-pel unit, the distance is 2 in the half-pel unit, and the distance is 4 in the quarter-pel uint.

P field picture: A P picture coded with interlaced field coding mode.

P frame picture: A P picture coded with progressive or interlaced frame coding mode.

P picture; predictive-coded picture: A picture coded using motion compensated prediction from past or future reference progressive/interlaced frame. A P picture can contain macroblocks that are inter-coded and macroblocks that are intra-coded.

past reference frame (field): A past reference frame (field) is a reference frame (field) that occurs at an earlier time than the current picture in display order.

picture: Source, coded or reconstructed image data. A source or reconstructed picture consists of three rectangular matrices of 8-bit numbers representing the luma and two color difference signals. For progressive video, a picture is identical to a frame, while for interlaced video, a picture may refer to a frame, or the top field or the bottom field of the frame depending on the context.

picture boundary: The boundary of an area of the picture is a border line that seperates it from outside of the picture. This border line consists of the first row, the last row, the first column and the last column of the picture.

picture format up conversion: A process of converting picture from 8-bit 4:2:0 format to at least one of the following: higher spatial resolution, higher bit depth, and extended color difference format.

predicted block: A reference block which is used as the prediction of the currently-decoded block for motion compensation.

prediction error: The difference between the actual value of a sample or data element and its predictor.

progressive: The property of frames where all the samples of the frame represent the same instance in time.

quality base layer: A layer represents the lowest quality of reconstructed picture among quality layers within a picture. The slice data where QUALITY_LAYER_ID == 0 shall be corresponding to quality base layer.

quality enhancement layer: A layer represents the highest quality of reconstructed picture among quality layers within a picture. The slice data where QUALITY_LAYER_ID is greater than 0 shall be corresponding to quality enhancement layer.

quality layer: A layer shall contain one or more quality layers, when quality refinement process is used. The quality layers are determined by the syntax element QUALITY_LAYER_ID in slice.

quality refinement: A process to enhance quality of reconstructed sample by using refinement data.

quantize; quantization: A process in which the continuous range of values of an input signal is divided into non-overlapping (but not necessarily equal) subranges, and a discrete, unique value is assigned to each subrange. A unique index is generated to represent this value.

reconstructed picture: A reconstructed picture is obtained by decoding a coded picture. A reconstructed picture is either a reconstructed frame (when decoding a frame picture), or one field of a reconstructed frame (when decoding a field picture). If the coded picture is a field picture, then the reconstructed picture is the top field or the bottom field of the reconstructed frame.

reference field: a field picture which is used as a reference field for motion compensation and is indicated by the header of the current field picture.

reference frame: a frame picture which is used as a reference frame for motion compensation and is indicated by the header of the current frame picture.

reference frame list: a set or buffer which have the candidate frames to be used as reference frames.

reference picture index: An index into a set of reference pictures.

residual layer stream: A stream compliant to the specification that carries compressed residual signals.

residual mapping and scaling: A process of converting the decoded residual signals through a linear scaling function during the reconstruction process of high-fidelity pictures .

resolution up-conversion: The process which converts the resolution of the lower layer picture to the resolution of the upper layer picture.

rounding: The process of adjusting the bias before a division (or shift) operation.

run: The number of zero coefficients preceding a non-zero coefficient, in the scan order. The absolute value of the non-zero coefficient is called "level".

sample domain: The set of locations or the grid where the picture sample can be located in the given picture format. This domain includes the outside of the picture and the sample grid of the outside is the same as the sample grid. The sample domain of the luma samples in the 4:2:0 format is the same as that of the luma samples in the 4:2:2 format. However, the sample domain of the chroma samples in the 4:2:0 format is different from that of the chroma samples in the 4:2:2 format because those chroma samples have the is different phase location.

skipped macroblock: A macroblock for which no data are encoded.

slice: An integer number of macroblocks ordered consecutively in the raster scan within a particular slice. These macroblocks within this slice are also consecutive in the raster scan within the picture. One or more

slices are ordered in the increasing number of first macroblock locations of slices in the raster scan within the picture.

stereo conversion: A process to obtain the prediction frame (field) for current view (current layer) picture

temporal prediction: A prediction derived from the previously reconstructed frame (field) of the current view (current layer) during the stereo conversion.

tone-mapping: The process which maps one set of colors to another. In this document, the tone-mapping maps the set of colors of the lower layer picture to the set of colors of the upper layer picture.

top field: One of two fields that comprise a frame. Each line of a top field is spatially located immediately above the corresponding line of the bottom field.

up-sampling: The process which converts the smaller resolution picture to larger resolution picture by interpolation and is used in the resolution up-conversion process.

upper layer: A layer that uses the current layer as a reference layer.

video buffering verifier (VBV): A hypothetical decoder that is conceptually connected to the output of the encoder. Its purpose is to provide a constraint on the variability of the data rate that an encoder or editing process may produce.

view prediction A prediction derived from the frame (field) of the other-side view (base layer) during the stereo conversion.

5.10 Number System

The following positional number systems are used to represent a certain number in this specification.

XYZ A number in positional number system with a radix of 10 (decimal number system). *XYZ* is the same as $X*10^2 + Y*10 + Z$. *X*, *Y* and *Z* shall be one among 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.

0x*XYZ* A number in positional number system with a radix of 16 (Hexadecimal number system). 0x*XYZ* is the same as $X*16^2 + Y*16 + Z$ in the decimal number system. *X*, *Y* and *Z* shall be one among 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A (or a), B (or b), C (or c), D (or d), E (or e) and F (or f). “0x” is the prefix for representing the hexadecimal number.

*XYZ*b A number in positional number system with a radix of 2 (binary number system). *XYZ*b is the same as $X*2^2 + Y*2 + Z$ in the decimal number system. *X*, *Y* and *Z* shall be 0 or 1. “b” is the suffix for representing the binary number. The number of digits shall be also number of bits to represent a value. For examples, 10b shall use 2 bits to represent 2 and 010b shall use 3 bits to represent 2.

6 Picture Sampling and Overall Bitstream Structure

6.1 Picture Sampling

The 4:2:0 picture sampling shall be made in the same way as SMPTE ST 421. In addition to the 4:2:0 color difference format, the Y, Cb, Cr 4:2:2/4:4:4 format shall be supported in the residual layer with the residual sample of each video component as 8-bit. For the 4:2:2 format, there shall be a total eight blocks in each macroblock, 4 for Y, 2 for Cb, and 2 for Cr. Figure 2 and Figure 3 show color difference locations in a progressive frame and in interlaced fields, respectively. In Figure 3, since the top field consists of even lines in a frame and the bottom field consists of odd lines in a frame, the relationship between lines of top field and bottom field is illustrated, but any space or line between two consecutive lines in each field shall not exist. This 4:2:2 picture sampling shall be made in the same way as Figure 6-4 of Recommendation ITU-T H.264. For the 4:4:4 format, there shall be a total twelve blocks in each macroblock, 4 for Y, 4 for Cb, and 4 for Cr and the phases of Y,Cb, Cr in each pixel shall be identical.

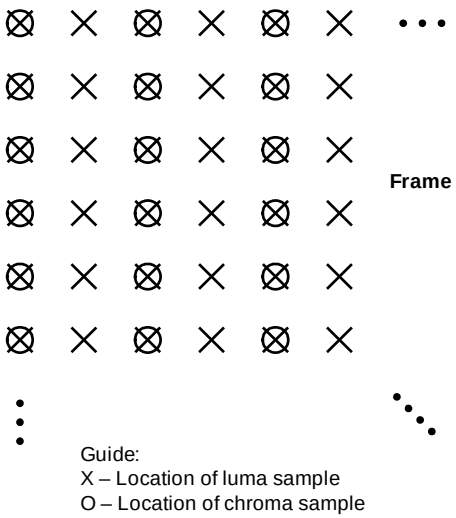

Figure 2 – Vertical and horizontal locations of 4:2:2 luma and color difference samples in a frame

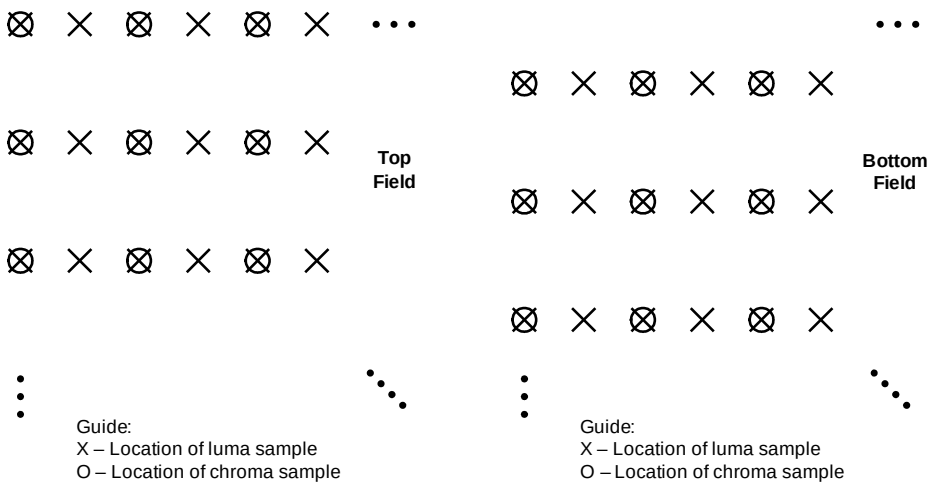

Figure 3 – Vertical and horizontal sampling locations of 4:2:2 samples in top and bottom fields

6.2 Hierarchical Elements of Bitstream

The bitstream syntax of the each enhancement layer shall consist of the hierarchical layers: sequence, pictures, slices, macroblocks and block. The sequence header shall be used as the entry point which shall be the random access point. Every picture of a progressive sequence shall be coded as the frame picture. However, the interlace sequence shall include field pictures and the frame pictures. A slice shall contain one or more contiguous macroblocks in the raster-scan order.

6.3 Bitstream Construction Constraints

A bitsteam shall be subjected to the constraints described in Annex F

7 Sequence Bitstream Syntax and Semantics

The syntax and semantics of the sequence parameters of the residual layer stream are specified in this section.

7.1 Sequence-level Syntax and Semantics

A sequence layer shall convey the sequence-level header specifying sequence-level parameters for decoding the compressed bitstreams. The bitstream syntax of the sequence layer shall be as defined in Table 1.

Table 1 – Sequence layer bitstream

SEQUENCE LAYER( ) {	Number of bits	Descriptor	Reference
LAYER_ID_SEQ	4	uimsbf	7.1.1
REF_LAYER_ID	4	uimsbf	7.1.2
PROFILE	3	uimsbf	7.1.3
LEVEL	4	uimsbf	7.1.4
PICTURE_WIDTH	16	uimsbf	7.1.5
PICTURE_HEIGHT	16	uimsbf	7.1.6
FRAMERATEIND	1	uimsbf	7.1.7
if (FRAMERATEIND == 0) {			
FRAMERATENR	8	uimsbf	7.1.8
FRAMERATEDR	4	uimsbf	7.1.9
} else {			
FRAMERATEEXP	16	uimsbf	7.1.10
}			
COLORDIFF_FORMAT	2	uimsbf	7.1.11
BIT_DEPTH_MINUS_8	4	uimsbf	7.1.12
LOWER_PICTURE_WIDTH	16	uimsbf	7.1.13
LOWER_PICTURE_HEIGHT	16	uimsbf	7.1.14
LOWER_FRAMERATEIND	1	uimsbf	7.1.15
if(LOWER_FRAMERATEIND == 0) {			
LOWER_FRAMERATENR	8	uimsbf	7.1.16
LOWER_FRAMERATEDR	4	uimsbf	7.1.17
} else {			
LOWER_FRAMERATEEXP	16	uimsbf	7.1.18
}			
LOWER_COLORDIFF_FORMAT	2	uimsbf	7.1.19
LOWER_BIT_DEPTH_MINUS_8	4	uimsbf	7.1.20
LOWER_LAYER_PRED_FLAG	1	uimsbf	7.1.21
if((PICTURE_WIDTH == LOWER_PICTURE_WIDTH) &&			

(PICTURE _HEIGHT == LOWER_PICTURE_HEIGHT) &&			
(LOWER_LAYER_PRED_FLAG == 0)) {			
BIT_UPCONVERSION	2	uimsbf	7.1.22
if**(**BIT_UPCONVERSION == 1 ||			
BIT_UPCONVERSION == 3**) {**			
ENHANCEMENT_LAYER_TONE_MAP_FLAG	1	uimsbf	7.1.23
}			
if(BIT_UPCONVERSION == 3) {			
SMOOTHING_WND	2	uimsbf	7.1.24
SMOOTHING_TH_MINUS_1	2	uimsbf	7.1.25
}			
else {			
if(BIT_DEPTH_MINUS_8 != LOWER_BIT_DEPTH_MINUS_8) {			
ENHANCEMENT_LAYER_TONE_MAP_FLAG	1	uimsbf	7.1.23
}			
}			
COLOR_DIFFERENCE_UPCONVERSION	2	uimsbf	7.1.26
if(ENHANCEMENT_LAYER_TONE_MAP_FLAG ||			
((PICTURE _HEIGHT == LOWER_PICTURE_HEIGHT) &&			
(PICTURE _HEIGHT == LOWER_PICTURE_HEIGHT) &&			
(BIT_UPCONVERSION == 2))) {			
Y_NUM_LINEAR_SEGMENTS	16	uimsbf	7.1.27
Y_LENGTH_OF_FLC_X_MINUS_1	4	uimsbf	7.1.28
Y_LENGTH_OF_FLC_Y_MINUS_1	4	uimsbf	7.1.29
for(i=0; i <= Y_NUM_LINEAR_SEGMENTS; i++) {			
Y_MAP_PARA_X[i]	variable size	uimsbf	7.1.30
Y_MAP_PARA_Y[i]	variable size	uimsbf	7.1.31
}			
U_NUM_LINEAR_SEGMENTS	16	uimsbf	7.1.32
U_LENGTH_OF_FLC_X_MINUS_1	4	uimsbf	7.1.33
U_LENGTH_OF_FLC_Y_MINUS_1	4	uimsbf	7.1.34
for(i=0; i <= U_NUM_LINEAR_SEGMENTS; i++) {			
U_MAP_PARA_X[i]	variable size	uimsbf	7.1.35
U_MAP_PARA_Y[i]	variable size	uimsbf	7.1.36
}			
V_NUM_LINEAR_SEGMENTS	16	uimsbf	7.1.37
V_LENGTH_OF_FLC_X_MINUS_1	4	uimsbf	7.1.38
V_LENGTH_OF_FLC_Y_MINUS_1	4	uimsbf	7.1.39
for(i=0; i <= V_NUM_LINEAR_SEGMENTS; i++) {			
V_MAP_PARA_X[i]	variable size	uimsbf	7.1.40
V_MAP_PARA_Y[i]	variable size	uimsbf	7.1.41
}			
}			
NUM_REF_PICTURES	4	uimsbf	7.1.42
INTERLACE_FLAG	1	uimsbf	7.1.43
LOWER_INTERLACE_FLAG	1	uimsbf	7.1.44
if(INTERLACE_FLAG == 1) {			

PAFF_FLAG	1	uimsbf	7.1.45
}			
QUALITY_REFINEMENT_FLAG	1	uimsbf	7.1.46
SEQ_INTRA_PRED_FLAG	1	uimsbf	7.1.47
if(((PICTURE_WIDTH == LOWER_PICTURE_WIDTH) &&			
(PICTURE _HEIGHT == LOWER_PICTURE_HEIGHT)) &&			
(NUM_LAYERS==1)) {			
STEREOSCOPIC_VIEW_FLAG	1	uimsbf	7.1.48
if (STEREOSCOPIC_VIEW_FLAG==1**)** {			
IS_LEFT_VIEW_FLAG	1	uimsbf	7.1.49
}			
}			
NUM_OF_SCAN_ORDERS	4	uimsbf	7.1.50
for(i = 0; i < NUM_OF_SCAN_ORDERS; i++) {			
for(j=0; j < 64;j++) {			
SEQ_ORDER_DIFF_FLAG[i][j]	1	uimsbf	7.1.51
if(SEQ_ORDER_DIFF_FLAG[i][j] == 1) {			
SEQ_SCAN_ORDER[i][j]	6	uimsbf	7.1.52
}			
}			
}			
if(INTERLACE_FLAG && !LOWER_INTERLACE_FLAG) {			
P2I_METHOD	1	uimsbf	7.1.53
if (P2I_METHOD == 1){			
P2I_WEIGHTING	1	uimsbf	7.1.54
if (P2I_WEIGHTING == 1){			
P2I_WEIGHTING_FACTOR_MINUS_2	3	uimsbf	7.1.55
}			
}			
}			
QMATRIX_EXTENSION_FLAG	1	uimsbf	7.1.56
if(QMATRIX_EXTENSION_FLAG == 1) {			
for(i=0; i < 64; i++) {			
QMAT_MINUS_1[i]	6	uimsbf	7.1.57
}			
}			
DISPLAY_EXT	1	uimsbf	7.1.58
if (DISPLAY_EXT==1) {			
DISP_HORIZ_SIZE	14	uimsbf	7.1.58
DISP_VERT_SIZE	14	uimsbf	7.1.58
ASPECT_RATIO_FLAG	1	uimsbf	7.1.58
if (ASPECT_RATIO_FLAG==1) {			
ASPECT_RATIO	4	uimsbf	7.1.58
if (ASPECT_RATIO == 15) {			
ASPECT_HORIZ_SIZE	8	uimsbf	7.1.58
ASPECT_VERT_SIZE	8	uimsbf	7.1.58
}			
}			
COLOR_FORMAT_FLAG	1	uimsbf	7.1.58
if (COLOR_FORMAT_FLAG ==1) {			
COLOR_PRIM	8	uimsbf	7.1.58
TRANSFER_CHAR	8	uimsbf	7.1.58
MATRIX_COEF	8	uimsbf	7.1.58

}			
}			
HRD_PARAM_ FLAG	1	uimsbf	7.1.58
if (HRD_PARAM_ FLAG == 1) {			
HRD_NUM_LEAKY_BUCKETS	5	uimsbf	7.1.58
BIT_RATE_EXPONENT	4	uimsbf	7.1.58
BUFFER_SIZE_EXPONENT	4	uimsbf	7.1.58
for(n=1; n <= HRD_NUM_LEAKY_BUCKETS; n++) {			
HRD_RATE[n **]**	16	uimsbf	7.1.58
HRD_BUFFER[n **]**	16	uimsbf	7.1.58
}			
}			
ALIGNtoBYTE ()			
}			

7.1.1 Layer Identifier for Sequence Layer (LAYER_ID_SEQ) (4 bits)

The syntax element LAYER_ID_SEQ shall identify the layer number of the current sequence. The range of LAYER_ID_SEQ shall be from 1 to 15. A sequence shall consist of the pictures with LAYER_ID that is the same as LAYER_ID_SEQ.

Note: The value 0 shall be reserved for the base layer.

7.1.2 Reference Layer Identifier (REF_LAYER_ID) (4 bits)

The syntax element REF_LAYER_ID shall identify the layer number of the lower layer referred to by the current picture. Since REF_LAYER_ID shall be less than LAYER_ID_SEQ, the maximum number of REF_LAYER_ID shall be LAYER_ID_SEQ minus one and the range of REF_LAYER_ID shall be from zero to LAYER_ID minus one.

7.1.3 Profile (PROFILE) (3 bits)

The syntax element PROFILE shall specify the profile of the current layer sequence. The meaning of each value shall be as defined in Table 2. The details are described in Annex D.1.

Table 2 – Meaning of PROFILE

PROFILE	Meaning
0	SMPTE Reserved
1	Main Profile
2	High Profile
3	High 4:2:2 Profile
4	High 4:4:4 Profile
5	High Intra 4:4:4 Profile
6	Stereo Profile
7	SMPTE Reserved

7.1.4 Level (LEVEL) (4 bits)

The syntax element LEVEL shall specify the decoding level for the current layer sequence. The meaning of each value shall be defined as Table 3. The details are described in Annex D.2.

Table 3 – Meaning of LEVEL

LEVEL	Meaning
0	Level 0
1	Level 1
2	Level 2
3	Level 3
4	Level 4
5	Level 5
6	Level 6
7	Level 7
8	Level 8
9	Level 9
10~15	SMPTE Reserved

7.1.5 Picture Width (PICTURE _WIDTH) (16 bits)

The syntax element PICTURE_WIDTH shall specify the horizontal size of the frame picture within the current residual layer sequence among the NUM_LAYERS layers in unit of pixels.

7.1.6 Picture Height (PICTURE _HEIGHT) (16 bits)

The syntax element PICTURE_HEIGHT shall specify the vertical size of the frame picture within the current residual layer sequence among the NUM_LAYERS layers in unit of pixels.

7.1.7 Frame Rate Indicator (FRAMERATEIND) (1 bit)

If FRAMERATEIND == 1, the frame rate of the current residual layer sequence shall be signaled explicitly by a 16 bit FRAMERATEEXP field. If FRAMERATEIND == 0, the frame rate of the current residual layer sequence shall be signaled by a numerator field (FRAMERATENR) and a denominator field (FRAMERATEDR), and the ratio of the two fields shall be taken to be the frame rate of the current residual layer sequence.

7.1.8 Frame Rate Numerator (FRAMERATENR) (8 bits)

The syntax element FRAMERATENR shall be present only if FRAMERATEIND == 0. FRAMERATENR shall indicate the frame rate numerator of the current residual layer sequence. The meaning of each value shall be as defined in Table 4.

Table 4 – Meaning of FRAMERATENR

FRAMERATENR	Value of Frame Rate Numerator
0	Forbidden
1	24 * 1000
2	25 * 1000
3	30 * 1000
4	50 * 1000
5	60 * 1000
6	48 * 1000
7	72 * 1000
8-255	SMPTE Reserved

7.1.9 Frame Rate Denominator (FRAMERATEDR) (4 bits)

The syntax element FRAMERATEDR shall be present only if FRAMERATEIND == 0. FRAMERATEDR shall indicate the frame rate denominator of the current residual layer sequence. The syntax element FRAMERATEDR shall be as defined in Table 5. The frame rate of the current residual layer sequence shall be the ratio of the FRAMERATENR to the FRAMERATEDR.

Table 5 – Meaning of FRAMERATEDR

FRAMERATEDR	Value of Frame Rate Denominator
0	Forbidden
1	1000
2	1001
3-15	SMPTE Reserved

7.1.10 Frame Rate Explicit (FRAMERATEEXP) (16 bits)

The syntax element FRAMERATEEXP shall be present only if FRAMERATEIND == 1. FRAMERATEEXP shall indicate the frame rate of the current residual layer sequence. This syntax element shall be an encoding of frame rate ranging from 0.03125 Hz to 2048 Hz in uniform steps of 0.03125 Hz.

7.1.11 Color difference Format (COLORDIFF_FORMAT) (2 bits)

The syntax element COLORDIFF_FORMAT shall specify the color difference/luma format to represent pictures of the current residual layer sequence among the NUM_LAYERS layers. The formats shall be as defined in Table 6.

Table 6 – Meaning of COLORDIFF_FORMAT

COLORDIFF_FORMAT	Format
0	4:2:0
1	4:2:2
2	4:4:4
3	SMPTE Reserved

7.1.12 Bit Depth (BIT_DEPTH_MINUS_8) (4 bits)

The syntax element BIT_DEPTH_MINUS_8 shall specify the bit depth minus 8 at the current layer sequence. The range of this syntax shall be from 0 to 8. BIT_DEPTH_MINUS_8 shall be greater than or equal to LOWER_BIT_DEPTH_MINUS_8. Here, the bit depth of the residuals, which is used as the input to the residual encoder or as the output of the residual decoder, shall be eight bits. This is noted in Figure 1 and Figure 47. After the residuals are generated using the current layer picture and the up-converted lower layer picture, these residuals shall be converted to eight bit samples as shown in Figure 1. In the reconstruction process, after the residuals are reconstructed by the residual decoder and converted by an inverse residual mapping/scaling of the encoding process, the reconstructed pictures shall be generated using these reconstructed residuals and the up-converted lower layer picture as shown in Figure 47. In Figure 1, the range of the residual shall be from 0 to 255 after the residual mapping/scaling process. In Figure 47, after the residual mapping/scaling processing, the range of residuals shall be the range corresponding to the bit depth of the current layer. The details of the residual mapping/scaling processing are described in Section 12.2.

7.1.13 Picture Width of Lower Layer (LOWER_PICTURE_WIDTH) (16 bits)

The syntax element LOWER_PICTURE_WIDTH shall specify the horizontal size of the frame picture within the lower layer sequence, referred by the current layer, in unit of pixels.

7.1.14 Picture Height of Lower Layer (LOWER_PICTURE_HEIGHT) (16 bits)

The syntax element LOWER_PICTURE_HEIGHT shall specify the vertical size of the frame picture within the lower layer sequence, referred by the current layer, in unit of pixels.

7.1.15 Frame Rate Indicator of Lower Layer (LOWER_FRAMERATEIND) (1 bit)

If LOWER_FRAMERATEIND == 1, the frame rate of the lower layer sequence shall be signaled explicitly by a 16 bit LOWER_FRAMERATEEXP field. If LOWER_FRAMERATEIND == 0, the frame rate of the lower layer sequence shall be signaled by a numerator field (LOWER_FRAMERATENR) and a denominator field (LOWER_FRAMERATEDR), and the ratio of the two fields shall be taken to be the frame rate of the lower layer sequence.

7.1.16 Frame Rate Numerator of Lower Layer (LOWER_FRAMERATENR) (8 bits)

The syntax element LOWER_FRAMERATENR shall be present only if LOWER_FRAMERATEIND == 0. LOWER_FRAMERATENR shall indicate the frame rate numerator of the lower layer sequence. The syntax element LOWER_FRAMERATENR shall be as defined in Table 7.

Table 7 – Meaning of LOWER_FRAMERATENR

LOWER_FRAMERATENR	Value of Frame Rate Numerator
0	Forbidden
1	24 * 1000
2	25 * 1000
3	30 * 1000
4	50 * 1000
5	60 * 1000
6	48 * 1000
7	72 * 1000
8-255	SMPTE Reserved

7.1.17 Frame Rate Denominator of Lower Layer (LOWER_FRAMERATEDR) (4 bits)

The syntax element LOWER_FRAMERATEDR shall be present only if LOWER_FRAMERATEIND == 0. LOWER_FRAMERATEDR shall indicate the frame rate denominator of the lower layer sequence. The syntax element LOWER_FRAMERATEDR shall be as defined in Table 8. The frame rate of the lower layer sequence shall be the ratio of LOWER_FRAMERATENR to LOWER_FRAMERATEDR.

Table 8 – Meaning of LOWER_FRAMERATEDR

LOWER_FRAMERATEDR	Value of Frame Rate Denominator
0	Forbidden
1	1000
2	1001
3-15	SMPTE Reserved

7.1.18 Frame Rate Explicit of Lower Layer (LOWER_FRAMERATEEXP) (16bits)

The syntax element LOWER_FRAMERATEEXP shall be present only if LOWER_FRAMERATEIND == 1. LOWER_FRAMERATEEXP shall indicate the frame rate of the lower layer sequence. This syntax element shall be an encoding of frame rate ranging from 0.03125 Hz to 2048 Hz in uniform steps of 0.03125 Hz.

7.1.19 Color difference Format of Lower Layer (LOWER_COLORDIFF_FORMAT) (2 bits)

The syntax element LOWER_COLORDIFF_FORMAT shall specifiy the color difference/luma format used to represent the lower layer pictures of the current residual layer sequence among the NUM_LAYERS layers. The LOWER_COLORDIFF_FORMAT shall be as defined in Table 6.

7.1.20 Bit Depth of Lower Layer (LOWER_BIT_DEPTH_MINUS_8) (4 bits)

The syntax element LOWER_BIT_DEPTH_MINUS_8 shall specify the bit depth of lower layer samples, referred by the current layer, minus 8. The range of this syntax shall be from 0 to 8.

7.1.21 1-D Filtered Prediction Flag for Lower Layer Pictures (LOWER_LAYER_PRED_FLAG) (1 bits)

The syntax element LOWER_LAYER_PRED_FLAG shall indicate whether or not the 1-D prediction filter on the reconstructed pictures of the lower layer shall be performed. If LOWER_LAYER_PRED_FLAG == 1 and both resolutions of the lower layer and the current layer are same each other, a 3-tap filter shall be applied in the horizontal direction for making the predicted pictures using the reconstructed pictures from the lower layer. If LOWER_LAYER_PRED_FLAG == 0 and the resolution of the lower layer is different from that of the current layer, the horizontal up-conversion and the vertical up-conversion shall use a set of the default filters in the resolution up-conversion process. Here, both of the default horizontal up-conversion filters and the default vertical up-conversion filters are same. If the resolution of the lower layer is different from the resolution of the current layer and LOWER_LAYER_PRED_FLAG == 1, the horizontal up-conversion shall use a set of the alternative filters and the vertical up-conversion shall use a set of the default filters in the resolution up-conversion process. In the case that both layers have the same resolution, the details are described in Section 12.1.5. In the case that both layers have different resolutions, the details are described in Section 12.1.3.

7.1.22 Bit Up-converting Method (BIT_UPCONVERSION) (2 bits)

The syntax element BIT_UPCONVERSION shall specify the bit depth up-converting method which is used to convert from the lower layer picture format to the current layer picture format for reconstructing output pictures with the residuals to be decoded. If BIT_UPCONVERSION == 0, the bit depth up-conversion shall not be performed. If BIT_UPCONVERSION == 1, the simple bit-shifting conversion method shall be used, if BIT_UPCONVERSION == 2, the specified tone-mapping method shall be used for the bit-depth extension, and if BIT_UPCONVERSION == 3, the smoothing method shall be applied to the bit-depth extension. The details are described in Section 12.1.1. When BIT_UPCONVERSION is not present, it shall be inferred to be 1. The meaning of each value shall be as defined in Table 9. If BIT_DEPTH_MINUS_8 is not equal to LOWER_BIT_DEPTH_MINUS_8, the BIT_UPCONVERSION shall not be equal to 0. Otherwise, the BIT_UPCONVERSION shall be zero.

Table 9 – Meaning of BIT_UPCONVERSION

BIT_UPCONVERSION	Bit Depth Extension Method
0	None
1	Simple Bit-shifting
2	Tone-Mapping
3	Smoothing

7.1.23 Tone Map Flag in Current Layer (ENHANCEMENT_LAYER_TONE_MAP_FLAG) (1 bit)

The syntax element ENHANCEMENT_LAYER_TONE_MAP_FLAG shall indicate whether or not the tone-mapping procedure shall be performed additionally when the resolution of the lower layer is different from the resolution of the current layer and both layers have different bit-depth samples, when LOWER_LAYER_PRED_FLAG == 1 and both resolutions of the lower layer and the current layer are same each other, or when both resolutions of the lower layer and the current layer are same each other and BIT_UPCONVERSION == 1 or BIT_UPCONVERSION == 3. If ENHANCEMENT_LAYER_TONE_MAP_FLAG == 1, the tone-mapping shall be applied after the lower layer samples are converted to the bit depth of the current layer. The details are described in Section 12.1.1.2.

When both resolutions of the lower layer and the current layer are same each other and BIT_UPCONVERSION == 2, the piece-wise linear tone-mapping procedure shall be also performed without the ENAHNCEMENT_LAYER_TONE_MAP_FLAG indication. In this case, the tone-mapping shall map the data from lower layer bit depth to bit depth of the current layer. The details are described in Section 12.1.1.2.

7.1.24 Smoothing Window Size (SMOOTHING_WND) (2 bits)

The syntax element SMOOTHING_WND shall specify the size of window when the smoothing method is selected for the bit depth extension. The range of this value shall be from one to three. Zero is SMPTE Reserved. If the size is one, the current pixel and 8 closest pixels shall be included in the operating window. If the size is two, the current pixel and 24 closest pixels shall be included in the window. If the size is three, the current pixel and 48 neighboring pixels shall be involved in the operation. The details are described in Section 12.1.1.3.

7.1.25 Smoothing Filter Threshold (SMOOTHING_TH_MINUS_1) (2 bits)

The syntax element SMOOTHING_TH_MINUS_1 shall specify the filter threshold control parameter when the smoothing method is selected for the bit depth extension. The filter threshold control parameter *T* shall be SMOOTHING_TH_MINUS_1 + 1. The range of SMOOTHING_TH_MINUS_1 shall be from zero to three. The details are described in Section 12.1.1.3.

7.1.26 Color Difference Up-converting Method (COLOR_DIFFERENCE_UPCONVERSION) (2 bits)

The syntax element COLOR_DIFFERENCE_UPCONVERSION shall specify the vertical direction color difference format up-conversion method to convert from the lower layer picture format to the current layer picture format for reconstructing output pictures with the residuals to be decoded. Here, the horizontal direction up-conversion shall be performed in the default way when the 4:2:2-to-4:4:4 or 4:2:0-to-4:4:4 conversion is needed. When the 4:2:0-to-4:4:4 conversion is needed, the vertical up-conversion and the horizontal up-conversion shall be performed sequentially. The details of the default horizontal up-conversion are described in Section 12.1.2. If COLOR_DIFFERENCE_UPCONVERSION == 0, the color difference format conversion shall not be performed. If COLOR_DIFFERENCE_UPCONVERSION == 1, the default color difference format up-conversion shall be used, while if COLOR_DIFFERENCE_UPCONVERSION == 2, each picture header within the sequence shall include the up-conversion method for the color difference format. This means each picture shall select the color difference up-conversion method independently. If the default filter is selected, the frame coding mode shall use the 4-tap bi-cubic frame filter, while the interlaced coding mode shall use the 4-tap bi-cubic interlaced filter. If COLOR_DIFFERENCE_UPCONVERSION == 2, COLOR_DIFFERENCE_UPCONV_IDX_U/COLOR_DIFFERENCE_UPCONV_IDX_V in Section 8.1.27, Section 8.1.28, Section 10.1.30 or Section 10.1.31 shall indicate which filter is used for each picture. The details are described in Section 12.1.2. The values are summarized in Table 10.

Table 10 – Meaning of COLOR_DIFFERENCE_UPCONVERSION

COLOR_DIFFERENCE_UPCONVERSION	Color Difference Extension Method
0	No
1	Default
2	Picture-level Adaptive
3	Reserved

7.1.27 Number of Linear-segments for Y Channel (Y_NUM_LINEAR_SEGMENTS) (16 bits)

The syntax element Y_NUM_LINEAR_SEGMENTS shall specify the number of intervals or segments which define the piece-wise linear maps when the additional tone-mapping is performed for the Y channel. This syntax shall be signaled when the tone-mapping is applied. The details are described in Section 12.1.1.2.

7.1.28 Length of MAP_PARA_X for Y Channel (Y_LENGTH_OF_FLC_X_MINUS_1) (4 bits)

The syntax element Y_LENGTH_OF_FLC_X_MINUS_1 shall specify the value which shall be the length of Y_MAP_PARA_X[i] minus one. Y_MAP_PARA_X[i] in Section 7.1.30 shall be the fixed length code with Y_LENGTH_OF_FLC_X_MINUS_1 + 1 bits. This syntax shall be signaled when the tone-mapping is applied.

7.1.29 Length of MAP_PARA_Y for Y Channel (Y_LENGTH_OF_FLC_Y_MINUS_1) (4 bits)

The syntax element Y_LENGTH_OF_FLC_Y_MINUS_1 shall specify the value which shall be the length of Y_MAP_PARA_Y[i] minus one. Y_MAP_PARA_Y[i] in Section 7.1.31 shall be the fixed length code with Y_LENGTH_OF_FLC_Y_MINUS_1 + 1 bits. This syntax shall be signaled when the tone-mapping is applied.

7.1.30 Tone-mapping Parameters for Input Signals of Y Channel (Y_MAP_PARA_X[i]) (variable size)

The syntax element Y_MAP_PARA_X[i]'s shall represent a series of input signal parameters for the piece-wise linear tone-mapping table of the Y channel. Each parameter Y_MAP_PARA_X[i] shall be coded with Y_LENGTH_OF_FLC_X_MINUS_1 + 1 bits. The details are described in Section 12.1.1.2.

7.1.31 Tone-mapping Parameters for Output Signals of Y Channel (Y_MAP_PARA_Y[i]) (variable size)

The syntax element Y_MAP_PARA_Y[i]'s shall represent a series of output signal parameters for the piece-wise linear tone-mapping table of the Y channel. Each parameter Y_MAP_PARA_Y[i] shall be coded with Y_LENGTH_OF_FLC_Y_MINUS_1 + 1 bits. The details are described in Section 12.1.1.2.

7.1.32 Number of Linear-segments for Cb Channel (U_NUM_LINEAR_SEGMENTS) (16 bits)

The syntax element U_NUM_LINEAR_SEGMENTS shall specifys the number of intervals or segments which define the piece-wise linear maps when the additional tone-mapping is performed for the Cb channel. This syntax shall be signaled when the tone-mapping is applied. The details are described in Section 12.1.1.2.

7.1.33 Length of MAP_PARA_X for Cb Channel (U_LENGTH_OF_FLC_X_MINUS_1) (4 bits)

The syntax element U_LENGTH_OF_FLC_X_MINUS_1 shall specify the length of U_MAP_PARA_X[i] minus one. U_MAP_PARA_X[i] in Section 7.1.35 shall be the fixed length code with U_LENGTH_OF_FLC_X_MINUS_1 + 1 bits. This syntax shall be signaled when the tone-mapping is applied.

7.1.34 Length of MAP_PARA_Y for Cb Channel (U_LENGTH_OF_FLC_Y_MINUS_1) (4 bits)

The syntax element U_LENGTH_OF_FLC_Y_MINUS_1 shall specify the length of U_MAP_PARA_Y[i] minus one. U_MAP_PARA_Y[i] in Section 7.1.36 shall be the fixed length code with U_LENGTH_OF_FLC_Y_MINUS_1 + 1 bits. This syntax shall be signaled when the tone-mapping is applied.

7.1.35 Tone-mapping Parameters for Input Signals of Cb Channel (U_MAP_PARA_X[i]) (variable size)

The syntax element U_MAP_PARA_X[i]'s shall represent a series of input signal parameters for the piece-wise linear tone-mapping table of the Cb channel. Each parameter U_MAP_PARA_X[i] shall be coded with U_LENGTH_OF_FLC_X_MINUS_1 + 1 bits. The details are described in Section 12.1.1.2.

7.1.36 Tone-mapping Parameters for Output Signals of Cb Channel (U_MAP_PARA_Y[i]) (variable size)

The syntax element U_MAP_PARA_Y[i]'s shall represent a series of output signal parameters for the piece-wise linear tone-mapping table of the Cb channel. Each parameter U_MAP_PARA_Y[i] shall be coded with U_LENGTH_OF_FLC_Y_MINUS_1 + 1 bits. The details are described in Section 12.1.1.2.

7.1.37 Number of Linear-segments for Cr Channel (V_NUM_LINEAR_SEGMENTS) (16 bits)

The syntax element V_NUM_LINEAR_SEGMENTS shall specify the number of intervals or segments which define the piece-wise linear maps when the additional tone-mapping is performed for the Cr channel. This syntax shall be signaled when the tone-mapping is applied. The details are described in Section 12.1.1.2.

7.1.38 Length of MAP_PARA_X for Cr Channel (V_LENGTH_OF_FLC_X_MINUS_1) (4 bits)

The syntax element V_LENGTH_OF_FLC_X_MINUS_1 shall specify the length of V_MAP_PARA_X[i] minus one. V_MAP_PARA_X[i] in section 7.1.40 shall be the fixed length code with V_LENGTH_OF_FLC_X_MINUS_1 + 1 bits. This syntax shall be signaled when the tone-mapping is applied.

7.1.39 Length of MAP_PARA_Y for Cr Channel (V_LENGTH_OF_FLC_Y_MINUS_1) (4 bits)

The syntax element V_LENGTH_OF_FLC_Y_MINUS_1 shall specify the length of V_MAP_PARA_Y[i] minus one. V_MAP_PARA_Y[i] in Section 7.1.41 shall be the fixed length code with V_LENGTH_OF_FLC_Y_MINUS_1 + 1 bits. This syntax shall be signaled when the tone-mapping is applied.

7.1.40 Tone-mapping Parameters for Input Signals of Cr Channel (V_MAP_PARA_X[i]) (variable size)

The syntax element V_MAP_PARA_X[i]'s shall represent a series of input signal parameters for the piece-wise linear tone-mapping table of the Cr channel. Each parameter V_MAP_PARA_X[i] shall be coded with V_LENGTH_OF_FLC_X_MINUS_1 + 1 bits. The details are described in Section 12.1.1.2.

7.1.41 Tone-mapping Parameters for Output Signals of Cr Channel (V_MAP_PARA_Y[i]) (variable size)

The syntax element V_MAP_PARA_Y[i]'s shall represent a series of output signal parameters for the piece-wise linear tone-mapping table of the Cr channel. Each parameter V_MAP_PARA_Y[i] shall be coded with V_LENGTH_OF_FLC_Y_MINUS_1 + 1 bits. The details are described in Section 12.1.1.2.

7.1.42 Number of Reference Pictures (NUM_REF_PICTURES) (4 bits)

The NUM_REF_PICTURES syntax element shall specify the maximum number of the reference frames which are available during decoding each picture of the current layer. When this value is zero, all pictures shall be decoded as the intra-picture. The range of this value shall be from 0 to 9.

7.1.43 Interlaced Flag (INTERLACE_FLAG) (1 bit)

The syntax element INTERLACE_FLAG shall indicate whether the sequence is coded in interlace mode or not. A value of 1 indicates interlaced mode.

7.1.44 Interlace Flag of Lower Layer (LOWER_INTERLACE_FLAG) (1 bit)

The syntax element LOWER_INTERLACE_FLAG shall indicate whether the lower layer sequence is coded in interlace mode or not. The value of 1 indicates interlaced mode.

7.1.45 Picture-level Adaptive Frame/Field Coding Flag (PAFF_FLAG) (1 bit)

The syntax element PAFF_FLAG shall indicate whether this sequence supports the picture-level adaptive frame/field coding mode. If this value is zero, all pictures in the interlaced sequence shall be decoded as the field picture. Otherwise, each picture shall be decoded as frame coding mode or field coding mode.

7.1.46 Quality Refinement Flag (QUALITY_REFINEMENT_FLAG) (1 bit)

The syntax element QUALITY_REFINEMENT_FLAG shall indicate whether quality refinement is used in the residual block reconstruction process. When quality refinement is used (QUALITY_REFINEMENT_FLAG == 1), residual samples of the same position in a current layer picture may be reconstructed by decoding multiple slices. The details are described in Section 9.1.4.6, Section 9.2.4, Section 11.1.4 and Section 11.2.4.

7.1.47 Sequence-level Intra Prediction Flag (SEQ_INTRA_PRED_FLAG) (1 bit)

The syntax element SEQ_INTRA_PRED_FLAG shall indicate whether the intra prediction is used in intra pictures. If SEQ_INTRA_PRED_FLAG == 0, the intra prediction shall not be allowed. Otherwise, the picture-level intra prediction flag (PIC_INTRA_PRED_FLAG) shall determine whether or not the intra-prediction is performed.

7.1.48 Stereoscopic Picture Flag (STEREOSCOPIC_VIEW_FLAG) (1 bit)

The syntax element STEREOSCOPIC_VIEW_FLAG shall indicate whether pictures in the current layer and lower layer are considered as each view of stereo-view pictures or not. If this value is one, all pictures in the lower and current layers shall be coded as one of stereo-views of stereoscopic pictures. Otherwise, all pictures shall be coded as single view pictures.

7.1.49 Left View Flag (IS_LEFT_VIEW_FLAG) (1 bit)

The syntax element IS_LEFT_VIEW_FLAG shall indicate whether pictures in the current layer are considered as left view of a stereo-view picture or not. If this value is zero, all pictures in the lower layer shall be decoded as the right-view pictures and all pictures in the current layer shall be decoded as the left-view pictures. Otherwise, all pictures in the lower layer shall be decoded as the left-view pictures and all pictures in the current layer shall be decoded as the right-view pictures.

7.1.50 Number of Scanning Orders (NUM_OF_SCAN_ORDERS) (4 bits)

The syntax element NUM_OF_SCAN_ORDERS shall specify how many optional scanning modes are defined and supported at the current layer.

7.1.51 Scanning Order Difference Flag at Sequence Layer (SEQ_ORDER_DIFF_FLAG) (1 bit)

The syntax element SEQ_ORDER_DIFF_FLAG shall indicate whether or not the n-th element of an optional scanning order is equal to the n-th element of the default scanning order, where n means an arbitrary index of the scanning order element. When SEQ_ORDER_DIFF_FLAG == 1, the n-th position value of the scanning order shall be different from that of the default scanning order. When the sequence is progressive, the default scanning order is depicted in Figure 15, while when the sequence is interlace, the default scanning order is depicted in Figure 38.

7.1.52 Sequence Layer Scanning Order (SEQ_SCAN_ORDER) (6 bits)

The syntax element SEQ_SCAN_ORDER shall specify the scanning order of the n-th element when SEQ_ORDER_DIFF_FLAG== 1.

7.1.53 Progressive to Interlaced Conversion Method (P2I_METHOD) (1 bit)

The syntax element P2I_METHOD, when set to 1, shall indicate whether each picture header within the interlaced sequence includes P2I_METHOD_FLAG. When P2I_METHOD == 1, the header of each interlaced picture shall indicate whether the interlaced pictures are generated using one frame or using two successive frames of the lower layer. The details are described in Section 10.1.18.

7.1.54 Progressive to Interlaced Conversion Weighting (P2I_WEIGHTING) (1 bit)

The syntax element P2I_WEIGHTING, when set to 1, shall indicate that the interlaced pictures are generated using the weighted interpolation of the two successive frames of the lower layer. When P2I_WEIGHTING == 0, the interlaced pictures shall be generated with equal weighting factors for the interpolation of the two successive frames of the lower layer. The details are described in Section 12.1.3.3.

7.1.55 Weighting Factor for Progressive to Interlaced Conversion (P2I_WEIGHTING_FACTOR_MINUS_2) (3 bits)

The syntax element P2I_WEIGHTING_FACTOR_MINUS_2 shall specify the value of weighting factor when P2I_WEIGHTING == 1. The weighting factor parameter shall be P2I_WEIGHTING_FACTOR_MINUS_2 + 2. The details are described in Section 12.1.3.3.

7.1.56 Quantization Matrix Extension Flag (QMATRIX_EXTENSION_FLAG) (1 bit)

The syntax element QMATRIX_FLAG shall indicate whether an additional quantization matrix is used for the current residual layer sequence or not. If QMATRIX_FLAG == 1, the additional quantization matrix shall be described by QMAT_MINUS_1[i]. Each layer can define the additional quantization matrix independently.

7.1.57 Quantization Matrix Component (QMAT_MINUS_1[i]) (6 bits)

The syntax element QMAT_MINUS_1[i] shall specify the i-th component of the additional quantization matrix when QMATRIX_FLAG == 1. The range of QMAT_MINUS_1[i] shall be from 0 to 63. Each component QMAT[i] of the quantization matrix shall be QMAT_MINUS_1[i] + 1. The details are described in Section 9.1.4.4.

7.1.58 Other Fields

The following fields shall be as defined in SMPTE ST 421, Section 6.1:

1. Display Extension Flag (DISPLAY_EXT) (1 bit)
2. Horizontal Display Size of Picture (DISP_HORIZ_SIZE) (14 bits)
3. Vertical Display Size of Picture (DISP_VERT_SIZE) (14 bits)
4. Sample Aspect Ratio Indicator Flag (ASPECT_RATIO_FLAG) (1 bit)
5. Sample Aspect Ratio (ASPECT_RATIO) (4 bits)
6. Aspect Width (ASPECT_HORIZ_SIZE) (8 bits)
7. Aspect Height (ASPECT_VERT_SIZE) (8 bits)
8. Color Format Indicator Flag (COLOR_FORMAT_FLAG) (1 bit)
9. Color Primaries (COLOR_PRIM) (8 bits)
10. Transfer Characteristics (TRANSFER_CHAR) (8 bits)
11. Matrix Coefficients (MATRIX_COEF) (8 bits)
12. Hypothetical Reference Decoder Indicator Flag (HRD_PARAM_FLAG) (1 bit)
13. HRD Number of Leaky Buckets (HRD_NUM_LEAKY_BUCKETS) (5 bits)
14. Bit Rate Exponent (BIT_RATE_EXPONENT) (4 bits)
15. Buffer Size Exponent (BUFFER_SIZE_EXPONENT) (4 bits)
16. HRD Bit Rate (HRD_RATE[n]) (16 bits)
17. HRD Buffer Size (HRD_BUFFER[n]) (16 bits)

8 Progressive Bitstream Syntax and Semantics

The syntax and semantics of the picture level, slice level, macroblock level, and block level of the frame pictures in the residual layer stream are specified in this section. Since the picture shall consist of one or more slices, the syntax and semantics of a picture level shall be referred to by one or more slices.

8.1 Picture-level Syntax and Semantics

Table 11 defines the bit stream syntax of the progressive picture layer.

Table 11 – Progressive picture layer bitstream

PICTURE LAYER( ) {	Number of bits	Descriptor	Reference
LAYER_ID	4	uimsbf	8.1.1
PICTURE_ID	8	uimsbf	8.1.2
CODED_ORDER	8	uimsbf	8.1.3
LAYER_ENTRY_FLAG	1	uimsbf	8.1.4
if(QUALITY_REFINEMENT_FLAG == 1) {			
MAX_QUALITY_LAYER_ID	4	uimsbf	8.1.5
QUALITY_REFERENCE_PICTURE_FLAG	1	uimsbf	8.1.6
}			
REFERENCED_PICTURE_FLAG	1	uimsbf	8.1.7
if(REFERENCED_PICTURE_FLAG == 1) {			
REMOVE_ENTRY_FLAG	1	uimsbf	8.1.8
if(REMOVE_ENTRY_FLAG == 1) {			

REMOVE_PICTURE_ID	8	uimsbf	8.1.9
}			
}			
if(LAYER_ENTRY_FLAG == 0) {			
MC_FLAG	1	uimsbf	8.1.10
}			
if(MC_FLAG == 1) {			
B_PICTURE_FLAG	1	uimsbf	8.1.11
if(B_PICTURE_FLAG == 0) {			
PRED_DIRECTION	1	uimsbf	8.1.12
}			
MV_MODE	variable size	vlclbf	8.1.13
for(i = 0; i < B_PICTURE_FLAG+1;i++) {			
REFERENCE_PICTURE_ID[i]	8	uimsbf	8.1.14
}			
}			
else {			
if(SEQ_INTRA_PRED_FLAG == 1) {			
PIC_INTRA_PRED_FLAG	1	uimsbf	8.1.15
}			
}			
SCAN_ORDER_TYPE	variable size	vlclbf	8.1.16
if(SCAN_ORDER_TYPE == 1) {			
SEQ_SCAN_ORDER_IDX	4	uimsbf	8.1.17
}			
else if(SCAN_ORDER_TYPE == 2) {			
for(i=0; i < 64;i++) {			
PIC_ORDER_DIFF_FLAG[i]	1	uimsbf	8.1.18
if(PIC_ORDER_DIFF_FLAG[i] == 1) {			
PIC_SCAN_ORDER[i]	6	uimsbf	8.1.19
}			
}			
}			
QP_PICTURE_UNIFORM_FLAG	1	uimsbf	8.1.20
QP_CHANNEL_UNIFORM_FLAG	1	uimsbf	8.1.21
if(QP_CHANNEL_UNIFORM_FLAG == 1) {			
QP_PICTURE	8	uimsbf	8.1.22
}			
else {			
QP_PICTURE_Y	8	uimsbf	8.1.23
QP_PICTURE_U	8	uimsbf	8.1.24
QP_PICTURE_V	8	uimsbf	8.1.25
}			
CHANNEL_SKIP_MODE	8	uimsbf	8.1.26
if(is_adaptive_color_difference_upconversion()) {			
COLOR_DIFFERENCE_UPCONV_IDX_U	1	uimsbf	8.1.27
COLOR_DIFFERENCE_UPCONV_IDX_V	1	uimsbf	8.1.28
}			
RES_SCALING_PRESENT_FLAG	1	uimsbf	8.1.29
if(RES_SCALING_PRESENT_FLAG == 1) {			
if(!CHANNEL_SKIP_MODE & 0x80) {			
SCALE_Y	9	uimsbf	8.1.30

SHIFT_Y	16	uimsbf	8.1.31
NORM_Y	4	uimsbf	8.1.32
}			
if(!CHANNEL_SKIP_MODE & 0x40) {			
SCALE_U	9	uimsbf	8.1.33
SHIFT_U	16	uimsbf	8.1.34
NORM_U	4	uimsbf	8.1.35
}			
if(!CHANNEL_SKIP_MODE & 0x20) {			
SCALE_V	9	uimsbf	8.1.36
SHIFT_V	16	uimsbf	8.1.37
NORM_V	4	uimsbf	8.1.38
}			
}			
if (STEREOSCOPIC_VIEW_FLAG == 1) {			
for('all macroblocks') {			
if(MC_FLAG = = 1) {			
VIEW_PRED_FLAG	1	uimsbf	8.1.39
}			
PRED_BLK_SIZE_FLAG	1	uimsbf	8.1.40
if(PRED_BLK_SIZE_FLAG == 0) {			
DMVDx	variable size	evlclbf	8.1.41
DMVDy	variable size	evlclbf	8.1.42
}			
else {			
for(i = 0; i < 4;i++) {			
DMVDx[i]	variable size	evlclbf	8.1.41
DMVDy[i]	variable size	evlclbf	8.1.42
}			
}			
}			
}			
ALIGNtoBYTE ()			

8.1.1 Layer Identifier (LAYER_ID) (4 bits)

The syntax element LAYER_ID shall identify the layer number of the current picture. The range of LAYER_ID shall be from 1 to 15. A picture shall consist of one or more than one slice with LAYER_ID_SLICE in Section 8.2.1 where LAYER_ID_SLICE is the same as this LAYER_ID

8.1.2 Picture Identifier (PICTURE_ID) (8 bits)

The syntax element PICTURE_ID shall identify the current picture. PICTURE_ID shall be described with the display order of the current picture. The range of PICTURE_ID shall be from 0 to 255. The increment between PICTURE_IDs of two consecutive pictures shall be any positive number. This means the PICTURE_ID of the next picture may not be the PICTURE_ID of the current picture plus one. However the increment between two PICTURE_IDs of two consecutive pictures shall be constant in a sequence. This shall be assigned in the wraparound counting way. For example, if the PICTURE_ID of the current picture is 254 and the increment of

PICTURE_ID is 4, the PICTURE_ID of the next displayed picture shall be two. Whenever LAYER_ENTRY_FLAG == 1, PICTURE_ID shall be zero.

8.1.3 Coded-order (CODED_ORDER) (8 bits)

The syntax element CODED_ORDER shall be described with the coded-order number. This shall be also assigned in the wraparound counting way similar to PICTURE_ID. The range of the value shall be from 0 to 255. However, when the 256-th picture in coded order is coded, its CODED_ORDER shall be zero and the CODED_ORDER of the next coded-ordered picture shall be one. The increment between two consecutively coded-ordered pictures shall be one. Whenever LAYER_ENTRY_FLAG == 1, CODED_ORDER shall be zero.

8.1.4 Layer Entry Flag (LAYER_ENTRY_FLAG) (1 bit)

The syntax element LAYER_ENTRY_FLAG, when set to 1, shall indicate that the current picture and subsequent-coded pictures do not use the previously-coded pictures as the reference picture. It also means the current coded picture does not use any reference pictures for motion compensation. This flag, when set to 1, shall indicate an entry point at each layer. If LAYER_ENTRY_FLAG == 1, any other information of the current layer shall not exist between this picture header and the sequence header of current layer. The bitstream configuration is explained in Annex F .

If the lower layer picture of the current picture is entry point for random access, the syntax element LAYER_ENTRY_FLAG of the current picture shall be equal to one for random access. The detail is described in Section 9.4.

8.1.5 Maximum Quality Layer Identifier (MAX_QUALITY_LAYER_ID) (4 bit)

The syntax element MAX_ QUALITY_LAYER_ID shall be present only if QUALITY_REFINEMENT_FLAG == 1. MAX_QUALITY_LAYER_ID shall specify that the maximum value of the syntax QUALITY_LAYER_ID specified in Section 8.2.2.

8.1.6 Quality Reference Picture Flag (QUALITY_REFERENCE_PICTURE_FLAG) (1 bit)

The syntax element QUALITY_REFERENCE_PICTURE_FLAG shall be present only if QUALITY_REFINEMENT_FLAG == 1. If QUALITY_REFERENCE_PICTURE_FLAG == 1, the current picture shall use the reference pictures marked as the quality enhancement reference instead of the reference pictures marked as the quality base layer reference within the reference picture buffer. Otherwise, the current picture shall use the reference pictures marked as the quality base layer reference instead of the reference pictures marked as the quality enhancement layer reference within the reference picture buffer. The details are described in Section 9.4 and Section 11.4.

8.1.7 Referenced Picture Flag (REFERENCED_PICTURE_FLAG) (1 bit)

The syntax element REFERENCED_PICTURE_FLAG, when set to 1, shall indicate that the current picture is used as the reference picture for the next coded-order pictures. If QUALITY_REFERENCE_PICTURE_FLAG == 0, both of the quality base layer and the quality enhancement layer of the current picture shall be used as the reference pictures. If QUALITY_REFERENCE_PICTURE_FLAG == 1, the quality enhancement layer of the current picture shall be used as the reference picture.

8.1.8 Removing Entry Flag (REMOVE_ENTRY_FLAG) (1 bit)

The syntax element REMOVE_ENTRY_FLAG, when set to 1, shall indicate that one picture of the reference frame list which contains the reconstructed pictures for temporal-predictive coding of the future picture is removed in the list. The reference frame list contains the decoded frames which are used as reference for decoding the subsequent frames in the coded-order.

8.1.9 Removing Picture Identifier (REMOVE_PICTURE_ID) (8 bits)

The syntax element REMOVE_PICTURE_ID shall specify which picture is removed in the reference frame list. The picture identified with REMOVE_PICTURE_ID shall be removed. If REFERENCED_PICTURE_FLAG == 1 and REMOVE_ENTRY_FLAG == 1, the picture identified with REMOVE_PICTURE_ID shall be replaced by the current decoded picture after decoding the current picture.

8.1.10 Motion Compensation Flag (MC_FLAG) (1 bit)

The syntax element MC_FLAG, when set to 1, shall indicate that the motion compensation is employed for decoding the current picture. Even if the reference frame list is not empty, when this value is zero, the current picture shall be decoded like intra picture. Whenever LAYER_ENTRY_FLAG == 1, MC_FLAG shall be set to zero.

8.1.11 B-picture Flag (B_PICTURE_FLAG) (1 bit)

The syntax element B_PICTURE_FLAG, when set to 1, shall indicate that the current frame is coded as a bidirectionally predictive-coded picture (B picture). If B_PICTURE_FLAG == 0, the current frame shall be coded as a P picture.

8.1.12 Predictive Direction for P Picture (PRED_DIRECTION) (1 bit)

The syntax element PRED_DIRECTION shall indicate the temporal direction of the motion compensated prediction for P-picture. If PRED_DIRECTION is set to 0, a past picture shall be selected as a reference frame for the current picture (forward P-picture coding mode). Otherwise, a future picture shall be selected as a reference frame (backward P-picture coding mode).

8.1.13 MV Mode (MV_MODE) (variable size)

The syntax element MV_MODE shall specify the precision of the motion vector and shall be as defined in Table 12.

Table 12 – Meaning of MV_MODE

MV_MODE	Codeword	Precision of MV
0	0b	Quarter-pel
1	10b	Half-pel
2	11b	One-pel

8.1.14 Reference Picture Identifier (REFERENCE_PICTURE_ID[i]) (8 bits)

The syntax element REFERENCE_PICTURE_ID[i] shall identify which reconstructed picture in the reference frame list is used for the reference picture in the current layer for motion compensation. This value points the PICTURE_ID of the reference picture. If the current frame is coded with B picture coding mode, REFERENCE_PITURE_ID[0] and REFERENCE_PITURE_ID[1] shall be selected as reference pictures. REFERENCE_PITURE_ID[0] and REFERENCE_PITURE_ID[1] point a past picture and the future picture respectively. If the current frame is coded with P picture coding mode, which has only one reference frame, REFERENCE_PICTURE_ID[0] shall be selected as a reference picture.

8.1.15 Picture-level Intra Prediction Flag (PIC_INTRA_PRED_FLAG) (1 bit)

The syntax element PIC_INTRA_PRED_FLAG shall be present only if MC_FLAG == 0 and SEQ_INTRA_PRED_FLAG == 1. PICTURE_INTRA_PRED_FLAG shall indicate whether the intra prediction

is used in the current picture. If PIC_INTRA_PRED_FLAG == 0, the intra prediction shall not be used in the current picture. Otherwise, the intra prediction shall be used in the current picture. When PIC_INTRA_PRED_FLAG is not present, it shall be 0.

8.1.16 Scanning Order Type (SCAN_ORDER_TYPE) (variable size)

The syntax element SCAN_ORDER_TYPE shall specify which type of scanning order is used for the current picture. If SCAN_ORDER_TYPE == 0, the default order for the frame coding shall be selected. When SCAN_ORDER_TYPE == 1, one scanning order among the predefined-scanning orders in Section 7.1.50, Section 7.1.51 and Section 7.1.52 shall be selected. When SCAN_ORDER_TYPE == 2, this current picture shall employ the picture-level user-defined scanning order which is defined in Section 8.1.18 and Section 8.1.19. These values are summarized in Table 13.

Table 13 – Meaning of SCAN_ORDER_TYPE

SCAN_ORDER_TYPE	Codeword	Scanning Order Type
0	1b	Default Order
1	01b	One among the Sequence-level Predefined Orders
2	00b	Picture-level User-defined Order

8.1.17 Scanning Order Index (SEQ_SCAN_ORDER_IDX) (4 bits)

The syntax element SEQ_SCAN_ORDER_IDX shall indicate which scanning order among the sequence-level scanning orders is applied to decode the current picture. The number of the sequence-level scanning orders is specified in Section 7.1.50. This value is the index of the optional scanning modes.

8.1.18 Scanning Order Difference Flag at Picture Layer (PIC_ORDER_DIFF_FLAG) (1 bit)

This syntax element PIC_ORDER_DIFF_FLAG shall be the same as SEQ_ORDER_DIFF_FLAG in Section 7.1.51 except for that this flag is for the picture layer.

8.1.19 Picture Layer Scanning Order (PIC_SCAN_ORDER) (6 bits)

This syntax element PIC_SCAN_ORDER shall be the same as SEQ_ SCAN_ORDER in Section 7.1.52 except for that this field is for the picture layer.

8.1.20 Uniform Picture Quantizer Flag (QP_PICTURE_UNIFORM_FLAG) (1 bit)

The syntax element QP_PICTURE_UNIFORM_FLAG shall indicate whether or not the QP varies spatially across the picture. If QP_PICTURE_UNIFORM_FLAG == 1, the QP does not vary spatially across the picture. If QP_PICTURE_UNIFORM_FLAG == 0, the quantization parameter shall be decided at the macroblock level. The details are described in Section 9.1.1.1.

8.1.21 Uniform Channel Quantizer Flag (QP_CHANNEL_UNIFORM_FLAG) (1 bit)

The syntax element QP_CHANNEL_UNIFORM_FLAG shall indicate whether or not the QP varies spatially across the color channels. If QP_ CHANNEL_UNIFORM == 1, the QP does not vary spatially across the color channels. The details are described in Section 9.1.1.1.

8.1.22 Picture Quantizer Parameter (QP_PICTURE) (8 bits)

The syntax element QP_PICTURE shall specify the picture-level quantizer parameter for the entire picture. This parameter is used as the scale in inverse quantization. If QP_CHANNEL_UNIFORM_FLAG == 1, this

value shall be the default quantization scale for all color channels. The range of QP_PICTURE shall be from 0 to 255.

8.1.23 Picture Quantizer Parameter for Y Channel (QP_PICTURE_Y) (8 bits)

The syntax element QP_PICTURE_Y shall specify the picture-level quantizer parameter for the Y channel. This parameter shall be used as the default scale in inverse quantization for the Y channel when QP_CHANNEL_UNIFORM == 0. The range of QP_PICTURE shall be from 0 to 255.

8.1.24 Picture Quantizer Parameter for Cb Channel (QP_PICTURE_U) (8 bits)

The syntax element QP_PICTURE_U shall specify the picture-level quantizer parameter for the Cb channel. This parameter shall be used as the default scale in inverse quantization for the Cb channel when QP_CHANNEL_UNIFORM == 0. The range of QP_PICTURE shall be from 0 to 255.

8.1.25 Picture Quantizer Parameter for Cr Channel (QP_PICTURE_V) (8 bits)

The syntax element QP_PICTURE_V shall specify the picture-level quantizer parameter for the Cr channel. This parameter shall be used as the default scale in inverse quantization for the Cr channel when QP_CHANNEL_UNIFORM == 0. The range of QP_PICTURE shall be from 0 to 255.

8.1.26 Channel Skip Mode (CHANNEL_SKIP_MODE) (8 bits)

The syntax element CHANNEL_SKIP_MODE shall specify which channels among luma and color difference channels are skipped. The most significant three bits shall be used and the least significant five bits shall be reserved. If the most significant bit is set to one, the luma channel shall not be coded and skipped. If the second most significant bit is set to one, then the Cb channel shall be skipped, and if the third most significant bit is set to one, then the Cr channel shall be skipped.

8.1.27 Color Difference Up-conversion Method for Cb Channel (COLOR_DIFFERENCE_UPCONV_IDX_U) (1 bit)

The syntax element COLOR_DIFFERENCE_UPCONV_IDX_U is the index which shall indicate a 4-tab bi-cubic interpolation filter or a 2-tap bi-linear interpolation filter to be used for the Cb channel color difference format extension in the vertical direction. The 4-tap filter shall be selected when it is zero, while the 2-tap filter shall be selected when it is one. The details are described in Section 12.1.2.1.

Table 14 – Meaning of COLOR_DIFFERENCE_UPCONV_IDX_U/COLOR_DIFFERENCE_UPCONV_IDX_V in progressive picture

COLOR_DIFFERENCE_UPCONV_IDX_U COLOR_DIFFERENCE_UPCONV_IDX_V	Filter Coefficients
0	{-2, 9, 28, -3}, {-3, 28, 9, -2}
1	{1, 3}, {3, 1}

8.1.28 Color Difference Up-conversion Method for Cr Channel (COLOR_DIFFERENCE_UPCONV_IDX_V) (1 bit)

The syntax element COLOR_DIFFERENCE_UPCONV_IDX_V is the index which shall indicate a 4-tab bi-cubic interpolation filter or a 2-tap bi-linear interpolation filter to be used for the Cr channel color difference format extension in the vertical direction. The 4-tap filter shall be selected when it is zero, while the 2-tap filter shall be selected when it is one. The details are described in Section 12.1.2.1.

8.1.29 Residual Mapping/Scaling Flag (RES_SCALING_PRESENT_FLAG) (1 bit)

The syntax element RES_SCALING_PRESENT_FLAG, when set to 1, shall indicate that residual mapping/scaling parameters are present in the bitstream. The details are described in Section 12.2.

8.1.30 Residual Scaling Parameter for Y Channel (SCALE_Y) (9 bits)

The syntax element SCALE_Y shall specify the scaling parameter for residuals of the Y channel. The range of SCALE_Y shall be from 0 to 511 and SCALE_Y shall be coded with a 9 bit fixed-length code. The default value of SCALE_Y shall be 0. *Scale* in Section 12.2 shall be SCALE_Y plus one for the Y channel. The details are described in Section 12.2.

8.1.31 Residual Shift Parameter for Y Channel (SHIFT_Y) (16 bits)

The syntax element SHIFT_Y shall specify the shifting parameter for residuals of the Y channel. The range of SHIFT_Y shall be from 0 to 65535 and SHIFT_Y shall be coded with a 16 bit fixed-length code. The default value of SHIFT_Y shll be 128. *Shift* in Section 12.2 shall be SHIFT_Y plus one for the Y channel. The details are described in Section 12.2.

8.1.32 Residual Norm Parameter for Y Channel (NORM_Y) (4 bits)

The syntax element NORM_Y shall specify the normalization parameter for residuals of the Y channel. The range of NORM_Y shall be from 0 to 15 and NORM_Y shall be coded with a 4 bit fixed-length code. The default value of NORM_Y shall be 0. *Norm* in Section 12.2 shall be NORM_Y plus one for the Y channel. The details are described in Section 12.2.

8.1.33 Residual Scaling Parameter for Cb Channel (SCALE_U) (9 bits)

The syntax element SCALE_U shall specify the scaling parameter for residuals of the Cb channel. The range of SCALE_U shall be from 0 to 511 and SCALE_U shall be coded with a 9 bit fixed-length code. The default value of SCALE_U shall be 0. *Scale* in section 12.2 shall be SCALE_U plus one for the Cb channel. The details are described in Section 12.2.

8.1.34 Residual Shift Parameter for Cb Channel (SHIFT_U) (16 bits)

The syntax element SHIFT_U shall specify the shifting parameter for residuals of the Cb channel. The range of SHIFT_U shall be from 0 to 65535 and SHIFT_U shall be coded with a 16 bit fixed-length code. The default value of SHIFT_U shall be 128. *Shift* in Section 12.2 shall be SHIFT_U plus one for the Cb channel. The details are described in Section 12.2.

8.1.35 Residual Norm Parameter for Cb Channel (NORM_U) (4 bits)

The syntax element NORM_U shall specify the normalization parameter for residuals of the Cb channel. The range of NORM_U shall be from 0 to 8 and NORM_U shall be coded with a 4 bit fixed-length code. The default value of NORM_U shall be 0. *Norm* in Section 12.2 shall be NORM_U plus one for the Cb channel. The details are described in Section 12.2.

8.1.36 Residual Scaling Parameter for Cr Channel (SCALE_V) (9 bits)

The syntax element SCALE_V shall specify the scaling parameter for residuals of the Cr channel. The range of SCALE_V shall be from 0 to 511 and SCALE_V shall be coded with a 9 bit fixed-length code. The default value of SCALE_V shall be 0. *Scale* in Section 12.2 shall be SCALE_V plus one for the Cr channel. The details are described in Section 12.2.

8.1.37 Residual Shift Parameter for Cr Channel (SHIFT_V) (16 bits)

The syntax element SHIFT_V shall specify the shifting parameter for residuals of the Cr channel. The range of SHIFT_V shall be from 0 to 65535 and SHIFT_V shall be coded with a 16 bit fixed-length code. The default value of SHIFT_V shall be 128. *Shift* in Section 12.2 shall be SHIFT_V plus one for the Cr channel. The details are described in Section 12.2.

8.1.38 Residual Norm Parameter for Cr Channel (NORM_V) (4 bits)

The syntax element NORM_V shall specify the normalization parameter for residuals of the Cr channel. The range of NORM_V shall be from 0 to 15 and NORM_V shall be coded with a 4 bit fixed-length code. The default value of NORM_V shall be 0. *Norm* in Section 12.2 shall be NORM_V plus one for the Cr channel. The details are described in Section 12.2.

8.1.39 View Prediction Flag (VIEW_PRED_FLAG) (1 bit)

The syntax element VIEW_PRED_FLAG shall be present only if STEREOSCOPIC_VIEW_FLAG == 1 and MC_FLAG == 1. VIEW_PRED_FLAG shall indicate whether the current macroblock uses the view prediction or the temporal prediction. When VIEW_PRED_FLAG is not present, it shall be inferred to be equal to one. If VIEW_PRED_FLAG == 1, the current macroblock shall use the view prediction. If VIEW_PRED_FLAG == 0, the current macroblock shall use the temporal prediction. The details are described in Section 12.1.4.

8.1.40 Prediction Block Size Flag (PRED_BLK_SIZE_FLAG) (1 bit)

The syntax element PRED_BLK_SIZE_FLAG shall be present only if STEREOSCOPIC_VIEW_FLAG == 1. PRED_BLK_SIZE_FLAG shall indicate whether the current macroblock is coded with one disparity/motion vector or four disparity/motion vectors. If PRED_BLK_SIZE_FLAG == 0, the current macroblock shall be coded with one disparity/motion vector. If PRED_BLK_SIZE_FLAG == 1, the current macroblock shall be coded four disparity/motion vectors. The details are described in Section 12.1.4.

8.1.41 Disparity/Motion Vector Differential in X-axis (DMVDx) (variable size)

The syntax element DMVDx shall specify the disparity/motion vector differential in the horizontal direction. If VIEW_PRED_FLAG == 1, DMVDx shall mean the disparity vector differential in the horizontal direction. If VIEW_PRED_FLAG == 0, DMVDx shall mean the motion vector differential in the horizontal direction. DMVDx shall be decoded using exponential-Golomb codes as described in Section 12.1.4.1. The range of the DMVDx shall be specified by limitations on MV [H] in Section D.2.

8.1.42 Disparity/Motion Vector Differential in Y-axis (DMVDy) (variable size)

The syntax element DMVDy shall specify the disparity/motion vector differential in the vertical direction. If VIEW_PRED_FLAG == 1, DMVDy shall mean the disparity vector differential in the vertical direction. If VIEW_PRED_FLAG == 0, DMVDy shall mean the motion vector differential in the vertical direction. DMVDy shall be decoded using exponential-Golomb codes as described in Section 12.1.4.1. The range of the DMVDy shall be specified by limitations on MV [V] in Section D.2.

8.2 Slice-level Syntax and Semantics

Table 15 shows the bitstream configuration that makes up the slice layer. The picture shall consist of one or more than one slices. The layer identifier of the picture level LAYER_ID in Section 8.1.1 and that of the slice level LAYER_ID_SLICE in Section 8.2.1 shall be same. When the quality refinement is not used or the slice is in quality base layer, the number of the coded macroblocks by each slice shall be identical to the total number of macroblocks in a picture. Otherwise, the number of the coded macroblocks by the slice may not be identical to the total number of macroblocks in a picture.

Table 15 – Progressive slice layer bitstream

SLICE LAYER( ) {	Number of bits	Descriptor	Reference
LAYER_ID_SLICE	4	uimsbf	8.2.1
if(QUALITY_REFINEMENT_FLAG == 1) {			
QUALITY_LAYER_ID	4	uimsbf	8.2.2
}			
FIRST_MB_X	10	uimsbf	8.2.3
FIRST_MB_Y	10	uimsbf	8.2.4
if(QP_CHANNEL_UINFORM_FLAG == 1) {			
DQP_SLICE	variable size	evlclbf	8.2.5
}			
else {			
DQP_SLICE_Y	variable size	evlclbf	8.2.6
DQP_SLICE_U	variable size	evlclbf	8.2.7
DQP_SLICE_V	variable size	evlclbf	8.2.8
}			
for('all macroblocks within a slice') {			
MB LAYER()			Table 16
}			
TerminateBit()			
ALIGNtoBYTE ()			
}			

8.2.1 Layer Identifier for Slice (LAYER_ID_SLICE) (4 bits)

The syntax element LAYER_ID_SLICE shall identify the layer number of the current slice. This value shall be the same as LAYER_ID of the picture which includes the current slice.

8.2.2 Quality Layer Identifier (QUALITY_LAYER_ID) (4 bits)

The syntax element QUALITY_LAYER_ID shall be present only if QUALITY_REFINEMENT_FLAG == 1. This syntax element shall specify the quality layer number of the current slice. QUALITY_LAYER_ID shall be in the range of 0 to MAX_QUALITY_LAYER_ID. When QUALITY_LAYER_ID is equal to 0, the slice shall be corresponding to the quality base layer. When QUALITY_LAYER_ID is greater than 0, the slice shall be corresponding to the quality enhancement layer. The reconstructed residual samples of the quality base layer construct the quality base layer picture. In the quality enhancement layer, the slice shall contain refinement data to enhance quality of the quality base layer picture. These data shall be used to reconstruct the residual samples of the quality enhancement layer picture. If the maximum number of QUALITY_LAYER_ID among the received slices at a decoder is 0 for the current picture reconstruction, the quality base layer picture shall be indentical to the quality enhancement layer picture.

8.2.3 X-Coordinate of the First MB in Slice (FIRST_MB_X) (10bits)

The syntax element FIRST_MB_X shall specify the x-coordinate of the first macroblock in the current slice. If the first macroblock in the current slice is the *N*-th macroblock in the horizontal direction and the *M*-th macroblock in the vertical direction, FIRST_MB_X shall be *N*.

8.2.4 Y-Coordinate of the First MB in Slice (FIRST_MB_Y) (10bits)

The syntax element FIRST_MB_Y shall specify the y-coordinate of the first macroblock in the current slice. If the first macroblock in the current slice is the *N*-th macroblock in the horizontal direction and the *M*-th macroblock in the vertical direction, FIRST_MB_Y shall be *M*.

8.2.5 Differential of Slice-level Quantizer Parameter (DQP_SLICE) (variable size)

The syntax element DQP_SLICE shall specify the differential of the quantizer parameter for the entire slice. Additionally, this syntax element DQP_SLICE shall be decoded using exponential-Golomb codes and this decoded value shall be used to generate the slice-level quantizer parameter. This generated slice-level quantizer parameter shall be used as the scale in inverse quantization. If QP_CHANNEL_UNIFORM_FLAG == 1, this generated slice-level quantizer parameter shall be the default quantization scale for all color channels. The details are described in Section 9.1.2.1.

8.2.6 Differential of Slice-level Quantizer Parameter for Luma Channel (DQP_SLICE_Y) (variable size)

The syntax element DQP_SLICE_Y shall specify the differential of the quantizer parameter for the luma channel of the entire slice. Additionally, this syntax element DQP_SLICE_Y shall be decoded using exponential-Golomb codes and this decoded value shall be used to generate the slice-level quantizer parameter for the luma channel. This generated slice-level parameter shall be used as the default scale in inverse quantization for the luma channel when QP_CHANNEL_UNIFORM == 0. The details are described in Section 9.1.2.1.

8.2.7 Differential of Slice-level Quantizer Parameter for Cb Channel (DQP_SLICE_U) (variable size)

The syntax element DQP_SLICE_U shall specify the differential of the quantizer parameter for the Cb channel of the entire slice. Additionally, this syntax element DQP_SLICE_U shall be decoded using exponential-Golomb codes and this decoded value shall be used to generate the slice-level quantizer parameter for the Cb channel. This generated parameter shall be used as the default scale in inverse quantization for the Cb channel when QP_CHANNEL_UNIFORM == 0. The details are described in Section 9.1.2.1

8.2.8 Differential of Slice-level Quantizer Parameter for Cr Channel (DQP_SLICE_V) (variable size)

The syntax element DQP_SLICE_V shall specify the differential of the quantizer parameter for the Cr channel of the entire slice. Additionally, this syntax element DQP_SLICE_V shall be decoded using exponential-Golomb codes and this decoded value shall be used to generate the slice-level quantizer parameter for the Cr channel. This generated parameter shall be used as the default scale in inverse quantization for the Cr channel when QP_CHANNEL_UNIFORM == 0. The details are described in Section 9.1.2.1

8.3 Macroblock-level Syntax and Semantics

Table 16 defines the bitstream syntax of the progressive macroblock layer.

Table 16 – Progressive macroblock layer bitstream

MB LAYER( ) {	Number of bits	Descriptor	Reference
if(is_beginning _of_slice() || is_previousMB_NotSkip()) {			
do {			
SKIP_RUN	variable size	avlclbf	8.3.1
}while (SKIP_RUN == 4)			
}			
if((is_not_skipMB() && QUALITY_REFINEMENT_FLAG == 0) ||			
(is_not_skipMB() && QUALITY_LAYER_ID == 0 &&			
QUALITY_REFINEMENT_FLAG == 1)) {			
if(MC_FLAG == 1) {			
MB_SIG_MODE	variable size	avlclbf	8.3.2
if(MB_SIG_MODE == 4) {			
MB_SIG_4	2	uimsbf	8.3.3
If(MB_SIG_4 == 3) {			
if(is_UpperMB_Inter() || is_LeftMB_Inter()){			
MB_MODE_1	1	uimsbf	8.3.4
}			
}			
}			
if(B_PICTURE_FLAG && ExistMV()) {			
B_PRED_MODE	2	uimsbf	8.3.5
}			
}			
else {			
if(PIC_INTRA_PRED_FLAG == 1) {			
if(!is_UpperMB_Skip() || !is_LeftMB_Skip()) {			
IPRED_MODE	variable size	uimsbf or avlclbf	8.3.6
}			
}			
INTRA_SIG	variable size	uimsbf or avlclbf	8.3.7
}			
for(i=0; i < number of MVs;i++) {			
MVDx[i]	variable size	evlclbf	8.3.8
MVDy[i]	variable size	evlclbf	8.3.9
if(B_PICTURE_FLAG && (B_PRED_MODE == 3)) {			
MVD2x[i]	variable size	evlclbf	8.3.10
MVD2y[i]	variable size	evlclbf	8.3.11
}			
}			
}			
if(ExistCodedCoeff()) {			
if(Exist_Y_Channel() && Exist_UV_Channel()){			
YUV_CSP	variable size	avlclbf	8.3.12
}			
if(Exist_Y_Coeff()) {			
Y_CSP	variable size	avlclbf	8.3.13
if(Exist_Y0Y1_Coeff()) {			
Y0Y1_CSP	variable size	avlclbf	8.3.14

}			
if(Exist_Y2Y3_Coeff()) {			
Y2Y3_CSP	variable size	avlclbf	8.3.15
}			
}			
if(Exist_UV_Coeff()) {			
if(Exist_U_Channel() && Exist_V_Channel()) {			
UV_CSP	variable size	avlclbf	8.3.16
}			
if(is422()) {			
if(Exist_U_Coeff()) {			
U0U1_CSP	variable size	avlclbf	8.3.17
}			
if(Exist_V_Coeff()) {			
V0V1_CSP	variable size	avlclbf	8.3.18
}			
}			
if(is444()) {			
if(Exist_U_Coeff()) {			
U_CSP	variable size	avlclbf	8.3.19
if(Exist_U0U1_Coeff()) {			
U0U1_CSP	variable size	avlclbf	8.3.17
}			
If(Exist_U2U3_Coeff()) {			
U2U3_CSP	variable size	avlclbf	8.3.20
}			
}			
if(Exist_V_Coeff()) {			
V_CSP	variable size	avlclbf	8.3.21
if(Exist_V0V1_Coeff()) {			
V0V1_CSP	variable size	avlclbf	8.3.18
}			
If(Exist_V2V3_Coeff()){			
V2V3_CSP	variable size	avlclbf	8.3.22
}			
}			
}			
}			
if(QP_PICTURE_UNIFORM_FLAG == 0) {			
DQP_MB	variable size	evlclbf	8.3.23
}			
for('all coded blocks in MB') {			
BLOCK LAYER()			Table 34
}			
}			

8.3.1 Skipped-macroblock Run (SKIP_RUN) (variable size)

The syntax element SKIP_RUN shall specify how many macroblocks are skipped, and shall be as defined in Table 17. This value shall be from zero to four. SKIP_RUN shall be decoded using adaptive VLC code. If the number of skipped macroblocks is greater than four, SKIP_RUN shall be repeated until all skipped macroblocks are counted. The skipped macroblocks described by SKIP_RUNs shall not be decoded. The decoding SKIP_RUN is explained in Section 9.1.3.1 and Section 9.2.3.1.

Table 17 – Meaning of SKIP_RUN

Decoded value of SKIP_RUN	Number of Skipped Macroblocks
0	0
1	1
2	2
3	3
4	4

8.3.2 MB-level Coded Significant Pattern and MB Mode (MB_SIG_MODE) (variable size)

The syntax element MB_SIG_MODE shall be jointly defined for both the macroblock-level coded significant pattern (CSP) and the macroblock mode. The decoded MB_SIG_MODE value shall be from zero to four, and shall be as define in Table 18. The meaning of '1' in the "MB-level CSP" column of Table 18 is that any magnitude among all quantized coefficients within a macroblock is not greater than one, and one or more quantized coefficients in the macroblock are not zero. The meaning of '2' is that a macroblock includes one or more quantized coefficients whose magnitudes are greater than one. In the MB coding mode column, INTRA means the intra-coded macroblock and 1-MV means the inter-predicted macroblock with one motion vector except for the interpolated mode in the B picture. When the current macroblock is coded as the interpolated mode in the B picture, 1-MV means the inter-predicted macroblock with 2 motion vectors, one of which is from the past picture and the other from the future picture. The procedure for decoding MB_SIG_MODE is described at Figure 29, Figure 33 and Figure 34 in Section 9.3.

Table 18 – Meaning of MB_SIG_MODE

Decoded value of MB_SIG_MODE	MB-level CSP	MB Coding Mode
0	1	INTRA
1	1	1-MV
2	2	INTRA
3	2	1-MV
4	Defined with MB_SIG_4 in Section 8.3.3	

8.3.3 MB-level Coded Significant Pattern (MB_SIG_4) (2 bits)

The syntax element MB_SIG_4 shall specify the macroblock-level coded significant pattern (CSP) when a macroblock is coded with four motion vectors or when a macroblock has no significant coefficient, and shall be as defined in Table 19. 4-MV means the inter-predicted macroblock with 4 motion vectors except for the interpolated mode in the B picture. When the current macroblock is coded as the interpolated mode in the B picture, 4-MV means the inter-predicted macroblock with 8 motion vectors, 4 of which are from the past picture and the rest 4 vectors from the future picture. In the column "MB-level CSP," '1' and '2' have the same meaning as MB_SIG_MODE, and '0' means a macroblock has no significant coefficients. When MB_SIG_4 is '11b' and the above macroblock or the left macroblock of the current macroblock is coded as inter mode, the MB-level CSP of the macroblock shall be zero and the macroblock coding mode shall be determined by MB_MODE_1 flag in Section 8.3.4. However, when MB_SIG_4 is '11b' and the left macroblock and the above macroblock of the current macroblock are coded as intra mode or are outside the coded picture area, the MB-level CSP of the macroblock shall be zero and the current macroblock shall be coded as inter mode with one motion vector.

Table 19 – Meaning of MB_SIG_4

MB_SIG_4	MB-level CSP	MB Coding Mode
0	0	4-MV
1	1	4-MV
2	2	4-MV
3	0	INTRA or 1-MV

8.3.4 MB Mode in Non-significant Macroblock (MB_MODE_1) (1 bit)

The syntax element MB_MODE_1 shall specify the macroblock coding mode when the current macroblock has no significant quantized coefficients. '0' indicates the macroblock is coded in INTRA mode, while '1' indicates the macroblock is coded in 1-MV mode. Here, the meaning of 1-MV shall be the same as MB_SIG_MODE.

8.3.5 Prediction Mode for B picture (B_PRED_MODE) (2 bits)

The syntax element B_PRED_MODE shall specify the prediction mode when the current picture is coded with one or more motion vectors in the B picture. B_PRED_MODE shall be coded according to Table 20. In the direct mode (when B_PRED_MODE == DIRECT_MODE), the prediction for the current macroblock shall be obtained using motion vectors of neighboring blocks/macroblocks without motion vectors of the current macroblock. The details are described in Section 9.2.3.7.

In the forward mode (when B_PRED_MODE == FORWARD_MODE) or the backward mode (when B_PRED_MODE == BACKWARD_MODE), the prediction for the current macroblock shall be obtained in the same way as the P picture coding. This mode also supports the motion vector scaling in Section 10.3.12 and Section 11.2.3.1 when the current picture is coded as a field picture.

In the interpolated mode (when B_PRED_MODE == INTERPOLATED_MODE), the forward and the backward motion vectors shall be explicitly coded within the bitstream and the prediction for the current macroblock shall be interpolated from the two reference pictures. The interpolation operation in this mode shall be pixel-wise average operation with the two referenced pictures. If the current macroblock is coded with 1-MV mode, 2 motion vectors, each of which points to the macroblocks each reference frame, shall be used for interpolation. If the current macroblock is coded with 4-MV mode, 8 motion vectors shall be used and each 4 motion vectors points to the 4 blocks of each reference frame. The details are described in Section 9.2.3.8.

Table 20 – Meaning of B_PRED_MODE

B_PRED_MODE	MB Prediction Mode
0	DIRECT_MODE
1	FORWARD_MODE
2	BACKWARD_MODE
3	INTERPOLATED_MODE

8.3.6 Intra Prediction Mode (IPRED_MODE) (variable size)

The syntax element IPRED_MODE shall specify the intra prediction mode for both luma and color difference in the macroblock. IPRED_MODE shall be coded according to Table 21. If only one of the upper and the left macroblock of current macroblock is coded as the skip mode, IPRED_MODE shall be decoded as 1-bit flag. If both the upper and left macroblocks of the current macroblock are not coded as the skip mode, IPRED_MODE shall be decoded using the adaptive VLC code with three alphabets described at Figure 28, Figure 33 and Figure 34 in Section 9.3. In the intra prediction mode column, DEFAULT means that the intra prediction uses the fixed value defined in Section 12.2. HORZ and VERT mean that the intra prediction is

performed in the horizontal direction and the vertical direction, respectively. When IPRED_MODE is not present (PIC_INTRA_PRED_FLAG is equal to 0 or both the upper and left macroblocks of the current macroblock are coded as skip mode), it shall be inferred to be DEFAULT mode. If QUALITY_LAYER_ID is greater than 0, IPRED_MODE shall not be present and intra prediction shall not be invoked. The detailed intra prediction process is described in Section 9.1.3.2.

Table 21 – Meaning of IPRED_MODE

Condition	Coding Method	IPRED_MODE	Intra Prediction Mode
is_UpperMB_Skip() == 1 && is_LeftMB_Skip() == 0	1-bit Fixed Length Coding	0	DEFAULT
		1	HORZ
is_UpperMB_Skip() == 0 && is_LeftMB_Skip() == 1	1-bit Fixed Length Coding	0	DEFAULT
		1	VERT
is_UpperMB_Skip() == 0 && is_LeftMB_Skip() == 0	Adaptive VLC Coding	0	DEFAULT
		1	HORZ
		2	VERT
is_UpperMB_Skip() == 1 && is_LeftMB_Skip() == 1	N/A	N/A	DEFAULT

8.3.7 MB-level Coded Significant Pattern in Intra Picture (INTRA_SIG) (variable size)

The syntax element INTRA_SIG shall specify the macroblock-level coded significant pattern (CSP) when the current-coded picture is the intra picture. INTRA_SIG shall be coded according to IPRED_MODE as specified in section 8.3.6. If IPRED_MODE == DEFAULT, INTRA_SIG shall be decoded as 1-bit flag. Otherwise (IPRED_MODE == HORZ or IPRED_MODE == VERT), INTRA_SIG shall be decoded using the adaptive VLC code with three alphabets described at Figure 28, Figure 33 and Figure 34 in Section 9.3. The following table describes the meaning of INTRA_SIG. In the column “MB-level CSP,” ‘0’, ‘1’ and ‘2’ have the same meaning as MB_SIG_4.

Table 22 – Meaning of INTRA_SIG

Condition	Coding Method	INTRA_SIG	MB-level CSP
IPRED_MODE == DEFAULT	1-bit Fixed Length Coding	0	1
		1	2
IPRED_MODE == HORZ or IPRED_MODE == VERT	Adaptive VLC Coding	0	0
		1	1
		2	2

8.3.8 Motion Vector Differential in X-axis (MVDx) (variable size)

The syntax element MVDx shall specify the motion vector differential in the horizontal direction. This syntax shall be used for the forward motion vector when the past frame is selected as the reference frame in the P picture coding mode or FORWARD is selected in the B picture coding mode, while this syntax shall be used for the backward motion vector when the future frame is selected as the reference frame in the P picture coding mode or BACKWARD is selected in the B picture coding mode. Additionally, this syntax shall be used to describe the backward motion vector in the interpolated mode of B picture. MVDx shall be decoded using exponential-Golomb codes as described in Section 9.2.3.3. The range of the MVDx shall be specified by limitations on MV [H] in Annex D.2.

8.3.9 Motion Vector Differential in Y-axis (MVDy) (variable size)

The syntax element MVDy shall specify the motion vector differential in the vertical direction. This syntax shall be used for the forward motion vector when the past frame is selected as the reference frame in the P picture coding mode or FORWARD is selected in the B picture coding mode, while this syntax shall be used for the backward motion vector when the future frame is selected as the reference frame in the P picture coding mode or BACKWARD is selected in the B picture coding mode. Additionally, this syntax shall be used to describe the backward motion vector in the interpolated mode of B picture. MVDy shall be decoded using exponential-Golomb codes as described in Section 9.2.3.3. The range of the MVDy shall be specified by limitations on MV [V] in Annex D.2.

8.3.10 Motion Vector Differential of X-axis for Interpolated Mode (MVD2x) (variable size)

The syntax element MVD2x shall specify the motion vector differential in the horizontal direction for the interpolated mode. In the interpolated mode, MVDx is for the forward vector while MVD2x is for the backward vector. MVD2x shall be decoded using exponential-Golomb codes as described in Section 9.2.3.3. The range of the MVD2x shall be specified by limitations on MV [H] in Annex D.2

8.3.11 Motion Vector Differential of Y-axis for Interpolated Mode (MVD2y) (variable size)

The syntax element MVD2y shall specify the motion vector differential in the vertical direction for the interpolated mode. In the interpolated mode, MVDy is for the forward motion vector while MVD2y is for the backward vector. MVD2y shall be decoded using exponential-Golomb codes as described in Section 9.2.3.3. The range of the MVD2y shall be specified by limitations on MV [V] in section D.2.

8.3.12 Coded Significant Pattern in Macroblock (YUV_CSP) (variable size)

The syntax element YUV_CSP shall be jointly coded with two values: the coded significant pattern of the luma blocks and the coded significant pattern of color difference blocks, and shall be as defined in Table 23 YUV_CSP describes the existence of significant quantized coefficients and types of significant coefficients in the luma blocks of the macroblock. Also YUV_CSP does the same thing for the color difference blocks of the macroblock. MB-level CSP shall be obtained from the previous other syntaxes such as MB_SIG_MODE, MB_SIG_4 and INTRA_SIG. The luma's CSP and the color difference's CSP describe CSP of each color component. The meaning of CSP in Table 23 is the same as MB_SIG_MODE or MB_SIG_4. If MB-level CSP is zero, this YUV_CSP shall not be decoded and the luma's CSP and the color difference's CSP shall be set to zero implicitly. If MB-level CSP is one, both luma's CSP and color difference's CSP shall be extracted from the decoded value of YUV_CSP using equation (8-1). If MB-level CSP is two, both luma's CSP and color difference's CSP shall be extracted using equation (8-2). The procedure for decoding YUV_CSP is described in Section 9.3. This syntax shall be omitted when the Y channel is skipped or when all color difference channels are skipped by CHANNEL_SKIP_MODE.

$$\begin{aligned} \text{Luma's CSP} &= (\text{YUV\_CSP} + 1) \,\&\, 0\text{x}01 \\ \text{Color difference's CSP} &= (\text{YUV\_CSP} + 1) >> 2 \end{aligned} \tag{8-1}$$

$$\begin{aligned} \text{Luma's CSP} &= \text{YUV\_CSP} \,\&\, 0\text{x}03 \\ \text{Color difference's CSP} &= (\text{YUV\_CSP} \,\&\, 0\text{x}0\text{C}) >> 2 \end{aligned} \tag{8-2}$$

Table 23 – Meaning of YUV_CSP

Decoded value of YUV_CSP	MB-level CSP	Luma's CSP	Color Difference's CSP
N/A	0	0	0
0	1	1	0
1		0	1
2		1	1
10 (1010b)	2	2	2
9 (1001b)		1	2
8 (1000b)		0	2
6 (0110b)		2	1
2 (0010b)		2	0

8.3.13 Coded Significant Pattern in Luma Blocks (Y_CSP) (variable size)

The syntax element Y_CSP shall be jointly coded with two values: the coded significant patterns of the upper luma blocks and the lower luma blocks. Y_CSP describes the existence of significant quantized coefficients and types of significant coefficients in the upper luma blocks of the macroblock. Also Y_CSP does the same thing for the lower luma blocks of the macroblock. The following table describes the meaning of Y_CSP. In the following table, the luma's CSP shall be obtained from YUV_CSP in Section 8.3.12. The upper luma blocks' CSP and the lower blocks' CSP describe CSP of each luma block row. The meaning of CSP in Table 24 is the same as YUV_CSP. If the luma's CSP of YUV_CSP is zero, this Y_CSP shall not be decoded and the upper luma blocks' CSP and the lower luma blocks' CSP shall be set to zero implicitly. If the luma's CSP of YUV_CSP is one, how to extract both upper luma blocks' CSP and lower luma blocks' CSP from the decoded value of Y_CSP is described in equation (8-3). If the luma's CSP of YUV_CSP is two, how to extract both upper luma blocks' CSP and lower luma blocks' CSP is described in equation (8-4). The procedure for decoding Y_CSP is described in Section 9.3. This syntax shall be omitted when the Y channel is skipped by CHANNEL_SKIP_MODE.

$$\begin{aligned} \text{Upper Luma Blocks' CSP} &= (\text{Y\_CSP} + 1)\ \&\ 0\text{x}01 \\ \text{Lower Luma Blocks' CSP} &= (\text{Y\_CSP} + 1) >> 2 \end{aligned} \tag{8-3}$$

$$\begin{aligned} \text{Upper Luma Blocks' CSP} &= \text{Y\_CSP}\ \&\ 0\text{x}03 \\ \text{Lower Luma Blocks' CSP} &= (\text{Y\_CSP}\ \&\ 0\text{x}0\text{C}) >> 2 \end{aligned} \tag{8-4}$$

Table 24 – Meaning of Y_CSP

Decoded value of Y_CSP	Luma's CSP of YUV_CSP	Upper Luma Blocks' CSP	Lower Luma Blocks' CSP
N/A	0	0	0
0	1	1	0
1		0	1
2		1	1
10 (1010b)	2	2	2
9 (1001b)		1	2
8 (1000b)		0	2
6 (0110b)		2	1
2 (0010b)		2	0

If all color difference channels are skipped, MB-level CSP which is obtained from MB_SIG_MODE, MB_SIG_4 and INTRA_SIG, shall be considered as luma's CSP of YUV_CSP in Table 24 and Y_CSP shall be decoded with MB-level CSP.

8.3.14 Coded Significant Pattern in Luma Upper Blocks (Y0Y1_CSP) (variable size)

The syntax element Y0Y1_CSP shall be jointly coded with two values: the coded significant patterns of the first and second luma blocks, and shall be coded as defined in Table 25. Y0Y1_CSP describes the existence of significant quantized coefficients and types of significant coefficients in the first luma block of the macroblock. Also Y0Y1_CSP does the same thing for the second luma block of the macroblock. The upper luma blocks' CSP shall be obtained from Y_CSP in Section 8.3.13. The first luma block's CSP and the second block's CSP describe CSP of each luma block. The meaning of CSP in Table 25 is the same as YUV_CSP. If the upper luma blocks' CSP of Y_CSP is zero, this Y0Y1_CSP shall not be decoded and the first luma block's CSP and the second luma block's CSP shall be set to zero implicitly. If Y0Y1_CSP is one, both the first luma block's CSP and the second luma block's CSP shall be obtained from equation (8-5). If Y0Y1_CSP is two, both the first luma block's CSP and the second luma block's CSP shall be obtained from equation (8-6). The procedure for decoding Y0Y1_CSP is described in Section 9.3. This syntax shall be omitted when the Y channel is skipped by CHANNEL_SKIP_MODE.

$$\text{The first Luma Block's CSP} = (\text{Y0Y1\_CSP} + 1)\ \&\ 0\text{x}01$$
$$\text{The second Luma Block's CSP} = (\text{Y0Y1\_CSP} + 1) >> 2 \quad (8\text{-}5)$$

$$\text{The first Luma Block's CSP} = \text{Y0Y1\_CSP}\ \&\ 0\text{x}03$$
$$\text{The second Luma Block's CSP} = (\text{Y0Y1\_CSP}\ \&\ 0\text{x}0\text{C}) >> 2 \quad (8\text{-}6)$$

Table 25 – Meaning of Y0Y1_CSP

Decoded value of Y0Y1_CSP	Upper Luma blocks' CSP of Y_CSP	The first Luma Block's CSP	The second Luma Block's CSP
N/A	0	0	0
0	1	1	0
1		0	1
2		1	1
10 (1010b)	2	2	2
9 (1001b)		1	2
8 (1000b)		0	2
6 (0110b)		2	1
2 (0010b)		2	0

8.3.15 Coded Significant Pattern in Luma Lower Blocks (Y2Y3_CSP) (variable size)

The syntax element Y2Y3_CSP shall be jointly coded with two values: the coded significant patterns of the third and fourth luma blocks, and shall be coded as defined in Table 26. Y2Y3_CSP describes the existence of significant quantized coefficients and types of significant coefficients in the third luma block of the macroblock. Also Y2Y3_CSP does the same thing for the fourth luma block of the macroblock. The lower luma blocks' CSP shall be obtained from Y_CSP in Section 8.3.13. The third luma block's CSP and the fourth block's CSP describe CSP of each luma block. The meaning of CSP in Table 26 is the same as YUV_CSP. If the lower luma blocks' CSP of Y_CSP is zero, this Y2Y3_CSP shall not be decoded and the third luma block's CSP and the fourth luma block's CSP shall be set to zero. If the lower luma blocks' CSP of Y_CSP is one, both the third luma block's CSP and the fourth luma block's CSP shall be extracted from the decoded value of Y2Y3_CSP using equation (8-7). If the lower luma blocks' CSP of Y_CSP is two, both the third luma block's CSP and the fourth luma block's CSP shall be extracted using equation (8-8). The procedure for decoding Y2Y3_CSP is described in Section 9.3. This syntax shall be omitted when the Y channel is skipped by CHANNEL_SKIP_MODE.

$$\text{The third Luma Block's CSP} = (\text{Y2Y3\_CSP} + 1)\ \&\ 0\text{x}01$$
$$\text{The fourth Luma Block's CSP} = (\text{Y2Y3\_CSP} + 1) >> 2 \quad (8\text{-}7)$$

$$\text{The third Luma Block's CSP} = \text{Y2Y3\_CSP}\ \&\ 0\text{x}03$$
$$\text{The fourth Luma Block's CSP} = (\text{Y2Y3\_CSP}\ \&\ 0\text{x}0\text{C}) >> 2 \quad (8\text{-}8)$$

Table 26 – Meaning of Y2Y3_CSP

Decoded value of Y2Y3_CSP	Lower Luma blocks' CSP of Y_CSP	The third Luma Block's CSP	The fourth Luma Block's CSP
N/A	0	0	0
0	1	1	0
1		0	1
2		1	1
10 (1010b)	2	2	2
9 (1001b)		1	2
8 (1000b)		0	2
6 (0110b)		2	1
2 (0010b)		2	0

8.3.16 Coded Significant Pattern in Color Difference Blocks (UV_CSP) (variable size)

The syntax element UV_CSP shall be jointly coded with two values: the coded significant patterns of the Cb color difference component and the Cr color difference component, and shall be coded as defined in Table 27. UV_CSP describes the existence of significant quantized coefficients and types of significant coefficients in the Cb color difference blocks of the macroblock. Also UV_CSP does the same thing for the Cr color difference blocks of the macroblock. The color difference's CSP shall be obtained from YUV_CSP in Section 8.3.12. The Cb color difference's CSP and the Cr color difference's CSP describe CSP of each color difference component. The meaning of CSP in Table 27 is the same as YUV_CSP. If the color difference's CSP of YUV_CSP is zero, this UV_CSP shall not be decoded and the Cb color difference's CSP and the Cr color difference's CSP shall be set to zero. If the color difference's CSP of YUV_CSP is one, both the Cb color difference's CSP and the Cr color difference's CSP shall be obtained from the decoded value of UV_CSP like as equation (8-9). If the color difference's CSP of YUV_CSP is two, both the Cb color difference's CSP and the Cr color difference's CSP shall be obtained using equation (8-10). The procedure for decoding UV_CSP is described in Section 9.3.

Since the each color difference component consists of only one single block when the color difference format is 4:2:0, additional CSPs for color difference components shall not be required. However, when the color difference format is 4:2:2 or 4:4:4, additional CSPs for color difference components shall be required because each color difference component consists of two blocks or four blocks.

This syntax shall be omitted when at least one channel between Cb and Cr channels is skipped by CHANNEL_SKIP_MODE.

$$\text{Cb Color Difference's CSP} = (\text{UV\_CSP} + 1)\ \&\ 0\text{x}01$$
$$\text{Cr Color Difference's CSP} = (\text{UV\_CSP} + 1) >> 2 \quad (8\text{-}9)$$

$$\text{Cb Color Difference's CSP} = \text{UV\_CSP}\ \&\ 0\text{x}03$$
$$\text{Cr Color Difference's CSP} = (\text{UV\_CSP}\ \&\ 0\text{x}0\text{C}) >> 2 \quad (8\text{-}10)$$

Table 27 – Meaning of UV_CSP

Decoded value of UV_CSP	Color Difference's CSP of YUV_CSP	Cb Color Difference's CSP	Cr Color Difference's CSP
N/A	0	0	0
0	1	1	0
1		0	1
2		1	1
10 (1010b)	2	2	2
9 (1001b)		1	2
8 (1000b)		0	2
6 (0110b)		2	1
2 (0010b)		2	0

If the Y channel is skipped, MB-level CSP, which is obtained from MB_SIG_MODE, MB_SIG_4 and INTRA_SIG, shall be considered as color difference's CSP of YUV_CSP in Table 27 and UV_CSP shall be decoded with MB-level CSP.

If one between the Cb channel and Cr channel is skipped and the Y channel is not skipped, UV_CSP shall be skipped, and color difference's CSP of YUV_CSP in Table 23 or Table 27 shall specify Cb color difference's CSP or Cr color difference's CSP, not skipped channel, in Table 27.

If one between the Cb channel and Cr channel is skipped and the Y channel is skipped, YUV_CSP, Y_CSP, Y0Y1_CSP, Y2Y3_CSP and UV_CSP shall be skipped, and MB-level CSP shall specify Cb color difference's CSP or Cr color difference's CSP, not skipped channel, in Table 27. Therefore, we need only U0U1_CSP or V0V1_CSP when the color difference format is 4:2:2. For the 4:4:4 color difference format, we need U_CSP, U0U1_CSP and U2U3_CSP when the Y channel and the Cr channel are skipped. V_CSP, V0V1_CSP and V2V3_CSP shall be needed when the Y channel and the Cb channel are skipped for the 4:4:4 color difference format.

8.3.17 Coded Significant Pattern in Color Difference Blocks for Cb Channel (U0U1_CSP) (variable size)

The syntax element U0U1_CSP shall be jointly coded with two values: the coded significant patterns of the first and second block of Cb color difference blocks when the color difference format is 4:2:2 or 4:4:4, and shall be coded as defined in Table 28. U0U1_CSP describes the existence of significant quantized coefficients and types of significant coefficients in the first block of the Cb color difference blocks. Also U0U1_CSP does the same thing for the second block of the Cb color difference blocks. The Cb color difference's CSP shall be obtained from UV_CSP in section 8.3.16 when the color difference format is 4:2:2, or from U_CSP in Section 8.3.19 when the color difference format is 4:4:4. The first and second Cb color difference blocks' CSPs describe CSP of each Cb block. The meaning of CSP in Table 28 is the same as YUV_CSP. If the Cb color difference's CSP of UV_CSP is zero when the color difference format is 4:2:2, this U0U1_CSP shall not be decoded and the first and second Cb color difference blocks' CSPs shall be set to zero. If the upper Cb blocks' CSP of U_CSP in Section 8.3.19 is zero when the color difference format is 4:4:4, this U0U1_CSP shall not be decoded and the first and second Cb color difference blocks' CSPs shall be set to zero. If the Cb color difference's CSP of UV_CSP (4:2:2) or U_CSP (4;4:4) is one, both the first and second Cb color difference blocks' CSPs shall be obtained from the decoded value of U0U1_CSP like as equation (8-11). If Cb color difference's CSP of UV_CSP (4:2:2) or U_CSP (4;4:4) is two, both the first and second Cb color difference blocks' CSPs shall be obtained using equation (8-12). The procedure for decoding U0U1_CSP is described in Section 9.3. This syntax shall be omitted when the Cb channel is skipped by CHANNEL_SKIP_MODE.

$$\begin{aligned} \text{The First Cb Block's CSP} &= (\text{U0U1\_CSP} + 1)\ \&\ 0\text{x}01 \\ \text{The Second Cb Block's CSP} &= (\text{U0U1\_CSP} + 1) >> 2 \end{aligned} \tag{8-11}$$

$$\text{The First Cb Block's CSP} = \text{U0U1\_CSP} \,\&\, 0x03$$
$$\text{The Second Cb Block's CSP} = (\text{U0U1\_CSP} \,\&\, 0x0C) >> 2 \quad (8\text{-}12)$$

Table 28 – Meaning of U0U1_CSP

Decoded value of U0U1_CSP	Cb Color Difference's CSP of UV_CSP (4:2:2) or U_CSP (4;4:4)	The First Cb Block's CSP	The Second Cb Block's CSP
N/A	0	0	0
0	1	1	0
1		0	1
2		1	1
10 (1010b)	2	2	2
9 (1001b)		1	2
8 (1000b)		0	2
6 (0110b)		2	1
2 (0010b)		2	0

8.3.18 Coded Significant Pattern in Color Difference Blocks for Cr Channel (V0V1_CSP) (variable size)

The syntax element V0V1_CSP shall be jointly coded with two values: the coded significant patterns of the first and second block of Cr color difference blocks when the color difference format is 4:2:2 or 4:4:4, and shall be coded as defined in Table 29. V0V1_CSP describes the existence of significant quantized coefficients and types of significant coefficients in the first block of the Cr color difference blocks. Also V0V1_CSP does the same thing for the second block of the Cr color difference blocks. The Cr color difference's CSP shall be obtained from UV_CSP in Section 8.3.16 when the color difference format is 4:2:2, or from V_CSP in Section 8.3.21 when the color difference format is 4:4:4. The first and second Cr color difference blocks' CSPs describe CSP of each Cr block. The meaning of CSP in Table 29 is the same as YUV_CSP. If the Cr color difference's CSP of UV_CSP is zero when the color difference format is 4:2:2, this V0V1_CSP shall not be decoded and the first and second Cr color difference blocks' CSPs shall be set to zero. If the upper Cr blocks' CSP of V_CSP in Section 8.3.21 is zero when the color difference format is 4:4:4, this U0U1_CSP shall not be decoded and the first and second Cr color difference blocks' CSPs shall be set to zero. If the Cr color difference's CSP of UV_CSP (4:2:2) or V_CSP (4:4:4) is one, both the first and second Cr color difference blocks' CSPs shall be obtained from the decoded value of V0V1_CSP using equation (8-13). If the Cr color difference's CSP of UV_CSP (4:2:2) or V_CSP (4:4:4) is two, both the first and second Cr color difference blocks' CSPs shall be obtained using equation (8-14). The procedure for decoding V0V1_CSP is described in Section 9.3. This syntax shall be omitted when the Cr channel is skipped by CHANNEL_SKIP_MODE.

$$\text{The First Cr Block's CSP} = (\text{V0V1\_CSP} + 1) \,\&\, 0x01$$
$$\text{The Second Cr Block's CSP} = (\text{V0V1\_CSP} + 1) >> 2 \quad (8\text{-}13)$$
$$\text{The First Cr Block's CSP} = \text{V0V1\_CSP} \,\&\, 0x03$$
$$\text{The Second Cr Block's CSP} = (\text{V0V1\_CSP} \,\&\, 0x0C) >> 2 \quad (8\text{-}14)$$

Table 29 – Meaning of V0V1_CSP

Decoded value of V0V1_CSP	Cr Color Difference's CSP of UV_CSP (4:2:2) or V_CSP (4:4:4)	The First Cr Block's CSP	The Second Cr Block's CSP
N/A	0	0	0
0	1	1	0
1		0	1
2		1	1
10 (1010b)	2	2	2
9 (1001b)		1	2
8 (1000b)		0	2
6 (0110b)		2	1
2 (0010b)		2	0

8.3.19 Coded Significant Pattern in Cb Channel (U_CSP) (variable size)

The syntax element U_CSP shall be jointly coded with two values: the coded significant patterns of the upper Cb blocks and the lower Cb blocks when the color difference format is 4:4:4, and shall be coded as defined in Table 30. U_CSP describes the existence of significant quantized coefficients and types of significant coefficients in the upper blocks of the Cb color difference blocks. Also U_CSP does the same thing for the lower blocks of the Cb color difference blocks. The Cb color difference's CSP shall be obtained from UV_CSP in Section 8.3.16. The upper Cb blocks' CSP and the lower Cb blocks' CSP describe CSP of each Cb block row. The meaning of CSP in Table 30 is the same as YUV_CSP. If the Cb color difference's CSP of UV_CSP is zero, this U_CSP shall not be decoded and the upper Cb blocks' CSP and the lower Cb blocks' CSP shall be set to zero. If the Cb color difference's CSP of UV_CSP is one, both the upper Cb blocks' CSP and the lower Cb blocks' CSP shall be obtained from the decoded value of U_CSP using equation (8-15). If the Cb color difference's CSP of UV_CSP is two, both the upper Cb blocks' CSP and the lower Cb blocks' CSP shall be obtained using equation (8-16). The procedure for decoding U_CSP is described in Section 9.3. This syntax shall be omitted when the Cb channel is skipped by CHANNEL_SKIP_MODE.

$$\begin{aligned} \text{Upper Cb Blocks' CSP} &= (\text{U\_CSP} + 1)\ \&\ 0\text{x}01 \\ \text{Lower Cb Blocks' CSP} &= (\text{U\_CSP} + 1) >> 2 \end{aligned} \tag{8-15}$$

$$\begin{aligned} \text{Upper Cb Blocks' CSP} &= \text{U\_CSP}\ \&\ 0\text{x}03 \\ \text{Lower Cb Blocks' CSP} &= (\text{U\_CSP}\ \&\ 0\text{x}0\text{C}) >> 2 \end{aligned} \tag{8-16}$$

Table 30 – Meaning of U_CSP

Decoded value of U_CSP	Cb Color Difference's CSP of UV_CSP	Upper Cb Blocks' CSP	Lower Cb Blocks' CSP
N/A	0	0	0
0	1	1	0
1		0	1
2		1	1
10 (1010b)	2	2	2
9 (1001b)		1	2
8 (1000b)		0	2
6 (0110b)		2	1
2 (0010b)		2	0

8.3.20 Coded Significant Pattern in Lower Blocks for Cb Channel (U2U3_CSP) (variable size)

The syntax element U2U3_CSP shall be jointly coded with two values: the coded significant patterns of the third and fourth block of Cb color difference blocks when the color difference format is 4:4:4, and shall be coded as defined in Table 31. U2U3_CSP describes the existence of significant quantized coefficients and types of significant coefficients in the third block of the Cb color difference blocks. Also U2U3_CSP does the same thing for the fourth block of the Cb color difference blocks. The lower Cb blocks' CSP shall be obtained from U_CSP in Section 8.3.19. The third and fourth Cb color difference blocks' CSPs describe CSP of each Cb block. The meaning of CSP in Table 31 is the same as YUV_CSP. If the lower Cb blocks' CSP of U_CSP is zero, this U2U3_CSP shall not be decoded and the third and fourth Cb color difference blocks' CSPs shall be set to zero. If the lower Cb blocks' CSP of U_CSP is one, both the third and fourth Cb color difference blocks' CSPs shall be obtained from the decoded value of U2U3_CSP like as equation (8-17). If the lower Cb blocks' CSP of U_CSP is two, both the third and fourth Cb color difference blocks' CSPs shall be obtained using equation (8-18). The procedure for decoding U2U3_CSP is described in Section 9.3. This syntax shall be omitted when the Cb channel is skipped by CHANNEL_SKIP_MODE.

$$\begin{aligned} &\text{The Third Cb Block's CSP} = (\text{U2U3\_CSP} + 1)\ \&\ 0\text{x}01 \\ &\text{The Fourth Block's CSP} = (\text{U2U3\_CSP} + 1) >> 2 \end{aligned} \tag{8-17}$$

$$\begin{aligned} &\text{The Third Cb Block's CSP} = \text{U2U3\_CSP}\ \&\ 0\text{x}03 \\ &\text{The Fourth Cb Block's CSP} = (\text{U2U3\_CSP}\ \&\ 0\text{x}0\text{C}) >> 2 \end{aligned} \tag{8-18}$$

Table 31 – Meaning of U2U3_CSP

Decoded value of U2U3_CSP	Lower Cb blocks' CSP of U_CSP	The third Cb Block's CSP	The fourth Cb Block's CSP
N/A	0	0	0
0	1	1	0
1		0	1
2		1	1
10 (1010b)	2	2	2
9 (1001b)		1	2
8 (1000b)		0	2
6 (0110b)		2	1
2 (0010b)		2	0

8.3.21 Coded Significant Pattern in Cr Channel (V_CSP) (variable size)

The syntax element V_CSP shall be jointly coded with two values: the coded significant patterns of the upper Cr blocks and the lower Cr blocks when the color difference format is 4:4:4, and shall be coded as defined in Table 32. V_CSP describes the existence of significant quantized coefficients and types of significant coefficients in the upper blocks of the Cr color difference blocks. Also V_CSP does the same thing for the lower blocks of the Cr color difference blocks. The Cr color difference's CSP shall be obtained from UV_CSP in Section 8.3.16. The upper Cr blocks' CSP and the lower Cr blocks' CSP describe CSP of each Cr block row. The meaning of CSP in Table 32 is the same as YUV_CSP. If the Cr color difference's CSP of UV_CSP is zero, this V_CSP shall not be decoded and the upper Cr blocks' CSP and the lower Cr blocks' CSP shall be set to zero. If the Cr Color difference's CSP of UV_CSP is one, both the upper Cr blocks' CSP and the lower Cr blocks' CSP shall be extracted from the decoded value of V_CSP using equation (8-19). If the Cr Color difference's CSP of UV_CSP is two, both the upper Cr blocks' CSP and the lower Cr blocks' CSP shall be extracted using equation (8-20). The procedure for decoding V_CSP is described in Section 9.3. This syntax shall be omitted when the Cr channel is skipped by CHANNEL_SKIP_MODE.

$$\text{Upper Cr Blocks' CSP} = (\text{V\_CSP} + 1)\ \&\ 0\text{x}01 \quad (8\text{-}19)$$

$$\text{Lower Cr Blocks' CSP} = (\text{V\_CSP} + 1) >> 2$$

$$\text{Upper Cr Blocks' CSP} = \text{V\_CSP}\ \&\ 0\text{x}03 \quad (8\text{-}20)$$

$$\text{Lower Cr Blocks' CSP} = (\text{V\_CSP}\ \&\ 0\text{x}0\text{C}) >> 2$$

Table 32 – Meaning of V_CSP

Decoded value of V_CSP	Cr Color Difference's CSP of UV_CSP	Upper Cr Blocks' CSP	Lower Cr Blocks' CSP
N/A	0	0	0
0	1	1	0
1		0	1
2		1	1
10 (1010b)	2	2	2
9 (1001b)		1	2
8 (1000b)		0	2
6 (0110b)		2	1
2 (0010b)		2	0

8.3.22 Coded Significant Pattern in Lower Blocks for Cr Channel (V2V3_CSP) (variable size)

The syntax element V2V3_CSP shall be jointly coded with two values: the coded significant patterns of the third and fourth block of Cr color difference blocks when the color difference format is 4:4:4, and shall be coded as defined in Table 33. V2V3_CSP describes the existence of significant quantized coefficients and types of significant coefficients in the third block of the Cr color difference blocks. Also V2V3_CSP does the same thing for the fourth block of the Cr color difference blocks. The lower Cr blocks' CSP shall be obtained from V_CSP in Section 8.3.21. The third and fourth Cr color difference blocks' CSPs describe CSP of each Cr block. The meaning of CSP in Table 33 is the same as YUV_CSP. If the lower Cr blocks' CSP of V_CSP is zero, this V2V3_CSP shall not be decoded and the third and fourth Cr color difference blocks' CSPs shall be set to zero. If the lower Cr blocks' CSP of V_CSP is one, both the third and fourth Cr color difference blocks' CSPs shall be obtained from the decoded value of V2V3_CSP using equation (8-21). If the Lower Cr blocks' CSP of V_CSP is two, both the third and fourth Cr color difference blocks' CSPs shall be obtained using equation (8-22).The procedure for decoding V2V3_CSP is described in Section 9.3. This syntax shall be omitted when the Cr channel is skipped by CHANNEL_SKIP_MODE.

$$\text{The Third Cr Block's CSP} = (\text{V2V3\_CSP} + 1)\ \&\ 0\text{x}01 \quad (8\text{-}21)$$

$$\text{The Fourth Cr Block's CSP} = (\text{V2V3\_CSP} + 1) >> 2$$

$$\text{The Third Cr Block's CSP} = \text{V2V3\_CSP}\ \&\ 0\text{x}03 \quad (8\text{-}22)$$

$$\text{The Fourth Cr Block's CSP} = (\text{V2V3\_CSP}\ \&\ 0\text{x}0\text{C}) >> 2$$

Table 33 – Meaning of V2V3_CSP

Decoded value of V2V3_CSP	Lower Cr blocks' CSP of V_CSP	The third Cr Block's CSP	The fourth Cr Block's CSP
N/A	0	0	0
0	1	1	0
1		0	1
2		1	1
10 (1010b)	2	2	2
9 (1001b)		1	2
8 (1000b)		0	2
6 (0110b)		2	1
2 (0010b)		2	0

8.3.23 Differential of MB-level Quantizer Parameter (DQP_MB) (variable size)

The syntax element DQP_MB shall specify the differential of the quantizer parameter for the current macroblock when QP_PICTURE_UNIFORM_FLAG == 0. Additionally, this syntax element DQP_MB shall be decoded using exponential-Golomb codes and this decoded value shall be used to generate the macroblock-level quantizer parameter. The details are described in Section 9.1.3.3.

8.4 Block-level Syntax and Semantics

Table 34, Table 35, Table 36, Table 37 and Table 38 define the bitstream syntax of the block layer. The block-level data shall be coded hierarchically. Table 35 shows the largest coefficient significant type (largest CST) of the related coefficients. These largest CSTs shall be represented by 0, 1, or 2. '0' means all levels of the related coefficients are zero. '1' means the maximum magnitude among the levels of the related coefficients is one. Finally, '2' means the maximum magnitude among levels of the related coefficients is greater than one. In Table 35, the number of "Range in effect" indicates the order of coefficients stored in the buffer before the scanning of Section 9.1.4.3 or Section 11.1.4.

Table 34 – Block layer bitstream

BLOCK LAYER( ) {	Number of bits	Descriptor	Reference
DC_AC_CST	variable size	avlclbf	8.4.1
if(DC_CST >= 1) {			Table 35
COEFF_LEVEL[0]	variable size	mevlclbf	8.4.16
}			
if(AC63 >= 1) {			Table 35
AC_CST	variable size	avlclbf	8.4.2
if(ACp1 >= 1) {			Table 35
ACp1_CST	variable size	avlclbf	8.4.3
CALL_COEFF_LEVEL(1, 4)	variable size	mevlclbf	Table 36
}			
if(ACp2 >= 1) {			Table 35
ACp2_CST	variable size	avlclbf	8.4.4
for(i=1; i<4; i++) {			
if(L1[i]==1) {			Table 35
L1[i]_CST	variable size	avlclbf	8.4.6
}			
else if(L1[i]==2) {			Table 35

CALL_2SYM_L1(i)	variable size	avlclbf	Table 37
}			
CALL_COEFF_LEVEL(i*4, i*4+4)	variable size	mevlclbf	Table 36
}			
}			
if(ACp3 >= 1) {			Table 35
ACp3_CST	variable size	avlclbf	8.4.5
if(L2[1] >= 1) {			Table 35
if(L2[1] == 1) {			Table 35
L2[1]_CST	variable size	avlclbf	8.4.7
}			
else {			
CALL_2SYM_L2(1)	variable size	avlclbf	Table 38
}			
for(i=4; i<8; i++) {			
if(L1[i]==1) {			Table 35
L1[i]_CST	variable size	avlclbf	8.4.6
}			
else if(L1[i]==2) {			Table 35
CALL_2SYM_L1(i)	variable size	avlclbf	Table 37
}			
CALL_COEFF_LEVEL(i*4, i*4+4)	variable size	mevlclbf	Table 36
}			
} // end of “if(L2[1] >= 1)”			
if(L2[2] >= 1) {			Table 35
if(L2[2] == 1) {			Table 35
L2[2]_CST	variable size	avlclbf	8.4.8
}			
else {			
CALL_2SYM_L2(2)	variable size	avlclbf	Table 38
}			
for(i=8; i<12; i++) {			
if(L1[i]==1) {			Table 35
L1[i]_CST	variable size	avlclbf	8.4.6
}			
else if(L1[i]==2) {			Table 35
CALL_2SYM_L1(i)	variable size	avlclbf	Table 37
}			
CALL_COEFF_LEVEL(i*4, i*4+4)	variable size	mevlclbf	Table 36
}			
} // end of “if(L2[2] >= 1)”			
if(L2[3] >= 1) {			Table 35
if(L2[3] == 1) {			Table 35
L2[3]_CST	variable size	avlclbf	8.4.9
}			
else {			
CALL_2SYM_L2(3)	variable size	avlclbf	Table 38
}			
for(i=12; i<16; i++) {			
if(L1[i]==1) {			Table 35
L1[i]_CST	variable size	avlclbf	8.4.6
}			
else if(L1[i]==2) {			Table 35

CALL_2SYM_L1(i)	variable size	avlclbf	Table 37
}			
CALL_COEFF_LEVEL(i*4, i*4+4)	variable size	mevlclbf	Table 36
}			
} // end of “if(L2[3] >= 1)”			
} // end of “if(ACp3 >= 1)”			
} // end of “if(AC63 >= 1)”			
} // end of “BLOCK LAYER()”			

Table 35 – The largest coefficient significant type (LCST) for 8x8 coefficients

LCST	Range in effect	The largest CST of coefficients corresponding to “Range in effect”
DC_CST	0	The CST of coefficient of DC
AC_CST[i]	Only one in 1~63	The CST of the i-th AC coefficient
AC63	1 ~ 63	The largest CST among all AC coefficients
ACp1	1~3	The largest CST of the first partial AC coefficients
ACp2	4~15	The largest CST of the second partial AC coefficients
ACp3	16~63	The largest CST of the third partial AC coefficients
L1[1]	4~7	The largest CST of the first partial coefficients of ACp2
L1[2]	8~11	The largest CST of the second partial coefficients of ACp2
L1[3]	12~15	The largest CST of the third partial coefficients of ACp2
L2[1]	16~31	The largest CST of the first partial coefficients of ACp3
L2[2]	32~47	The largest CST of the second partial coefficients of ACp3
L2[3]	48~63	The largest CST of the third partial coefficients of ACp3
L1[4]	16~19	The largest CST of the first partial coefficients of L2[1]
L1[5]	20~23	The largest CST of the second partial coefficients of L2[1]
L1[6]	24~27	The largest CST of the third partial coefficients of L2[1]
L1[7]	28~31	The largest CST of the fourth partial coefficients of L2[1]
L1[8]	32~35	The largest CST of the first partial coefficients of L2[2]
L1[9]	36~39	The largest CST of the second partial coefficients of L2[2]
L1[10]	40~43	The largest CST of the third partial coefficients of L2[2]
L1[11]	44~47	The largest CST of the fourth partial coefficients of L2[2]
L1[12]	48~51	The largest CST of the first partial coefficients of L2[3]
L1[13]	52~55	The largest CST of the second partial coefficients of L2[3]
L1[14]	56~59	The largest CST of the third partial coefficients of L2[3]
L1[15]	60~63	The largest CST of the fourth partial coefficients of L2[3]

Table 36 – Level coding of coefficients

CALL_COEFF_LEVEL (start, end) {	Number of bits	Descriptor	Reference
for(i=start; i<end; i++) {			
if(AC_CST[i] >= 1) {			Table 35
COEFF_LEVEL[i]	variable size	mevlclbf	8.4.16
}			
}			
}			

Table 37 – Symbol coding for two symbols in the coefficient position of layer 1

CALL_2SYM_L1(i) {	Number of bits	Descriptor	Reference
if(i_th_component_of_Layer1_Significant()) {			5.8
L1AC4_CST[i]	variable size	avlclbf	8.4.13
}			
if(Last_3_Coeffs_of_L1AC4_CST[i]_Significant()) {			5.8
L1AC3_CST[i]	variable size	avlclbf	8.4.14
}			
if(Last_2_Coeffs_of_L1AC4_CST[i]_Significant()) {			5.8
L1AC2_CST[i]	variable size	avlclbf	8.4.15
}			
}			

Table 38 – Symbol coding for two symbols in the coefficient position of layer 2

CALL_2SYM_L2(i) {	Number of bits	Descriptor	Reference
if(i_th_component_of_Layer2_Significant()) {			5.8
L2AC16_CST[i]	variable size	avlclbf	8.4.10
}			
if(Last_12_Coeffs_of_L2AC16_CST[i]_Significant()) {			5.8
L2AC12_CST[i]	variable size	avlclbf	8.4.11
}			
if(Last_8_Coeffs_of_L2AC12_CST[i]_Significant()) {			5.8
L2AC8_CST[i]	variable size	avlclbf	8.4.12
}			
}			

8.4.1 DC/AC Coefficient-Significant Types (DC_AC_CST) (variable size)

The syntax element DC_AC_CST shall specify the coefficient-significant types of DC and AC coefficients, and shall be as defined in Table 39. DC_CST and AC63 shall be obtained from the decoded value of DC_AC_CST using equation (8-23) for the case of the i^{th} block-level CSP being one, and using equation (8-24) for the case of the i^{th} block-level CSP being two. When the i^{th} block-level CSP is zero, both the DC_CST and the AC63 shall be zero. DC_CST and AC63 are explained in Table 35. The decoding method is the same as YUV_CSP. The coefficient-significant type (CST) of DC represented by DC_CST shall be from zero to two. ‘0’ means the DC level is zero, ‘1’ means the magnitude of DC level is one, and ‘2’ means the magnitude of DC level is greater than one. The coefficient-significant types (CSTs) of AC represented by AC63 also shall be from zero to two. In the case of AC cofficients, ‘0’ means all AC coefficient levels are zero, ‘1’ means any magnitude among all AC coefficient levels is not greater than one, and ‘2’ means the block has

more than one AC coefficient whose magnitude is greater than one. This syntax shall be decoded using adaptive VLC coding, and the luma blocks and the color difference blocks have different probability models. The i^{th} block-level CSP shall be obtained from the MB-level other syntaxes. The CSPs of luma blocks shall be from Y0Y1_CSP or Y2Y3_CSP. For color difference blocks, UV_CSP determines the CSPs of the corresponding color difference blocks when the color difference format is 4:2:0, while when the color difference format is 4:2:2, U0U1_CSP and V0V1_CSP determine the CSPs of Cb blocks and Cr blocks, respectively. The procedure for decoding DC_AC_CST is described in Section 9.3.

$$DC\_CST = (DC\_AC\_CST + 1) >> 1$$
$$AC63 = 1 - (DC\_AC\_CST \,\&\, 0x01) \quad (8\text{-}23)$$

$$DC\_CST = (DC\_AC\_CST \,\&\, 0x0C) >> 2$$
$$AC63 = DC\_AC\_CST \,\&\, 0x03 \quad (8\text{-}24)$$

Table 39 – Meaning of DC_AC_CST

Decoded value of DC_AC_CST	i^{th} Block-level CSP	DC_CST	AC63
0	1	0	1
1		1	0
2		1	1
10 (1010b)	2	2	2
9 (1001b)		2	1
8 (1000b)		2	0
6 (0110b)		1	2
2 (0010b)		0	2

8.4.2 AC Coefficient-Significant Types (AC_CST) (variable size)

The syntax element AC_CST shall specify the coefficient-significant types of 63 AC coefficients, and shall be as defined in Table 40. This syntax indicates coefficient-significant types of three parts of ACs: ACp1, ACp2, and ACp3 represented in Table 35. ACp1, ACp2, and ACp3 shall be obtained from the decoded value of AC_CST using equation (8-25). The coefficient-significant type shall be from zero to two. The meaning of each value is similar to the AC coefficient part of DC_AC_CST. '0' means the AC coefficients of the corresponding part have no significant coefficient. '1' means any magnitude among the AC coefficient levels of the corresponding part is not greater than one. '2' means the corresponding AC coefficient part has more than one AC coefficient level whose magnitude is greater than one. This syntax shall be decoded using the adaptive VLC coding, and the luma blocks and the color difference blocks have the same probability model. The AC63 shall be obtained from the syntax of DC_AC_CST. If AC63 is zero, this AC_CST shall not be decoded and the significant types of all AC coefficients shall be zero. The procedure for decoding AC_CST is described in Section 9.3.

$$ACp1 = (AC\_CST \,\&\, 0x30) >> 4$$
$$ACp2 = (AC\_CST \,\&\, 0x0C) >> 2 \quad (8\text{-}25)$$
$$ACp3 = AC\_CST \,\&\, 0x03$$

Table 40 – Meaning of AC_CST

Decoded value of AC_CST	AC63	ACp1	ACp2	ACp3
N/A	0	0	0	0
1	1	0	0	1
4	1	0	1	0
5	1	0	1	1
16	1	1	0	0
17	1	1	0	1
20	1	1	1	0
21	1	1	1	1
2	2	0	0	2
6	2	0	1	2
8	2	0	2	0
9	2	0	2	1
10	2	0	2	2
18	2	1	0	2
22	2	1	1	2
24	2	1	2	0
25	2	1	2	1
26	2	1	2	2
32	2	2	0	0
33	2	2	0	1
34	2	2	0	2
36	2	2	1	0
37	2	2	1	1
38	2	2	1	2
40	2	2	2	0
41	2	2	2	1
42	2	2	2	2

8.4.3 Significant Types of the 1st, 2nd, and 3rd AC Coefficients (ACp1_CST) (variable size)

The syntax element ACp1_CST shall specify the coefficient-significant types of the coefficients from AC[1] to AC[3], and shall be as defined in Table 41. Three CSTs of AC[1], AC[2], and AC[3] shall be obtained from the

decoded value of ACp1_CST using equation (8-26). The decoding method shall be the same as AC_CST. The coefficient-significant type shall be from zero to two, and each meaning is explained in Section 8.4.2. This syntax shall be decoded using the adaptive VLC coding, and the luma blocks and the color difference blocks have the same probability model. The ACp1 shall be obtained from the syntax of AC_CST. The procedure for decoding ACp1_CST is described in Section 9.3.

$$\begin{aligned} \text{CST of AC}[1] &= (\text{ACp1\_CST} \,\&\, 0\text{x}30) >> 4 \\ \text{CST of AC}[2] &= (\text{ACp1\_CST} \,\&\, 0\text{x}0\text{C}) >> 2 \\ \text{CST of AC}[3] &= \text{ACp1\_CST} \,\&\, 0\text{x}03 \end{aligned} \tag{8-26}$$

Table 41 – Meaning of ACp1_CST

Decoded value of ACp1_CST	ACp1	CST of AC[1]	CST of AC[2]	CST of AC[3]
N/A	0	0	0	0
1	1	0	0	1
4	1	0	1	0
5	1	0	1	1
16	1	1	0	0
17	1	1	0	1
20	1	1	1	0
21	1	1	1	1
2	2	0	0	2
6	2	0	1	2
8	2	0	2	0
9	2	0	2	1
10	2	0	2	2
18	2	1	0	2
22	2	1	1	2
24	2	1	2	0
25	2	1	2	1
26	2	1	2	2
32	2	2	0	0
33	2	2	0	1
34	2	2	0	2
36	2	2	1	0
37	2	2	1	1
38	2	2	1	2

40	2	2	2	0
41	2	2	2	1
42	2	2	2	2

8.4.4 Significant Types of 4th to 15th AC Coefficients (ACp2_CST) (variable size)

The syntax element ACp2_CST shall specify the coefficient-significant types of the coefficients from AC[4] to AC[15], and shall be as defined in Table 42. L1[1], L1[2], and L1[3] shall be obtained from the decoded value of ACp2_CST using equation (8-27). The decoding method shall be the same as AC_CST. The coefficient-significant type shall be from zero to two, and each meaning is explained in section 8.4.2. This syntax shall be decoded using the adaptive VLC coding, and the luma blocks and the color difference blocks have the same probability model. The ACp2 shall be obtained from the syntax of AC_CST. In Table 42, L1[1], L1[2] and L1[3] are explained in Table 35. The procedure for decoding ACp2_CST is described in Section 9.3.

$$\begin{aligned} L1[1] &= (ACp2\_CST \,\&\, 0x30) >> 4 \\ L1[2] &= (ACp2\_CST \,\&\, 0x0C) >> 2 \\ L1[3] &= ACp2\_CST \,\&\, 0x03 \end{aligned} \tag{8-27}$$

Table 42 – Meaning of ACp2_CST

Decoded value of ACp2_CST	ACp2	L1[1]	L1[2]	L1[3]
N/A	0	0	0	0
1	1	0	0	1
4	1	0	1	0
5	1	0	1	1
16	1	1	0	0
17	1	1	0	1
20	1	1	1	0
21	1	1	1	1
2	2	0	0	2
6	2	0	1	2
8	2	0	2	0
9	2	0	2	1
10	2	0	2	2
18	2	1	0	2
22	2	1	1	2
24	2	1	2	0
25	2	1	2	1

26	2	1	2	2
32	2	2	0	0
33	2	2	0	1
34	2	2	0	2
36	2	2	1	0
37	2	2	1	1
38	2	2	1	2
40	2	2	2	0
41	2	2	2	1
42	2	2	2	2

8.4.5 Significant Types of 16th to 63th AC Coefficients (ACp3_CST) (variable size)

The syntax element ACp3_CST shall specify the coefficient-significant types of the coefficients from AC[16] to AC[63], and shall be as defined in Table 43. Three symbols of L2[1], L2[2], and L2[3] shall be obtained from the decoded value of ACp3_CST using equation (8-28). The decoding method shall be the same as AC_CST. The coefficient-significant type shall be from zero to two and each meaning is explained in Section 8.4.2. This syntax shall be decoded using the adaptive VLC coding, and the luma blocks and the color difference blocks have the same probability model. The ACp3 shall be obtained from the syntax of AC_CST. In Table 43, L2[1], L2[2] and L2[3] are explained in Table 35 and Figure 11 of Section 9.1.4.1. The procedure for decoding ACp3_CST is described in Section 9.3.

$$\begin{aligned} L2[1] &= (ACp3\_CST \,\&\, 0x30) >> 4 \\ L2[2] &= (ACp3\_CST \,\&\, 0x0C) >> 2 \\ L2[3] &= ACp3\_CST \,\&\, 0x03 \end{aligned} \tag{8-28}$$

Table 43 – Meaning of ACp3_CST

Decoded value of ACp3_CST	ACp3	L2[1]	L2[2]	L2[3]
N/A	0	0	0	0
1	1	0	0	1
4	1	0	1	0
5	1	0	1	1
16	1	1	0	0
17	1	1	0	1
20	1	1	1	0
21	1	1	1	1
2	2	0	0	2

6	2	0	1	2
8	2	0	2	0
9	2	0	2	1
10	2	0	2	2
18	2	1	0	2
22	2	1	1	2
24	2	1	2	0
25	2	1	2	1
26	2	1	2	2
32	2	2	0	0
33	2	2	0	1
34	2	2	0	2
36	2	2	1	0
37	2	2	1	1
38	2	2	1	2
40	2	2	2	0
41	2	2	2	1
42	2	2	2	2

8.4.6 Significant Types of 4 AC Coefficients (L1[i]_CST) (variable size)

The syntax element L1[i]_CST shall specify the coefficient-significant types of four AC coefficients, and shall be as defined in Table 44. This syntax indicates coefficient-significant types of four ACs: AC[4i], AC[4i+1], AC[4i+2] and AC[4i+3]. The variable ‘i’ is valid from 1 to 15. CSTs of AC[4i], AC[4i+1], AC[4i+2], and AC[4i+3] shall be obtained from the decoded value of L1[i]_CST using equation (8-29), when L1[i] is one. For the case of L1[i] being two, the method obtaining the four CSTs is explained in Section 8.4.13, Section 8.4.14 and Section 8.4.15. The coefficient-significant type shall be from zero to two. ‘0’ means the corresponding AC coefficient is not significant coefficient. ‘1’ means the corresponding AC coefficient level is not greater than one. ‘2’ means the corresponding AC coefficient level is greater than one. This syntax shall be decoded using the adaptive VLC coding, and the luma blocks and the color difference blocks have the same probability model. The L1[i] shall be obtained from the syntax of ACp2_CST, L2[1]_CST, L2[2]_CST or L2[3]_CST. The procedure for decoding L1[i]_CST is described in Section 9.3.

$$\begin{aligned} \text{CST of } AC[4i] &= (L1[i]\_CST \,\&\, 0xC0) >> 6 \\ \text{CST of } AC[4i+1] &= (L1[i]\_CST \,\&\, 0x30) >> 4 \\ \text{CST of } AC[4i+2] &= (L1[i]\_CST \,\&\, 0x0C) >> 2 \\ \text{CST of } AC[4i+3] &= L1[i]\_CST \,\&\, 0x03 \end{aligned} \tag{8-29}$$

Table 44 – Meaning of L1[i]_CST

Decoded value of L1[i]_CST	L1[i]	CST of AC[4i]	CST of AC[4i+1]	CST of AC[4i+2]	CST of AC[4i+3]
N/A	0	0	0	0	0
1	1	0	0	0	1
4	1	0	0	1	0
5	1	0	0	1	1
16	1	0	1	0	0
17	1	0	1	0	1
20	1	0	1	1	0
21	1	0	1	1	1
64	1	1	0	0	0
65	1	1	0	0	1
68	1	1	0	1	0
69	1	1	0	1	1
80	1	1	1	0	0
81	1	1	1	0	1
84	1	1	1	1	0
85	1	1	1	1	1

8.4.7 Significant Types of 16th to 31th AC Coefficients (L2[1]_CST) (variable size)

The syntax element L2[1]_CST shall specify the coefficient-significant types of 16 AC coefficients corresponding to L1[4], L1[5], L1[6] and L1[7]. L2[1]_CST shall be as defined in Table 45. L1[4], L1[5], L1[6] and L1[7] shall be obtained from the decoded value of L2[1]_CST using equation (8-30), when L2[1] is one. For the case of L2[1] being two, the method obtaining L1[4], L1[5], L1[6] and L1[7] is explained in Section 8.4.10, Section 8.4.11 and Section 8.4.12. In Table 35 and Figure 11 of Section 9.1.4.1, L1[4], L1[5], L1[6], L1[7] and L2[1] are explained. The decoding method shall be the same as L1[i]_CST. The coefficient-significant type shall be from zero to two, and each meaning is explained in Section 8.4.2. This syntax shall be decoded using the adaptive VLC coding, and the luma blocks and the color difference blocks have the same probability model. The L2[1] shall be obtained from the syntax of ACp3_CST. The procedure for decoding L2[1]_CST is described in Section 9.3.

$$\begin{aligned} L1[4] &= (L2[1]\_CST \,\&\, 0xC0) >> 6 \\ L1[5] &= (L2[1]\_CST \,\&\, 0x30) >> 4 \\ L1[6] &= (L2[1]\_CST \,\&\, 0x0C) >> 2 \\ L1[7] &= L2[1]\_CST \,\&\, 0x03 \end{aligned} \tag{8-30}$$

Table 45 – Meaning of L2[1]_CST

Decoded value of L2[1]_CST	L2[1]	L1[4]	L1[5]	L1[6]	L1[7]
N/A	0	0	0	0	0
1	1	0	0	0	1
4	1	0	0	1	0
5	1	0	0	1	1
16	1	0	1	0	0
17	1	0	1	0	1
20	1	0	1	1	0
21	1	0	1	1	1
64	1	1	0	0	0
65	1	1	0	0	1
68	1	1	0	1	0
69	1	1	0	1	1
80	1	1	1	0	0
81	1	1	1	0	1
84	1	1	1	1	0
85	1	1	1	1	1

8.4.8 Significant Types of 32th to 47th AC Coefficients (L2[2]_CST) (variable size)

The syntax element L2[2]_CST shall specify the coefficient-significant types of 16 AC coefficients corresponding to L1[8], L1[9], L1[10] and L1[11]. L2[2]_CST shall be as defined in Table 46. L1[8], L1[9], L1[10] and L1[11] shall be obtained from the decoded value of L2[2]_CST using equation (8-31), when L2[2] is one. For the case of L2[2] being two, the method obtaining L1[8], L1[9], L1[10] and L1[11] is explained in Section 8.4.10, Section 8.4.11 and Section 8.4.12. In Table 35 and Figure 11 of Section 9.1.4.1, L1[8], L1[9], L1[10], L1[11] and L2[2] are explained. The decoding method shall be the same as L1[i]_CST. The coefficient-significant type shall be from zero to two, and each meaning is explained in Section 8.4.2. This syntax shall be decoded using the adaptive VLC coding, and the luma blocks and the color difference blocks have the same probability model. The L2[2] shall be obtained from the syntax of ACp3_CST. The procedure for decoding L2[2]_CST is described in Section 9.3.

$$\begin{aligned} L1[8] &= (L2[2]\_CST \,\&\, 0xC0) >> 6 \\ L1[9] &= (L2[2]\_CST \,\&\, 0x30) >> 4 \\ L1[10] &= (L2[2]\_CST \,\&\, 0x0C) >> 2 \\ L1[11] &= L2[2]\_CST \,\&\, 0x03 \end{aligned} \tag{8-31}$$

Table 46 – Meaning of L2[2]_CST

Decoded value of L2[2]_CST	L2[2]	L1[8]	L1[9]	L1[10]	L1[11]
N/A	0	0	0	0	0
1	1	0	0	0	1
4	1	0	0	1	0
5	1	0	0	1	1
16	1	0	1	0	0
17	1	0	1	0	1
20	1	0	1	1	0
21	1	0	1	1	1
64	1	1	0	0	0
65	1	1	0	0	1
68	1	1	0	1	0
69	1	1	0	1	1
80	1	1	1	0	0
81	1	1	1	0	1
84	1	1	1	1	0
85	1	1	1	1	1

8.4.9 Significant Types of 48th to 63th AC Coefficients (L2[3]_CST) (variable size)

The syntax element L2[3]_CST shall specify the coefficient-significant types of 16 AC coefficients corresponding to L1[12], L1[13], L1[14] and L1[15]. L2[3]_CST shall be as defined in Table 47. L1[12], L1[13], L1[14] and L1[15] shall be obtained from the decoded value of L2[3]_CST using equation (8-32), when L2[3] is one. For the case of L2[3] being two, the method obtaining L1[12], L1[13], L1[14] and L1[15] is explained in Section 8.4.10, Section 8.4.11 and Section 8.4.12. In Table 35 and Figure 11 of Section 9.1.4.1, L1[12], L1[13], L1[14], L1[15] and L2[3] are explained. The decoding method shall be the same as L1[i]_CST. The coefficient-significant type shall be from zero to two, and each meaning is explained in Section 8.4.2. This syntax shall be decoded using the adaptive VLC coding, and the luma blocks and the color difference blocks have the same probability model. The L2[3] shall be obtained from the syntax of ACp3_CST. The procedure for decoding L2[3]_CST is described in Section 9.3.

$$\begin{aligned} L1[12] &= (L2[3]\_CST \,\&\, 0xC0) >> 6 \\ L1[13] &= (L2[3]\_CST \,\&\, 0x30) >> 4 \\ L1[14] &= (L2[3]\_CST \,\&\, 0x0C) >> 2 \\ L1[15] &= L2[3]\_CST \,\&\, 0x03 \end{aligned} \tag{8-32}$$

Table 47 – Meaning of L2[3]_CST

Decoded value of L2[3]_CST	L2[3]	L1[12]	L1[13]	L1[14]	L1[15]
N/A	0	0	0	0	0
1	1	0	0	0	1
4	1	0	0	1	0
5	1	0	0	1	1
16	1	0	1	0	0
17	1	0	1	0	1
20	1	0	1	1	0
21	1	0	1	1	1
64	1	1	0	0	0
65	1	1	0	0	1
68	1	1	0	1	0
69	1	1	0	1	1
80	1	1	1	0	0
81	1	1	1	0	1
84	1	1	1	1	0
85	1	1	1	1	1

8.4.10 Significant Types of 16 AC Coefficients in Layer 2(L2AC16_CST[i]) (variable size)

The syntax element L2AC16_CST[i] shall specify the coefficient-significant types of 16 AC coefficients among 48 AC coefficients in the layer 2, and shall be as defined in Table 48. Those 16 AC coefficients shall be from AC[16i] to AC[16i+15] where the variable 'i' is valid from 1 to 3. The decoding method is the same as DC_AC_CST except that this syntax shall be present when L2[i] obtained from ACp3_CST is two, and the luma blocks and the color difference blocks have the same probability model. This syntax indicates coefficient-significant types of two parts of AC coefficients. One part is corresponding to the first 4 AC coefficients from AC[16i] to AC[16i+3]. The largest CST of these four AC coefficients is represented in L1[4i]. The other part is corresponding to the rest 12 AC coefficients from AC[16i+4] to AC[16i+15]. The CST of the second part shall specify the CST of the rest 12 AC coefficients. In other word, this second part shall be the largest value among L1[4i+1], L1[4i+2], L1[4i+3]. Both L1[4i] and the CST of the others (L1[4i+1], L1[4i+2], and L1[4i+3]) shall be obtained from the decoded value of L2AC16_CST[i] using equation (8-33) for the case of L2[i] being two. When L2[i] is one, it is explained in the Section 8.4.7, Section 8.4.8, and Section 8.4.9 how to obtain L1[4i], L1[4i+1], L1[4i+2], and L1[4i+3]. If L2[i] is zero, this L2AC16_CST[i] shall not be decoded and the corresponding CSTs from AC[16i] to AC[16i+15] shall be zero. The coefficient-significant type shall be from zero to two, and each meaning is explained in Section 8.4.2. This syntax shall be also decoded using the adaptive VLC coding. The L2[i] shall be obtained from the syntax of ACp3_CST. L1[x] and L2[x] shown in Table 48 are explained in Table 35 and Figure 11 of Section 9.1.4.1. The procedure for decoding L2AC16_CST[i] is described in Section 9.3.

$$\begin{aligned} L1[4i] &= (L2AC16\_CST[i] \,\&\, 0x0C) >> 2 \\ CST\ of\ L1[4i+1] \sim L1[4i+3] &= L2AC16\_CST[i] \,\&\, 0x03 \end{aligned} \tag{8-33}$$

Table 48 – Meaning of L2AC16_CST[i]

Decoded value of L2AC16_CST[i]	L2[i]	L1[4i]	CST of L1[4i+1]~L1[4i+3]
N/A	0	0	0
10 (1010b)	2	2	2
9 (1001b)		2	1
8 (1000b)		2	0
6 (0110b)		1	2
2 (0010b)		0	2

8.4.11 Significant Types of the Last 12 AC Coefficients in Layer 2(L2AC12_CST[i]) (variable size)

The syntax element L2AC12_CST[i] shall specify the coefficient-significant types of 12 AC coefficients among 48 AC coefficients in the layer 2, and shall be as defined in Table 49. Those 12 AC coefficients shall be from AC[16i+4] to AC[16i+15] where the variable 'i' is valid from 1 to 3. The decoding method shall be the same as DC_AC_CST except that the luma blocks and the color difference blocks have the same probability model. This syntax indicates coefficient-significant types of two parts of AC coefficients. One part is corresponding to the four AC coefficients from AC[16i+4] to AC[16i+7]. The largest CST of these four AC coefficients is represented in L1[4i+1]. The other part is corresponding to the rest eight AC coefficients from AC[16i+8] to AC[16i+15]. The CST of the second part shall specify the CST of the rest eight AC coefficients. In other words, this second part shall be the largest value between L1[4i+2] and L1[4i+3]. Both L1[4i+1] and the CST of the others (L1[4i+2] and L1[4i+3]) shall be obtained from the decoded value of L2AC12_CST[i] using equation (8-34) for the case of the CST of L1[4i+1]~L1[4i+3] being one, and using equation (8-35) for the case of the CST of L1[4i+1]~L1[4i+3] being two. If the CST of L1[4i+1]~L1[4i+3] is zero, this L2AC12_CST[i] shall not be decoded and the corresponding CSTs from AC[16i+4] to AC[16i+15] shall be zero. The coefficient-significant type shall be from zero to two, and each meaning is explained in Section 8.4.2. This syntax shall be also decoded using the adaptive VLC coding. The CST of L1[4i+1]~L1[4i+3] shall be obtained from L2AC16_CST[i]. L1[x] shown in Table 49 is explained in Table 35 and Figure 11 of Section 9.1.4.1. The procedure for decoding L2AC12_CST[i] is described in Section 9.3.

$$L1[4i+1] = ((L2AC12\_CST[i]+1) >> 1 \quad (8\text{-}34)$$

$$CST\ of\ L1[4i+2]\ and\ L1[4i+3] = 1 - ((L2AC12\_CST[i] \,\&\, 0x01)$$

$$L1[4i+1] = (L2AC12\_CST[i] \,\&\, 0x0C) >> 2 \quad (8\text{-}35)$$

$$CST\ of\ L1[4i+2]\ and\ L1[4i+3] = L2AC12\_CST[i] \,\&\, 0x03$$

Table 49 – Meaning of L2AC12_CST[i]

Decoded value of L2AC12_CST[i]	CST of L1[4i+1]~L1[4i+3]	L1[4i+1]	CST of L1[4i+2] and L1[4i+3]
N/A	0	0	0
0	1	0	1
1		1	0
2		1	1
10 (1010b)	2	2	2
9 (1001b)		2	1
8 (1000b)		2	0
6 (0110b)		1	2
2 (0010b)		0	2

8.4.12 Significant Types of the Last 8 AC Coefficients in Layer 2(L2AC8_CST[i]) (variable size)

The syntax element L2AC8_CST[i] shall specify the coefficient-significant types of 8 AC coefficients among 48 AC coefficients in the layer 2, and shall be as defined in Table 50. Those eight AC coefficients shall be from AC[16i+8] to AC[16i+15] where the variable 'i' is valid from 1 to 3. The decoding method shall be the same as L2AC12_CST[i]. This syntax indicates coefficient-significant types of two parts of AC coefficients. One part is corresponding to the four AC coefficients from AC[16i+8] to AC[16i+11]. The largest CST of these four AC coefficients is represented in L1[4i+2]. The other part is corresponding to rest four AC coefficients from AC[16i+12] to AC[16i+15]. The CST of the second part shall specify the CST of the rest four AC coefficients. In other word, this second part shall be the largest CST among AC[16i+12], AC[16i+13], AC[16i+14] and AC[16i+15]. This second part's CST is represented in L1[4i+3]. Both L1[4i+2] and L1[4i+3] shall be obtained from the decoded value of L2AC8_CST[i] using equation (8-36) for the case of the CST of L1[4i+2] and L1[4i+3] being one, and using equation (8-37) for the case of the CST of L1[4i+2] and L1[4i+3] being two. If the CST of L1[4i+2] and L1[4i+3] is zero, this L2AC8_CST[i] shall not be decoded and the corresponding CSTs from AC[16i+8] to AC[16i+15] shall be zero. The coefficient-significant type shall be from zero to two, and each meaning is explained in Section 8.4.2. This syntax shall be also decoded using the adaptive VLC coding, and the luma blocks and the color difference blocks have the same probability model. The CST of L1[4i+2] and L1[4i+3] shall be obtained from L2AC12_CST[i]. L1[x] shown in Table 50 is explained in Table 35 and Figure 11 of Section 9.1.4.1. The procedure for decoding L2AC8_CST[i] is described in Section 9.3.

$$L1[4i+2] = ((L2AC8\_CST[i]+1) >> 1$$
$$L1[4i+3] = 1 - ((L2AC8\_CST[i] \,\&\, 0x01) \quad (8\text{-}36)$$

$$L1[4i+2] = (L2AC8\_CST[i] \,\&\, 0x0C) >> 2$$
$$L1[4i+3] = L2AC8\_CST[i] \,\&\, 0x03 \quad (8\text{-}37)$$

Table 50 – Meaning of L2AC8_CST[i]

Decoded value of L2AC8_CST[i]	CST of L1[4i+2] and L1[4i+3]	L1[4i+2]	L1[4i+3]
N/A	0	0	0
0	1	0	1
1		1	0
2		1	1
10 (1010b)	2	2	2
9 (1001b)		2	1
8 (1000b)		2	0
6 (0110b)		1	2
2 (0010b)		0	2

8.4.13 Significant Types of 4 AC Coefficients in Layer 1(L1AC4_CST[i]) (variable size)

The syntax element L1AC4_CST[i] shall specify the coefficient-significant types of four AC coefficients among 60 AC coefficients in the layer 1, and shall be as defined in Table 51. Those four AC coefficients shall be from AC[4i] to AC[4i+3] where the variable 'i' is valid from 1 to 15. The decoding method shall be the same as L2AC16_CST[i] except that this syntax shall be present when L1[i] obtained from ACp2_CST, L2[1]_CST, L2[2]_CST, L2[3]_CST, L2AC16_CST[i], L2AC12_CST[i] or/and L2AC8_CST[i] is two. This syntax indicates coefficient-significant types of two parts of AC coefficients: one part is corresponding to the first AC coefficient AC[4i] and the other part is corresponding to three AC coefficients from AC[4i+1] to AC[4i+3]. Both the CST of AC[4i] and the CST of the others (AC[4i+1], AC[4i+2], and AC[4i+3]) shall be obtained from the decoded value of L1AC4_CST[i] using equation (8-38) for the case of L1[i] being two. When L1[i] is one, it is explained in Section 8.4.6 how to obtain the four CSTs of AC[4i], AC[4i+1], AC[4i+2], and AC[4i+3]. If L1[i] is zero, this

L1AC4_CST[i] shall not be decoded and the corresponding CSTs from AC[4i] to AC[4i+3] shall be zero. The coefficient-significant type shall be from zero to two, and each meaning is explained in Section 8.4.2. This syntax shall be also decoded using the adaptive VLC coding, and the luma blocks and the color difference blocks have the same probability model. L1[x] in Table 51 is explained in Table 35 and Figure 11 of Section 9.1.4.1. The procedure for decoding L1AC4_CST[i] is described in Section 9.3.

$$\begin{aligned} \text{CST of } AC[4i] &= (L1AC4\_CST[i] \,\&\, 0x0C) >> 2 \\ \text{CST of } AC[4i+1] \sim AC[4i+3] &= L1AC4\_CST[i] \,\&\, 0x03 \end{aligned} \tag{8-38}$$

Table 51 – Meaning of L1AC4_CST[i]

Decoded value of L1AC4_CST[i]	L1[i]	CST of AC[4i]	CST of AC[4i+1]~AC[4i+3]
N/A	0	0	0
10 (1010b)	2	2	2
9 (1001b)		2	1
8 (1000b)		2	0
6 (0110b)		1	2
2 (0010b)		0	2

8.4.14 Significant Types of the Last 3 AC Coefficients in Layer 1(L1AC3_CST[i]) (variable size)

The syntax element L1AC3_CST[i] shall specify the coefficient-significant types of three AC coefficients among 60 AC coefficients in the layer 1, and shall be as defined in Table 52. Those three coefficients shall be from AC[4i+1] to AC[4i+3] where the variable 'i' is valid from 1 to 15. The decoding method shall be the same as L2AC12_CST[i]. This syntax indicates coefficient-significant types of two parts of AC coefficients: one part is corresponding to AC[4i+1] and the other part is corresponding to two AC coefficients (AC[4i+2] and AC[4i+3]). Both the CST of AC[4i+1] and the CST of the others (AC[4i+2] and AC[4i+3]) shall be obtained from the decoded value of L1AC3_CST[i] using equation (8-39) for the case of the CST of AC[4i+1]~AC[4i+3] being one, and using equation (8-40), for the case of the CST of AC[4i+1]~AC[4i+3] being two. If the CST of AC[4i+1]~AC[4i+3] is zero, this L1AC3_CST[i] shall not be decoded and the corresponding CSTs from AC[4i+1] to AC[4i+3] shall be zero. The coefficient-significant type shall be from zero to two, and each meaning is explained in Section 8.4.2. This syntax shall be also decoded using the adaptive VLC coding, and the luma blocks and the color difference blocks have the same probability model. The CST of AC[4i+1]~AC[4i+3] shall be obtained from L1AC4_CST[i]. The procedure for decoding L1AC3_CST[i] is described in Section 9.3.

$$\begin{aligned} \text{CST of } AC[4i+1] &= ((L1AC3\_CST[i]+1) >> 1 \\ \text{CST of } AC[4i+2] \text{ and } AC[4i+3] &= 1 - ((L1AC3\_CST[i] \,\&\, 0x01) \end{aligned} \tag{8-39}$$

$$\begin{aligned} \text{CST of } AC[4i+1] &= (L1AC3\_CST[i] \,\&\, 0x0C) >> 2 \\ \text{CST of } AC[4i+2] \text{ and } AC[4i+3] &= L1AC3\_CST[i] \,\&\, 0x03 \end{aligned} \tag{8-40}$$

Table 52 – Meaning of L1AC3_CST[i]

Decoded value of L1AC3_CST[i]	CST of AC[4i+1]~AC[4i+3]	CST of AC[4i+1]	CST of AC[4i+2] and AC[4i+3]
N/A	0	0	0
0	1	0	1
1		1	0
2		1	1
10 (1010b)	2	2	2
9 (1001b)		2	1
8 (1000b)		2	0
6 (0110b)		1	2
2 (0010b)		0	2

8.4.15 Significant Types of the Last 2 AC Coefficients in Layer 1(L1AC2_CST[i]) (variable size)

The syntax element L1AC2_CST[i] shall specify the coefficient-significant types of two AC coefficients among 60 AC coefficients in the layer 1, and shall be as defined in Table 53. Those two coefficients shall be AC[4i+2] and AC[4i+3] where the variable 'i' is valid from 1 to 15. The decoding method shall be the same as L2AC8_CST[i]. This syntax indicates coefficient-significant types of two parts of AC coefficients: one part is corresponding to AC[4i+2] and the other part is corresponding to AC[4i+3]. Both the CST of AC[4i+2] and the CST of AC[4i+3] shall be obtained from the decoded value of L1AC2_CST[i] using (8-41) for the case of the CST of AC[4i+2] and AC[4i+3] being one, and using equation (8-42) for the case of the CST of AC[4i+2] and AC[4i+3] being two. If the CST of AC[4i+2] and AC[4i+3] is zero, this L1AC2_CST[i] shall not be decoded and the corresponding CSTs (AC[4i+2] and AC[4i+3]) shall be zero. The coefficient-significant type shall be from zero to two, and each meaning is explained in Section 8.4.2. This syntax shall be also decoded using the adaptive VLC coding, and the luma blocks and the color difference blocks have the same probability model. The CST of AC[4i+2] and AC[4i+3] shall be obtained from L1AC3_CST[i]. The procedure for decoding L1AC2_CST[i] is described in Section 9.3.

$$\begin{aligned} \text{CST of } AC[4i+2] &= ((\text{L1AC2\_CST}[i]+1) >> 1 \\ \text{CST of } AC[4i+3] &= 1 - ((\text{L1AC2\_CST}[i] \,\&\, 0x01) \end{aligned} \tag{8-41}$$

$$\begin{aligned} \text{CST of } AC[4i+2] &= (\text{L1AC2\_CST}[i] \,\&\, 0x0C) >> 2 \\ \text{CST of } AC[4i+3] &= \text{L1AC2\_CST}[i] \,\&\, 0x03 \end{aligned} \tag{8-42}$$

Table 53 – Meaning of L1AC2_CST[i]

Decoded value of L1AC2_CST[i]	CST of AC[4i+2] and AC[4i+3]	CST of AC[4i+2]	CST of AC[4i+3]
N/A	0	0	0
0	1	0	1
1		1	0
2		1	1
10 (1010b)	2	2	2
9 (1001b)		2	1
8 (1000b)		2	0
6 (0110b)		1	2
2 (0010b)		0	2

8.4.16 Coefficient Level (COEFF_LEVEL[i]) (variable size)

The syntax COEFF_LEVEL[i] shall specify the level and sign of a non-zero transform coefficients. This syntax element COEFF_LEVEL[i] shall be decoded when DC or any AC component is significant. COEFF_LEVEL[i] is decoded as described in Section 9.1.4.2. The range of the level is described in Section 9.1.4.2 and Section 9.1.4.5. All syntaxes except for COEFF_LEVEL[i] shall be used to indicate which AC components have more than zero value.

9 Progressive Bitstream Decoding Process

The decoding process of I and P frame pictures of the residual layer stream is specified in this section.

9.1 Progressive I Frame Picture Decoding

Section 9.1.1 defines the picture layer decoding, Section 9.1.2 defines the slice layer decoding, Section 9.1.3 defines the macroblock layer decoding, and Section 9.1.4 defines the block layer decoding.

9.1.1 Picture Layer Decoding at Progressive I Frame Picture

Section 8.1 defines the elements in the intra picture layer header. The following sections provide extra detail for those elements.

9.1.1.1 Picture-level Quantization Parameters

At the picture level, two syntax elements are used to signal if the quantizer varies spatially or across the color channels. QP_PICTURE_UNIFORM_FLAG and QP_CHANNEL_UNIFORM_FLAG are explained in Section 8.1.20 and Section 8.1.21, respectively.

The quantization parameters for the picture shall be decoded depending on QP_CHANNEL_UNIFORM_FLAG as shown in Figure 4. The range of the quantization parameters shall be from 0 to 255.

```
if (QP_ CHANNEL_UNIFORM_FLAG == 1) {
      QP_PICTURE = get_bits(8 bits); //Quantization parameter for all channels
}
else {
      QP_PICTURE_Y = get_bits(8 bits); // Quantization parameter for Y channel
      QP_PICTURE_U = get_bits(8 bits); // Quantization parameter for Cb channel
      QP_PICTURE_V = get_bits(8 bits); // Quantization parameter for Cr channel
}
```

Figure 4 – Pseudo-code for picture-level quantization parameters

9.1.2 Slice Layer Decoding at Progressive I Frame Picture

Section 8.2 defines the elements in the slice header of the inter frame picture. The following sections provide extra detail for those elements.

9.1.2.1 Slice-level Quantization Parameters

At the picture level, two syntax elements are used to signal if the quantizer varies spatially or across the color channels. QP_PICTURE_UNIFORM_FLAG and QP_CHANNEL_UNIFORM_FLAG are explained in Section 8.1.20 and Section 8.1.21, respectively.

The quantization parameters for the slice are decoded depending on QP_CHANNEL_UNIFORM_FLAG as shown in Figure 5. Here, the slice-level quanitization parameters are the modified version of QP_PICTURE, QP_PICTURE_U, QP_PICTURE_Y and QP_PICTURE_V.

```
if (QP_ CHANNEL_UNIFORM_FLAG == 1) {
      DQP_SLICE = decode_DQP_vlc();
       QP_SLICE = QP_PICTURE + DQP_SLICE; //Quantization parameter for all channels
       if(QP_SLICE > 255) QP_SLICE = 255;
       if(QP_SLICE < 0) QP_SLICE = 0;
}
else {
       DQP_SLICE_Y = decode_DQP_vlc();
       DQP_SLICE_U = decode_DQP_vlc();
       DQP_SLICE_V = decode_DQP_vlc()

      QP_SLICE_Y = QP_PICTURE_Y + DQP_SLICE_Y; // Quantization parameter for Y channel
      If(QP_SLICE_Y > 255) QP_SLICE_Y = 255;
       if(QP_SLICE_Y < 0) QP_SLICE_Y = 0;

      QP_SLICE_U = QP_PICTURE_U + DQP_SLICE_U; // Quantization parameter for Cb channel
      if(QP_SLICE_U > 255) QP_SLICE_U = 255;
       if(QP_SLICE_U < 0) QP_SLICE_U = 0;

      QP_SLICE_V = QP_PICTURE_V + DQP_SLICE_V; // Quantization parameter for Cr channel
      if(QP_SLICE_V > 255) QP_SLICE_V = 255;
       if(QP_SLICE_V < 0) QP_SLICE_V = 0;
}
```

Figure 5 – Pseudo-code for slice-level quantization parameters

In Figure 5, the function *decode_DQP_vlc*() shall decode DQP_SLICE, DQP_SLICE_Y, DQP_SLICE_U and DQP_SLICE_V using the pseudo-code of Figure 6. The range of the quanization parameters QP_SLICE, QP_SLICE_Y, QP_SLICE_U and QP_SLICE_V shall be from 0 to 255.

```
//DQP represents the differential of the quanizer parameter
decode_DQP_vlc() {
  for(numZeros=0; ; numZeros++)
      if(get_bits(1) == 1)
         break;

 if(numZeros > 0) {
      val = (1 << numZeros) + get_bits(numZeros) - 1;
      if ( (val & 1) == 1)
         DQP = (val+1) >> 1;
      else
         DQP = -(val >> 1);
 }
 else
     DQP = 0;

  return DQP;
 }
```

Figure 6 – Pseudo-code of DQP decoding

If QP_PICTURE_UNIFORM_FLAG == 0, it shall be possible that the quantization parameter varies spatially across the slice and each QP shall be decided at the macroblock level, therefore the quantization parameter varies spatially across the slice. Otherwise these modified quantization parameters shall be fixed within a slice. The details are described in Section 9.1.3.3.

9.1.3 Macroblock Layer Decoding at Progressive I Frame Picture

The macroblock layer shall specify the SKIP_RUN, IPRED_MODE, INTRA_SIG, YUV_CSP, Y_CSP and UV_CSP. If the color difference format is 4:2:2, U0U1_CSP and V0V1_CSP shall be additionally specified. If the color difference format is 4:4:4, U_CSP, V_CSP, U2U3_CSP and V2V3_CSP shall be additionally specified.

9.1.3.1 Skipped-macroblock Run (SKIP_RUN)

This value shall be decoded when the current macroblock is the beginning of each slice or the previous macroblock is not skipped macroblocks. When the macroblock-level CSP is zero in the intra picture, the current macroblock shall be the skipped macroblock. If the number of skipped macroblocks is greater than four, SKIP_RUN shall be repeated until all skipped macroblocks are counted. For example, when the number of skipped macroblocks is 10, SKIP_RUN shall be coded three times. The first SKIP_RUN is four, the second SKIP_RUN is also four and the third SKIP_RUN is two. SKIP_RUN shall be decoded using adaptive VLC code with five alphabets. The procedure for decoding SKIP_RUN is described at Figure 29, Figure 33 and Figure 34 in Section 9.3.

9.1.3.2 Intra Prediction

The selected IPRED_MODE as specified in Section 8.3.6 shall be applied to both luma and color difference blocks. If IPRED_MODE == HORZ, predicted samples shall be obtained by extrapolating the right edge samples of the left macroblock into current macroblock. If IPRED_MODE == VERT, predicted samples shall be obtained by extrapolating the bottom edge samples of the upper macroblock into current macroblock. The extrapolation for luma components is illuminated in Figure 7. If IPRED_MODE == DEFAULT, all predicted samples shall be assigned to the value *Shift* as defined in Section 12.2.

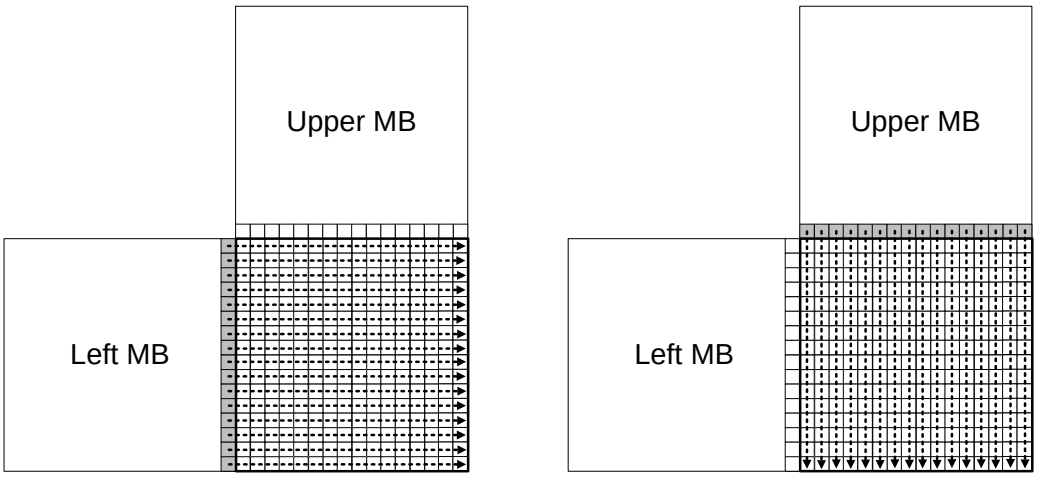

Figure 7 – Intra prediction mode (in case of luma)

The pseudo-code in Figure 8 describes the intra prediction process.

```
// Rec[][] represents previously reconstructed luma or color difference samples in the current picture
// Pred[][] represents predicted luma or color difference samples in the current picture
// Shift represents the value defined in Section 12.2
// (iXCoord, iYCoord) represents the luma or color difference location given in full-sample units of the top left corner of the current macroblock
// Height represents the height for luma or color difference of the current macroblock
// Width represents the width for luma or color difference of the current macroblock

if(IPRED_MODE == HORZ) {
      for(row = 0; row < Height; row++)
            for(col = 0; col < Width; col++)
                  Pred[iXCoord+col][iYCoord+row] = Rec[iXCoord – 1][iYCoord+row];
}
else if(IPRED_MODE == VERT) {
      for(row = 0; row < Height; row++)
            for(col = 0; col < Width; col++)
                  Pred[iXCoord+col][iYCoord+row] = Rec[iXCoord+col][iYCoord-1];
}
else { //IPRED_MODE == DEFAULT
      for(row = 0; row < Height; row++)
            for(col = 0; col < Width; col++)
                  Pred[iXCoord+col][iYCoord+row] = Shift;

}
```

Figure 8 – Pseudo-code for intra prediction process

9.1.3.3 Macroblock-level Quantization Parameter

If QP_PICTURE_UNIFORM_FLAG == 0, the quantization parameters at the macroblock level shall be specified. This parameter specifying procedure consists of two steps: prediction and updating.

Prediction:
At the macroblock, the quantization parameter of the macroblock shall be predicted from the quantization parameters of the left and top macroblock of the current macroblock.

The prediction scheme is defined by the pseudo-code in Figure 9. Here, QP_SLICE, QP_SLICE_Y, QP_SLICE_U and QP_SLICE_V are derived in Section 9.1.2.1.

```
if (isLeftNeighbor() && !is_LeftMB_Skip()) {
     if(QP_CHANNEL_UNIFORM_FLAG == 1) {
            QPPred = QP[LeftNeighbor];
     }
     else {
            QPPred_Y = QP_Y[LeftNeighbor];
            QPPred_U = QP_U[LeftNeighbor];
            QPPred_V = QP_V[LeftNeighbor];
    }
}
else if (isTopNeighbor() && !is_UpperMB_Skip()) {
     if(QP_CHANNEL_UNIFORM_FLAG == 1) {
           QPPred = QP[TopNeighbor];
     }
     else {
            QPPred_Y = QP_Y[TopNeighbor];
            QPPred_U = QP_U[TopNeighbor];
            QPPred_V = QP_V[TopNeighbor];
    }
}
else {
    if(QP_CHANNEL_UNIFORM_FLAG == 1) {
             QPPred = QP_SLICE;
   }
   else {
           QPPred_Y = QP_SLICE_Y;
           QPPred_U = QP_SLICE_U;
           QPPred_V = QP_SLICE_V;
   }
}
```

Figure 9 – Pseudo-code for predicting quantization parameter of the current MB

Updating:

First QP_PICTURE_UNIFORM_FLAG shall be used to indicate whether the quantization parameters of the current macroblock are equal to the slice-level quantization parameters. If QP_PICTURE_UNIFORM_FLAG == 1, the quantization parameters of the current macroblock shall be the same as the slice-level quantization parameters.

```
if (QP_PICTURE_UNIFORM_FLAG == 1) {
        if(QP_CHANNEL_UNIFORM_FLAG == 1) {
                QP_MB = QP_SLICE;
        }
        else{
                QP_MB_Y = QP_SLICE_Y;
                QP_MB_U = QP_SLICE_U;
                QP_MB_V = QP_SLICE_V;
        }
else {
        if(QP_CHANNEL_UNIFORM_FLAG == 1) {
                DQP_MB = decode_DQP_vlc()
                QP_MB = QPPred + DQP_MB;
                if(QP_MB > 255) QP_MB = 255;
                if(QP_MB < 0) QP_MB = 0;
        }
        else {
                DQP_MB = decode_DQP_vlc()

                QP_MB_Y = QPPred_Y + DQP_MB:
                if(QP_MB_Y > 255) QP_MB_Y = 255;
                if(QP_MB_Y < 0) QP_MB_Y = 0;

                QP_MB_U = QPPred_U + DQP_MB;
                if(QP_MB_U > 255) QP_MB_U = 255;
                if(QP_MB_U < 0) QP_MB_U = 0;

                QP_MB_V = QPPred_V + DQP_MB
                if(QP_MB_V > 255) QP_MB_V = 255;
                if(QP_MB_V < 0) QP_MB_V = 0;
        }
}
```

Figure 10 – Pseudo-code for selecting MB-level quantization parameter

In Figure 10, the function *decode_DQP_vlc_code()* is defined in Figure 6.

9.1.4 Block Layer Decoding at Progressive I Frame Picture

9.1.4.1 Layers of Coefficient Levels

In order to decode coefficient levels, the coefficient-significant type of each coefficient is necessary. All syntaxes except for COEFF_LEVEL[i] in the block layer specify the coefficient-significant types of all coefficients within the block.

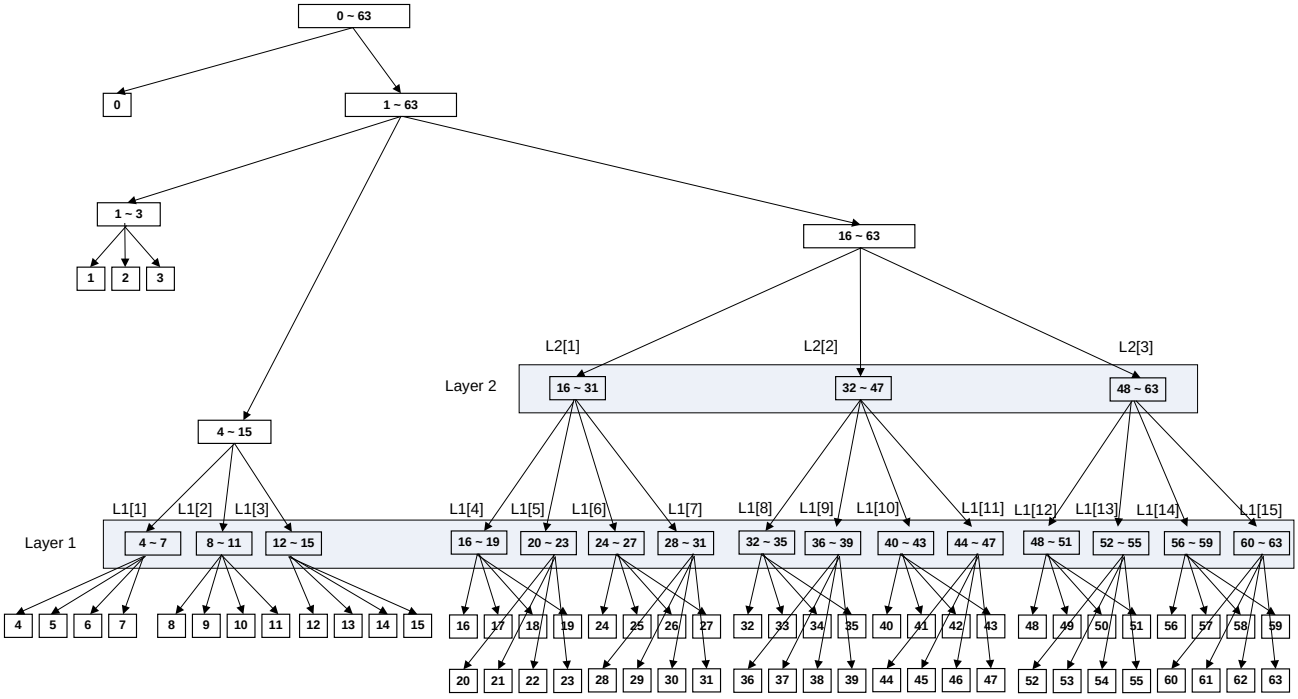

Figure 11 – Layers of AC coefficient levels

Figure 11 defines layers of some AC levels. The layer 1 shall consist of 15 elements each of which has 4 AC levels. The CST of each element is called L1[i]. L1[1] represents the CST of AC[4]~AC[7] levels not AC[1]~AC[4]. L1[i] represents the CST of the subsequent 4 AC levels of L1[i-1]. Therefore, L1[i] represents the CST of AC[4i], AC[4i+1], AC[4i+2] and AC[4i+3] levels. For example, L1[5] represents the CST of AC[20], AC[21], AC[22] and AC[23]. In Figure 11, there shall be three CST elements in the layer 2: L2[1], L2[2] and L2[3]. Each element represents the CST of 4 elements of the layer 1. So L2[i] represents the CST of 16 AC levels. L2[i] shall consist of L1[4i], L1[4i+1], L1[4i+2] and L1[4i+3]. L2[i] represents the CST of the subsequent 16 AC levels of L2[i-1]. For instance, L2[i] represents the CST of AC[16i]~AC[16i+15]. In Figure 11, the box of '0~63' is corresponding to DC_AC_CST in Section 8.4.1. The box of '1~63' represents CSTs of three groups: AC[1]~AC[3], AC[4~15], and AC[16~63]. This box is corresponding to AC_CST in Section 8.4.2. The box of '1~3' is corresponding to the CST of the first three AC coefficients represented by ACp1_CST in Section 8.4.3. The box of '16~63' represents CSTs of three elements (L2[1], L2[2] and L2[3]) at the layer 2, which is corresponding to ACp3_CST in Section 8.4.5. The box of '4~15' represents CSTs of L1[1], L1[2] and L1[3], which is represented by ACp2_CST in Section 8.4.4. The L2[i] represents the CST of L1[4i], L1[4i+1], L1[4i+2] and L1[4i+3], and the L1[i] represents CSTs of AC[4i], AC[4i+1], AC[4i+2] and AC[4i+3]. Each L2[i] is specified by ACp3_CST, and each L1[i] (for i > 3) is specified by L2[1]_CST, L2[2]_CST, L2[3]_CST, L2AC16_CST[i], L2AC12_CST[i] or/and L2AC8_CST[i] in Section 8.4.7, Section 8.4.8, Section 8.4.9, Section 8.4.10, Section 8.4.11 and Section 8.4.12, respectively.

9.1.4.2 Decoding Coefficient Levels

After decoding significant types of coefficients are decided, all significant coefficient levels shall be decoded according to their CSTs. If the CSP of a coefficient level is '1', only the sign information is necessary to decode its coefficient level. When the CST of a coefficient level is '2', the magnitude of the corresponding coefficient level abs(*level*) shall be signaled as the modified Exponential Golomb code:

1. if $2 \leq$ abs(*level*) ≤ 17, abs(*level*) shall be signalled as "abs(*level*) - 2" binary 0s followed by 1;

2. if abs(*level*) > 17, abs(*level*) shall be signalled as a prefix part of 16 binary 0s followed by a code to signal BITS and followed by a fixed length code of BITS; BITS (BITS > 0) shall be signalled as "BITS - 1" binary 0s followed by a binary 1; abs(*level*) shall be reconstructed as abs(*level*) = 16 + get_bits(BITS) + (1 << BITS)
3. one bit shall be used to signal the sign of *level*.

The decoding procedure of the coefficient level is specified in the pseudo-ode of Figure 12 where CST is the coefficient-significant type.

```
decode_coeff_symbol() {
  level = 0;
  if(CST == 0) return 0;
  if(CST == 1) {
     level = 1;
  }
  else {
        while(get_bits(1) == 0){
               level++;
               if(level == 16) break;
        }
         if(level == 16)
               level = decode_escape_symbol();
         else
               level += 2;
   }

   sign = get_bit(1);
   if(sign == 1)
      level = - level;
   return level
}
```

Figure 12 – Pseudo-code of coefficient decoding

When the level is greater than 17, the level shall be decoded in the escape decoding mode. In the escape mode, '0000000000000000b' shall be used as the prefix for the larger levels. The pseudo-code of Figure 13 shall specify how the levels in escape mode are decoded. The level shall be in the range of integer value from -2048 to 2047, inclusive (see Section 9.1.4.5).

```
decode_escape_symbol() {
  BITS = 1;
  while(get_bits(1)==0)
         BITS++;

   level= 16 + get_bits(BITS) + (1 << BITS);
   return level
}
```

Figure 13 – Pseudo-code of coefficient decoding in escape mode

9.1.4.3 Scanning of AC Coefficients

The coefficient level decoding in the previous section makes a one-dimensional array of the quantized transform coefficients. The elements in this array shall be converted to an 8x8 two-dimensional array for the inverse transform. A scanning array shall be used to scan out the coefficients of the one-dimensional array into the 8x8 array, where the DC always maps to the position (0, 0). Figure 14 shows the procedure for scanning out the coefficients.

```
//level[64] represents coefficient array after level decoding
//coeff[8] [8] represents transform coefficient block
//scan_array[64] represents mapping used to scan out coefficients
for(i=0; i<64; i++) {
  index = scan_array[i];
  row = index/8;
  col = index%8;
  coeff[row][col] = level[i];
}
```

Figure 14 – Pseudo-code for scanning

One of the scanning arrays shall be used to scan out the one-dimensional array depending on the syntax elements SCAN_ORDER TYPE and SEQ_SCAN_ORDER_INDEX in Section 8.1.16 and Section 8.1.17. According to those syntax elements, a user-defined scanning array can be used. When SCAN_ORDER_TYPE == 0, the default scanning matrix for the progressive picture shall be used. For example, Figure 15 shows the mapping array used to convert the one-dimensional to two-dimensional scanning pattern shown in Figure 16 and corresponds to scan_array[] of Figure 14 when the default scanning pattern is selected.

0	1	4	5	16	17	20	21
2	3	6	7	18	19	22	23
8	10	12	13	24	25	28	29
9	11	14	15	26	27	30	31
32	34	40	41	48	49	52	53
33	35	42	43	50	51	54	55
36	38	44	45	56	57	60	61
37	39	46	47	58	59	62	63

Figure 15 – Default scanning array for progressive pictures

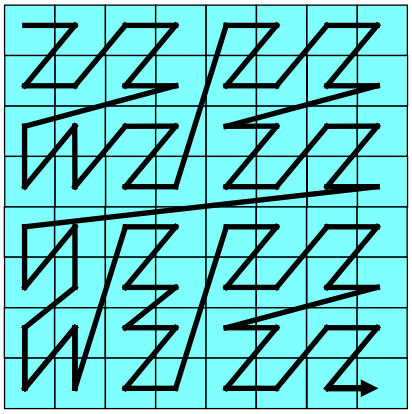

Figure 16 – Default scanning pattern for progressive pictures

9.1.4.4 Inverse Quantization for DC/AC Coefficient

Section 9.1.1.1, Section 9.1.2.1 and Section 9.1.3.3 describe the picture-level, the slice-level and the macro-block level quantization parameters, respectively. Figure 17 describes how to calculate quantization parameters for the corresponding block.

```
if(QP_PICTURE_UNIFORM_FLAG == 1) {
   if(QP_CHANNEL_UNIFORM_FLAG == 1) {
            quant = QP_SLICE;
   }
   else {
       if(current block is Y block) {
            quant = QP_SLICE_Y;
       }
       else if(current block is U block) {
            quant = QP_SLICE_U;
       }
       else if(current block is V block) {
            quant = QP_SLICE_V;
       }
   }
}
else {
      if(QP_CHANNEL_UNIFORM_FLAG == 1) {
          quant = QP_MB;
      }
      else {
          if(current block is Y block) {
              quant = QP_MB _Y;
          }
          else if(current block is U block) {
              quant = QP_MB _U;
          }
          else if(current block is V block) {
              quant = QP_MB _V;
          }
      }
}
```

Figure 17 – Pseudo-code to determine the quantization parameters for the corresponding block

In Figure 17, quant shall be the quantization parameter of the current block. The non-zero quantized coefficients shall be inversely quantized according to the pseudo-code in Figure 18.

```
//quant_coeff[i]: quantized coefficient
//dequant_coeff[i]: dequantized coefficient
//quant: quantization step or scale
//QMAT[i]: quantization matrix
//QMAT_MINUS_1[i]: QMAT[i] – 1 which is defined in Section 7.1.57

if(quant == 0) {
      for(i = 0; i < 64; i++) {
           dequant_coeff[i] = quant_coeff[i];
      }
}
else {
       if(QMATRIX_EXTENSION_FLAG == 0) {
             for(i = 0; i < 64; i++) {
                  if(quant_coeff[i] != 0)
                      dequant_coeff[i] = quant_coeff[i] * quant;
                  else
                      dequant_coeff[i] = 0;
             }
       }
       else {
             for(i = 0; i < 64; i++){
                  if(quant_coeff[i] != 0) {
                      QMAT[i] = QMAT_MINUS_1[i]+1;
                      dequant_coeff[i] = quant_coeff[i] * QMAT[i] * quant;
                  }
                  else {
                       dequant_coeff[i] = 0;
                  }
             }
      }
}
```

Figure 18 – Pseudo-code for inverse quantization

9.1.4.5 Inverse Transform

When the value of *quant* decribed in Section 9.1.4.4 is not equal to zero and the reconstruction of the transform coefficients is finished, the resulting 8 x 8 blocks shall be processed by a separable two-dimensional inverse transform. The inverse transform is similar, but not identical to an IDCT. The transform matrix defined in equation (9-1) shall be used for an 8 point inverse transformation. This transform shall be the same as the transform of SMPTE ST 421.

$$T = \begin{bmatrix} 12 & 12 & 12 & 12 & 12 & 12 & 12 & 12 \\ 16 & 15 & 9 & 4 & -4 & -9 & -15 & 16 \\ 16 & 6 & -6 & -16 & -16 & -6 & 6 & 16 \\ 15 & -4 & -16 & -9 & 9 & 16 & 4 & -15 \\ 12 & -12 & -12 & 12 & 12 & -12 & -12 & 12 \\ 9 & -16 & 4 & 15 & -15 & -4 & 16 & -9 \\ 6 & -16 & 16 & -6 & -6 & 16 & -16 & 6 \\ 4 & -9 & 15 & -16 & 16 & -15 & 9 & -4 \end{bmatrix} \quad \textbf{(9-1)}$$

The formula of equation (9-2) shall be used to compute the inverse transform 8x8 blocks.

$$\begin{aligned} E &= (D \cdot T + 4) >> 3 \\ R &= (T' \cdot E + C + 64) >> 7 \end{aligned} \quad \textbf{(9-2)}$$

where operator '$\cdot$' represents matrix multiplication, operator '>>' represents arithmetic right shift performed entry-wise on a matrix, and *T* represents the transpose matrix for *T*. *D* is the inverse quantized transform coefficient block which is the input block. The inverse quantized transform coefficient value shall not exceed the range of -2048 to 2047, inclusive. *E* is the intermediate matrix. The values in the intermediate matrix shall be in the range of -4096 to 4095, inclusive. *C* is defined in equation (9-3).

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad \textbf{(9-3)}$$

R is the inverse transformed block which is the output block.

When the value of *quant* described in Section 9.1.4.4 is equal to zero, the output block *R* shall be the same as the input block *D* that consists of the reconstructed transform coefficients without the inverse quantization.

9.1.4.6 Residual Block Reconstruction for Intra Picture

In order to reconstruct residual block, the output of the inverse transform shall be added to the result of the intra-prediction. The pseudo-code in Figure 19 describes the reconstruction process for the intra picture.

```
// rec[][] represents reconstructed residual samples in the current picture
// Pred[][] represents predicted luma or color difference samples in the current picture
// Idct[][] represents output of the inverse DCT for luma or color difference samples in current picture
// (iXCoord, iYCoord) represents the luma or color difference location given in full-sample units of the top left corner of the current macroblock
// Height represents the height for luma or color difference of the current macroblock
// Width represents the width for luma or color difference of the current macroblock

for(row = 0; row < Height; row++) {
    for(col = 0; col < Width; col++) {
        x = iXCoord+col;
        y = iYCoord+row;
        rec[x][y] = CLIP(Pred[x][y] + Idct[x][y]);
    }
}
```

Figure 19 – Pseudo-code for residual block reconstruction in the intra picture

If the syntax element QUALITY_REFINEMENT_FLAG specifies that the quality refinement is used, the following process shall be invoked. In the quality based layer, a residual block shall be reconstructed by adding the result of intra-prediction to the inverse transformed block as described in Figure 19. The reconstructed residual samples shall construct the quality base layer picture. The collocated inverse transform blocks of higher quality layers to the block of the quality base layer shall be used to refine the reconstructed residual block. The pseudo-code in Figure 20 describes the quality refinement process for the intra picture. After the quality refinement process, the reconstructed residual samples shall construct a quality enhancement layer picture.

```
// rec[][] represents reconstructed residual samples for the current picture
// Idct[k][][] represents output of the inverse DCT for luma or color difference samples in k'th quality layer
// Pred[][] represents predicted luma or color difference samples in the current picture
// (iXCoord, iYCoord) represents the luma or color difference location given in full-sample units of the top left corner of the current macroblock
// Height represents the height for luma or color difference of the current macroblock
// Width represents the width for luma or color difference of the current macroblock
// Num_quality_layer[i][j] represents the number of quality layer for the residue sample located at (i,j) position in the current picture

for(row = 0; row < Height; row++) {
    for(col = 0; col < Width; col++) {
        x = iXCoord+col;
        y = iYCoord+row;
        N = num_quality_layer[x][y];
        rec[x][y] = CLIP(Pred[x][y] + Idct[0][x][y])
        if (N > 0)
        {
            for(layer = 0; layer < N; layer++)
            {
                rec[x][y] = CLIP(rec[x][y] + Idct[layer+1][x][y])
            }
        }
    }
}
```

Figure 20 – Pseudo-code for residual block reconstruction using quality refinement in the intra picture

9.2 Progressive Inter Frame Picture Decoding

Section 9.2.1 defines the picture layer decoding, Section 9.2.2 defines the slice layer decoding, Section 9.2.3 defines the macroblock layer decoding, and Section 9.2.4 defines the block layer decoding.

9.2.1 Picture Layer Decoding at Progressive Inter Frame Picture

Section 8.1 defines the elements in the inter frame picture layer header. Most elements are similar to those of the intra frame picture in Section 9.1.1. The reference picture selection process and the management of the decoded frame buffer are described in Section 9.4.

9.2.2 Slice Layer Decoding at Progressive Inter Frame Picture

Section 8.2 defines the elements in the slice header of the inter frame picture. All elements shall be the same as the intra frame picture in Section 9.1.2.

9.2.3 Macroblock Layer Decoding at Progressive Inter Frame Picture

The Inter Frame picture decoding at the macroblock layer shall be the same as the intra picture as defined in Section 9.1.3, except as provided in this section.

9.2.3.1 Skipped-macroblock Run (SKIP_RUN)

The decoding procedure of SKIP_RUN shall be the same as Section 9.1.3.1. However, the skipped macroblocks shall be compensated with the predicted motion vector from neighboring macroblocks in the inter frame. If the horizontal location of the reference frame in full-pel unit pointed by the horizontal component of the predicated motion vector is less than -15 or greater than PICTURE_WIDTH-1, the horizontal component shall be set to zero, and if the vertical location of the reference frame in full-pel unit pointed by the vertical component of the predicated motion vector is less than -15 or greater than PICTURE_HEIGHT-1, the vertical component shall be set to zero.

In the P picture coding, the number of the predicted motion vectors shall be only one from forward or backward reference frame depending on PRED_DIRECTION. The skipped macroblocks shall be compensated with this motion vector.

In the B picture coding, the number of the predicted motion vectors shall be one or two. If some of neighboring macroblocks have forward motion vectors, the forward-predicted motion vector shall be derived from those forward motion vectors. Similarly, if some of neighboring macroblocks have backward motion vectors, the backward-predicted motion vector shall be derived from those backward motion vectors.

If all of the neighboring macroblocks have only forward motion vectors, the skipped macroblocks shall be compensated with only the forward-predicted motion vector.

If all of the neighboring macroblocks have only backward motion vectors, the skipped macroblocks shall be compensated with only the backward-predicted motion vector.

If both the forward-predicted motion vector and the backward-predicted motion vector exist, the skipped macroblocks shall be compensated using the averaged macroblock of both the forward-compensated macroblock and the backward-compensated macroblock with the forward-predicted motion vector and the backward-predicted motion vector, respectively. This shall be the same as the interpolated mode in Section 9.2.3.8 except that this mode uses the predicted motion vectors instead of the explicitly coded motion vectors.

If the above and left blocks of the skipped macroblock are coded in intra mode or outside frame, this skipped macroblock shall have zeros without motion compensation by the predicted motion vector.

9.2.3.2 Motion Vector Prediction

Motion vectors shall be computed by adding the motion vector differential (MVD or MVD2) in Section 8.3.8, Section 8.3.9, Section 8.3.10 and Section 8.3.11 to a motion vector predictor. The unit of all motion vectors shall be as defined in Section 8.1.13 and Section 10.1.15. The motion vector predictor shall be derived from some of neighboring motion vectors.

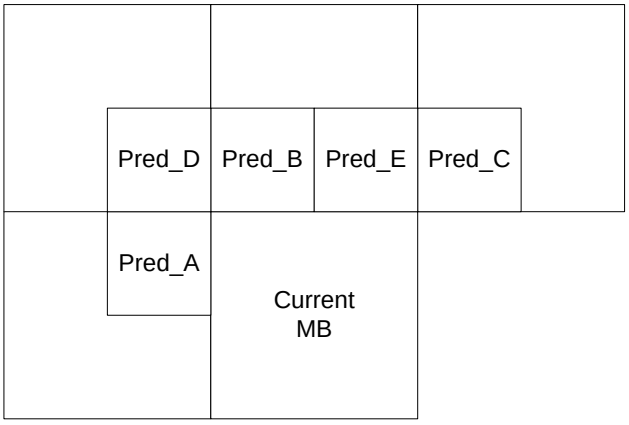

Figure 21 – Candidate motion vectors for 1-MV macroblocks

There shall be two cases for the motion vector prediction: motion vector prediction in P picture and motion vector prediction in B picture. In the P picture coding mode, since all motion vectors are only the forward motion vectors or the backward motion vectors with only one reference picture, only one predicted motion vector exists. For the B-picture coding mode, since the forward and backward predictions are available, both forward and backward predictions can exist.

In the B-picture coding mode, the forward-predicted motion vector shall be derived from the neighboring macroblocks which have forward motion vectors while the backward-predicted motion vector shall be derived from the neighboring macroblocks which have backward motion vectors. If all of the neighboring macroblocks have only forward motion vectors, the forward-predicted vector shall be derived like the P-picture coding mode and the backward-predicted vector shall not exist. On the contrary, if all of the neighboring macroblocks have only backward motion vectors, the backward-predicted vector shall be derived like the P-picture coding mode and the forward-predicted vector shall not exist.

Figure 21 shows the candidate motion vectors for an 1-MV macroblock. The neighboring macroblocks are either 1-MV or 4-MV macroblocks. The figure shows the candidate motion vectors assuming the neighbors are 4-MV macroblocks. If any of the neighbors is an 1-MV macroblock, the motion vector for the macroblocks shall be taken to be the vectors for the entire macroblock.

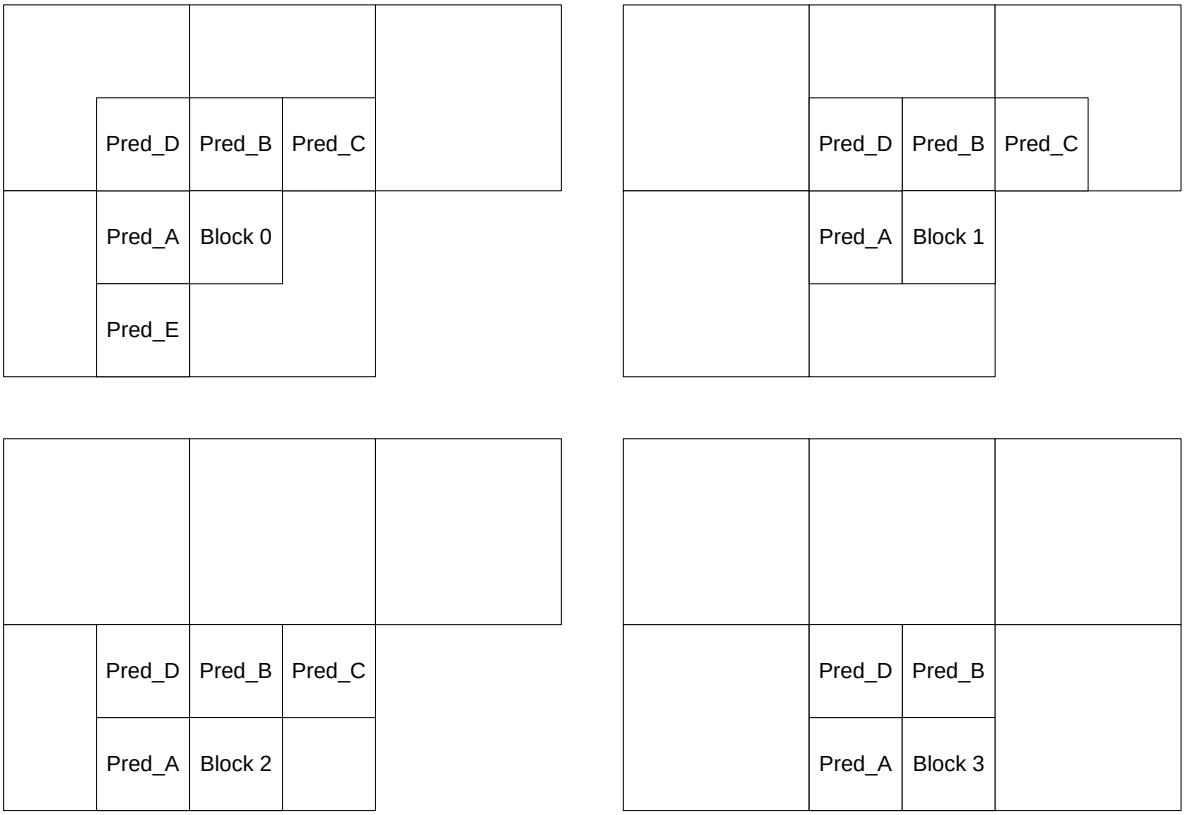

Figure 22 – Candidate motion vectors for 4-MV macroblocks

Figure 22 shows the candidate motion vectors for each of the 4 luma blocks in a 4-MV macroblock. The neighboring macroblocks are either 1-MV or 4-MV macroblocks. If the candidate predictor is intra, outside the frame boundary, or part of a different slice, the candidate predictor shall be regarded as unavailable in the P picture coding mode. For the B picture coding, the candidate predictor with the backward motion vector shall be regarded as unavailable for forward-predicted motion vector, while the candidate predictor with the forward motion vector shall be regarded as unavailable for backward-predicted motion vector. The pseudo-code of Figure 23 specifies the process for calculating the motion vector predictors. At most 3 motion vector predictor candidates are used for deriving motion vector predictor. In the B picture coding mode, this prediction shall be applied two times for the forward prediction and the backward prediction.

```
neighbour_count = 0
if (Pred_A is available) {
  mvx[neighbour_count] = mvx[Pred_A]
  mvy[neighbour_count] = mvy[Pred_A]
  neighbour_count ++
}
if (Pred_B is available) {
  mvx[neighbour_count] = mvx[Pred_B]
  mvy[neighbour_count] = mvy[Pred_B]
  neighbour_count ++
}
if (!block 3 of 4-MV macroblock && Pred_C is available) {
  mvx[neighbour_count] = mvx[Pred_C]
  mvy[neighbour_count] = mvy[Pred_C]
  neighbour_count ++
}
if (neighbour_count<3 && Pred_D is available) {
  mvx[neighbour_count] = mvx[Pred_D]
  mvy[neighbour_count] = mvy[Pred_D]
  neighbour_count ++
}
if (neighbour_count<3 && block 0 of 4-MV macroblock && Pred_E is available) {
  mvx[neighbour_count] = mvx[Pred_E]
  mvy[neighbour_count] = mvy[Pred_E]
  neighbour_count ++
}
if (neighbour_count<3 && 1-MV macroblock && Pred_E is available) {
  mvx[neighbour_count] = mvx[Pred_E]
  mvy[neighbour_count] = mvy[Pred_E]
  neighbour_count ++
}
if (neighbour_count == 0){
  mvpx = 0
  mvpy = 0
}
else if (neighbour_count < 3){
  mvpx = mvx[0]
  mvpy = mvy[0]
}
else {
   if(|mvx[0]- mvx[1]| < |mvx[1]- mvx[2]|) {
     mvpx = mvx[0]
   }
   else {
     mvpx = mvx[2]
   }
   if(|mvy[0]- mvy[1]| < |mvy[1]- mvy[2]|) {
     mvpy = mvy[0]
   }
   else {
     mvpy = mvy[2]
   }
}
```

Figure 23 – Pseudo-code for motion vector predictor derivation

9.2.3.3 Motion Vector

In order to calculate the motion vectors of the current macroblock, the motion vector differentials (MVDx, MVDy, MVD2x and MVD2y) shall be decoded. The pseudo-code of Figure 24 describes the procedure for decoding MVDx, MVDy, MVD2x or MVD2y. MVD2x and MVD2y shall be decoded whenever the current macroblcok is coded with the interpolated mode.

```
//MVD represents the motion vector differential in x-axis or y-axis
decode_MVD_symbol() {
  for(numZeros=0; ; numZeros++)
      if(get_bits(1) == 1)
          break;

  if(numZeros > 0) {
      val = (1 << numZeros) + get_bits(numZeros) - 1;
      if ( (val & 1) == 1)
          MVD = (val+1) >> 1;
      else
          MVD = -(val >> 1);
  }
  else
      MVD = 0;

   return MVD;
}
```

Figure 24 – Pseudo-code of MVD decoding

After decoding MVD, the motion vector shall be calculated according to equation (9-4):

$$MVx = PMVx + MVDx$$
$$MVy = PMVy + MVDy \quad \textbf{(9-4)}$$

where *PMVx* and *PMVy* are motion vector predictors which are obtained from section 9.2.3.2, and *MVDx* and *MVDy* are the motion vector differentials.

If the current macroblock is coded as the interpolated mode, *MVx* and *MVy* are components of the forward motion vector and the backward motion vector shall be additionally calculated according to equation (9-5):

$$MV2x = PMV2x + MVD2x$$
$$MV2y = PMV2y + MVD2y \quad \textbf{(9-5)}$$

where *PMV2x* and *PMV2y* are backward motion vector predictors which are obtained from Section 9.2.3.2, and *MVD2x* and *MVD2y* are the backward motion vector differentials.

9.2.3.4 Motion compensation

The motion vectors calculated in Section 9.2.3.3 shall be used to obtain the predicted block in the reference frame. The horizontal and vertical motion vector components represent the displacement between the block currently being decoded and the corresponding location in the reference frame. Positive values represent locations that are below and to the right of the current location. Negative values represent locations that are above and to the left of the current location. The actual reconstructed motion vector shall be used by subsequent macroblocks as a reference for the motion vector predictor calculation. The motion vector shall be adjusted to point the area [-18:PICTURE_WIDTH+17] x [-18:PICTURE_HEIGHT+17] before the motion compensation. Here, each location in the outside of the actual image shall be filled with the nearest image pixel.

The following operation shall be applied to obtain the predicted luma and color difference samples.

- Let (iXCoord, iYCoord) be the spatial location given in full-sample units of the top left corner of the current macroblock or block (e.g. if the current macroblock is located on the 2nd row and 3rd column, then iXCoord = 2 * 16 and iYCoord = 1 * 16).

For each luma sample location (0 <= x < BlockWidth, 0 <= y < BlockHeight), the corresponding predicted luma sample value shall be derived as follows:

- Let (*MVx*, *MVy*) be the motion vector obtained from Section 9.2.3.3
- Let (iMvX, iMvY) be the reconstructed luma motion vector in quarter pixel unit.
  If MV_MODE == 2, iMvX = *MVx* << 2 and iMvY = *MVy* << 2.
  If MV_MODE == 1, iMvX = *MVx* << 1 and iMvY = *MVy* <<1.
  If MV_MODE == 0, iMvX = *MVx* and iMvY = *MVy*.
- Let (xIntLuma, yIntLuma) be the spatial luma location in full-sample units, and (xFracLuma, yFracLuma) be an offset given in quarter sample units.

Then the spatial location of predicted luma block shall be computed according to the following equations.

- xIntLuma = (iXCoord * 4 + iMvX) >> 2
- yIntLuma = (iYCoord * 4 + iMvY) >> 2

- xFracLuma = iMvX & 3
- yFracLuma = iMvY & 3

The predicted luma sample values are derived by invoking the process as in Section 9.2.3.5.

For each color difference sample location (0 <= x < BlockWidthChr, 0 <= y < BlockHeightChr), the corresponding predicted color difference sample value shall be derived as follows:

- Let (*MVx*, *MVy*) be the motion vector obtained from Section 9.2.3.3.
- Let (iMvX, iMvY) be the reconstructed color difference motion vector in quarter pixel unit.
  If MV_MODE == 2, iMvX = *MVx* << 2 and iMvY = *MVy* << 2.
  If MV_MODE == 1, iMvX = *MVx* << 1 and iMvY = *MVy* << 1.
  If MV_MODE == 0, iMvX = *MVx* and iMvY = *MVy*.
- Let (xIntChr, yIntChr) be the spatial color difference location in full-sample units, and (xFracChr, yFracChr) be an offset given in quarter sample unit.

Then the spatial location of predicted color difference block shall be computed according to the following equations.

If COLORDIFF_FORMAT == 0 (4:2:0), then the following operations shall be performed.

- xIntChr = (iXCoord * 4 + iMvX) >> 3
- yIntChr = (iYCoord * 4 + iMvY) >> 3

- xFracChr = iMvX & 7
- yFracChr = iMvY & 7

If COLORDIFF_FORMAT == 1 (4:2:2), then the following operations shall be performed.

- xIntChr = (iXCoord * 4 + iMvX) >> 3
- yIntChr = (iYCoord * 4 + iMvY) >> 2

- xFracChr = iMvX & 7
- yFracChr = iMvY & 3

If COLORDIFF_FORMAT == 2 (4:4:4), then the following operations shall be performed.

- xIntChr = (iXCoord * 4 + iMvX) >> 2
- yIntChr = (iYCoord * 4 + iMvY) >> 2

- xFracChr = iMvX & 3
- yFracChr = iMvY & 3

The predicted color difference sample values are derived by invoking the process in Section 9.2.3.6.

9.2.3.5 Luma sample interpolation process

Integer pixel motion vector values do not require the computation of interpolated pixels. For non-integer pixel motion vector values, samples at fractional sample positions shall be derived from the 4-nearest integer pixel position using a bi-linear interpolation filter as shown in Figure 25.

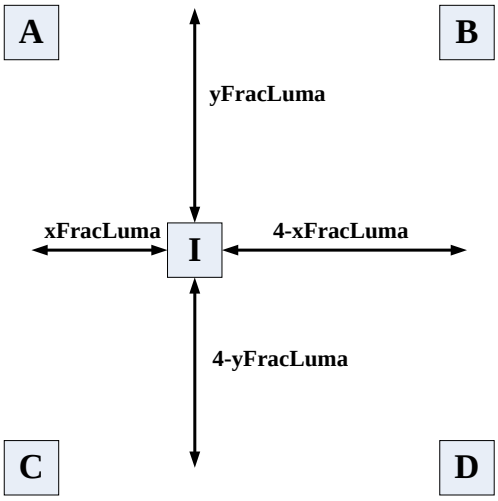

Figure 25 – Interpolation of Luma fourth-sample positions

Each sample in the reference picture shall be mapped by using the motion vector in full-sample unit and offset value in Section 9.2.3.4. Using a bi-linear interpolation filter, each luma sample at fractional sample positions shall be interpolated according to the following:

LumaInterpolated = ((4 – xFracLuma) * (4 – yFracLuma) * A + xFracLuma * (4 – yFracLuma) * B
+ (4 – xFracLuma) * yFracLuma * C + xFracLuma * yFracLuma * D + 8) >> 4

where A, B, C and D are the neighboring integer-position samples.

9.2.3.6 Color difference sample interpolation process

The color difference sample interpolation process also employs a bi-linear interpolation like the luma sample interpolation. If the precision of the motion vectors is quarter-pel, the range of xFracChr shall be from zero to seven when COLORDIFF_FORMAT == 0 (4:2:0) or COLORDIFF_FORMAT == 1 (4:2:2), while the range of xFracChr shall be from zero to three when COLORDIFF_FORMAT == 2 (4:4:4). The range of the yFracChr shall be from zero to seven if COLORDIFF_FORMAT == 0 (4:2:0), while the range of the yFracChr shall be from zero to three if COLORDIFF_FORMAT == 1 (4:2:2) or if COLORDIFF_FORMAT == 2 (4:4:4).

Each color difference sample at fractional sample shall be derived as follows. A, B, C and D are the same as shown in Figure 25.

If COLORDIFF_FORMAT == 0, the interpolated color difference sample shall be
ColorDifferenceInterpolated = ((8 – xFracChr) * (8 – yFracChar) * A + xFracChr * (8 – yFracChar) * B
+ (8 – xFracChr) * yFracChar * C + xFracChr * yFracChar * D + 32) >> 6

If COLORDIFF_FORMAT == 1, the interpolated color difference sample shall be
ColorDifferenceInterpolated = ((8 – xFracChr) * (4 – yFracChar) * A + xFracChr * (4 – yFracChar) * B
+ (8 – xFracChr) * yFracChar * C + xFracChr * yFracChar * D + 16) >> 5

If COLORDIFF_FORMAT == 2, the interpolated color difference sample shall be
ColorDifferenceInterpolated = ((4 – xFracChr) * (4 – yFracChar) * A + xFracChr * (4 – yFracChar) * B
+ (4 – xFracChr) * yFracChar * C + xFracChr * yFracChar * D + 8) >> 4

9.2.3.7 Direct mode for B picture

The direct mode shall be selected for decoding the current macroblock whenever B_PRED_MODE == 0 and B_PICTURE_FLAG == 1. This mode shall not use the explicitly coded motion vectors but use the predicted motion vectors. The motion vector prediction method is defined in Section 9.2.3.2. There are four cases for prediction for the current macroblock depending on the motion vector prediction.

If all of the neighboring macroblocks have only forward motion vectors, the current macroblock shall be compensated with only the forward-predicted motion vector.

If all of the neighboring macroblocks have only backward motion vectors, the current macroblock shall be compensated with only the backward-predicted motion vector.

If both the forward-predicted motion vector and the backward-predicted motion vector exist, the current macroblock shall be compensated with the interpolated macroblock from both the forward-predicted and the backward-predicted motion vector. The interpolated macroblock shall be made using equation (9-6)

If the above and left blocks of the current macroblock are coded in intra mode or outside frame, both the forward-predicted motion vector and the backward-predicted motion vector shall be set to zeros. The current macroblock shall be compensated with these zero motion vectors. The interpolated macroblock shall be made using equation (9-6).

Unlike the skip mode, this mode shall use the transmitted transform coefficients with the compensated data to reconstruct residuals

9.2.3.8 Interpolated mode for B Picture

The interpolated mode shall be selected for decoding the current macroblock whenever B_PRED_MODE == 3 and B_PICTURE_FLAG == 1. This mode shall use two explicitly-coded motion vectors for the motion compensation. One motion vector points to a past reference picture and the other one points to a future reference picture. Each prediction shall be performed with each motion vector to each reference frame in the same manner as described in Section 9.2.3.4. The interpolation from each prediction shall be a pixel average operation with round-up to compute the pixels in the motion compensated macroblock

$$I(x, y) = (P1(x, y) + P2(x, y) + 1) >> 1 \quad \textbf{(9-6)}$$

where $I(x, y)$ is the interpolated pixel from both predicted macroblocks, $P1(x, y)$ is the pixel of the first predicted macroblock, and $P2(x, y)$ is the pixel of the second predicted macroblock.

9.2.4 Block Layer Decoding at Progressive Inter Frame Picture

The decoding process for coded blocks of progressive inter pictures shall be the same as described in Section 9.1.4 except for the residual reconstruction. In P pictures, the output of the inverse transform shall be added to the result of the motion-compensation. The pseudo-code in Figure 26 describes the reconstruction process for the inter picture.

```
// rec[][] represents reconstructed residual samples in the current picture
// Idct[][] represents output of the inverse DCT for luma or color difference samples in current picture
// MC[][] represents motion-compensated data in the current picture
// (iXCoord, iYCoord) represents the luma or color difference location given in full-sample units of the top left corner of the current macroblock
// Height represents the height for luma or color difference of the current macroblock
// Width represents the width for luma or color difference of the current macroblock

for(row = 0; row < Height; row++) {
    for(col = 0; col < Width; col++) {
            x = iXCoord+col;
            y = iYCoord+row;
           rec[x][y] = CLIP(Idct[x][y] + MC[x][y]);
    }
}
```

Figure 26 – Pseudo-code for residual block reconstruction in the inter picture

If the syntax element QUALITY_REFINEMENT_FLAG specifies that the quality refinement is used, the following process is invoked. In the quality based layer, a residual block shall be reconstructed by adding the the motion-compensated block to the inverse transformed block as decribed in Figure 26. The reconstructed residual samples shall construct the quality base layer picture. The collocated inverse transform blocks of higher quality layers to the block of the quality base layer shall be used to refine the inverse transform block of the quality base layer. Finally, the refined inverse transform block shall be added to the result of the motion-compensated block. The pseudo-code in Figure 27 describes progressive refinement process for inter picture. After quality refinement process, the reconstructed residual samples shall construct quality enhancement layer picture.

```
// rec[][] represents reconstructed residual samples in the current picture
// Idct[k][][] represents output of inverse DCT for luma or color difference samples in k'th quality layer
// Idct_qe[][] represents final output of inverse DCT for luma or color difference samples of quality enhancement layer
// MC[][] represents motion-compensated data in the current picture
// (iXCoord, iYCoord) represents the luma or color difference location given in full-sample units of the top left corner of the current macroblock
// Height represents the height for luma or color difference of the current macroblock
// Width represents the width for luma or color difference of the current macroblock
// Num_quality_layer[i][j] represents the number of quality layer for the residue sample located at (i,j) position in the current picture

for(row = 0; row < Height; row++) {
    for(col = 0; col < Width; col++) {
        x = iXCoord+col;
        y = iYCoord+row;
        N = Num_quality_layer[x][y];
        Idct_qe[x][y] = Idct[0][x][y];
        for(layer = 0; layer < N; layer++)
        {
            Idct_qe[x][y] = Idct_qe[x][y] + Idct[layer+1][x][y]
        }
        rec[x][y] = CLIP(Idct_qe[x][y] + MC[x][y]);
    }
}
```

Figure 27 – Pseudo-code for residual block reconstruction using quality refinement in the inter picture

9.3 Decoding Process for Adaptive Variable Length Code (VLC)

This is the specification for the VLC algorithm used in this document. It is commonly known as "Adaptive Huffman Coding". Various syntax elements are coded using the adaptive VLC code. The syntax elements include SKIP_RUN, IPRED_MODE, INTRA_SIG, and syntaxes for the coded significant pattern (CSP) in the macroblock layer; the syntax elements for MB-level CSP: MB_SIG_MODE, YUV_CSP, Y_CSP, Y0Y1_CSP, Y2Y3_CSP, UV_CSP, U0U1_CSP and V0V1_CSP in the macroblock layer; and the coefficient-significant type (CST) of block layer which are represented by: DC_AC_CST, L2AC16_CST[i], L2AC12_CST[i], L2AC8_CST[i], L1AC4_CST[i], L1AC3_CST[i], L1AC2_CST[i], AC_CST, ACp1_CST, ACp2_CST, ACp3_CST, L1[i]_CST, L2[1]_CST, L2[2]_CST, L2[3]_CST. DC_AC_CST, L2AC16_CST[i], L2AC12_CST[i], L2AC8_CST[i], L1AC4_CST[i], L1AC3_CST[i] and L1AC2_CST[i] use the adaptive VLC code with alphabet size of 3 or 5. AC_CST, ACp1_CST, ACp2_CST, and ACp3_CST use the adaptive VLC code with alphabet size of 7 or 19. L1[i]_CST, L2[1]_CST, L2[2]_CST, L2[3]_CST use the adaptive VLC code with alphabet size of 15.

IPRED_MODE, INTRA_SIG use the adaptive VLC code alphabet size of three. SKIP_RUN and MB_SIG_MODE use the adaptive VLC code alphabet size of five. Other syntaxes use the alphabet code of three or five, where the selection of the alphabet code is determined by the other syntax elements previously decoded For example, the alphabet size of YUV_CSP shall be determined by MB-level CSP induced from MB_SIG_MODE, MB_SIG_4 or INTRA_SIG. In the case of Y_CSP, the alphabet size shall be determined by the decoded YUV_CSP.

Table 54 shows the VLC code table when the alphabet size is three. The pseudo code of Figure 28 shows the decoding procedure of the VLC code with three alphabets. Figure 28 includes the definition of the structure AdaptiveVLC3 which is the set of variables for adaptive VLC decoding with three alphabets.

Table 54 – VLC codeword with 3 alphabets

Value	Binary VLC Codeword
0	0b
1	10b
2	11b

```
// AdaptiveVLC3: structure for adaptive VLC decoding with 3 alphabets
// Adapt_vlc3[19]: 19 structures for adaptive VLC decoding with 3 alphabets
//                 19 syntaxes are decoded with adaptive VLC code with 3 alphabets
// Occurrence[3]: probability models for 3 alphabets for each syntax
// Symbol[3]: storage for three alphabets
// SWAP(): the exchange function of Symbols and Occurrence
//          for decoding syntax with a probability model
//

typedef struct {
   UINT16   Occurrence[3];
   UINT16   Symbol[3];
} AdaptiveVLC3;

AdaptiveVLC3 Adapt_vlc3[19]; // declare 19 structure variables for adaptive VLC coding


SWAP(&A, &B) {
        Temp = A;
        A = B;
        B = Temp;
}

AdaptiveVLC3_reset(idx) {
       for(i=0; i < 3; i++) {
            Adapt_vlc3[idx].Occurrence[i]=0;
            Adapt_vlc3[idx].Symbol[i]= i;
        }
}

AdaptiveVLC3_update(idx) {
       while(Adapt_vlc3[idx].Occurrence[0] > 256) {
             for(i=0; i < 3;i++)
                  Adapt_vlc3[idx].Occurrence[i] >>= 1;
       }
}

AdaptiveVLC3_decodeSymbol(idx) {
    Value = decode_vlc_code();// using Table 54

     Symbol = Adapt_vlc3[idx].Symbol[Value];
     Adapt_vlc3[idx].Occurrence[Value]++; //Update Occurrence

    // Make sure
    // Adapt_vlc3[idx].Occurrence[Value-1] >= Adapt_vlc3[idx].Occurrence[Value]
    if(Value > 0 && Adapt_vlc3[idx].Occurrence[Value] > Adapt_vlc3[idx].Occurrence[Value-1]) {
                SWAP(Adapt_vlc3[idx].Symbol[Value], Adapt_vlc3[idx].Symbol[Value-1]);
                SWAP(Adapt_vlc3[idx].Occurrence[Value], Adapt_vlc3[idx].Occurrence[Value-1]);
    }

    return Symbol;
}
```

Figure 28 – Pseudo-code for adaptive VLC decoding with 3 alphabets

In Figure 28, ***AdaptiveVLC3_reset()*** and ***AdaptiveVLC3_update()*** are defined for initializing/re-initializing the adaptive VLC decoder and updating the probability model, respectively. The function ***AdaptiveVLC3_decodeSymbol()*** decodes the adaptive VLC code. Each element of the array 'Occurrence[]' has the frequency of each symbol. This array shall be sorted in descending order after decoding the VLC code. In the function ***AdaptiveVLC3_decodeSymbol()***, the function *decode_vlc_code()* decodes Value using the VLC table of Table 54 and idx shall indicate which syntax is decoding among syntaxes of Table 59 where idx also means the index of the probability model of the corresponding syntax.

Table 55 shows the VLC code table with five alphabets. The pseudo code of Figure 29 shows the decoding procedure of the VLC code with five alphabets. Figure 29 includes the definition of the structure AdaptiveVLC5 which is the set of variables for adaptive VLC decoding with five alphabets.

Table 55 – VLC codeword with 5 alphabets

Value	Binary VLC Codeword (Table_Index == 0)	Binary VLC Codeword (Table_Index == 1)	Binary VLC Codeword (Table_Index == 2)
0	0b	0b	00b
1	10b	100b	01b
2	110b	101b	10b
3	1110b	110b	110b
4	1111b	111b	111b

In Figure 29, ***AdaptiveVLC5_reset()*** and ***AdaptiveVLC5_update()*** are defined for initializing/re-initializing the adaptive VLC decoder and updating the probability model, respectively. The ***AdaptiveVLC5_update()*** of Figure 29 is different from that of Figure 28. The differences are induced from three different alphabet tables. When the decoder selects the VLC code with five alphabets, one table among three different tables of Table 55 shall be selected. The selection shall be adaptively changed based on average bit length. The function ***AdaptiveVLC5_update()*** also finds the table which has the minimum average bit length at the updating time. The function ***AdaptiveVLC5_decodeSymbol()*** decodes each symbol based on the selection of the table and the probability model. In the function ***AdaptiveVLC5_decodeSymbol()***, the function *decode_vlc_code()* decodes Value using the VLC table of Table 55 and idx shall indicate which syntax is decoding among syntaxes of Table 60 where idx also means the index of the probability model of the corresponding syntax.

```
// AdaptiveVLC5: structure for adaptive VLC decoding with 5 alphabets
// Adapt_vlc5[24]: 24 structures for adaptive VLC decoding with 5 alphabets
//                 24 syntaxes are decoded with adaptive VLC code with 5 alphabets
// Occurrence[5]: probability models for 5 alphabets for each syntax
// Symbol[5]: storage for 5 alphabets
// initSymbol5[5]: the initial symbol values for the case of using 5 alphabets when idx < 22
// Table_Index: the selected table index of 3 tables of Table 55
// SWAP(): the exchange function of Symbols and Occurrence
//         for decoding syntax with a probability model
//

typedef struct {
   UINT16   Occurrence[5];
   UINT16   Symbol[5];
   UINT16   Table_Index;
} AdaptiveVLC5;

AdaptiveVLC5 Adapt_vlc5[24]; // declare 24 structure variables for adaptive VLC coding
```

```
SWAP(&A, &B)
{
        Temp = A;
        A = B;
        B= Temp;
}

AdaptiveVLC5_reset(idx) {
        initSymbol5[5] = {10, 9, 8, 6, 2 };
        Adapt_vlc5[idx].Table_Index = 0;
        if(idx < 22) {
                for(i=0; i < 5; i++) {
                        Adapt_vlc5[idx].Occurrence[i]=0;
                        Adapt_vlc5[idx].Symbol[i]= initSymbol5[i];
                }
        }
        else if(idx == 33 || idx == 34) {
                for(i=0; i < 5; i++) {
                        Adapt_vlc5[idx].Occurrence[i]=0;
                        Adapt_vlc5[idx].Symbol[i]= i;
                }
        }
}

AdaptiveVLC5_update(idx) {
        while(Adapt_vlc5[idx].Occurrence[0] > 256) {
            for(i=0; i < 5;i++)
                Adapt_vlc5[idx].Occurrence[i] >>= 1;
        }
        //Table Selection Logic
        MinCost = Cost = Adapt_vlc5[idx].Occurrence[0]
                        + 2 * Adapt_vlc5[idx].Occurrence[1]
                        + 3 * Adapt_vlc5[idx].Occurrence[2]
                        + 4 * Adapt_vlc5[idx].Occurrence[3]
                        + 4 * Adapt_vlc5[idx].Occurrence[4];
        Adapt_vlc5[idx].Table_Index = 0;

        Cost = Adapt_vlc5[idx].Occurrence[0]
                        +3 * Adapt_vlc5[idx].Occurrence[1]
                        +3 * Adapt_vlc5[idx].Occurrence[2]
                        +3 * Adapt_vlc5[idx].Occurrence[3]
                        +3 * Adapt_vlc5[idx].Occurrence[4];
        if(Cost < MinCost) {
                Adapt_vlc5[idx].Table_Index = 1; MinCost = Cost ;
        }

        Cost = 2 * Adapt_vlc5[idx].Occurrence[0]
                        +2 * Adapt_vlc5[idx].Occurrence[1]
                        +2 * Adapt_vlc5[idx].Occurrence[2]
                        +3 * Adapt_vlc5[idx].Occurrence[3]
                        +3 * Adapt_vlc5[idx].Occurrence[4];
        if(Cost < MinCost) {
                Adapt_vlc5[idx].Table_Index = 2; MinCost = Cost ;
        }
}
```

```
AdaptiveVLC5_decodeSymbol(idx) {
    Value = decode_vlc_code(); // using Table 55

     Symbol = Adapt_vlc5[idx].Symbol[Value];
     Adapt_vlc5[idx].Occurrence[Value]++; //Update Occurrence

    //Make sure
    // Adapt_vlc5[idx].Occurrence[Value-1] >=Adapt_vlc5[idx].Occurrence[Value]
    if(Value > 0 && Adapt_vlc5[idx].Occurrence[Value] > Adapt_vlc5[idx].Occurrence[Value-1]) {
          SWAP(Adapt_vlc5[idx].Symbol[Value], Adapt_vlc5[idx].Symbol[Value-1]);
          SWAP(Adapt_vlc5[idx].Occurrence[Value], Adapt_vlc5[idx].Occurrence[Value-1]);
    }

    return  Symbol;
}
```

Figure 29 – Pseudo-code for adaptive VLC decoding with 5 alphabets

Table 56 shows the VLC code table with seven alphabets. The pseudo code of Figure 30 shows the decoding procedure of the VLC code with seven alphabets. Figure 30 includes the definition of the structure AdaptiveVLC7 which is the set of variables for adaptive VLC decoding with seven alphabets.

Table 56 – VLC codeword with 7 alphabets

Value	Binary VLC Codeword (Table_Index == 0)	Binary VLC Codeword (Table_Index == 1)	Binary VLC Codeword (Table_Index == 2)
0	10b	0b	10b
1	00b	111b	01b
2	111b	100b	00b
3	110b	1101b	110b
4	010b	1100b	1110b
5	0111b	1011b	11111b
6	0110b	1010b	11110b

In Figure 30, ***AdaptiveVLC7_reset()*** and ***AdaptiveVLC7_update()*** are defined for initializing/re-initializing the adaptive VLC decoder and updating the probability model, respectively. When the decoder selects the VLC code with seven alphabets, one table among three different tables of Table 56 shall be selected. The selection shall be adaptively changed based on average bit length. The function ***AdaptiveVLC7_update()*** also finds the table which has the minimum average bit length at the updating time. The function ***AdaptiveVLC7_decodeSymbol()*** decodes each symbol based on the selection of the table and the probability model. In the function ***AdaptiveVLC7_decodeSymbol()***, the function *decode_vlc_code()* decodes Value using the VLC table of Table 56 and idx shall indicate which syntax is decoding among syntaxes of Table 61 where idx also means the index of the probability model of the corresponding syntax.

```
// AdaptiveVLC7: structure for adaptive VLC decoding with 7 alphabets
// Adapt_vlc7[4]: 4 structures for adaptive VLC decoding with 7 alphabets
//                    4 syntaxes are decoded with adaptive VLC code with 7 alphabets
// Occurrence[7]: probability models for 7 alphabets for each syntax
// Symbol[7]: storage for 7 alphabets
// initSymbol7[7]: the initial symbol values for the case of using 7 alphabets
// BITS_Adapt_vlc7[3][7]: the number of bits for the selected table of Table 56 and symbol
// Table_Index: the selected table index of 3 tables of Table 56
// SWAP(): the exchange function of Symbols and Occurrence
//                 for decoding syntax with a probability model
//

typedef struct {
   UINT16   Occurrence[7];
   UINT16   Symbol[7];
   UINT16   Table_Index;
} AdaptiveVLC7;

AdaptiveVLC7 Adapt_vlc7[4]; // declare 4 structure variables for adaptive VLC coding

SWAP(&A, &B)
{
        Temp = A;
        A = B;
        B= Temp;
}

AdaptiveVLC7_reset(idx) {
        initSymbol7[7] = { 1, 4, 5, 16, 17, 20, 21 };
        Adapt_vlc7[idx].Table_Index = 0;
        for(i=0; i < 7; i++) {
          Adapt_vlc7[idx].Occurrence[i]=0;
          Adapt_vlc7[idx].Symbol[i] = initSymbol7[i];
        }
}

AdaptiveVLC7_update(idx) {
      BITS_Adapt_vlc7[3][7] = {       { 2, 2, 3, 3, 3, 4, 4 },
                                      { 1, 3, 3, 4, 4, 4, 4 },
                                      { 2, 2, 2, 3, 4, 5, 5 }     };
      while(Adapt_vlc7[idx].Occurrence[0] > 256) {
          for(i=0; i < 7;i++)
              Adapt_vlc7[idx].Occurrence[i] >>= 1;
      }
      //Table Selection Logic
      MinCost = 0xffffffff;
      for(tmpTable = 0; tmpTable < 3; tmpTable++) {
              Cost = 0;
              for(i = 0; i < 7; i++) {
                      Cost += Adapt_vlc7[idx].Occurrence[i] * BITS_ Adapt_vlc7[tmpTable][i];
              }
              if(Cost < MinCost) {
                      MinCost = Cost;
                      Adapt_vlc7[idx].Table_Index = tmpTable;
              }
      }
```

```
}

AdaptiveVLC7_decodeSymbol(idx) {
    Value = decode_vlc_code(); // using Table 56

     Symbol = Adapt_vlc7[idx].Symbol[Value];
     Adapt_vlc7[idx].Occurrence[Value]++; //Update Occurrence

    //Make sure
    // Adapt_vlc7[idx].Occurrence[Value-1] >=  Adapt_vlc7[idx].Occurrence[Value]
    if(value > 0 && Adapt_vlc7[idx].Occurrence[Value] > Adapt_vlc7[idx].Occurrence[Value-1]) {
          SWAP(Adapt_vlc7[idx].Symbol[Value], Adapt_vlc7[idx].Symbol[Value-1]);
          SWAP(Adapt_vlc7[idx].Occurrence[Value], Adapt_vlc7[idx].Occurrence[Value-1]);
    }

    return  Symbol;
}
```

Figure 30 – **Pseudo-code for adaptive VLC decoding with 7 alphabets**

Table 57 shows the VLC code table with 15 alphabets. The pseudo code of Figure 31 shows the decoding procedure of the VLC code with 15 alphabets. Figure 31 includes the definition of the structure AdaptiveVLC15 which is the set of variables for adaptive VLC decoding with 15 alphabets.

Table 57 – **VLC codeword with 15 alphabets**

Value	Binary VLC Codeword (Table_Index == 0)	Binary VLC Codeword (Table_Index == 1)	Binary VLC Codeword (Table_Index == 2)
0	01b	10b	0b
1	110b	00b	101b
2	001b	110b	100b
3	000b	011b	1110b
4	1110b	0101b	1100b
5	1001b	11111b	11010b
6	11110b	01001b	111110b
7	10111b	01000b	111101b
8	10110b	111101b	111100b
9	10101b	111100b	110110b
10	10100b	111010b	1111111b
11	10001b	111001b	1101111b
12	10000b	111000b	1101110b
13	111111b	1110111b	11111101b
14	111110b	1110110b	11111100b

In Figure 31, ***AdaptiveVLC15_reset()*** and ***AdaptiveVLC15_update()*** are defined for initializing/re-initializing the adaptive VLC decoder and updating the probability model, respectively. When the decoder selects the VLC code with 15 alphabets, one table among three different tables of Table 57 shall be selected. The selection shall be adaptively changed based on average bit length. The function ***AdaptiveVLC15_update()*** also finds the table which has the minimum average bit length at the updating time. The function ***AdaptiveVLC15_decodeSymbol()*** decodes each symbol based on the selection of the table and the probability model. In the function ***AdaptiveVLC15_decodeSymbol()***, the function *decode_vlc_code()* decodes Value using the VLC table of Table 57 and idx shall indicate which syntax is decoding among syntaxes of Table 62 where idx also means the index of the probability model of the corresponding syntax.

```
// AdaptiveVLC15: structure for adaptive VLC decoding with 15 alphabets
// Adapt_vlc15[5]: 5 structures for adaptive VLC decoding with 15 alphabets
//                      5 syntaxes are decoded with adaptive VLC code with 15 alphabets
// Occurrence[15]: probability models for 15 alphabets for each syntax
// Symbol[15]: storage for 15 alphabets
// initSymbol15[15]: the initial symbol values for the case of using 15 alphabets
// BITS_Adapt_vlc15[3][15]: the number of bits for the selected table of Table 57 and symbol
// Table_Index: the selected table index of 3 tables of Table 57
// SWAP(): the exchange function of Symbols and Occurrence
//                  for decoding syntax with a probability model
//

typedef struct {
   UINT16   Occurrence[15];
   UINT16   Symbol[15];
   UINT16   Table_Index;
} AdaptiveVLC15;

AdaptiveVLC15 Adapt_vlc15[5]; // declare 5 structure variables for adaptive VLC coding

SWAP(&A, &B)
{
          Temp = A;
          A = B;
          B= Temp;
}

AdaptiveVLC15_reset(idx) {
          initSymbol15[15] = { 1, 4, 5, 16, 17, 20, 21, 64, 65, 68, 69, 80, 81, 84, 85 };
          Adapt_vlc15[idx].Table_Index = 0;
          for(i=0; i < 15; i++) {
             Adapt_vlc15[idx].Occurrence[i]=0;
             Adapt_vlc15[idx].Symbol[i] = initSymbol15[i];
          }
}

AdaptiveVLC15_update(idx) {
        BITS_Adapt_vlc15[3][15] = {       { 2, 3, 3, 3, 4, 4, 5, 5, 5, 5, 5, 5, 6, 6 },
                                          { 2, 2, 3, 3, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7 },
                                          { 1, 3, 3, 4, 4, 5, 6, 6, 6, 6, 7, 7, 7, 8, 8 }          };
        while(Adapt_vlc15[idx].Occurrence[0] > 256) {
            for(i=0; i < 15;i++)
                Adapt_vlc15[idx].Occurrence[i] >>= 1;
        }
        //Table Selection Logic
        MinCost = 0xffffffff;
        for(tmpTable = 0; tmpTable < 3; tmpTable++) {
                    Cost = 0;
                    for(i = 0; i < 15; i++) {
                                Cost += Adapt_vlc15[idx].Occurrence[i] * BITS_ Adapt_vlc15[tmpTable][i];
                    }
                    if(Cost < MinCost) {
                                MinCost = Cost;
                                Adapt_vlc15[idx].Table_Index = tmpTable;
                    }
        }
```

```
}

AdaptiveVLC15_decodeSymbol(idx) {
    Value = decode_vlc_code(); // using Table 57

     Symbol = Adapt_vlc15[idx].Symbol[Value];
     Adapt_vlc15[idx].Occurrence[Value]++; //Update Occurrence
    //Make sure
    // Adapt_vlc15[idx].Occurrence[Value-1] >=  Adapt_vlc15[idx].Occurrence[Value]
    if(Value > 0 && Adapt_vlc15[idx].Occurrence[Value] > Adapt_vlc15[idx].Occurrence[Value-1]) {
          SWAP(Adapt_vlc15[idx].Symbol[Value], Adapt_vlc15[idx].Symbol[Value-1]);
          SWAP(Adapt_vlc15[idx].Occurrence[Value], Adapt_vlc15[idx].Occurrence[Value-1]);
    }

    return  Symbol;
}
```

Figure 31 – **Pseudo-code for adaptive VLC decoding with 15 alphabets**

Table 58 shows the VLC code table with 19 alphabets. The pseudo code of Figure 32 shows the decoding procedure of the VLC code with 19 alphabets. Figure 32 includes the definition of the structure AdaptiveVLC19 which is the set of variables for adaptive VLC decoding with 19 alphabets.

Table 58 – **VLC codeword with 19 alphabets**

Value	Binary VLC Codeword (Table_Index == 0)	Binary VLC Codeword (Table_Index == 1)
0	10b	110b
1	011b	101b
2	001b	011b
3	1110b	010b
4	1100b	1111b
5	0001b	1110b
6	0000b	1000b
7	11110b	0010b
8	11011b	0001b
9	11010b	10011b
10	01010b	00111b
11	01001b	00110b
12	01000b	00000b
13	111111b	100101b
14	010110b	100100b
15	1111101b	0000100b
16	1111100b	0000101b
17	0101111b	0000110b
18	0101110b	0000111b

In Figure 32, ***AdaptiveVLC19_reset()*** and ***AdaptiveVLC19_update()*** are defined for initializing/re-initializing the adaptive VLC decoder and updating the probability model, respectively. When the decoder selects the VLC code with 19 alphabets, one table between two different tables of Table 58 shall be selected. The selection shall be adaptively changed based on average bit length. The function ***AdaptiveVLC19_update()*** also finds the table which has the minimum average bit length at the updating time. The function ***AdaptiveVLC19_decodeSymbol()*** decodes each symbol based on the selection of the table and the probability model. In the function ***AdaptiveVLC19_decodeSymbol()***, the function *decode_vlc_code()*

decodes Value using the VLC table of Table 58 and idx shall indicate which syntax is decoding among syntaxes of Table 63 where idx also means the index of the probability model of the corresponding syntax.

```
// AdaptiveVLC19: structure for adaptive VLC decoding with 19 alphabets
// Adapt_vlc19[4]: 4 structures for adaptive VLC decoding with 19 alphabets
//                      4 syntaxes are decoded with adaptive VLC code with 19 alphabets
// Occurrence[19]: probability models for 19 alphabets for each syntax
// Symbol[19]: storage for 19 alphabets
// initSymbol19[19]: the initial symbol value for the case of usning the 19 alphabets
// BITS_Adapt_vlc19[2][19]: the number of bits for the selected table of Table 58 and symbol
// Table_Index: the selected table index of 2 tables of Table 58
// SWAP():the exchange function of Symbols and Occurrence
//                 for decoding syntax with a probability model
//

typedef struct {
   UINT16   Occurrence[19];
   UINT16   Symbol[19];
   UINT16   Table_Index;
} AdaptiveVLC19;

AdaptiveVLC19 Adapt_vlc19[4]; // declare 4 structure variables for adaptive VLC coding

SWAP(&A, &B)
{
          Temp = A;
          A = B;
          B= Temp;
}

AdaptiveVLC19_reset(idx) {
          initSymbol19[19] = { 2, 6, 8, 9, 10, 18, 22, 24, 25, 26, 32, 33, 34, 36, 37, 38, 40, 41, 42 };
          Adapt_vlc19[idx].Table_Index = 0;
          for(i=0; i < 19; i++) {
             Adapt_vlc19[idx].Occurrence[i]=0;
             Adapt_vlc19[idx].Symbol[i] = initSymbol19[i];
          }
}

AdaptiveVLC19_update(idx) {
        BITS_Adapt_vlc19[2][19] = {       { 2, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 6, 6, 7, 7, 7, 7 },
                                          { 3, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5, 5, 6, 6, 7, 7, 7, 7 }          };
        while(Adapt_vlc19[idx].Occurrence[0] > 256) {
            for(i=0; i < 19;i++)
                Adapt_vlc19[idx].Occurrence[i] >>= 1;
        }
        //Table Selection Logic
        MinCost = 0xffffffff;
        for(tmpTable = 0; tmpTable < 2; tmpTable++) {
                    Cost = 0;
                    for(i = 0; i < 19; i++) {
                                Cost += Adapt_vlc19[idx].Occurrence[i] * BITS_ Adapt_vlc19[tmpTable][i];
                    }
                    if(Cost < MinCost) {
                                MinCost = Cost;
                                Adapt_vlc19[idx].Table_Index = tmpTable;
```

```
                    }
            }
}

AdaptiveVLC19_decodeSymbol(idx) {
    Value = decode_vlc_code(); // using Table 58

     Symbol = Adapt_vlc19[idx].Symbol[Value];
     Adapt_vlc19[idx].Occurrence[Value]++; //Update Occurrence

    //Make sure
    // Adapt_vlc19[idx].Occurrence[Value -1] >= Adapt_vlc19[idx].Occurrence[Value]
    if(Value > 0 && Adapt_vlc19[idx].Occurrence[Value] > Adapt_vlc19[idx].Occurrence[Value-1]) {
          SWAP(Adapt_vlc19[idx].Symbol[Value], Adapt_vlc19[idx].Symbol[Value-1]);
          SWAP(Adapt_vlc19[idx].Occurrence[Value], Adapt_vlc19[idx].Occurrence[Value-1]);
     }

    return  Symbol;
}
```

Figure 32 – **Pseudo-code for adaptive VLC decoding with 19 alphabets**

The probability models of the adaptive VLC codes shall be initialized at the beginning of each slice as shown in Figure 33. There shall be 56 models for adaptive VLC codes. 19 models shall be used for adaptive VLC codes with 3 alphabets, 24 models shall be used for adaptive VLC codes with 5 alphabets, 4 models shall be used for adaptive VLC codes with 7 alphabets, 5 models shall be used for adaptive VLC codes with 15 alphabets, and 4 models shall be used for adaptive VLC codes with 19 alphabets. This means that 19 kinds of syntaxes are decoded using the 3-alphabet VLC code, 24 kinds of syntaxes are decoded using the 5-alphabet VLC code, 4 kinds of syntaxes are decoded using the 7-alphabet VLC code, 4 kinds of syntaxes are decoded using the 19-alphabet VLC code, and 5 kinds of syntaxes are decoded using the 15-alphabet VLC code.

```
resetModel() {
  for(idx = 0; idx < 22; idx++) {
    if(idx == 7 || idx == 10 || idx == 13 || idx == 16 || idx == 19)
          continue;
    AdaptiveVLC3_reset(idx);
  }

  for(idx = 0; idx < 22; idx++) {
    AdaptiveVLC5_reset(idx);
  }

  for(idx = 22; idx < 26; idx++) {
    AdaptiveVLC7_reset(idx);
    AdaptiveVLC19_reset(idx);
  }

  for(idx = 26; idx < 31; idx++) {
    AdaptiveVLC15_reset(idx);
  }
  AdaptiveVLC3_reset(31);
  AdaptiveVLC3_reset(32);
  AdaptiveVLC5_reset(33);
  AdaptiveVLC5_reset(34);
}
```

Figure 33 – **Pseudo-code for model reset at new slice**

The probability models of adaptive VLC codes shall be updated every 16 nonzero macroblocks as shown in Figure 34 except for the probability model of SKIP_RUN in Table 60. The probability model update for SKIP_RUN shall beperformed whenever SKIP_RUN is decoded. After SKIP_RUN is decoded, the function ***AdaptiveVLC5_update(34)*** shall be performed.

```
updateModel() {
  for(idx = 0; idx < 22; idx++) {
    if(idx == 7 || idx == 10 || idx == 13 || idx == 16 || idx == 19)
          continue;
    AdaptiveVLC3_update(idx);
  }

  for(idx = 0; idx < 22; idx++) {
     AdaptiveVLC5_update(idx);
  }

  for(idx = 22; idx < 26; idx++) {
    AdaptiveVLC7_update(idx);
    AdaptiveVLC19_update(idx);
  }

  for(idx = 26; idx < 31; idx++) {
    AdaptiveVLC15_update(idx);
  }
  AdaptiveVLC3_update(31);
  AdaptiveVLC3_update(32);
  AdaptiveVLC5_update(33);
}
```

Figure 34 – Pseudo-code for model update per 16 nonzero macroblocks

19 types of syntaxes use the 3-alphabet code and 24 types of syntaxes use the 5-alphabet code. Table 59 and Table 60 define which syntax element uses which model index.

Table 59 – Probability model for VLC-coded syntaxes with three alphabets

Model Index	Syntax	Comment
0	YUV_CSP	
1	Y_CSP	
2	Y0Y1_CSP, Y2Y3_CSP	
3	UV_CSP, U_CSP, V_CSP	
4	U0U1_CSP, V0V1_CSP U2U3_CSP, V2V3_CSP	
5	DC_AC_CST	for Luma block
6	DC_AC_CST	for Color Difference block
7	N/A	Not Assigned
8	L2AC12_CST[1]	
9	L2AC8_CST[1]	
10	N/A	Not Assigned
11	L2AC12_CST[2]	
12	L2AC8_CST[2]	
13	N/A	Not Assigned
14	L2AC12_CST[3]	
15	L2AC8_CST[3]	

16	N/A	Not Assigned
17	L1AC3_CST[1~3]	
18	L1AC2_CST[1~3]	
19	N/A	Not Assigned
20	L1AC3_CST[4~15]	
21	L1AC2_CST[4~15]	
22~30	N/A	Not Assigned
31	IPRED_MODE	When Is_UpperMB_Skip() == 0 && Is_LeftMB_Skip() == 0 && (QUALITY_REFINEMENT_FLAG == 0 || (QUALITY_REFINEMENT_FLAG == 1 && QUALITY_LAYER_ID == 0))
32	INTRA_SIG	When IPRED_MODE != 0

Table 60 – Probability model for VLC-coded syntaxes with five alphabets

Model Index	Syntax	Comment
0	YUV_CSP	
1	Y_CSP	
2	Y0Y1_CSP, Y2Y3_CSP	
3	UV_CSP, U_CSP, V_CSP	
4	U0U1_CSP, V0V1_CSP, U2U3_CSP, V2V3_CSP	
5	DC_AC_CST	for Luma block
6	DC_AC_CST	for Color Difference block
7	L2AC16_CST[1]	
8	L2AC12_CST[1]	
9	L2AC8_CST[1]	
10	L2AC16_CST[2]	
11	L2AC12_CST[2]	
12	L2AC8_CST[2]	
13	L2AC16_CST[3]	
14	L2AC12_CST[3]	
15	L2AC8_CST[3]	
16	L1AC4_CST[1~3]	
17	L1AC3_CST[1~3]	
18	L1AC2_CST[1~3]	
19	L1AC4_CST[4~15]	
20	L1AC3_CST[4~15]	
21	L1AC2_CST[4~15]	
22~32	N/A	Not Assigned
33	MB_SIG_MODE	
34	SKIP_RUN	

4 types of syntaxes use the 7-alphabet code, 5 types of syntaxes use the 15-alphabet code, and 4 types of syntaxes use the 19-alphabet code. Table 61, Table 62 and Table 63 define which syntax element uses which model index.

Table 61 – Probability model for VLC-coded syntaxes with 7 alphabets

Model Index	Syntax	Comment
0~21	N/A	Not Assigned
22	AC_CST	
23	ACp1_CST	
24	ACp2_CST	
25	ACp3_CST	

Table 62 – Probability model for VLC-coded syntaxes with 15 alphabets

Model Index	Syntax	Comment
0~25	N/A	Not Assigned
26	L2[1]_CST	
27	L2[2]_CST	
28	L2[3]_CST	
29	L1[i]_CST	i=1, 2, and 3
30	L1[i]_CST	4<= i < 16

Table 63 – Probability model for VLC-coded syntaxes with 19 alphabets

Model Index	Syntax	Comment
0~21	N/A	Not Assigned
22	AC_CST	
23	ACp1_CST	
24	ACp2_CST	
25	ACp3_CST	

The syntaxes decoded from Table 59 (except for model indices of 31 and 32), Table 60 (except for model indices of 33 and 34), Table 61, Table 62 and Table 63 have dependences. The final symbol values from these syntaxes depend on the previously-decoded syntax elements. The dependency on the previously-decoded syntax elements is described in Section 8.3 and Section 8.4 in detail.

After decoding syntax elements corresponding to index values from 0 to 30, the corresponding decoded values have multiple meanings which are also explained in Section 8.3 and Section 8.4. For example, the decoded YUV_CSP represents: either the luma's CSP or the color difference's CSP. The pseudo-code of Figure 35 shows how these syntaxes shall be decoded. In this pseudo-code, the function ***AdaptiveVLC3_decodeSymbol()***, ***AdaptiveVLC5_decodeSymbol()***, ***AdaptiveVLC7_decodeSymbol()***, ***AdaptiveVLC15_decodeSymbol()*** and ***AdaptiveVLC19_decodeSymbol()*** are defined in Figure 28, Figure 29, Figure 30, Figure 31 and Figure 32, respectively. For example, in the case of idx == 0, the function ***decodeAdaptiveVLC***() shall decode YUV_CSP and type_symbol shall be the MB-level CSP induced from MB_SIG_MODE and MB_SIG. In the case of idx == 1, the function ***decodeAdaptiveVLC***() shall decode Y_CSP, and the input value type_symbol shall be the luma's CSP which is induced from ***value2*** decided when decoding YUV_CSP.

```
//type_symbol represents which alphabet code is selected
//              among 3-alphabet, 5-alphabet, 7-alphabet, 15-alphabet, or 19-alphabet when idx < 31..
//            0: No codeword
//            1: 3-alphabet code, 7-alphabet code or 15-alphabet code
//            2: 5-alphabet code or 19-alphabet code
//idx represents one among the model indices from Table 59 to Table 63.
//value1, value2, value3 and value4 are the decoded values.
//One Symbol values (value1) is obtained when idx == 31, idx == 32 or idx == 34
//Two symbol values (value1 and value2) are obtained when 3-alphabet or 5-alphabet code is used
//Three symbol values (value1, value2 and value3) are obtained
//                                          when 7-alphabet or 19-alphabet code is used
//Four symbol values (value1, value2 value3 and value4) are obtained when 15-alphabet code.
decodeAdaptiveVLC(type_symbol, idx, &value1, &value2, &value3, &value4) {
  if(idx <  5)  {
      if(type_symbol == 0)
            value1 = value2 = 0;
      else if(type_symbol == 1) {
            sym = AdaptiveVLC3_decodeSymbol(idx);
            value1 = ((sym+1) & 1)), value2 = (sym+1) >> 1;
      }
      else if(type_symbol == 2) {
            sym = AdaptiveVLC5_decodeSymbol(idx);
            value1 = sym & 0x03,   value2 = (sym & 0x0C) >> 2;
       }
  }
  else if(idx < 22) {
       if(type_symbol == 0)
            value1 = value2 = 0;
      else if(type_symbol == 1) {
            sym = AdaptiveVLC3_decodeSymbol(idx);
            value1 = (sym+1) >> 1, value2 = ((sym+1) & 1));
      }
      else if(type_symbol == 2) {
           sym = AdaptiveVLC5_decodeSymbol(idx);
           value1 = (sym & 0x0C) >> 2, value2 = sym & 0x03;
       }
  }
  else if(idx < 26) {
       if(type_symbol == 0)
          value1 = value2 = value3 = 0
      else if(type_symbol == 1) {
          sym = AdaptiveVLC7_decodeSymbol(idx);
          value1 = (sym & 0x30) >> 4, value2 = (sym & 0x0C) >> 2, value3 = sym & 0x03;
      }
      else {
          sym = AdaptiveVLC19_decodeSymbol(idx);
          value1 = (sym & 0x30) >> 4, value2 = (sym & 0x0C) >> 2, value3 = sym & 0x03;
      }
  }
  else if(idx < 31){
       if(type_symbol == 0)
            value1 = value2 = value3 = value4 = 0;
       else {
            sym = AdaptiveVLC15_decodeSymbol(idx);
            value1 = (sym & 0xC0) >> 6, value2 = (sym & 0x30) >> 4;
```

```
                ***value3*** = (sym & 0x0C) >> 2, ***value4*** = sym & 0x03;
        }
    }
    else {
        if(idx == 31 || idx == 32) { // IPRED_MODE or INTRA_SIG
            ***value1*** = AdaptiveVLC3_decodeSymbol(idx);
        }
        else if(idx == 34) { // SKIP_RUN (See Table 17)
            // ***value1*** = # of skipped macroblocks
            ***value1*** = AdaptiveVLC5_decodeSymbol(idx);
        }
        else if(idx == 33) { // MB_SIG_MODE
            //***value1***: MB-level CSP (See Table 18)
            //***value2***: MB Coding Mode (See Table 18)
            sym = AdaptiveVLC5_decodeSymbol(idx);
            if(sym < 4) {
                ***value1*** = (sym >> 1) + 1;
                ***value2*** = ((sym & 0x01) == 1)? '1-MV' : 'INTRA';
            }
            else {
                sym = get_bits(2); // MB_SIG_4 (See Table 19)
                if(sym < 3) {
                    ***value1*** = sym;
                    ***value2*** = '4-MV';
                }
                else {
                    ***value1*** = 0;
                    sym = get_bits(1); // MB_MODE_1 (See section 8.3.4)
                    if(sym == 0) ***value2*** = 'INTRA';
                    else ***value2*** = '1-MV';
                }
            }
        }
    }
}
```

Figure 35 – Pseudo-code for decoding CST or CSP syntaxes

9.4 Reference Frame List Management

There shall be only a single reference frame list for each layer. The frames at the current layer use only the frames of the current layer and not any other layer as the references because each layer shall be decoded independently as shown in Figure 36. The decoded picture buffer of layer 0 (base layer) shall be operated by the rule of the codec standard which is used to decode the base layer. The decoded picture buffer operation of other layers except for the layer 0 is described in this section.

Since the current layer has only one reference frame list, if LAYER_ENTRY_FLAG == 1, during decoding reference frame list at the current layer shall be made empty and the current frame shall be decoded in intra-coding mode. If REFERENCED_PICTURE_FLAG == 1, the current frame shall be inserted in the reference frame list after decoding the current frame. If REMOVE_ENTRY_FLAG == 1, one frame in the reference frame list shall be removed which is indicated by REMOVE_PICTURE_ID. Therefore, when the reference frame list is full and REFERENCED_PICTURE_FLAG == 1, then REMOVE_ENTRY_FLAG shall be set to 1 and the frame indicated by REMOVE_PICTURE_ID shall be replaced with the current decoded frame.

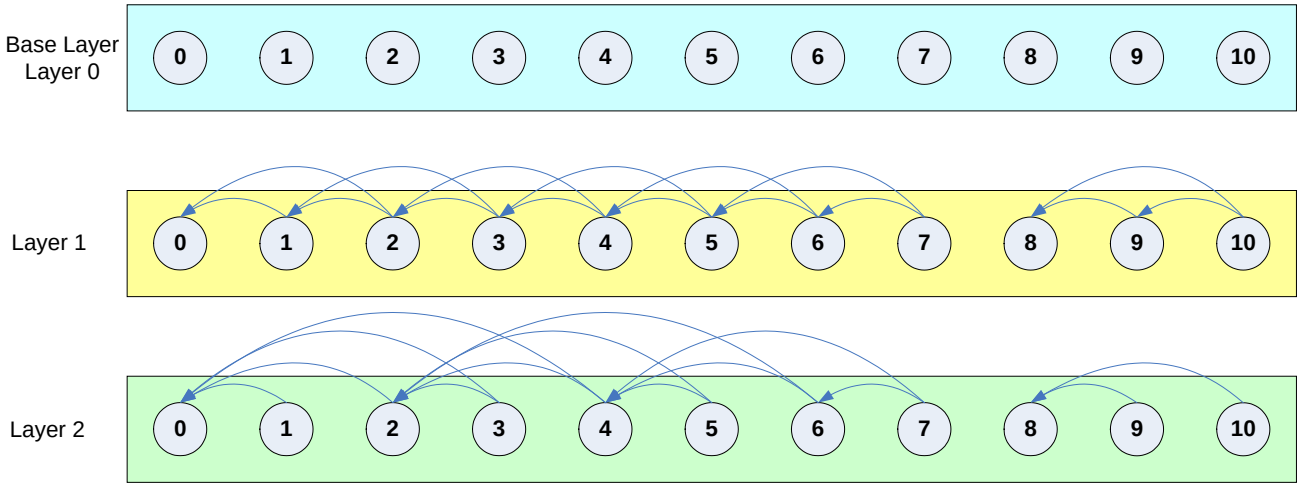

Figure 36 – Referencing operation in three-layer coding

When decoding frames, the first few frames shall occupy the empty locations of the reference frame list until the number of the coded pictures, whose REFERENCED_PICTURE_FLAG == 1, is equal to the size of the reference frame list, defined by NUM_REF_PICTURES.

The referencing operations are illustrated in Figure 36 when the number of total layers is two. In this case, layer 1 has the reference frame buffer with NUM_REF_PICTURES == 2 and all decoded frames can be used as references. Layer 2 also has the reference frame buffer with NUM_REF_PICTURES == 2 and even-numbered decoded frames can be used as references. Also in this case, the frame which is identified with REMOVE_PICTURE_ID in the corresponding buffer is removed from the reference frame list when the buffer is full at each layer, and the new frame is inserted. The frame to be removed shall be signaled with REMOVE_PICTURE_ID when REMOVE_ENTRY_FLAG is set to 1.

When decoding frames at layer 1, frame 0 is decoded in the intra-coding mode and occupies the first location of the reference frame list. Then frame 1 is decoded in the inter-coding mode using frame 0 as a reference and occupies the second location of the reference frame list. Since there is only one frame in the reference frame list, even if NUM_PREF_PICTURES == 2, frame 1 is decoded using one reference. When frame 2 is decoded, frame 0 or frame 1 can be used as a reference. After decoding frame 2, because the buffer is full, frame 0 is selected and removed from the reference frame list. The decoded frame 2 shall be inserted into the first location of the reference frame list. This procedure is repeated until a sequence is terminated or the frame where LAYER_ENTRY_FLAG == 1 is decoded. If LAYER_ENTRY_FLAG == 1 when frame 8 is decoded, all frames in the reference frame list are removed, frame 8 is decoded in intra-coding mode and frame 8 occupies the first location in the reference frame list.

When decoding frames at layer 2, frame 0 is decoded in the intra-coding mode and occupies the first location in the reference frame list of layer 2. Then frame 1 is decoded in the inter-coding mode using frame 0 as a reference but frame 1 is not inserted into the reference frame list. When frame 2 is decoded in the inter-coding mode using frame 0 as a reference, it is inserted into the second location of the reference frame list. If there is only one frame in the reference frame list, even if we set NUM_REF_PICTURES == 2, frame 2 is decoded using one reference which is frame 0. Frame 3 can use frame 0 or frame 2 as a reference. Frame 4 can use frame 0 or frame 2 as a reference. After decoding frame 4, frame 0 in the reference frame list is replaced with frame 4. This procedure is repeated until a sequence is terminated or the frame whose LAYER_ENTRY_FLAG == 1 is decoded. In Figure 36, LAYER_ENTRY_FLAG == 1 at frame 8 which is decoded in the intra-coding mode, all previous reconstructed frames in the reference frame list are removed, and frame 8 occupies the first location in the reference frame list at layer 2.

In the direct mode, the skip mode or the interpolated mode of the B picture coding mode, the past and the future frames can be referenced. Figure 36 shows the referencing operation when the only the forward

prediction is available. Figure 37 shows the referencing operation when the B picture coding is available. In Figure 37, numbers of the circles indicate the display order of each frame, the solid lines represent the forward prediction, and the dashed lines represent the backward prediction.

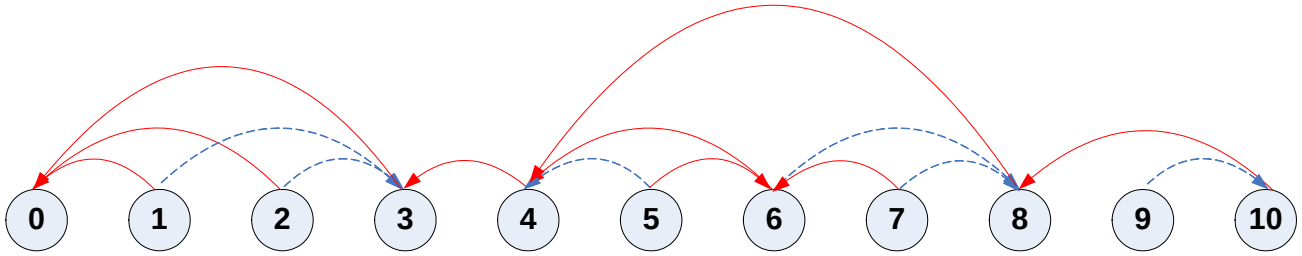

Figure 37 – Referencing operation when the forward and backward predictions are available

In this example, we assume NUM_REF_PICTURES == 3. The frame 0 is decoded in intra-coding mode, it is inserted in the reference frame list and REFERENCED_PICTURE_FLAG is set to 1. The frame 3 is decoded in the inter-coding mode with only the forward prediction. The frame 3 uses the frame 0 as a reference, the frame 3 also is inserted in the reference frame list and REFERENCED_PICTURE_FLAG is set to 1. The frame 1 and frame 2 are decoded in the B picture coding mode. The frame 1 and the frame 2 use the frame 0 and the frame 3 as references. Since the frame 1 and the frame 2 are not used as references for the other frames, both frames' REFERENCED_PICTURE_FLAGs are set to zero. The frame 4 is decoded in the P picture coding mode with forward prediction. Here, the frame 4 uses the frame 3 as a reference and the frame 4 is inserted into the reference frame list. When the frame 8 is decoded with the frame 4 in the forward-P picture coding mode, since the reference frame is full, the frame 0 is removed from the reference frame list and the frame 8 is inserted into the frame list after decoding the frame 8. The frame 6 is decoded in the B picture coding mode using the frame 4 and the frame 8. After decoding the frame 6, the frame 3 is removed from the reference frame list and the frame 6 is inserted into the reference frame list. The frame 5 is decoded with the frame 4 and the frame 6, and the frame 7 is decoded with the frame 6 and the frame 8. The frame 5 and the frame 7 are not inserted into the reference frame list. The frame 10 is decoded with the frame 8 in the forward-P picture coding mode and the frame 4 is removed from the reference frame list and the frame 10 is inserted to the reference frame list. Finally, the frame 9 is decoded with the frame 10 in the backward-P picture coding mode. If the frame 9 is used as a reference for future process, the frame 9 is inserted into the reference frame list and the frame 4 is removed from the reference frame list. Otherwise, the frame 9 is not inserted into the frame reference list. Therefore the decoding order for Figure 37 is frame 0, frame 3, frame 1, frame 2, frame 4, frame 8, frame 6, frame 5, frame 7, frame 10, and the frame 10. This decoding order shall be changed to the display order for the final rendering.

For random access, if LAYER_ENTRY_FLAG == 1 at the current frame of the current layer, LAYER_ENTRY_FLAG of the frame at the same time instance of the upper layer shall be set to one. In addition, the frame of the enhancement layer at the time instant of the random access point of the base layer shall have the picture header whose LAYER_ENTRY_FLAG is one.

When the quality refinement is used (i.e QUALITY_REFINEMENT_FLAG == 1), the current picture uses the reference pictures marked as the quality enhancement reference (QUALITY_REFERENCE_PICTURE_FLAG == 1) and the current frame is used as a reference for future decoding (i.e. REFERENCED_PICTURE_FLAG == 1), the reconstructed frame of the quality enhancement layer shall be inserted in the reference frame list. When the quality refinement is used (i.e QUALITY_REFINEMENT_FLAG == 1), the current picture uses the reference pictures marked as the quality base layer reference (QUALITY_REFERENCE_PICTURE_FLAG == 0) and the current frame is used as a reference for future decoding (i.e. REFERENCED_PICTURE_FLAG == 1), the both of the reconstructed quality base layer and the reconstructed quality enhancement layer of the current picture shall be inserted in the reference frame list. The quality base layer frame shall consist of the reconstructed residual samples by the slices of the quality base layer (i.e. QUALITY_LAYER_ID == 0). The quality enhancement layer frame shall consist of the reconstructed residue samples derived by the quality refinement process. In the reference frame list, the reconstructed frame of the quality base layer shall be

marked as the quality base reference and the reconstructed frame of the quality enhancement layer shall be marked as the quality enhancement reference. If REMOVE_PICTURE_ID indicates a frame in reference frame list, the reference frames marked as the quality base reference and marked as the quality enhancement reference shall be removed.

If QUALITY_REFERENCE_PICTURE_FLAG is equal to 1, the reference frames marked as quality enhancement reference shall be used as a reference. Otherwise, the reference frames marked as quality base reference shall be used as a reference.

10 Interlaced Bitstream Syntax and Semantics

The syntax and semantics of the picture level, slice level, macroblock level, and block level of the interlaced field pictures are specified in this section.

10.1 Picture-level Syntax and Semantics

Table 64 defines the bit stream syntax of the interlaced picture layer.

Table 64 – Interlaced picture layer bitstream

PICTURE LAYER( ) {	Number of bits	Descriptor	Reference
LAYER_ID	4	uimsbf	10.1.1
PICTURE_ID	8	uimsbf	10.1.2
CODED_ORDER	8	uimsbf	10.1.3
LAYER_ENTRY_FLAG	1	uimsbf	10.1.4
if(QUALITY_REFINEMENT_FLAG == 1) {			
MAX_QUALITY_LAYER_ID	4	uimsbf	10.1.5
QUALITY_REFERENCE_PICTURE_FLAG	1	uimsbf	10.1.6
}			
if(PAFF_FLAG == 1) {			
FRAME_CODED_FLAG	1	uimsbf	10.1.7
if(FRAME_CODED_FLAG == 0) {			
TOP_FIELD_FLAG	1	uimsbf	10.1.8
}			
}			
else {			
TOP_FIELD_FLAG	1	uimsbf	10.1.8
}			
if(TOP_FIELD_FLAG == 1 || FRAME_CODED_FLAG == 1) {			
REFERENCED_PICTURE_FLAG	1	uimsbf	10.1.9
if(REFERENCED_PICTURE_FLAG == 1) {			
REMOVE_ENTRY_FLAG	1	uimsbf	10.1.10
if(REMOVE_ENTRY_FLAG == 1) {			
REMOVE_PICTURE_ID	8	uimsbf	10.1.11
}			
}			
}			
if(LAYER_ENTRY_FLAG == 0) {			
MC_FLAG	1	uimsbf	10.1.12
}			
if(MC_FLAG == 1) {			
B_PICTURE_FLAG	1	uimsbf	10.1.13

if(B_PICTURE_FLAG == 0) {			
PRED_DIRECTION	1	uimsbf	10.1.14
}			
MV_MODE	variable size	vlclbf	10.1.15
for(i = 0; i < B_PICTURE_FLAG+1;i++) {			
REFERENCE_PICTURE_ID[i]	8	uimsbf	10.1.16
}			
}			
else {			
If(SEQ_INTRA_PRED_FLAG == 1) {			
PIC_INTRA_PRED_FLAG	1	uimsbf	10.1.17
}			
}			
if(P2I_METHOD == 1) {			
P2I_METHOD_FLAG	1	uimsbf	10.1.18
}			
SCAN_ORDER_TYPE	variable size	vlclbf	10.1.19
if(SCAN_ORDER_TYPE ==1) {			
SEQ_SCAN_ORDER_IDX	4	uimsbf	10.1.20
}			
else if(SCAN_ORDER_TYPE == 2) {			
for(i=0; i < 64;i++) {			
PIC_ORDER_DIFF_FLAG[i]	1	uimsbf	10.1.21
if(PIC_ORDER_DIFF_FLAG[i] == 1) {			
PIC_SCAN_ORDER[i]	6	uimsbf	10.1.22
}			
}			
}			
QP_PICTURE_UNIFORM_FLAG	1	uimsbf	10.1.23
QP_CHANNEL_UNIFORM_FLAG	1	uimsbf	10.1.24
if(QP_CHANNEL_UNIFORM_FLAG == 1) {			
QP_PICTURE	8	uimsbf	10.1.25
}			
else {			
QP_PICTURE_Y	8	uimsbf	10.1.26
QP_PICTURE_U	8	uimsbf	10.1.27
QP_PICTURE_V	8	uimsbf	10.1.28
}			
CHANNEL_SKIP_MODE	8	uimsbf	10.1.29
if(is_adaptive_color_difference upconversion()) {			
COLOR_DIFFERENCE_UPCONV_IDX_U	2	uimsbf	10.1.30
COLOR_DIFFERENCE_UPCONV_IDX_V	2	uimsbf	10.1.31
}			
RES_SCALING_PRESENT_FLAG	1	uimsbf	10.1.32
if(RES_SCALING_PRESENT_FLAG == 1) {			
if(!CHANNEL_SKIP_MODE & 0x80) {			
SCALE_Y	9	uimsbf	10.1.33
SHIFT_Y	16	uimsbf	10.1.34
NORM_Y	4	uimsbf	10.1.35
}			
if(!CHANNEL_SKIP_MODE & 0x40) {			
SCALE_U	9	uimsbf	10.1.36
SHIFT_U	16	uimsbf	10.1.37

NORM_U	4	uimsbf	10.1.38
}			
if(!CHANNEL_SKIP_MODE & 0x20) {			
SCALE_V	9	uimsbf	10.1.39
SHIFT_V	16	uimsbf	10.1.40
NORM_V	4	uimsbf	10.1.41
}			
}			
if (STEREOSCOPIC_VIEW_FLAG == 1) {			
for('all macroblocks') {			
if(MC_FLAG == 1){			
VIEW_PRED_FLAG	1	uimsbf	10.1.42
}			
PRED_BLK_SIZE_FLAG	1	uimsbf	10.1.43
if(PRED_BLK_SIZE_FLAG == 0) {			
DMVDx	variable size	evlclbf	10.1.44
DMVDy	variable size	evlclbf	10.1.45
}			
else {			
for(i = 0; i < 4; i++) {			
DMVDx[i]	variable size	evlclbf	10.1.44
DMVDy[i]	variable size	evlclbf	10.1.45
}			
}			
}			
}			
ALIGNtoBYTE ()			

10.1.1 Layer Identifier (LAYER_ID) (4 bits)

The syntax element LAYER_ID in an interlaced frame and interlaced field shall be the same as described in Section 8.1.1. In the interlaced field, two fields within a same frame have a same LAYER_ID.

10.1.2 Picture Identifier (PICTURE_ID) (8 bits)

The syntax element PICTURE_ID in an interlaced frame and interlaced field shall be the same as described in Section 8.1.2 except for the increment. In the interlaced field, the increment between two consecutive fields shall be one. In two fields within a same frame, the PICTURE_ID of the top field shall be assigned with an even number and the PICTURE_ID of the bottom field shall be assigned with the PICTURE_ID of the top field plus one. In the case of the interlaced frame, the PICTURE_ID of the current frame shall be assigned with an even number and the PICTURE_ID of the next interlaced frame or the interlaced top field shall be assigned with the PICTURE_ID of the current frame plus two.

10.1.3 Coded-order (CODED_ORDER) (8 bits)

The syntax element CODED_ORDER in an interlaced frame and interlaced field shall be the same as described in Section 8.1.3. In the interlaced field, two fields within a same frame have a same CODED_ORDER.

10.1.4 Layer Entry Flag (LAYER_ENTRY_FLAG) (1 bit)

The syntax element LAYER_ENTRY_FLAG in an interlaced frame and interlaced field shall be the same as described in Section 8.1.4.

10.1.5 Maximum Quality Layer Identifier (MAX_ QUALITY_LAYER_ID) (4 bit)

The syntax element LAYER_ENTRY_FLAG in an interlaced frame and interlaced field shall be the same as described in Section 8.1.5.

10.1.6 Quality Reference Picture Flag (QUALITY_REFERENCE_PICTURE _FLAG) (1 bit)

The syntax element QUALITY_REFERENCE_PICTURE _FLAG in an interlaced frame and interlaced field shall be the same as described in Section 8.1.6.

10.1.7 Frame-coded Mode (FRAME_CODED_FLAG) (1 bit)

The syntax element FRAME_CODED_FLAG shall indicate whether the current picture is coded in the frame coded mode or the field coded mode when PAFF_FLAG == 1. If FRAME_CODED_FLAG == 1, the current picture shall be coded in the frame coded mode, else the field coded mode. The details are explained in Section 11.1.1.1.

10.1.8 Polarity of the Current Field (TOP_FIELD_FLAG) (1 bit)

The syntax element TOP_FIELD_FLAG, when set, shall indicate that the current field is the top field when the current picture is coded in the field picture mode.

10.1.9 Referenced Picture Flag (REFERENCED_PICTURE_FLAG) (1 bit)

The syntax element REFERENCED_PICTURE_FLAG in an interlaced frame and interlaced field shall be almost the same as described in Section 8.1.7. The difference is that REFERENCED_PICTURE_FLAG shall appear when the current picture is the top field or the interlaced frame. If the current picture is the top field, the current top field and the bottom field, which is a paired field with the current top field, shall be used as the reference frame for the next coded-order pictures.

10.1.10 Removing Entry Flag (REMOVE_ENTRY_FLAG) (1 bit)

The syntax element REMOVE_ENTRY_FLAG in an interlaced frame and interlaced field shall be almost the same as described in Section 8.1.8. The difference is that REMOVE_ENTRY_FLAG shall appear when the current picture is the top field or the interlaced frame.

10.1.11 Removing Picture Identifier (REMOVE_PICTURE_ID) (8 bits)

The syntax element REMOVE_PICTURE_ID in an interlaced frame and interlaced field shall be almost the same as described in Section 8.1.9. The differences are that REMOVE_PICTURE_ID shall appear when the current picture is the top field or the interlaced frame, and the picture pointed with REMOVE_PICTURE_ID shall be the top field or the interlaced frame because the reference frame list is managed by the frame unit not field unit. Therefore, if the REMOVE_PICTURE_ID points the top field of a certain frame in the reference frame list, the frame including that top field shall be removed.

10.1.12 Motion Compensation Flag (MC_FLAG) (1 bit)

The syntax element MC_FLAG in an interlaced frame and interlaced field shall be the same as described in Section 8.1.10.

10.1.13 B-picture Flag (B_PICTURE_FLAG) (1 bit)

The syntax element B_PICTURE_FLAG in an interlaced frame and interlaced field shall be the same as described in Section 8.1.11.

10.1.14 Predictive Direction for P Picture (PRED_DIRECTION) (1 bit)

The syntax element PRED_DIRECTION in and interlaced frame and interlaced field shall be the same as described in Section 8.1.12.

10.1.15 MV Mode (MV_MODE) (variable size)

The syntax element MV_MODE in an interlaced frame and interlaced field shall be the same as described in Section 8.1.13.

10.1.16 Reference Picture Identifier (REFERENCE_PICTURE_ID[i]) (8 bits)

The syntax element REFERENCE_PICTURE_ID[i] in an interlaced frame and interlaced field shall be almost the same as described in Section 8.1.14. The difference is that REFERENCE_PICTURE_ID[i] shall point an interlaced frame or interlaced field. If REFERENCE_PICTURE_ID[i] is even number and the current picture is the field picture, the REFERENCE_PICTURE_ID[i] points top field. If REFERENCE_PICTURE_ID[i] is even number and the current picture is the frame picture, the REFERENCE_PICTURE_ID[i] points frame picture. If REFERENCE_PICTURE_ID[i] is odd number and the current picture is the field picture, the REFERENCE_PICTURE_ID[i] points bottom field.

If the current picture is a field picture with B picture coding mode, REFERENCE_PITURE_ID[0] and REFERENCE_PITURE_ID[1] shall be selected as reference pictures. Here, REFERENCE_PITURE_ID[0] and REFERENCE_PITURE_ID[1] point a past field picture and a future field picture. In case of the interlaced frame picture, REFERENCE_PICTURE_ID[i] shall be the same meaning as the progressive frame picture.

10.1.17 Picture-level Intra Prediction Flag (PIC_INTRA_PRED_FLAG) (1 bit)

The syntax element PIC_INTRA_RRED_FLAG, in an interlaced frame and interlaced field shall be the same as described in Section 8.1.15.

10.1.18 On/Off Flag of Progressive to Interlaced Conversion Method (P2I_METHOD_FLAG) (1 bit)

The syntax element P2I_METHOD_FLAG shall be present only if P2I_METHOD ==1. P2I_METHOD_FLAG shall indicate whether the interlaced pictures are generated using one frame or using two successive frames of the lower layer for generating the residual of the the current layer. When P2I_METHOD_FLAG == 1, the interlaced pictures shall be generated using two successive frames of the lower layer. Here, weighting factors for the interpolation of the two successive frames of the lower layer is defiend in Section 7.1.54 and Section 7.1.55. The details are described in Section 12.1.3.3.

10.1.19 Scanning Order Type (SCAN_ORDER_TYPE) (variable size)

The syntax element SCAN_ORDER_TYPE in an interlaced frame and interlaced field shall be the same as described in Section 8.1.16. However, the default order is different from the order of progressive case. When the current picture is field picture mode, the default scanning order is as shown in Figure 38. When the current picture is frame picture mode, the default scanning order is as shown in Figure 15.

10.1.20 Scanning Order Index (SEQ_SCAN_ORDER_IDX) (4 bits)

The syntax element SEQ_SCAN_ORDER_IDX in an interlaced frame and interlaced field shall be the same as described in Section 8.1.17.

10.1.21 Scanning Order Difference Flag at Picture Layer (PIC_ORDER_DIFF_FLAG) (1 bit)

The syntax element PIC_ORDER_DIFF_FLAG in an interlaced frame and interlaced field shall be the same as described in Section 8.1.18.

10.1.22 Picture Layer Scanning Order (PIC_SCAN_ORDER) (6 bits)

The syntax element PIC_SCAN_ORDER in an interlaced frame and interlaced field shall be the same as described in Section 8.1.19.

10.1.23 Uniform Picture Quantizer Flag (QP_PICTURE_UNIFORM_FLAG) (1 bit)

The syntax element QP_PICTURE_UNIFORM_FLAG in an interlaced frame and interlaced field shall be the same as described in Section 8.1.20.

10.1.24 Uniform Channel Quantizer Flag (QP_CHANNEL_UNIFORM_FLAG) (1 bit)

The syntax element QP_CHANNEL_UNIFORM_FLAG in an interlaced frame and interlaced field shall be the same as described in Section 8.1.21.

10.1.25 Picture Quantizer Parameter (QP_PICTURE) (8 bits)

The syntax element QP_PICTURE in an interlaced frame and interlaced field shall be the same as described in Section 8.1.22.

10.1.26 Picture Quantizer Parameter for Y Channel (QP_PICTURE_Y) (8 bits)

The syntax element QP_PICTURE_Y in an interlaced frame and interlaced field shall be the same as described in Section 8.1.23.

10.1.27 Picture Quantizer Parameter for Cb Channel (QP_PICTURE_U) (8 bits)

The syntax element QP_PICTURE_U in an interlaced frame and interlaced field shall be the same as described in Section 8.1.24.

10.1.28 Picture Quantizer Parameter for Cr Channel (QP_PICTURE_V) (8 bits)

The syntax element QP_PICTURE_V in an interlaced frame and interlaced field shall be the same as described in Section 8.1.25.

10.1.29 Channel Skip Mode (CHANNEL_SKIP_MODE) (8 bits)

The syntax element CHANNEL_SKIP_MODE in an interlaced frame and interlaced field shall be the same as described in Section 8.1.26.

10.1.30 Color Difference Up-Conversion Method for Cb Channel (COLOR_DIFFERENCE_UPCONV_IDX_U) (2 bits)

The syntax element COLOR_DIFFERENCE_UPCONV_IDX_U in an interlaced frame and interlaced field is slightly different from that of Section 8.1.27. This syntax shall indicate which filter is used for the vertical extension of the up-convert the Cb channel color difference among a 4-tap bi-cubic interlaced filter, a 4-tab bi-cubic frame filter (which is same as the frame coding), and a de-interlacing filter. The filter coefficients are shown in Table 65. Further details are described in Section 12.1.2.2.

Table 65 – Meaning of COLOR_DIFFERENCE_UPCONV_IDX_U/COLOR_DIFFERENCE_UPCONV_IDX_V in an interlaced picture

COLOR_DIFFERENCE_UPCONV_IDX_U COLOR_DIFFERENCE_UPCONV_IDX_V	Filter Coefficients
0	Field Interpolation
1	{-2, 9, 28, -3}, {-3, 28, 9, -2}
2	De-interlacing
3	SMPTE Reserved

10.1.31 Color Difference Up-Conversion Method for Cr Channel (COLOR_DIFFERENCE_UPCONV_IDX_V) (2 bits)

The syntax element COLOR_DIFFERENCE_UPCONV_IDX_V in an interlaced frame and interlaced field is slightly different from that of Section 8.1.28. This syntax shall indicate which filter is used for the vertical extension of the the Cr channel color difference among: a 4-tap bi-cubic interlaced filter, a 4-tab bi-cubic frame filter (which is same as the frame coding), and a de-interlacing filter. The filter coefficients are shown in Table 65. Further details are described in Section 12.1.2.2.

10.1.32 Residual Mapping/Scaling Flag (RES_SCALING_PRESENT_FLAG) (1 bit)

The syntax element RES_SCALING_PRESENT_FLAG, when set, shall indicate residual scaling is used. The details are described in Section 12.2.

10.1.33 Residual Scaling Parameter for Y Channel (SCALE_Y) (9 bits)

The syntax element SCALE_Y in an interlaced frame and interlaced field shall be the same as described in Section 8.1.30.

10.1.34 Residual Shift Parameter for Y Channel (SHIFT_Y) (16 bits)

The syntax element SHIFT_Y in an interlaced frame and interlaced field shall be the same as described in Section 8.1.31.

10.1.35 Residual Norm Parameter for Y Channel (NORM_Y) (4 bits)

The syntax element NORM_Y in an interlaced frame and interlaced field shall be the same as described in Section 8.1.32.

10.1.36 Residual Scaling Parameter for Cb Channel (SCALE_U) (9 bits)

The syntax element SCALE_U in an interlaced frame and interlaced field shall be the same as described in Section 8.1.33.

10.1.37 Residual Shift Parameter for Cb Channel (SHIFT_U) (16 bits)

The syntax element SHIFT_U in an interlaced frame and interlaced field shall be the same as described in Section 8.1.34.

10.1.38 Residual Norm Parameter for Cb Channel (NORM_U) (4 bits)

The syntax element NORM_U in an interlaced frame and interlaced field shall be the same as described in Section 8.1.35.

10.1.39 Residual Scaling Parameter for Cr Channel (SCALE_V) (9 bits)

The syntax element SCALE_V in an interlaced frame and interlaced field shall be the same as described in Section 8.1.36.

10.1.40 Residual Shift Parameter for Cr Channel (SHIFT_V) (16 bits)

The syntax element SHIFT_V in an interlaced frame and interlaced field shall be the same as described in Section 8.1.37.

10.1.41 Residual Norm Parameter for Cr Channel (NORM_V) (4 bits)

The syntax element NORM_V in an interlaced frame and interlaced field shall be the same as described in Section 8.1.38.

10.1.42 View Prediction Flag (VIEW_PRED_FLAG) (1 bit)

The syntax element VIEW_PRED_FLAG in an interlaced frame and interlaced field shall be the same as described in Section 8.1.39.

10.1.43 Prediction Block Size Flag (PRED_BLK_SIZE_FLAG) (1 bit)

The syntax element PRED_BLK_SIZE_FLAG in an interlaced frame and interlaced field shall be the same as described in Section 8.1.40.

10.1.44 Disparity/Motion Vector Differential in X-axis (DMVDx) (variable size)

The syntax element DMVDx in an interlaced frame and interlaced field shall be the same as described in Section 8.1.41.

10.1.45 Disparity/Motion Vector Differential in Y-axis (DMVDy) (variable size)

The syntax element DMVDy in an interlaced frame and interlaced field shall be the same as described in Section 8.1.42.

10.2 Slice-level Syntax and Semantics

The slice-level syntax shall be as defined in Table 66 for an interlaced frame and interlaced field, and shall be the same as described in Section 8.2.

Table 66 – Interlaced field/frame slice layer bitstream

SLICE LAYER( ) {	Number of bits	Descriptor	Reference
LAYER_ID_SLICE	4	uimsbf	10.2.1
if(QUALITY_REFINEMENT_FLAG == 1) {			
QUALITY_LAYER_ID	4	uimsbf	10.2.2
}			
FIRST_MB_X	10	uimsbf	10.2.3
FIRST_MB_Y	10	uimsbf	10.2.4
if(QP_CHANNEL_UINFORM_FLAG == 1) {			
DQP_SLICE	variable size	evlclbf	10.2.5
}			
else {			
DQP_SLICE_Y	variable size	evlclbf	10.2.6
DQP_SLICE_U	variable size	evlclbf	10.2.7
DQP_SLICE_V	variable size	evlclbf	10.2.8
}			
for('all macroblocks within a slice') {			
MB LAYER()			Table 67
}			
TerminateBit()			
ALIGNtoBYTE ()			
}			

10.2.1 Layer Identifier for Slice (LAYER_ID_SLICE) (4 bits)

The syntax element LAYER_ID of a slice in an interlaced frame and interlaced field shall be the same as described in Section 8.2.1.

10.2.2 Quality Layer Identifier (QUALITY_LAYER_ID) (4 bits)

The syntax element QUALITY_LAYER_ID of a slice in an interlaced frame and interlaced field shall be the same as described in Section 8.2.2.

10.2.3 X-Coordinate of the First MB in Slice (FIRST_MB_X) (10bits)

The syntax element FIRST_MB_X of a slice in an interlaced frame and interlaced field shall be the same as described in Section 8.2.3.

10.2.4 Y-Coordinate of the First MB in Slice (FIRST_MB_Y) (10bits)

The syntax element FIRST_MB_Y of a slice in an interlaced frame and interlaced field shall be the same as described in Section 8.2.4.

10.2.5 Differential of Slice-level Quantizer Parameter (DQP_SLICE) (variable size)

The syntax element DQP_SLICE of a slice in an interlaced frame and interlaced field shall be the same as described in Section 8.2.5.

10.2.6 Differential of Slice-level Quantizer Parameter for Luma Channel (DQP_SLICE_Y) (variable size)

The syntax element DQP_SLICE_Y of a slice in an interlaced frame and interlaced field shall be the same as described in Section 8.2.6.

10.2.7 Differential of Slice-level Quantizer Parameter for Cb Channel (DQP_SLICE_U) (variable size)

The syntax element DQP_SLICE_U of a slice in an interlaced frame and interlaced field shall be the same as described in Section 8.2.7.

10.2.8 Differential of Slice-level Quantizer Parameter for Cr Channel (DQP_SLICE_V) (variable size)

The syntax element DQP_SLICE_U of a slice in an interlaced frame and interlaced field shall be the same as described in Section 8.2.8.

10.3 Macroblock-level Syntax and Semantics

Table 67 defines the bitstream syntax of the macroblock layer.

Table 67 – Interlaced macroblock layer bitstream

MB LAYER( ) {	Number of bits	Descriptor	Reference
if(is_beginning_of_slice() || is_previousMB_NotSkip()) {			
do {			
SKIP_RUN	variable size	avlclbf	10.3.1
}while (SKIP_RUN == 4)			
}			
if((is_not_skipMB() && QUALITY_REFINEMENT_FLAG == 0) ||			
(is_not_skipMB() && QUALITY_LAYER_ID == 0 &&			
QUALITY_REFINEMENT_FLAG == 1)) {			
if(MC_FLAG == 1) {			
MB_SIG_MODE	variable size	avlclbf	10.3.2
if(MB_SIG_MODE == 4) {			
MB_SIG_4	2	uimsbf	10.3.3
If(MB_SIG_4 == 3) {			
if((is_UpperMB_Inter() || is_LeftMB_Inter()){			
MB_MODE_1	1	uimsbf	10.3.4
}			
}			
}			
if(B_PICTURE_FLAG && ExistMV()) {			
B_PRED_MODE	2	uimsbf	10.3.5
}			
}			
else {			
if(PIC_INTRA_PRED_FLAG == 1) {			
if(!is_UpperMB_Skip() || !is_LeftMB_Skip()){			
IPRED_MODE	variable size	uimsbf or avlclbf	10.3.6
}			
}			
INTRA_SIG	variable size	uimsbf or avlclbf	10.3.7

}			
for(i=0; i < number of MVs;i++) {			
MVDx[i]	variable size	evlclbf	10.3.8
MVDy[i]	variable size	evlclbf	10.3.9
if(B_PICTURE_FLAG && (B_PRED_MODE == 3)) {			
MVD2x[i]	variable size	evlclbf	10.3.10
MVD2y[i]	variable size	evlclbf	10.3.11
}			
}			
if(isAvailableMVSCALE()) {			
MV_SCALE_FLAG	1	uimsbf	10.3.12
}			
}			
if(ExistCodedCoeff()) {			
if(Exist_Y_Channel() && Exist_UV_Channel()){			
YUV_CSP	variable size	avlclbf	10.3.13
}			
if(Exist_Y_Coeff()) {			
Y_CSP	variable size	avlclbf	10.3.14
if(Exist_Y0Y1_Coeff()) {			
Y0Y1_CSP	variable size	avlclbf	10.3.15
}			
if(Exist_Y2Y3_Coeff()) {			
Y2Y3_CSP	variable size	avlclbf	10.3.16
}			
}			
if(Exist_UV_Coeff()) {			
if(Exist_U_Channel() && Exist_V_Channel()) {			
UV_CSP	variable size	avlclbf	10.3.17
}			
if(is422()) {			
if(Exist_U_Coeff()) {			
U0U1_CSP	variable size	avlclbf	10.3.18
}			
if(Exist_V_Coeff()) {			
V0V1_CSP	variable size	avlclbf	10.3.19
}			
}			
if(is444()) {			
if(Exist_U_Coeff()) {			
U_CSP	variable size	avlclbf	10.3.20
if(Exist_U0U1_Coeff()) {			
U0U1_CSP	variable size	avlclbf	10.3.18
}			
if(Exist_U2U3_Coeff()) {			
U2U3_CSP	variable size	avlclbf	10.3.21
}			
}			
if(Exist_V_Coeff()) {			
V_CSP	variable size	avlclbf	10.3.22
if(Exist_V0V1_Coeff()){			
V0V1_CSP	variable size	avlclbf	10.3.19
}			

if(Exist_V2V3_Coeff()){			
V2V3_CSP	variable size	avlclbf	10.3.23
}			
}			
}			
}			
if(QP_PICTURE_UNIFORM_FLAG == 0) {			
DQP_MB	variable size	evlclbf	10.3.24
}			
for('all coded blocks in MB') {			
BLOCK LAYER()			Table 34
}			
}			

10.3.1 Skipped-macroblock Run (SKIP_RUN) (variable size)

The syntax element SKIP_RUN in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.1.

10.3.2 MB-level Coded Significant Pattern and MB Mode (MB_SIG_MODE) (variable size)

The syntax element MB_SIG_MODE in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.2.

10.3.3 MB-level Coded Significant Pattern (MB_SIG_4) (2 bits)

The syntax element MB_SIG_4 in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.3.

10.3.4 MB Mode in Non-significant Macroblock (MB_MODE_1) (1 bit)

The syntax element MB_MODE_1 in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.4.

10.3.5 Prediction Mode for B-picture (B_PRED_MODE) (2 bits)

The syntax element B_PRED_MODE in a in a macroblock macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.5.

10.3.6 Intra Prediction Mode (IPRED_MODE) (variable size)

The syntax element IPRED_MODE in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.6.

10.3.7 MB-level Coded Significant Pattern in Intra Picture (INTRA_SIG) (variable size)

The syntax element INTRA_SIG in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.7.

10.3.8 Motion Vector Differential in X-axis (MVDx) (variable size)

The syntax element MVDx in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.8.

10.3.9 Motion Vector Differential in Y-axis (MVDy) (variable size)

The syntax element MVDy in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.9.

10.3.10 Motion Vector Differential of X-axis for Interpolated Mode (MVD2x) (variable size)

The syntax element MVD2x in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.10.

10.3.11 Motion Vector Differential of Y-axis for Interpolated Mode (MVD2y) (variable size)

The syntax element MVD2y in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.11.

10.3.12 Motion Vector Scaling Flag (MV_SCALE_FLAG) (1 bit)

The syntax element MV_SCALE_FLAG, when set, shall indicate that the current MB uses the motion vector scaling mode to predict residuals of the current macroblock from the reference pictures. The motion vector scaling mode uses the derived motion vector which refers to the other field of the same reference frame for the motion compensation when the current picture is coded in the interlaced field picture. The motion vector obtained from this current reference picture shall be modified to the motion vector to the auxiliary reference picture which is the opposite field of the frame that includes the current reference field. The current reference picture shall be decided by the picture layer elements, REFERENCE_PICTURE_ID[i]. The modification shall be performed by the linear scaling based on the time difference and the offset adjustment. This is necessary because the pixel locations of the top field and bottom field are not matched. The details are described in Section 11.2.3.1.

10.3.13 Coded Significant Pattern in Macroblock (YUV_CSP) (variable size)

The syntax element YUV_CSP in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.12.

10.3.14 Coded Significant Pattern in Luma Blocks (Y_CSP) (variable size)

The syntax element Y_CSP in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.13.

10.3.15 Coded Significant Pattern in Luma Upper Blocks (Y0Y1_CSP) (variable size)

The syntax element Y0Y1_CSP in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.14.

10.3.16 Coded Significant Pattern in Luma Lower Blocks (Y2Y3_CSP) (variable size)

The syntax element Y2Y3_CSP in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.15.

10.3.17 Coded Significant Pattern in Color Difference Blocks (UV_CSP) (variable size)

The syntax element UV_CSP in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.16.

10.3.18 Coded Significant Pattern in Color Difference Blocks for Cb Channel (U0U1_CSP) (variable size)

The syntax element U0U1_CSP in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.17.

10.3.19 Coded Significant Pattern in Color Difference Blocks for Cr Channel (V0V1_CSP) (variable size)

The syntax element V0V1_CSP in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.18.

10.3.20 Coded Significant Pattern in Cb Channel (U_CSP) (variable size)

The syntax element U_CSP in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.19.

10.3.21 Coded Significant Pattern in Lower Blocks for Cb Channel (U2U3_CSP) (variable size)

The syntax element U2U3_CSP in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.20.

10.3.22 Coded Significant Pattern in Cr Channel (V_CSP) (variable size)

The syntax element V_CSP in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.21.

10.3.23 Coded Significant Pattern in Lower Blocks for Cr Channel (V2V3_CSP) (variable size)

The syntax element V2V3_CSP in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.22.

10.3.24 Differential of MB-level Quantizer Parameter (DQP_MB) (variable size)

The syntax element DQP_MB in a macroblock of an interlaced frame and interlaced field shall be the same as described in Section 8.3.23.

10.4 Block-level Syntax and Semantics

The block layer syntax elements in interlaced frame and interlaced field pictures shall be identical to the corresponding syntax elements in progressive pictures as described in Section 8.4

11 Interlaced Bitstream Decoding Process

The decoding process of I and P field pictures shall be as specified in this section.

11.1 Interlaced I Frame/Field Picture Decoding

Section 11.1.1 defines the picture layer decoding, Section 11.1.2 defines the slice layer decoding, Section 11.1.3 defines the macroblock layer decoding, and Section 11.1.4 defines the block layer decoding.

11.1.1 Picture Layer Decoding at Interlaced I Frame/Field Picture

Table 64 defines the elements that consist of the intra frame/field picture layer header for interlaced sequences. Most elements are similar to the progressive sequences.

11.1.1.1 Picture-level Adaptive Frame/Field Coding

At the sequence layer header, PAFF_FLAG, when set, shall indicate this sequence supports the picture-level adaptive frame/field coding mode when INTERLACE_FLAG == 1. Under the picture-level adaptive frame/field coding mode, FRAME_CODE_FLAG in section 10.1.7 shall indicate whether the current picture is coded in the frame or field mode. If FRAME_CODE_FLAG == 1, all operations shall be the same as the frame picture in the progressive sequence. If PAFF_FLAG == 1 and FRAME_CODE_FLAG == 0, the current picture shall be decoded in the field picture modes. If INTERLACE_FLAG == 1 and PAFF_FLAG == 0, all pictures shall be decoded in field coding mode.

11.1.2 Slice Layer Decoding at Interlaced I Frame/Field Picture

Table 66 defines the elements in the intra frame/field picture layer header for interlaced sequences. All elements shall be the same as the progressive sequences.

11.1.3 Macroblock Layer Decoding at Interlaced I Frame/Field Picture

The decoding process for coded macroblocks of interlaced I frame/field pictures shall be the same as described in Section 9.1.3.

11.1.4 Block Layer Decoding at Interlaced I Frame/Field Picture

The decoding process for coded blocks of I frame/field pictures shall be the same as described in Section 9.1.4 except for the scanning order for field blocks as shown in Figure 39. Figure 38 shows the mapping array used to convert from the one-dimensional to two-dimensional scanning pattern shown in Figure 39 and corresponds to scan_array[] of Figure 14 when the default scanning pattern is selected (SCAN_ORDER_TYPE == 0) and the current picture is the field code mode. Despite the interlaced sequence, if the current picture is a frame picture, the default scanning is shown in Figure 16.

0	2	16	18	32	34	48	50
1	3	17	19	33	35	49	51
4	6	20	22	36	38	52	54
5	7	21	23	37	39	53	55
8	10	24	26	40	42	56	58
9	11	25	27	41	43	57	59
12	14	28	30	44	46	60	62
13	15	29	31	45	47	61	63

Figure 38 – Default scanning array for interlaced pictures

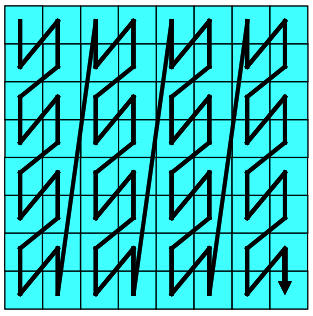

Figure 39 – Default scanning pattern for interlaced pictures

11.2 Interlaced Inter Frame/Field Picture Decoding

Section 11.2.1 defines the picture layer decoding, Section 11.2.2 defines the slice layer decoding, Section 11.2.3 defines the macroblock layer decoding, and Section 11.2.4 defines the block layer decoding.

11.2.1 Picture Layer Decoding at Interlaced Inter Frame/Field Picture

Table 64 defines the elements of the inter frame/field picture layer header for interlaced sequences. The details shall be the same as described in Section 9.1.1 and Section 11.1.1.

11.2.2 Slice Layer Decoding at Interlaced Inter Frame/Field Picture

Table 66 defines the elements in the inter frame/field picture layer header for interlaced sequences. All elements shall be the same as the progressive sequences.

11.2.3 Macroblock Layer Decoding at Interlaced Inter Frame/Field Picture

The decoding process for coded macroblocks of interlaced P frame/field pictures shall be the same as described in Section 9.2.3 except for the motion vector scaling mode which is signaled by MV_SCALE_FLAG.

11.2.3.1 Motion Vector Scaling

MV_SCALE_FLAG, when set, signals that the current macroblock refers to the other field not the current reference field. But this shall be performed only when the current MB is inter-macroblock not skipped, the current picture is field picture, the current macroblock is not coded with DIRECT_MODE or INTERPOLATED_MODE and the current reference field is not the first field of the current frame. The interlaced frame shall consist of the top field and the bottom field. So, if the current reference field is the field of the current frame and the current picture is the bottom field of the current frame, the MV scaling mode shall not be used. When MV_SCALE_FLAG == 1, the other field of the same reference frame shall be used as the auxiliary reference field and the predicted blocks shall be obtained from the auxiliary reference field with the modified motion vector. Figure 40 and Figure 41 show the main reference fields and the auxiliary reference fields when the current field is top or bottom field. The main reference field shall be indicated by REFERENCE_PICTURE_ID[i]. REFERENCE_PICTURE_ID[i] shall indicate which frame in the reference frame list is selected as a reference frame, and the least significant bit of REFERENCE_PICTURE_ID[i] shall indicate which field in the reference frame selected by REFERENCE_PICTURE_ID[i] is selected as a main reference field. As shown in the right of Figure 41, when the main reference field is the top field of the current frame, the auxiliary reference field is not used and the motion vector scaling mode is not selected. In this case, it is not necessary to signal MV_SCALE_FLAG and the reconstructed top field shall not be stored in the reference frame list until the bottom field of the current field is decoded.

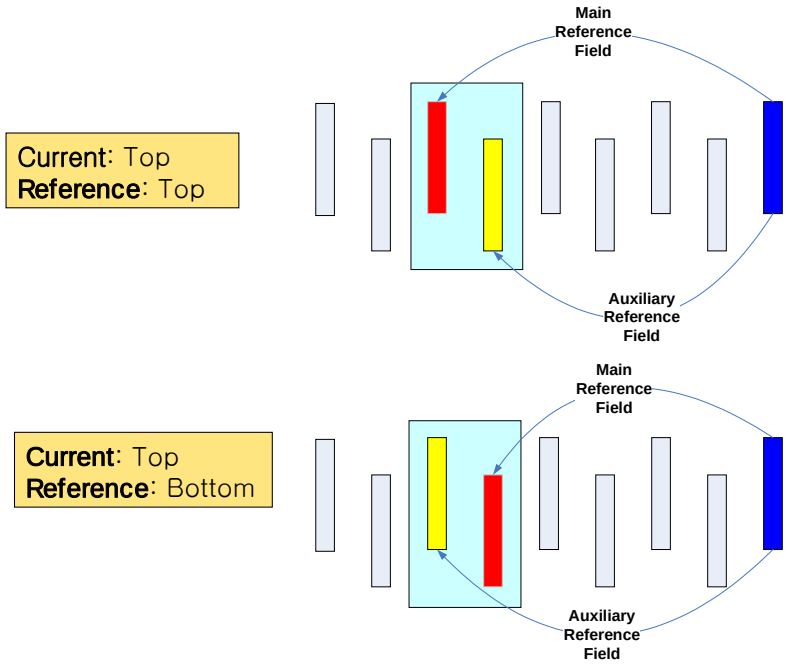

Figure 40 – Reference fields when the current field is the top field

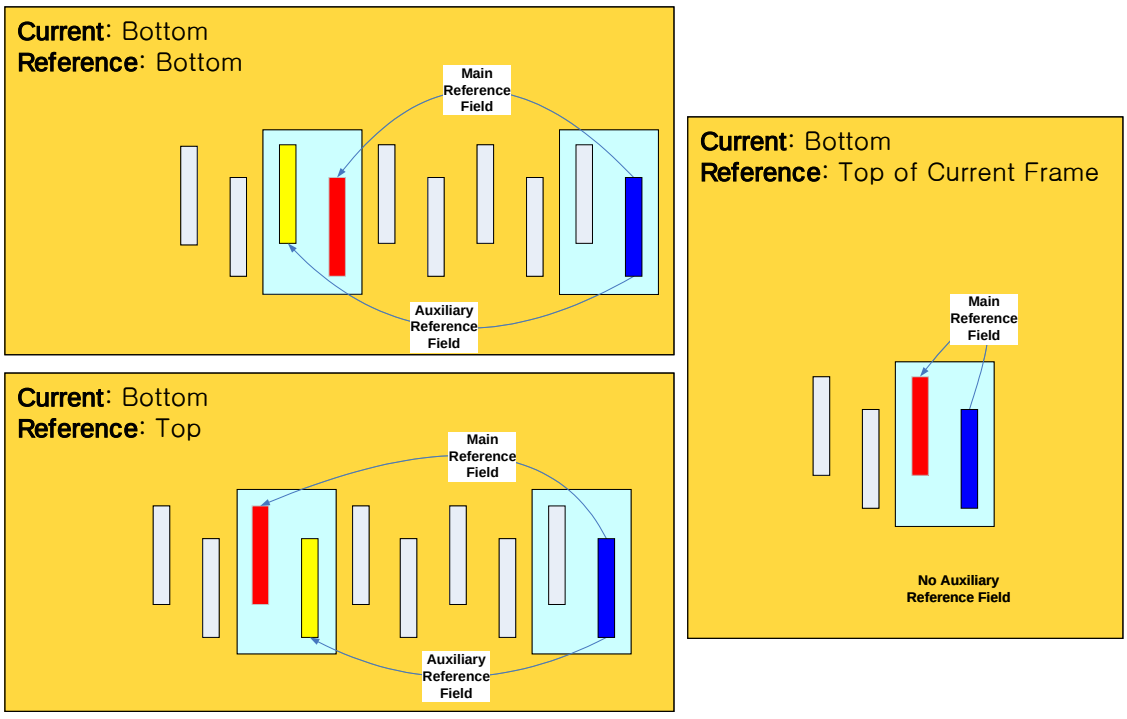

Figure 41 – Reference fields when the current field is the bottom field

There shall be eight cases for the motion vector scaling. Figure 43 and Figure 45 show the case that the main reference field has the same polarity (top versus bottom field) as the current field. Figure 44 and Figure 46 show the case that the main reference field has the different polarity from the current field. Additionally Figure 43 and Figure 44 show the case that a past picture is selected as the main reference field while Figure 45 and Figure 46 show the case that a future picture is selected as the main reference field.

Table 68 summarizes the temporal offset and signs of spatial offset for calculating the modified MVs of the four cases. In Table 68, the temporal offset is to calculate the distance from the current field to the auxiliary reference field. The spatial offset is to compensate for the mismatch between top field and bottom field. The sixth column in Table 68 indicates which formula is used to calculate the modified MV.

Table 68 – Offsets for MV Scaling

Cases	Current Field	Main Reference Field	Temporal Offset	Sign of Spatial Offset	Formula
Figure 43 (a)	Top	Past Top	-1	-	(11-1)
Figure 43 (b)	Bottom	Past Bottom	+1	+	
Figure 45 (a)	Top	Future Top	+1	-	
Figure 45 (b)	Bottom	Future Bottom	-1	+	
Figure 44 (a)	Top	Past Bottom	+1	+	(11-2)
Figure 44 (b)	Bottom	Past Top	-1	-	
Figure 46 (a)	Top	Future Bottom	-1	+	
Figure 46 (b)	Bottom	Future Top	+1	-	

When the main reference field has the same polarity as the current field, the modified motion vector shall be calculated by equation (11-1):

$$MV_x^{aux} = \frac{MV_x^{main} \cdot (dist + temporal\_offset) + \frac{1}{2} dist}{dist}$$
$$MV_y^{aux} = \frac{MV_y^{main} \cdot (dist + temporal\_offset) + \frac{1}{2} dist}{dist} + spatial\_offset \quad \textbf{(11-1)}$$

where spatial_offset is in half pel units, and its sign is described in Table 68. The temporal_offset value is defined in Table 68 and *dist* is the temporal field distance between the current field and the main reference field. The *dist* value shall be calculated by the pseudo-code of Figure 42.

```
if(B_PICTURE_FLAG == 0) {
   if(PRED_DIRECTION == 0) {
       if(PICTURE_ID > REFERENCE_PICTURE_ID[0]) {
            dist = PICTURE_ID - REFERENCE_PICTURE_ID[0];
       }
       else {
            dist = 256 - REFERENCE_PICTURE_ID[0] + PICTURE_ID;
       }
  }
  else{
      if(PICTURE_ID < REFERENCE_PICTURE_ID[0]) {
            dist = REFERENCE_PICTURE_ID[0] - PICTURE_ID;
      }
      else {
            dist = 256 - PICTURE_ID + REFERENCE_PICTURE_ID[0];
      }
  }
}
else {
 if(B_PRED_MODE == 1) {
      if(PICTURE_ID > REFERENCE_PICTURE_ID[0]) {
          dist = PICTURE_ID - REFERENCE_PICTURE_ID[0];
      }
      else {
          dist = 256 - REFERENCE_PICTURE_ID[0] + PICTURE_ID;
     }
 }
 else if(B_PRED_MODE == 2) {
     if(PICTURE_ID < REFERENCE_PICTURE_ID[1]) {
         dist = REFERENCE_PICTURE_ID[1] - PICTURE_ID;
    }
    else {
         dist = 256 - PICTURE_ID + REFERENCE_PICTURE_ID[1];
    }
 }
}
```

Figure 42 – Pseudo-code for calculating *dist*

If the precision of the motion vector is quarter-pel, the magnitude of the spatial_offset shall be 2. If the precision of the motion vector is half-pel, the magnitude of the spatial_offset shall be 1. The reason of this spatial_offset adjustment is that the locations of top-field and the bottom field are mismatched by the half-pel unit in the vertical direction.

When the main reference field has a different polarity from the current field, the modified motion vector shall be calculated by equation (11-2):

$$MV_x^{aux} = \frac{MV_x^{main} \cdot (dist + temporal\_offset) + \frac{1}{2} dist}{dist}$$
$$MV_y^{aux} = \frac{(MV_y^{main} + spatial\_offset) \cdot (dist + temporal\_offset) + \frac{1}{2} dist}{dist} \quad \textbf{(11-2)}$$

where *spatial_offset* is in half-pel units, and its sign is described in Table 68. The *temporal_offset* value is defined in Table 68 and *dist* shall be calculated using Figure 42.

If MV_MODE == 11b, the spatial_offset of the above motion vector scaling equations shall be zero.

After calculating the modified motion vectors, they are considered the motion vectors for the auxiliary reference field of the current macroblock. Therefore the predicted blocks in the auxiliary reference field shall be indicated by these modified motion vectors.

Since the number of motion vectors within a macroblock is one or four, when the MB_SIG_MODE indicates 1-MV mode, one motion vector shall be modified and used to obtain the predicted macroblock. If MB_SIG_MODE indicates 4-MV mode, four motion vectors shall be modified and used to obtain four predicted blocks. The motion vectors as the input of the motion vector scaling are defined with PMVx/PMVy and MVDx/MVDy.

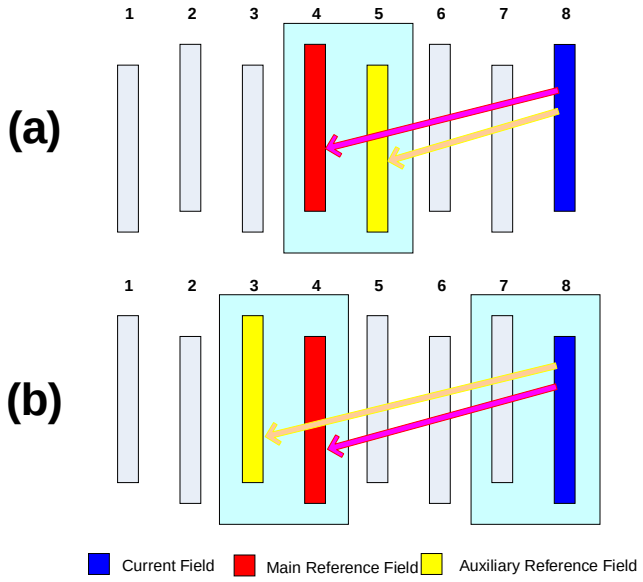

Figure 43 – MV scaling when main reference field in the past frame and current field have same polarities

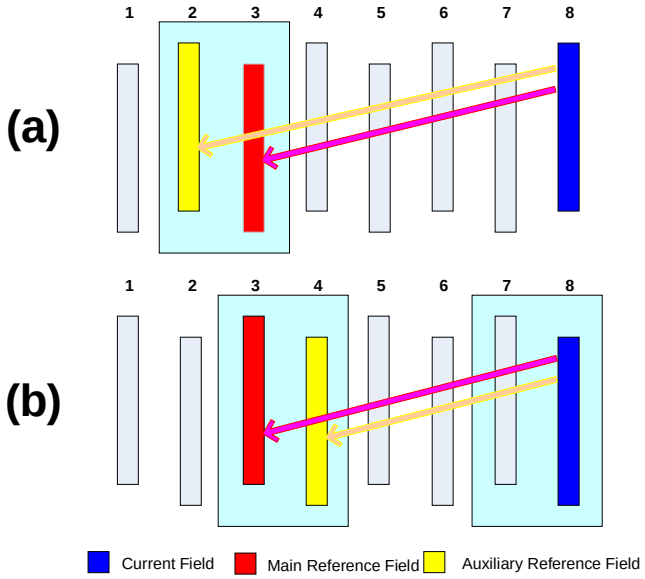

Figure 44 – MV scaling when main reference field in the past frame and current field have different polarities

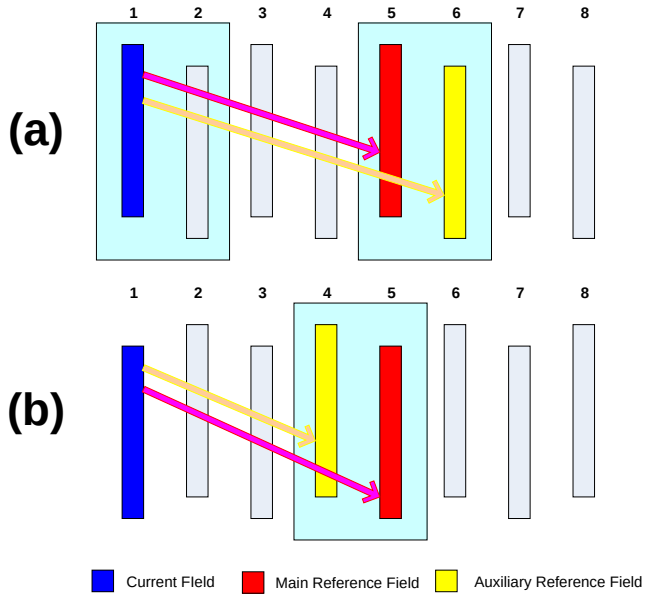

Figure 45 – MV scaling when main reference field in the future frame and current field have same polarities

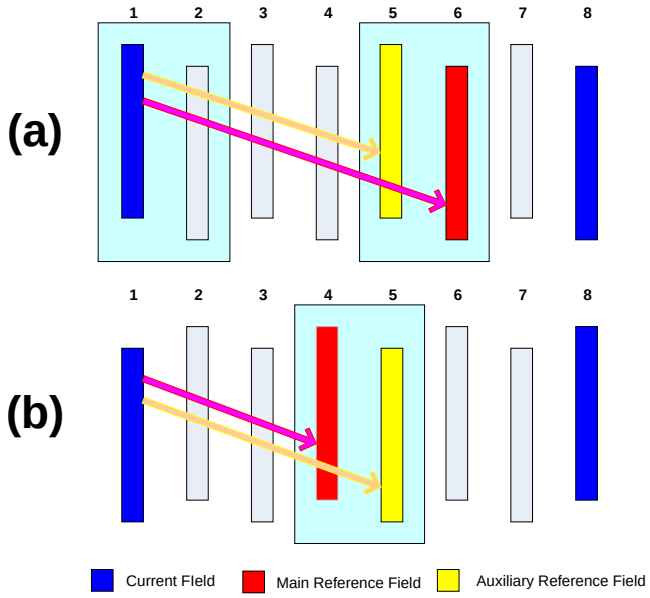

Figure 46 – MV scaling when main reference field in the future frame and current field have different polarities

11.2.4 Block Layer Decoding at Interlaced P Frame/Field Picture

The decoding process for coded blocks of interlaced P frame/field pictures shall be the same as described in Section 11.1.4 except for the residual reconstruction. In interlaced P frame/field pictures, the residual reconstruction process shall be the same as described in Section 9.2.4.

11.3 Adaptive VLC Coding

The adaptive VLC coding of an interlaced frame and interlaced field shall be the same as described in Section 9.3.

11.4 Reference Frame List Management

The reference frame list management of interlaced sequences shall be the same as described in Section 9.4 except for the field picture management. If the current picture is decoded in the frame mode, the reference frame list operation shall be same as the progressive sequences. When the current picture is decoded in the field mode, one frame shall be selected from the reference frame list and one field of the selected frame shall be the referenced field picture indicated by the last bit of REFERENCE_PICTURE_ID[i]. As described in Section 10.1.16, if REFERENCE_PICTURE_ID[i] is an even number, a top field shall be a referenced field. Otherwise, a bottom field shall be a reference field. Since reference frame list is managed by the frame unit not field unit, the decoded top and bottom field pair shall be considered as a frame and both fields shall be stored in the reference frame list when the REFERENCED_PICTURE_FLAG of the top field is one. Here, the PICTURE_ID of the top field shall be used as an identifier in the reference frame list.

12 High Fidelity Picture Reconstruction Process

The entire decoding process which includes the residual decoder is shown in Figure 47. Various video coding standards, such as ISO/IEC 13818-2 (MPEG-2), ISO/IEC 14496-2 (MPEG-4), SMPTE ST 421 (VC-1) or ITU-T H.264, can be used as the base layer decoder in Figure 47. The decoding process of each residual decoder is explained in Section 9 and Section 11. The reconstruction of the video of the base layer (layer 0) does not require any enhancement layer. For the reconstruction of the video of the second layer (layer 1), the base layer picture shall be up-converted and shall be added to the results of the residual mapping/scaling from the layer 1. Since the reconstructed residuals of the residual decoder are from 0 to 255, the reconstructed residuals shall be converted to the ragne corresponding to the bit depth of the layer 1. The third layer (layer 2) video shall be reconstructed by adding the results of the residual mapping/scaling from the layer 2 to the up-converted video of the second layer (layer 1) video. Here, the output of the residual decoder, which is described in the previous sections, shall be mapped or scaled before the addition to the lower layer. This section describes the format up-conversion process and the residual mapping/scaling process.

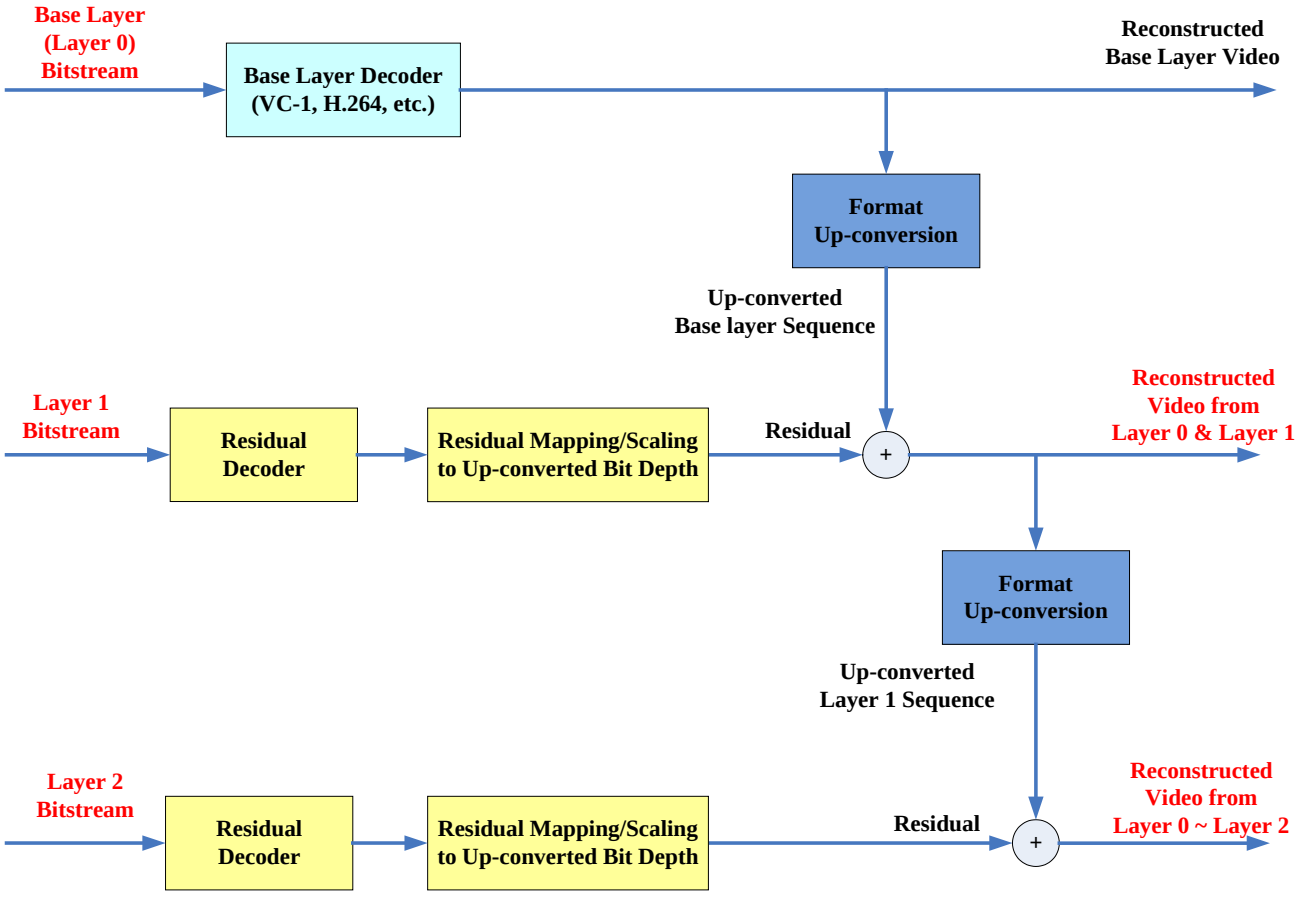

Figure 47 – Decoding process block diagram

12.1 Format Up-Conversion

This section specifies how the format up-conversion process in Figure 47 operates. There are bit depth up-conversion, resolution up-conversion, color difference up-conversion, and tone mapping in the format up-conversion process. Any combination is possible. If it is necessary to apply all functions in the format up-conversion process, bit depth up-conversion, resolution up-conversion and color difference up-conversion

shall be applied sequentially. If tone mapping is applied in the bit depth up-conversion, additional tone mapping shall not be applied.

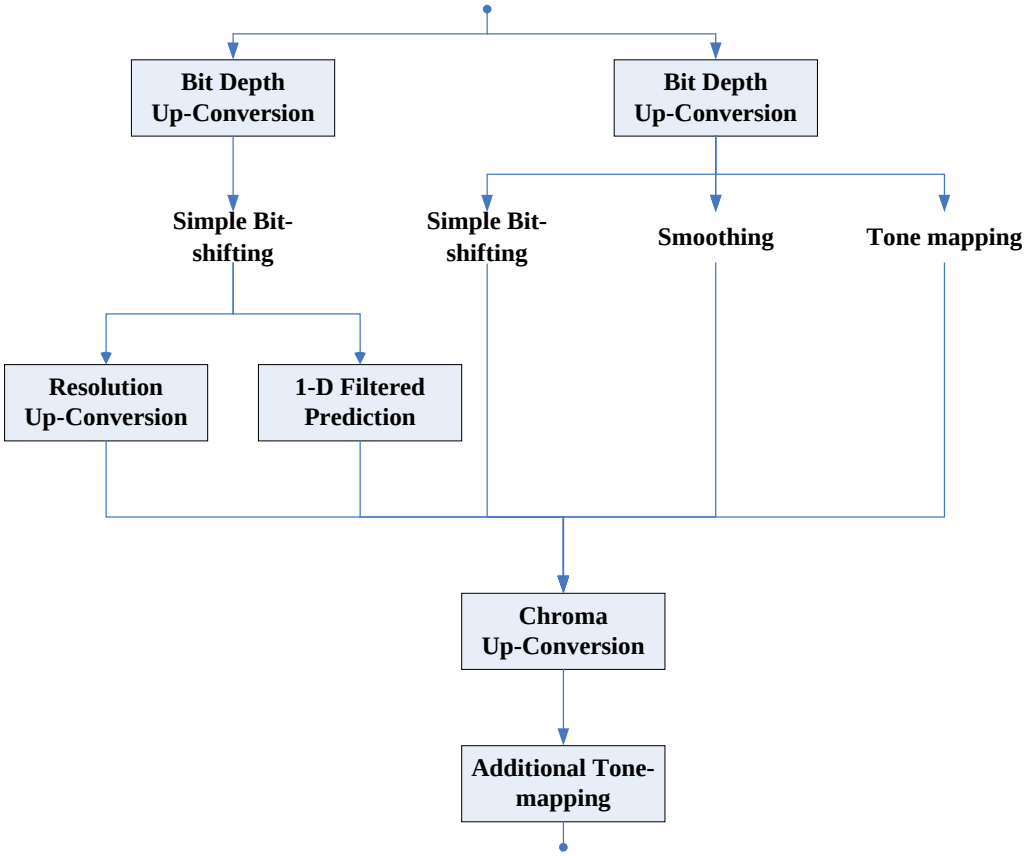

Figure 48 – Diagram for format up-conversion process

Figure 48 shows the order of the format up-conversion. When the bit depth up-conversion is applied with the resolution up-conversion or the 1-D filtered prediction on the reconstructed picture, the simple bit-shifting shall be used as the bit depth up-conversion method. When the bit depth up-conversion is applied without resolution up-conversion or the 1-D filtered prediction, the simple bit-shifting or the smoothing or the tone mapping shall be used as the bit depth up-conversion method.

12.1.1 Bit Depth Up-Conversion

12.1.1.1 Simple Bit-shifting

Simple bit-shifting shall be performed when BIT_DEPTH_MINUS_8 > LOWER_BIT_DEPTH_MINUS_8, the syntax BIT_UPCONVERSION == 01b and the both base and current layer resolutions are same. This operation is just the left shifting by “BIT_DEPTH_MINUS_8 – LOWER_BIT_DEPTH_MINUS_8” bits.

Additionally, this simple bit-shifting shall be performed when the lower layer bit depth is less than the current layer bit depth and the lower layer resolution is also different from the current layer resolution. In this case, the simple bit-shifting shall be performed prior to the resolution up-conversion.

The resolution up-sampling shall employ the spatial low-pass filter which is the following type of equation (**12-1**).

$$y = \frac{\sum_{i=1}^{N} a_i x_i + T/2}{T} \qquad where\ T = \sum_{i=1}^{N} a_i \tag{12-1}$$

Here, N is the number of the interpolation filter coefficients and a_i are the filter coefficients. When implementing the resolution up-conversion, if the denominator in the filter equation (**12-1**) is scaled down as in equation (**12-2**), the simple bit-shifting method shall be eliminated because equation (**12-2**) includes the same effect of the simple bit-shifting.

$$y = \frac{\sum_{i=1}^{N} a_i x_i + T >> (K+1)}{T >> K} \tag{12-2}$$

where K = BIT_DEPTH_MINUS_8 – LOWER_BIT_DEPTH_MINUS_8.

12.1.1.2 Tone Mapping

The tone-mapping is based on a series of piece-wise linear mappings. The basic idea is that there are a series of N points for which the mapping parameters are directly available in the bitstream, and the mapping for intermediate points is derived by linear interpolation.

For example, we can signal that N=5 and the mapping parameters for the 5 points are:

X_0 = 0, Y_0 = 10,
X_1 = 30, Y_1 = 20,
X_2 = 90, Y_2 = 100,
X_3 = 200, Y_3 = 220,
X_4 = 255, Y_4 = 230

For x in the range of (X_n, X_{n+1}), the linear interpolation between (Y_n, Y_{n+1}) shall be as defined in equation **(12-3**).

$$y = Y_n + \frac{\left[\left(Y_{n+1} - Y_n\right) * \left(x - X_n\right) + \left(X_{n+1} - X_n\right)/2\right]}{X_{n+1} - X_n} \tag{12-3}$$

For actual signaling of the map parameters, DPCM shall be used, i.e. we signal X_0, X_1-X_0, X_2-X_1, X_3-X_2, and X_4-X_3,. The same shall be applied to Y_s.

Here, X_0 and the following differences X_n-X_{n-1} for the Y channel shall be decoded as Y_MAP_PARA_X[i] in section 7.1.30 with Y_LENGTH_OF_FLC_X_MINUS_1+1 bits, while Y_0 and the following differences Y_n-Y_{n-1} shall be decoded as Y_MAP_PARA_Y[i] in section 7.1.31 with Y_LENGTH_OF_FLC_Y_MINUS_1+1 bits. U_MAP_PARA_X[i], U_LENGTH_OF_FLC_X_MINUS_1, U_MAP_PARA_Y[i] and U_LENGTH_OF_FLC_Y_MINUS_1 shall be decoded for the Cb channel, while V_MAP_PARA_X[i], V_LENGTH_OF_FLC_X_MINUS_1, V_MAP_PARA_Y[i] and V_LENGTH_OF_FLC_Y_MINUS_1 shall be decoded for the Cr channel in the same way as in the Y channel.

As a special case, for 8-bit samples, when N=256, the mapping is equivalent to a typical table-look-up.
N is never 0 and the number of the piece-wise intervals is N-1, so (N-1) shall be encoded instead of N as NUM_LINEAR_SEGMENTS in Section 7.1.27.

The tone mapping shall be applied in the following cases:

1) the current layer and the lower layer have the same resolution and BIT_UPCONVERSION == 2.
2) ENHANCEMENT_LAYER_TONE_MAP_FLAG == 1.

Depending on the above cases, the range of *X* is different. In case 1, the lower layer samples with lower layer bit depth are used as inputs to the tone-mapping procedure and the output results have the current layer bit depth. In case 2, after the base-layer samples are converted to the enhancement layer bit depth, the tone-mapping shall be applied, so the inputs to the tone-mapping have the same bit-depth as the current layer bit depth or the outputs of the tone-mapping.

In case 1, if color difference up-conversion is necessary, it shall be performed after this tone-mapping.

In case 2, ENHANCEMENT_LAYER_TONE_MAP_FLAG can be decoded when the lower layer and the current layer have the same resolution, and the BIT_UPCONVERSION == 1 or BIT_UPCONVERSION == 3. In this case, if ENHANCEMENT_LAYER_TONE_MAP_FLAG is not set to one, this additional tone-mapping shall not be required. Otherwise, after the simple bit-shifting or the smoothing operation, this additional tone-mapping shall be applied. The simple bit-shifting and the smoothing operation convert the data from the base-layer bit depth to enhancement-layer bit depth. If the color difference up-conversion is additionally necessary, the color difference up-conversion shall be performed before this tone-mapping.

In addition, ENHANCEMENT_LAYER_TONE_MAP_FLAG can be set when the resolution of the lower layer is different from the resolution of the current layer and the base-layer bit depth is also different from the current layer bit depth. In this case, the bit depth up-conversion and the resolution up-conversion shall be performed before the tone-mapping where the simple bit-shifting is selected as the bit depth up-conversion without signaling BIT_UPCONVERSION. Therefore, the input and output signals of the tone-mapping have the same bit depth. If the color difference up-conversion is additionally necessary, the color difference conversion shall be performed before this tone-mapping.

Note: If the spatial resolution of the lower layer picture is different from the current layer spatial resolution, the bit depth up-conversion shall be performed prior to the resolution up-conversion. Since a spatial up-sampling filter can make higher bit depth outputs by down-scaling the denominator of the filter equation. If this kind of filtering method is implemented, the simple bit-shifting can be omitted as explained in Section 12.1.1.1.

12.1.1.3 Smoothing

The spatial low-pass filter shall be represented in equation (12-4)

$$s'(x,y) = \frac{2^{BD-8}}{N} \sum_{i,j \in [-R,R]} w(i,j) \cdot s(x+i, y+j) \tag{12-4}$$

where $s(x, y)$ represents the input image sample value (e.g. at 8-bit), $w(i, j)$ represents the 2-D low pass filter with a normalization factor N, R represents the filter range, BD stands for the expected bit depth at the high fidelity layer (with BD ≥ 8), and $s'(x, y)$ represents the filtered image sample (with bit depth of BD).

The 2-D filter can be further broken down into a combination of 1-D filters along four directions: horizontal, vertical, NW-SE, and NE-SW, as shown in Figure 49. Samples not along these directions in the 2-D windows shall have $w(i, j) = 0$.

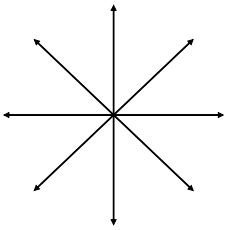

Figure 49 – Four direction for smoothing

In addition, there is a simple rule for each 1-D direction. For the horizontal direction as an example, *m* is the maximum absolute value of *i* that satisfies the constraint in equation (12-5):

$$\left|s(x+i,y)-s(x,y)\right| > T \quad \text{(with –R <= i <= R)} \tag{12-5}$$

where *T* is the filter threshold control parameter. The filter threshold control parameter *T* is SMOOTHING_TH_MINUS_1 + 1 as described in section 7.1.25 and the filter range parameter R is signaled in section 7.1.24. The *m* value shall be set to (*R*+1) in the case where it can not be obtained from equation (**12-5**). Then, the *w*(*i*, *j*) with j=0, -*R* <= *i* <= *R* shall be as defined in equation (12-6).

$$w(i,\ j) = \begin{cases} 1 & |i| < m \\ 0 & otherwise \end{cases} \tag{12-6}$$

When all the *w*(*i*, *j*) values are settled, the normalization factor *N* is ∑*w*(*i*, *j*).

The strength of the filter can be effectively controlled by *T* and *R*. *T* and *R* shall be in range of {1, 2, 3, 4} and {0, 1, 2, 3}, respectively. When *R*=0, there is no low-pass filtering. SMOOTHING_WND shall be *R* and SMOOTHING_TH_MINUS_1+1 shall be *T*. SMOOTHING_WND and SMOOTHING_TH_MINUS_1 are defined in Section 7.1.24 and Section 7.1.25, respectively.

Note: The advantage of the filter is to keep symmetric along each 1-D direction. The filter coefficients are set to 1 for simplicity.

12.1.2 Color Difference Up-conversion

There shall be three cases in the color difference up-conversion: 4:2:0-to-4:2:2 conversion, 4:2:2-to-444 conversion and 4:2:0-to-4:4:4 conversion. In the 4:2:0-to-4:2:2 conversion, the vertical color difference up-conversion only shall be performed. In the 4:2:2-to-4:4:4 conversion, the horizontal color difference up-conversion only shall be performed. In the 4:2:0-to-4:4:4 conversion, both vertical and horizontal conversions shall be performed sequentially. In this section, the vertical and horizontal conversions are defined independently because both conversions have different phase characteristics and the interlace features should be considered in the vertical direction. In the 4:2:0-to-4:4:4 conversion, we shall convert 4:2:0 to 4:2:2 and then convert 4:2:2 to 4:4:4.

12.1.2.1 Progressive Color Difference Up-conversion in Vertical Direction

The vertical color difference up-conversion method from 4:2:0 to 4:2:2 formats shall be selected based on COLOR_DIFFERENCE_UPCONV_IDX_U/COLOR_DIFFERENCE_UPCONV_IDX_V defined in Section 8.1.27 and Section 8.1.28.

Figure 50 shows the pixels that shall be used to compute the interpolated pixels for each case. S[n] denotes the interpolated pixel position for output 4:2:2 color difference samples. P[n] represents the pixel position in input 4:2:0 color difference samples.

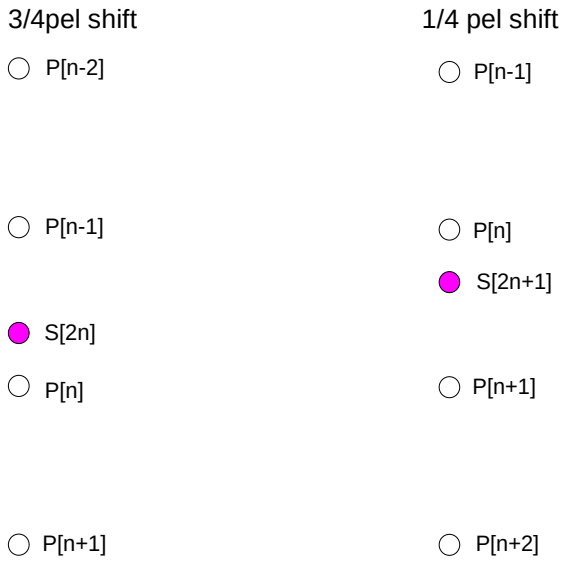

Figure 50 – Pixel shift corresponding to interpolated pixel position for progressive

If COLOR_DIFFERENCE_UPCONV_IDX_U/COLOR_DIFFERENCE_UPCONV_IDX_V == 0 then a 4-tab bi-cubic frame filter shall be used. The equations (**12-7**) and (**12-8**) shall specify the filtered result, S, for the possible shift locations of the interpolated pixel.

$$S[2n] = (-2*P[n-2] + 9*P[n-1] + 28*P[n] - 3*P[n+1] + 16) >> 5 \quad \text{(3/4 pel shift)} \quad (12\text{-}7)$$

$$S[2n+1] = (-3*P[n-1] + 28*P[n] + 9*P[n+1] - 2*P[n+2] + 16) >> 5 \quad \text{(1/4 pel shift)} \quad (12\text{-}8)$$

where n for S[n] is the index of the vertical position in the 4:2:2 color difference sample domain and n for P[n] is the index of the vertical position in the 4:2:0 color difference sample domain. If the height of the output picture is 2N, the range of n shall be from 0 to N-1.

If any pixels (P[n]) in the 4:2:0 sample domain are out of the picture boundary, then the boundary pixel shall be used instead of those pixels. When P[n-2] or P[n-1] is out of the picture boundary (in case of S[0], for example), P[n] shall be used instead of P[n-2] or P[n-1].

If COLOR_DIFFERENCE_UPCONV_IDX_U/COLOR_DIFFERENCE_UPCONV_IDX_V == 1 then a 2-tab bi-linear frame filter shall be used.

The equations (**12-9**) and (**12-10**) shall specify the filtered result, S[n], for the possible shift locations of the interpolated pixel.

$$S[2n] = (P[n-1] + 3*P[n] + 2) >> 2 \quad \text{(3/4 pel shift)} \quad (12\text{-}9)$$

$$S[2n+1] = (3*P[n] + P[n+1] + 2) >> 2 \quad \text{(1/4 pel shift)} \quad (12\text{-}10)$$

If the any pixels (P[n]) in the 4:2:0 sample domain are out of the picture boundary, then the boundary pixel shall be used instead of those pixels.

12.1.2.2 Interlaced Color Difference Up-conversion in Vertical Direction

The interlaced vertical color difference up-conversion method from 4:2:0 to 4:2:2 format shall be selected based on COLOR_DIFFERENCE_UPCONV_IDX_U/COLOR_DIFFERENCE_UPCONV_IDX_V in Section 10.1.30 and Section 10.1.31.

Figure 51 shows the pixels that shall be used to compute the interpolated pixels for each case. $S_T[n]$ and $S_B[n]$ denote the interpolated pixel position for the top and bottom field picture in the output 4:2:2 color

difference sample, respectively. $P_T[n]$ is the input 4:2:0 color difference sample in the top field and $P_B[n]$ is the input 4:2:0 color difference sample in the bottom field.

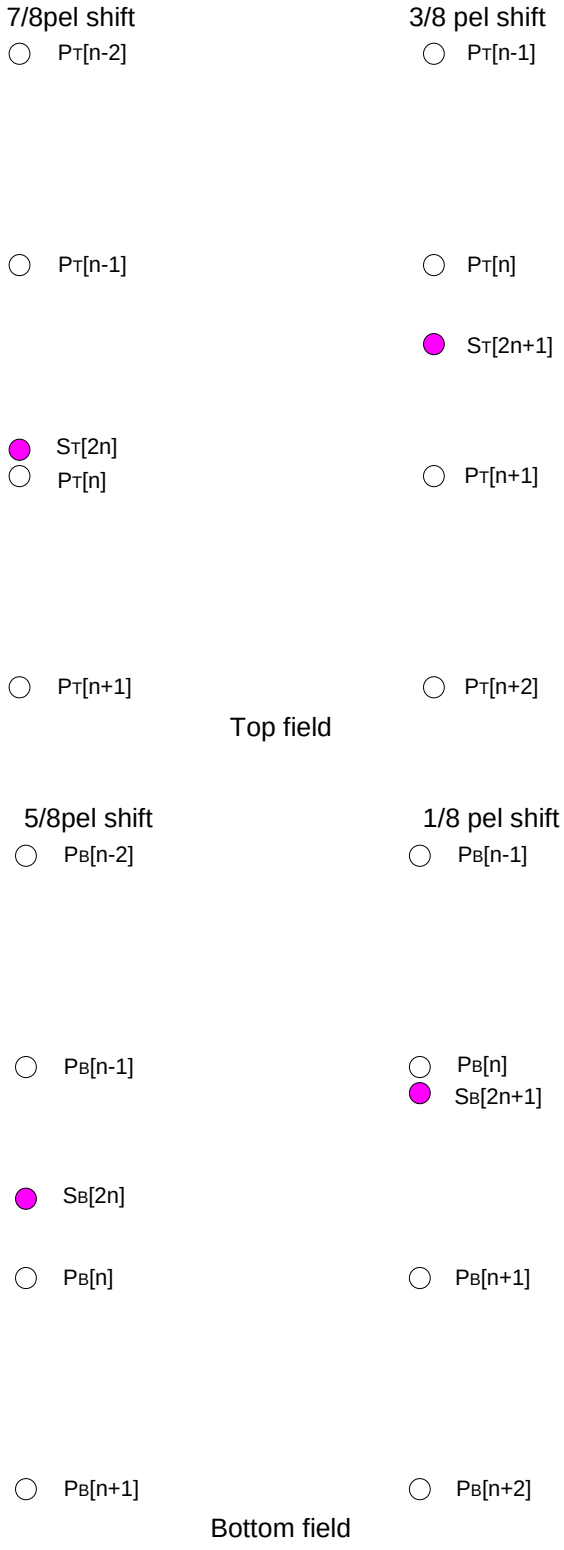

Figure 51 – Pixel shift corresponding to interpolated pixel position for interlace

If COLOR_DIFFERENCE_UPCONV_IDX_U/COLOR_DIFFERENCE_UPCONV_IDX_V == 0 then a 4-tap bi-cubic field filter shall be used. The equations (**12-11**), (**12-12**), (**12-13**) and (**12-14**) shall specify the filtered result, $S_T[n]$ and $S_B[n]$, for the possible shift locations of the interpolated pixel.

Top field case:

$$S_T[2n] = (-P_T[n-2] + 3*P_T[n-1] + 32*P_T[n] - 2*P_T[n+1] + 16) >> 5 \quad \text{(7/8 pel shift)} \quad (12\text{-}11)$$

$$S_T[2n+1] = (-2*PT[n-1] + 23*PT[n] + 13*PT[n+1] - 2*PT[n+2] + 16) >> 5 \quad \text{(3/8 pel shift)} \quad (12\text{-}12)$$

Bottom field case:

$$S_B[2n] = (-2*P_B[n-2] + 13*P_B[n-1] + 23*P_B[n] – 2*P_B[n+1] + 16) >> 5 \quad \text{(5/8 pel shift)} \quad (12\text{-}13)$$

$$S_B[2n+1] = (-2*PB[n-1] + 32*PB[n] + 3*PB[n+1] - PB[n+2] + 16) >> 5 \quad \text{(1/8 pel shift)} \quad (12\text{-}14)$$

where n for $S_T[n]$ or $S_B[n]$ is the index of the vertical position in the 4:2:2 color difference field sample domain and n for $P_T[n]$ or $P_B[n]$ is the index of the vertical position in the 4:2:0 color difference field sample domain. If the height of the output picture is 2N, the range of n shall be from 0 to N-1.

If the any pixels ($P_T[n]$ or $P_B[n]$) in the 4:2:0 sample domain are out of the picture boundary, then the boundary pixel shall be used instead of those pixels. When $P_T[n-2]$ or $P_T[n-1]$ is out of the top field picture boundary (in case of S[0], for example), $P_T[n]$ shall be used instead of $P_T[n-2]$ or $P_T[n-1]$. When $P_B[n-2]$ or $P_B[n-1]$ is out of the bottom field picture boundary (in case of $S_B[0]$, for example), $P_B[n]$ shall be used instead of $P_B[n-2]$ or $P_B[n-1]$.

If COLOR_DIFFERENCE_UPCONV_IDX_U/COLOR_DIFFERENCE_UPCONV_IDX_V == 1 then a 4-tap bi-cubic frame filter shall be used. In this case, the filtering process shall be identical to progressive case (Section 12.1.2.1). The 4-tap bi-cubic filter shall be applied to the input 4:2:0 color difference frame sample which shall be regarded as a sample in a frame picture.

If COLOR_DIFFERENCE_UPCONV_IDX_U/COLOR_DIFFERENCE_UPCONV_IDX_V == 2 then the de-interlacing frame filter shall be used.

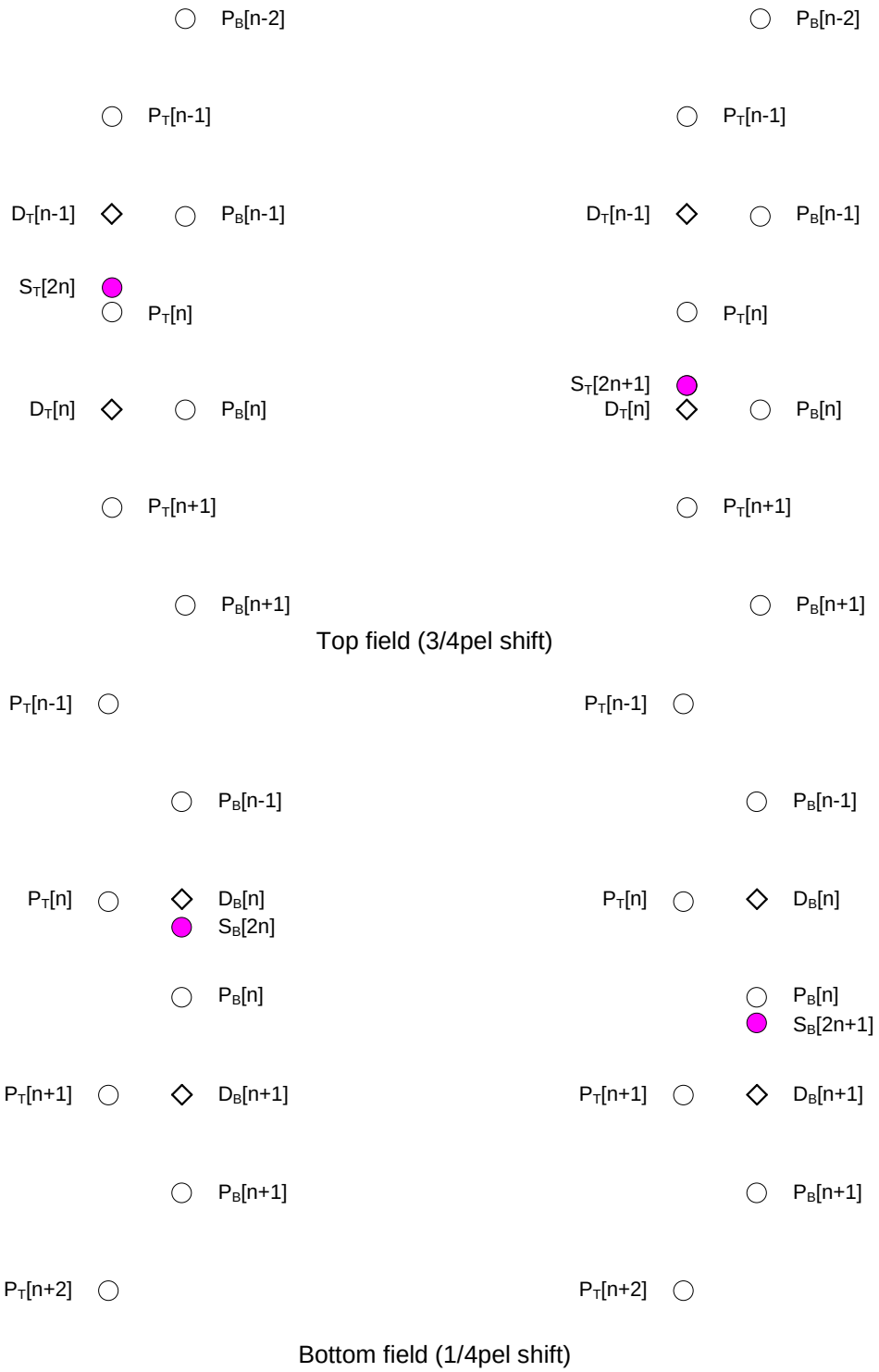

Figure 52 – Pixel position for deinterlacing filtering

Figure 52 shows the pixel position for deinterlacing filtering in each field. $P_T[n]$ is the input 4:2:0 color difference sample in the top field and $P_B[n]$ is the input 4:2:0 color difference sample in bottom field. For the pixel corresponding to opposite field position, the deinterlaced pixel $D_T[n]$ or $D_B[n]$ shall be obtained by equations (**12-15**) and (**12-16**).

$$D_T[n] = (-P_B[n-1] + 8*P_T[n] + 2*P_B[n] + 8*P_T[n+1] – P_B[n+1] + 8) >> 4 \quad \text{(Top field)} \tag{12-15}$$

$$D_B[n] = (-P_T[n-1] + 8*P_B[n-1] + 2*P_T[n] + 8*P_B[n] – P_T[n+1] + 8) >> 4 \quad \text{(Bottom field)} \tag{12-16}$$

The equations (**12-17**), (**12-18**), (**12-19**) and (**12-20**) shall specify the filtered result, S, for the possible shift locations of the interpolated pixel.

Top field case (3/4 pel shift):

$$S_T[2n] = (-2*P_T[n-1] + 9*D_T[n-1] + 28*P_T[n] - 3*D_T[n] + 16) >> 5 \tag{12-17}$$

$$S_T[2n+1] = (-2*D_T[n-1] + 9*P_T[n] + 28*D_T[n] - 3*P_T[n+2] + 16) >> 5 \tag{12-18}$$

Bottom field case (1/4 pel shift):

$$S_B[2n] = (-3*P_B[n-1] + 28*D_B[n] + 9*P_B[n] - 2*D_B[n+1] + 16) >> 5 \tag{12-19}$$

$$S_B[2n+1] = (-3*D_B[n] + 28*P_B[n] + 9*D_B[n+1] - 2*P_B[n+1] + 16) >> 5 \tag{12-20}$$

where $P_T[n]$ or $P_B[n]$ is the pixel of the 4:2:0 color difference frame sample and $D_T[n]$ or $D_B[n]$ is the de-interlaced filtered pixel. If the deinterlaced filter pixel is not used here, then this filtering process shall be identical to the 4-tap bi-cubic frame filter case. The locations of S_T or S_B in the vertical direction shall be same as the luma sample positions.

12.1.2.3 Color Difference Up-conversion in Horizontal Direction

The horizontal color difference up-conversion shall be performed when the color difference format of the lower layer is 4:2:0 or 4:2:2 and the color difference format of the upper layer is 4:4:4. When the 4:2:0-to-4:4:4 conversion is selected, the vertical up-conversion described in Section 12.1.2.1 or Section 12.1.2.2 shall be performed and then the horizontal up-conversion described in this section shall be performed. Figure 53 shows the pixel positions which shall be used to compute the interpolated pixels. P[n] represents the pixel positions of input color difference samples and S[n] denotes the interpolated pixel positions of output samples. Output samples shall be produced at potions 0 to 2N-1 for input samples at potions 0 to N-1 in horizontal axis.

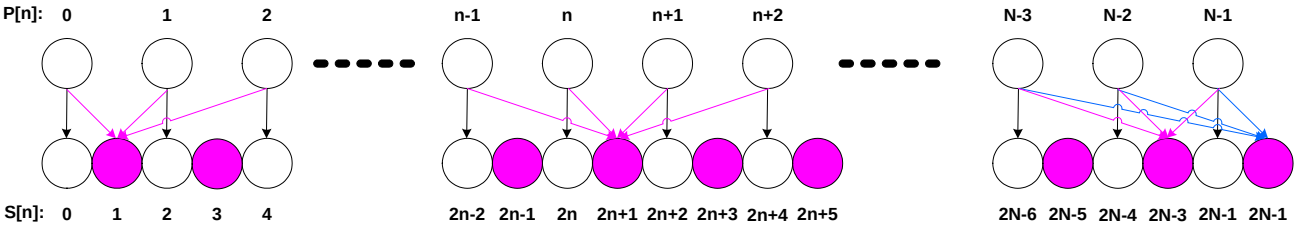

Figure 53 – Pixel position for horizontal color difference up-conversion

The equation (12-21) shall specify the filtered result, S, for locations 2n of the interpolation pixel.

$$S[2n] = P[n] \tag{12-21}$$

The equation (12-22) shall specify the filtered result, S for locations 2n+1 except positions 1, 2N-3, 2N-1 of the interpolated pixel

$$S[2n+1] = ((9(P[n] + P[n+1]) - (P[n-1]+P[n+2]) + 8) >> 4 \tag{12-22}$$

At positions 1, 2N-3 and 2N-1 of equation (12-22) where indices of the input samples point to the outside of the input sample domain, the filtered results shall be generated with the following equation (12-23), (12-24) and (12-25). The equation (12-23) shall specify the filtered result, S for location 1 of the interpolation pixel.

S[1] = ((9(P[0] + P[1]) - (P[0]+P[2]) + 8) >> 4 (12-23)

The equation (12-24) shall specify the filtered result, S for location 2N-3 of the interpolation pixel.

S[2N-3] = ((9(P[N-2] + P[N-1]) - (P[N-3]+P[N-1]) + 8) >> 4 (12-24)

The equation (12-25) shall specify the filtered result, S for locations 2N-1 of the interpolation pixel.

S[2N-1] = ((9(P[N-1] + P[N-1]) - (P[N-2]+P[N-1]) + 8) >> 4 (12-25)

12.1.3 Resolution Up-conversion

12.1.3.1 Progressive to Progressive Resolution Up-conversion

The predicted frame for the current layer frame shall be obtained by up-sampling the lower layer frame based on direct interpolation.

For the luma samples, the actual prediction sample *pred*[*X,Y*] at a position (*X,Y*) in the current layer shall be calculated as in equation (12-26):

pred[*X,Y*] = *interp*[*xI, yI*]
with

$$\begin{cases} xI = [X \cdot w_{lower} \cdot 16 + 8 \cdot (w_{lower} - w_{current})] // w_{current} \\ yI = [Y \cdot h_{lower} \cdot 16 + 8 \cdot (h_{lower} - h_{current})] // h_{current} \end{cases} \quad (12\text{-}26)$$

where *interp* is the sample array resulting from the 1/16-sample interpolation of the lower layer luma samples; $w_{current}$ and $h_{current}$ are the width and height of the current layer pictures, respectively; and w_{lower} and h_{lower} are width and height of the lower layer pictures, respectively.

A set of the default 4-tap filters shall be applied for the up-sampling process of the luma samples in the horizontal axis and the vertical axis when LOWER_LAYER_PRED_FLAG == 0. The default up-sampling 4-tap filter coefficients shall be as defined in Table 69. The normalization factor of the filters shall be 32.

Table 69 – Default filter coefficients for luma samples

phase	(4-tap) Interpolation filter coefficients			
	e[-1]	e[0]	e[1]	e[2]
0	0	32	0	0
1/16	-1	32	2	-1
2/16	-2	31	4	-1
3/16	-3	30	6	-1
4/16	-3	28	8	-1
5/16	-4	26	11	-1
6/16	-4	24	14	-2
7/16	-3	22	16	-3
8/16	-3	19	19	-3
9/16	-3	16	22	-3
10/16	-2	14	24	-4
11/16	-1	11	26	-4
12/16	-1	8	28	-3
13/16	-1	6	30	-3
14/16	-1	4	31	-2
15/16	-1	2	32	-1

When LOWER_LAYER_PRED_FLAG == 1, a set of the default 4-tap filters shall be applied for the up-sampling process of the luma samples in the vertical axis, and a set of the alternative 6-tap filters shall be applied for the up-sampling process of the luma samples in the horizontal axis. The alternative 6-tap up-sampling filter coefficients shall be as defined in Table 70. The normalization factor of the alternative filters shall be 128.

Table 70 – Alternative 6-tap filter coefficients for luma samples

phase	(6-tap) Interpolation filter coefficients					
	e[-2]	e[-1]	e[0]	e[1]	e[2]	e[3]
0	0	32	64	32	0	0
1/16	-1	30	65	35	0	-1
2/16	-2	27	64	38	2	-1
3/16	-3	24	63	41	4	-1
4/16	-3	22	61	43	6	-1
5/16	-4	18	59	47	9	-1
6/16	-4	16	58	50	10	-2
7/16	-3	16	57	51	10	-3
8/16	-3	13	54	54	13	-3
9/16	-3	10	51	57	16	-3
10/16	-2	10	50	58	16	-4
11/16	-1	9	47	59	18	-4
12/16	-1	6	43	61	22	-3
13/16	-1	4	41	63	24	-3
14/16	-1	2	38	64	27	-2
15/16	-1	0	35	65	30	-1

For the color difference samples, the actual prediction sample *predC*[*X, Y*] at a position (*X, Y*) in the current layer shall be calculated as in equation (**12-27**):

predC[*X, Y*] = *interpC*[*xI , yI*]
with

$$\begin{cases} xI = [X \cdot w_{lower,C} \cdot 16 + (8 + 4 \cdot p_{current,x}) \cdot w_{lower,C} - (8 + 4 \cdot p_{lower,x}) \cdot w_{current,C}] // w_{current,C} \\ yI = [Y \cdot h_{lower,C} \cdot 16 + (8 + 4 \cdot p_{current,y}) \cdot h_{lower,C} - (8 + 4 \cdot p_{lower,y}) \cdot h_{current,C}] // h_{current,C} \end{cases} \quad (12\text{-}27)$$

and *interpC* is the sample array resulting from the 1/16-sample interpolation of the lower layer color difference samples (*C* being either *Cr* or *Cb*). $w_{current,C}$, $h_{current,C}$, $w_{lower,C}$, $h_{lower,C}$ are the color difference geometric parameters, and $p_{lower,x}$, $p_{lower,y}$, $p_{current,x}$, $p_{current,y}$ are the horizontal and vertical color difference phase shifts in units of 1/4 sampling space, with respect to the luma pictures, of the lower and current layers.

A set of the default bilinear filters shall be applied for the up-sampling process of the color difference samples in the horizontal axis and the vertical axis when LOWER_LAYER_PRED_FLAG == 0. The default up-sampling bilinear filter coefficients shall be as defibed in Table 71. The normalization factor of the filters shall be 32.

Table 71 – Default bilinear filter coefficients for color difference samples

phase	Bilinear filter coefficients			
	e[-1]	e[0]	e[1]	e[2]
0	0	32	0	0
1/16	0	30	2	0
2/16	0	28	4	0
3/16	0	26	6	0
4/16	0	24	8	0
5/16	0	22	10	0
6/16	0	20	12	0
7/16	0	18	14	0
8/16	0	16	16	0
9/16	0	14	18	0
10/16	0	12	20	0
11/16	0	10	22	0
12/16	0	8	24	0
13/16	0	6	26	0
14/16	0	4	28	0
15/16	0	2	30	0

When LOWER_LAYER_PRED_FLAG == 1, a set of the default bilinear filters shall be applied for the up-sampling process of the color difference samples in the vertical axis, and a set of the alternative 4-tap filters shall be applied for the up-sampling process of the color difference samples in the horizontal axis. The alternative 4-tap up-sampling filter coefficients shall be as defined in Table 72. The normalization factor of the filters shall be 128.

Table 72 – Alternative 4-tap filter coefficients for color difference samples

phase	Bilinear filter coefficients			
	e[-1]	e[0]	e[1]	e[2]
0	32	64	32	0
1/16	30	62	34	2
2/16	28	60	36	4
3/16	26	58	38	6
4/16	24	56	40	8
5/16	22	54	42	10
6/16	20	52	44	12
7/16	18	50	46	14
8/16	16	48	48	16
9/16	14	46	50	18
10/16	12	44	52	20
11/16	10	42	54	22
12/16	8	40	56	24
13/16	6	38	58	26
14/16	4	36	60	28
15/16	2	34	62	30

12.1.3.2 Interlaced to Interlaced Resolution Up-conversion

All parameters ($w_{current}$, $h_{current}$, w_{lower}, h_{lower}), ($p_{lower,x}$, $p_{lower,y}$), and ($p_{current,x}$, $p_{current,y}$) correspond to frame-based pictures. The following two new parameters $\phi_{current,y}$ and $\phi_{lower,y}$ in equations (12-28) and (12-29) are defined for the generalized resampling processes.

$$\phi_{current,y} = \begin{cases} p_{current,y} - 1 & for\ current\ top\_field(4:2:0) \\ p_{current,y} + 1 & for\ current\ bot\_field(4:2:0) \\ 2 \cdot p_{current,y} & otherwise(current\ \ frame, etc.) \end{cases} \quad (12\text{-}28)$$

$$\phi_{lower,y} = \begin{cases} p_{lower,y} - 1 & for\ lower\ top\_field(4:2:0) \\ p_{lower,y} + 1 & for\ lower\ bot\_field(4:2:0) \\ 2 \cdot p_{lower,y} & otherwise(lower\ \ frame, etc.) \end{cases} \quad (12\text{-}29)$$

The parameters $\phi_{current,y}$ and $\phi_{lower,y}$ represent the color difference vertical phase position in units of 1/8 color difference samples of the current and lower pictures (either field or frame), respectively. It is required that the two corresponding field pictures be of the same parity (i.e., either top or bottom).

Given a luma sample in the current field at position (*X*, *Y*) in units of single luma samples, its corresponding position in the lower field (*xI*, *yI*) in units of 1/16 luma samples of the lower field shall be derived in equation (**12-30**)

$$\begin{cases} xI = [X \cdot w_{lower} \cdot 16 + 8 \cdot (w_{lower} - w_{current})] // w_{current} \\ yI = [Y \cdot h_{lower} \cdot 16 + 8 \cdot (h_{lower} - h_{current})] // h_{current} \end{cases} \quad (12\text{-}30)$$

where $w_{current}$ and $h_{current}$ are width and height of the current layer pictures, respectively; and w_{lower} and h_{lower} are width and height of the lower layer pictures, respectively. Based on the phase positions derived, the default up-sampling filters shall be chosen from Table 69 and the normalization factor of the filters shall be 32 when LOWER_LAYER_PRED_FLAG == 0. When LOWER_LAYER_PRED_FLAG == 1, the alternative 6-tap filters shall be chosen from Table 70 for the up-sampling in the vertical axis, the normalization factor of the alternative 6-tap filters shall be 128, and the up-sampling filters in the horizontal axis shall be chosen from Table 69.

Similarly, given a color difference sample in the current field at position (*X, Y*) in units of single color difference samples, its corresponding position in the lower field (xI, yl) in units of 1/16 color difference samples of the lower field shall be derived in equation (12-31)

$$\begin{cases} xI = [X \cdot w_{lower,C} \cdot 16 + (8 + 4 \cdot p_{current,x}) \cdot w_{lower,C} - (8 + 4 \cdot p_{lower,x}) \cdot w_{current,C}] // w_{current,C} \\ yI = [Y \cdot h_{lower,C} \cdot 16 + (8 + 2 \cdot \phi_{current,y}) \cdot h_{lower,C} - (8 + 2 \cdot \phi_{lower,y}) \cdot h_{current,C}] // h_{current,C} \end{cases} \quad (12\text{-}31)$$

Based on the phase positions derived, the default up-sampling filters shall be chosen from Table 71. The normalization factor of the filters shall be 32 when LOWER_LAYER_PRED_FLAG == 0. When LOWER_LAYER_PRED_FLAG == 1, the alternative 4-tap filters shall be chosen from **Table 72** for the up-sampling in the vertical axis, the normalization factor of the alternative 4-tap filters shall be 128, and the up-sampling filters in the horizontal axis shall be chosen from Table 71.

The interlaced-to-interlaced upsampling should be applied to the top field pair and the bottom field pair, respectively.

12.1.3.3 Progressive to Interlaced Resolution Up-conversion

For progressive-to-interlaced up-conversion, where the lower layer picture is a progressive frame and the current layer picture is two interlaced fields, the lower layer progressive frame shall be coincident with the current layer top field. The current bottom field has a time phase.

Given a luma sample in the current field at position (*X*, *Y*) in units of single luma samples, its corresponding position in the lower layer frame (*xI*, *yI*) in units of 1/16 luma samples of the lower layer frame can be derived in equation (12-32)

$$\begin{cases} xI = [X \cdot w_{lower} \cdot 16 + 8 \cdot (w_{lower} - w_{current})] // w_{current} \\ yI = [Y \cdot h'_{lower} \cdot 16 + 8 \cdot (h'_{lower} - h_{current})] // h_{current} \end{cases} \quad (12\text{-}32)$$

with

$$h'_{lower} = 2 \cdot h_{lower}$$

Based on the phase positions derived, the default up-sampling filters shall be chosen from Table 69 and the normalization factor of the filters shall be 32 when LOWER_LAYER_PRED_FLAG == 0. When LOWER_LAYER_PRED_FLAG == 1, the alternative 6-tap filters shall be chosen from Table 70 for the up-sampling in the vertical axis, the normalization factor of the alternative 6-tap filters shall be 128, and the up-sampling filters in the horizontal axis shall be chosen from Table 69.

Progressive to interlaced resolution up-conversion shall be achieved in two ways selected by the value of P2I_METHOD_FLAG in Section 10.1.18. When P2I_METHOD_FLAG == 1, the up-converted interlaced fields shall be generated by two lower layer frames. The lower layer frame shall be coincident with a top field of the

current layer picture. The bottom field of the current layer picture shall be generated by the lower layer frame and the next lower layer frame. In this case, the bottom field reflects the motion information as well and causes the residue signal between original sequence and up-converted sequence to be small. Additional memory is required to store one more frame. If P2I_METHOD_FLAG == 1 and P2I_WEIGHTING == 0, then the bottom field shall be interpolated from the two lower layer frames without a weighting factor. If P2I_METHOD_FLAG == 1 and P2I_WEIGHTING == 1, then the bottom field shall be generated from the weighted interpolation of the two lower layer frames. When P2I_METHOD_FLAG == 0 then only one frame shall be used to generate two fields of the current layer picture.

To simplify the whole resampling design, the up-sampling process is broken down into two stages. In the first stage, the lower layer frame shall be upsampled into a picture *P* with $w_{current}$ and $h_{current}/2$ using Table 69 or Table 70. In the second stage, the upsampled picture *P* shall be vertically upsampled to generate the current layer picture *C* which consists of the top field and bottom field defined in equations (12-33), (12-34), (12-35), (12-36), (12-37), and (12-38).

When P2I_METHOD_FLAG == 0,

Top field

$$P_C[2i] = (32 \cdot P_p[i] + 16) // 32 \tag{12-33}$$

Bottom field

$$\begin{aligned} P_C[2i+1] = (&P_P[i-3] - 5 \cdot P_P[i-2] + 20 \cdot P_P[i-1] \\ &+ 20 \cdot P_P[i] - 5 \cdot P_P[i+1] + P_P[i+2] + 16) // 32 \end{aligned} \tag{12-34}$$

where P_P is a pixel of the upsampled picture *P* and P_C is a pixel of the current layer picture *C.* The index *i* of the upsampled picture pixel P_P is the vertical index of the same column. If the index is less than zero, the index shall be replaced by zero. If the index is greater than $h_{current}$ -1, the index shall be replaced by $h_{current}$ -1.

When P2I_METHOD_FLAG == 1 and P2I_WEIGHTING == 0,

Top field

$$P_C[2i] = (32 \cdot P_p[i] + 16) // 32 \tag{12-35}$$

Bottom field

$$P_C[2i+1] = (P_P[i-1] + P_P[i] + P_{P,NEXT}[i-1] + P_{P,NEXT}[i] + 2) // 4 \tag{12-36}$$

where $P_{P,NEXT}$ is a pixel of the upsampled next picture P_{NEXT} and colocated with P_P.

When P2I_METHOD_FLAG == 1 and P2I_WEIGHTING == 1,

Top field

$$P_C[2i] = (32 \cdot P_p[i] + 16) // 32 \tag{12-37}$$

Bottom field

$$P_C[2i+1] = (\alpha \cdot P_P[i-1] + \alpha \cdot P_P[i] + P_{P,NEXT}[i-1] + P_{P,NEXT}[i] + \alpha + 1) // \{2(\alpha+1)\} \quad (12\text{-}38)$$

where α is defined as P2I_WEIGHTING_FACTOR_MINUS_2+2.

Similarly, given a color difference sample in the current field at position (*X, Y*) in units of single color difference samples, its corresponding position in the lower layer frame (xI, yI) in units of 1/16 color difference samples of the lower layer frame shall be derived in equation (12-39)

$$\begin{cases} xI = [X \cdot w_{lower,C} \cdot 16 + (8 + 4 \cdot p_{current,x}) \cdot w_{lower,C} - (8 + 4 \cdot p_{lower,x}) \cdot w_{current,C}] // w_{current,C} \\ yI = [Y \cdot h'_{lower,C} \cdot 16 + (8 + 2 \cdot \phi_{current,y}) \cdot h'_{lower,C} - (8 + 2 \cdot \phi_{lower,y}) \cdot h_{current,C}] // h_{current,C} \end{cases} \quad (12\text{-}39)$$

with

$$h'_{lower,C} = 2 \cdot h_{lower,C}$$

Based on the phase positions derived, the default up-sampling filters shall be chosen from Table 71. The normalization factor of the filters shall be 32 when LOWER_LAYER_PRED_FLAG == 0. When LOWER_LAYER_PRED_FLAG == 1, the alternative 4-tap filters shall be chosen from Table 72 for the up-sampling in the vertical axis, the normalization factor of the alternative 4-tap filters shall be 128, and the up-sampling filters in the horizontal axis shall be chosen from Table 71.

The upsampled picture *P* shall be vertically upsampled to generate the current layer picture *C* which consists of the top field and bottom field in equations (12-40), (12-41), (12-42), (12-43), (12-44), and (12-45).

When P2I_METHOD_FLAG == 0

Top field

$$P_C[2i] = (32 \cdot P_p[i] + 16) // 32 \quad (12\text{-}40)$$

Bottom field

$$P_C[2i+1] = (16 \cdot P_P[i-1] + 16 \cdot P_P[i] + 16) // 32 \quad (12\text{-}41)$$

When P2I_METHOD_FLAG == 1 and P2I_WEIGHTING == 0,

Top field

$$P_C[2i] = (32 \cdot P_p[i] + 16) // 32 \quad (12\text{-}42)$$

Bottom field

$$P_C[2i+1] = (P_P[i-1] + P_P[i] + P_{P,NEXT}[i-1] + P_{P,NEXT}[i] + 2) // 4 \quad (12\text{-}43)$$

When P2I_METHOD_FLAG == 1 and P2I_WEIGHTING == 1,

Top field

$$P_C[2i] = (32 \cdot P_P[i] + 16) // 32 \tag{12-44}$$

Bottom field

$$P_C[2i+1] = (\alpha \cdot P_P[i-1] + \alpha \cdot P_P[i] + P_{P,NEXT}[i-1] + P_{P,NEXT}[i] + \alpha + 1) // \{2(\alpha + 1)\} \tag{12-45}$$

where α is defined as P2I_WEIGHTING_FACTOR_MINUS_2+2.

12.1.3.4 Interlaced to Progressive Resolution Up-conversion

For up-conversion, the top field of a lower layer frame shall be up-converted to a first field of the same parity using the methods described in Section 12.1.3.2. The odd lines of the current layer frame shall be filled by the interpolation using the up-converted even lines. The odd lines of luma and color difference samples shall be filled by equation (**12-46**) and equation(**12-47**), respectively.

$$\begin{aligned} P_C[2i+1] = (&P_P[i-2] - 5 \cdot P_P[i-1] + 20 \cdot P_P[i] \\ &+ 20 \cdot P_P[i+1] - 5 \cdot P_P[i+2] + P_P[i+3] + 16) // 32 \end{aligned} \tag{12-46}$$

$$P_C[2i+1] = (16 \cdot P_P[i] + 16 \cdot P_P[i+1] + 16) // 32 \tag{12-47}$$

The bottom field of a lower layer frame shall be up-converted in the same way. After the up-conversion of the bottom field, the even lines of the current layer frame shall be filled by the interpolation using the up-converted odd lines. The even lines of luma and color difference samples shall be filled by equation (**12-48**) and equation (**12-49**), respectively.

$$\begin{aligned} P_C[2i] = (&P_P[i-3] - 5 \cdot P_P[i-2] + 20 \cdot P_P[i-1] \\ &+ 20 \cdot P_P[i] - 5 \cdot P_P[i+1] + P_P[i+2] + 16) // 32 \end{aligned} \tag{12-48}$$

$$P_C[2i] = (16 \cdot P_P[i-1] + 16 \cdot P_P[i] + 16) // 32 \tag{12-49}$$

The current layer picture has a double frame rate. Here, if the resolution of the current layer is the same as the resolution of the lower layer, the top field of the lower layer shall be up-converted to a frame picture with the same resolution and the bottome field of the lower layer shall be also up-converted to the next frame picture with the same resolution.

12.1.4 Stereoscopic Conversion

The prediction picture for the current layer picture shall be obtained by a macroblock-based view or temporal prediction. In this conversion process, the opposite-side view of the current one is considered as the base

layer. If VIEW_PRED_FLAG == 0 (described in Section 8.1.39) for the current macroblock, the temporal prediction shall be performed by referring previously reconstructed picture of the current view (current layer). If VIEW_PRED_FLAG == 1, the view prediction shall be performed by referring to the reconstructed picture of the opposite-side view (base layer). The decoding process shall be as shown in Figure 54. In Figure 54, the reconstruction of the video of the second layer (layer 1) shall be performed by adding the reconstructed residual from the layer 1 to the view-predicted video from the base layer or the temporal-predicted video from the previous reconstruction of the layer 1.

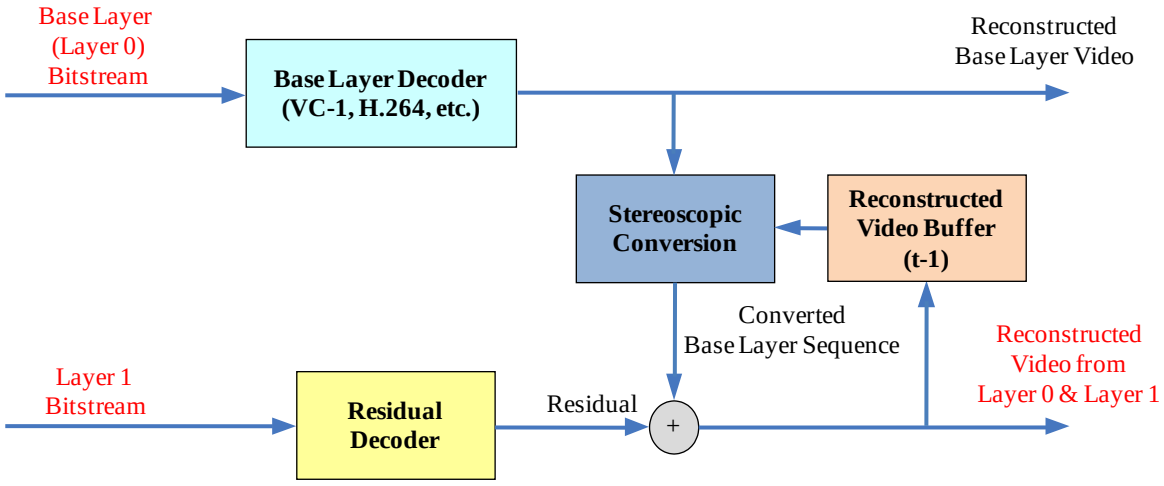

Figure 54 – Stereoscopic decoding process block diagram

12.1.4.1 Disparity/Motion Vector

In order to calculate the disparity or motion vector of the current macroblock, the disparity or motion vector differentials (DMVDx and DMVDy) shall be decoded. The decoding procedure of DMVDx shall be the same as the DMVDy. The pseudo-code of Figure 55 describes the procedure for decoding DMVDx or DMVDy.

```
//DMVD represents the disparity/motion vector differential in x-axis or y-axis
decode_DMVD_symbol() {
  for(numZeros=0; ; numZeros++)
      if(get_bits(1) == 1)
         break;

  if ( numZeros > 0) {
      val = (1 << numZeros) + get_bits(numZeros) - 1;
      if ( (val & 1) == 1)
         DMVD = (val+1) >> 1;
      else
         DMVD = -(val >> 1);
  }
  else
     DMVD = 0;

   return DMVD;
}
```

Figure 55 – Pseudo-code of DMVD decoding

After decoding DMVD, the disparity or motion vector shall be calculated according to equation (**12-50**):

$$DMVx = PDMVx + DMVDx$$
$$DMVy = PDMVy + DMVDy \quad (12\text{-}50)$$

where *PDMVx* and *PDMVy* are disparity or motion vector predictors obtained from Section 12.1.4.2, and *DMVDx* and *DMVDy* are the disparity or motion vector differentials.

12.1.4.2 Disparity/Motion Vector Prediction

The disparity/motion vector shall be computed by adding the disparity/motion vector differential (DMVD) in Section 8.1.41 and Section 8.1.42 to a disparity/motion vector predictor. The units for all disparity/motion vectors shall be as defined in Section 8.1.13 and Section 10.1.15. The disparity/motion vector predictor shall be derived from some of neighboring disparity or motion vectors.

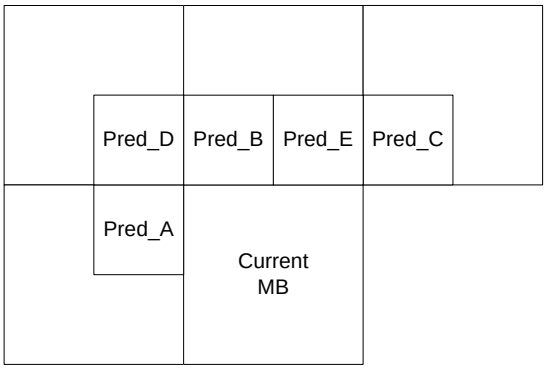

Figure 56 – Candidate disparity/motion vectors for 1-DMV macroblocks

Figure 56 shows the candidate disparity/motion vectors for an 1-DMV macroblocks. The neighboring macroblocks are either 1-DMV or 4-DMV macroblocks. The figure shows the candidate disparity/motion vectors assuming the neighbors are 4-DMV macroblocks. If any of the neighbors are 1-DMV macroblocks, the disparity/motion vector for the macroblocks shall be taken to be the disparity/motion vector for the entire macroblocks.

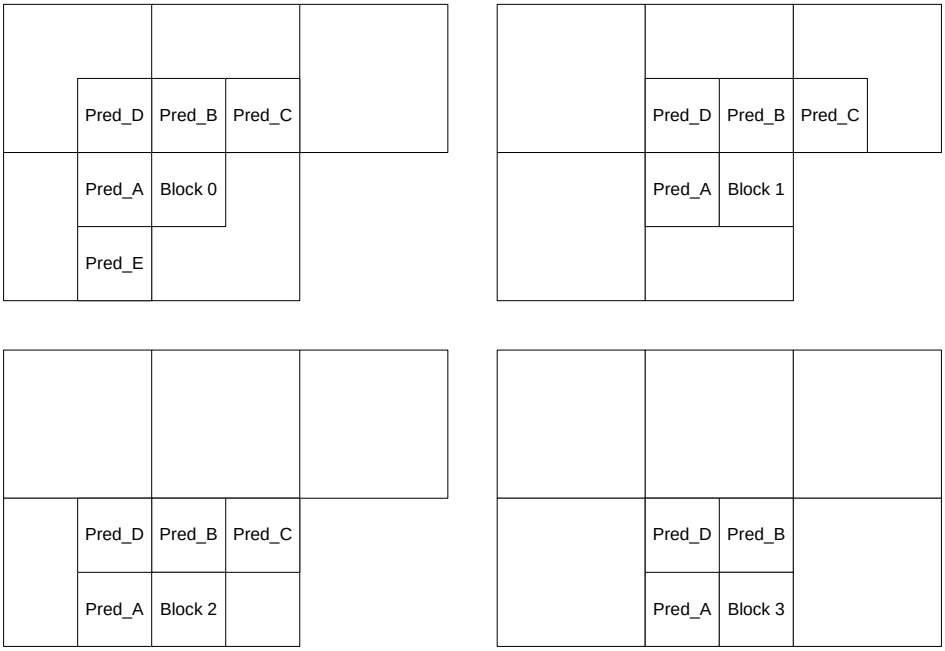

Figure 57 – Candidate disparity/motion vectors for 4-DMV macroblocks

Figure 57 shows the candidate disparity/motion vectors for each of the 4 luma blocks in a 4-DMV macroblock. The neighboring macroblocks are either 1-DMV or 4-DMV macroblocks. If the candidate disparity/motion vector is intra, outside the frame boundary, or part of a different slice, the candidate disparity/motion vector shall be regarded as unavailable. The pseudo-code of Figure 58 specifies the process for calculating disparity/motion vector predictors. At most 3 disparity/motion vector predictor candidates are used for deriving disparity/motion vector predictor.

```
neighbour_count = 0
if (Pred_A is available) {
  dmvx[neighbour_count] = dmvx[Pred_A]
  dmvy[neighbour_count] = dmvy[Pred_A]
  neighbour_count ++
}
if (Pred_B is available) {
  dmvx[neighbour_count] = dmvx[Pred_B]
  dmvy[neighbour_count] = dmvy[Pred_B]
  neighbour_count ++
}
if (!block 3 of 4-MV macroblock && Pred_C is available) {
  dmvx[neighbour_count] = dmvx[Pred_C]
  dmvy[neighbour_count] = dmvy[Pred_C]
  neighbour_count ++
}
if (neighbour_count<3 && Pred_D is available) {
  dmvx[neighbour_count] = dmvx[Pred_D]
  dmvy[neighbour_count] = dmvy[Pred_D]
  neighbour_count ++
}
if (neighbour_count<3 && block 0 of 4-DMV macroblock && Pred_E is available) {
  dmvx[neighbour_count] = dmvx[Pred_E]
```

```
  dmvy[neighbour_count] = dmvy[Pred_E]
  neighbour_count ++
}
if (neighbour_count<3 && 1-DMV macroblock && Pred_E is available) {
  dmvx[neighbour_count] = dmvx[Pred_E]
  dmvy[neighbour_count] = dmvy[Pred_E]
  neighbour_count ++
}
if (neighbour_count == 0){
  dmvpx = 0
  dmvpy = 0
}
else if (neighbour_count < 3){
  dmvpx = dmvx[0]
  dmvpy = dmvy[0]
}
else {
  if ( |dmvx[0] – dmvx[1]| < |dmvx[1] – dmvx[2]| ) {
    dmvpx = dmvx[0]
  }
  else {
    dmvpx = dmvx[2]
  }
  if ( |dmvy[0] – dmvy[1]| < |dmvy[1] – dmvy[2]| ) {
    dmvpy = dmvy[0]
  }
  else {
    dmvpy = dmvy[2]
  }
}
```

Figure 58 – Pseudo-code for disparity/motion vector predictor derivation

12.1.4.3 Disparity/Motion Compensation

The disparity/motion vector calculated in Section 12.1.4.1 shall be used to obtain the predicted block in the view/temporal reference frame. If VIEW_PRED_FLAG == 0, the predicted block shall be obtained from the temporal reference frame. If VIEW_PRED_FLAG == 1, the predicted block shall be obtained from the view reference frame. The horizontal and vertical disparity/motion vector components represent the displacement between the block currently being decoded and the corresponding location in the view/temporal reference frame. Positive values represent locations that are below and to the right of the current location. Negative values represent locations that are above and to the left of the current location. The actual reconstructed disparity/motion vector shall be used by subsequent macroblocks as a reference for disparity/motion vector predictor calculation. The motion vector shall be adjusted to point the area [-18:PICTURE_WIDTH+17] x [-18:PICTURE_HEIGHT+17] before the motion compensation. Here, each location in the outside of the actual image shall be filled with the nearest image pixel.

The following operation shall be applied to obtain the predicted luma and color difference samples.

- Let (iXCoord, iYCoord) be the spatial location given in full-sample units of the top left corner of the current macroblock or block (e.g. if the current macroblock is located on the 2nd row and 3rd column, then iXCoord = 2 * 16 and iYCoord = 1 * 16).

For each luma sample location (0 <= x < BlockWidth, 0 <= y < BlockHeight), the corresponding predicted luma sample value shall be derived as follows:

- Let (*DMVx*, *DMVy*) be the disparity/motion vector obtained from Section 12.1.4.1.
- Let (iDMvX, iDMvY) be the reconstructed luma disparity/motion vector in quarter pixel unit.
  If MV_MODE == 2, iDMvX = *DMVx* << 2 and iDMvY = *DMVy* << 2.
  If MV_MODE == 1, iDMvX = *DMVx* << 1 and iDMvY = *DMVy* <<1.
  If MV_MODE == 0, iDMvX = *DMVx* and iDMvY = *DMVy*.
- Let (xIntLuma, yIntLuma) be the spatial luma location in full-sample units, and (xFracLuma, yFracLuma) be an offset given in quarter sample units.

The spatial location of predicted luma block shall be computed according to the following equations.

- xIntLuma = (iXCoord * 4 + iDMvX) >> 2
- yIntLuma = (iYCoord * 4 + iDMvY) >> 2

- xFracLuma = iDMvX & 3
- yFracLuma = iDMvY & 3

The predicted luma sample values shall be derived by invoking the process in Section 12.1.4.4.

For each color difference sample location (0 <= x < BlockWidthChr, 0 <= y < BlockHeightChr), the corresponding predicted color difference sample value shall be derived as follows:

- Let (*DMVx*, *DMVy*) be the disparity/motion vector obtained from Section 12.1.4.1.
- Let (iDMvX, iDMvY) be the reconstructed color difference disparity/motion vector in quarter pixel units.
  If MV_MODE == 2, iDMvX = *DMVx* << 2 and iDMvY = *DMVy* << 2.
  If MV_MODE == 1, iDMvX = *DMVx* << 1 and iDMvY = *DMVy* << 1.
  If MV_MODE == 0, iDMvX = *DMVx* and iDMvY = *DMVy*.
- Let (xIntChr, yIntChr) be the spatial color difference location in full-sample units, and (xFracChr, yFracChr) be an offset given in quarter sample units.

The spatial location of the predicted color difference block shall be computed according to the following equations.

If COLORDIFF_FORMAT == 0 (4:2:0), then the following operations shall be performed.

- xIntChr = (iXCoord * 4 + iDMvX) >> 3
- yIntChr = (iYCoord * 4 + iDMvY) >> 3

- xFracChr = iDMvX & 7
- yFracChr = iDMvY & 7

If COLORDIFF_FORMAT == 1 (4:2:2), then the following operations shall be performed.

- xIntChr = (iXCoord * 4 + iDMvX) >> 3
- yIntChr = (iYCoord * 4 + iDMvY) >> 2

- xFracChr = iDMvX & 7
- yFracChr = iDMvY & 3

If COLORDIFF_FORMAT == 2 (4:4:4), then the following operations shall be performed.

- xIntChr = (iXCoord * 4 + iMvX) >> 2
- yIntChr = (iYCoord * 4 + iMvY) >> 2

- xFracChr = iMvX & 3
- yFracChr = iMvY & 3

The predicted color difference sample values shall be derived by invoking the process in Section 12.1.4.5.

12.1.4.4 Luma Sample Interpolation Process

Integer pixel disparity/motion vectors do not require the computation of interpolated pixels. For non-integer pixel disparity/motion vector values, samples at fractional sample positions shall be derived from 4-nearest integer pixel position with a bicubic interpolation filter.

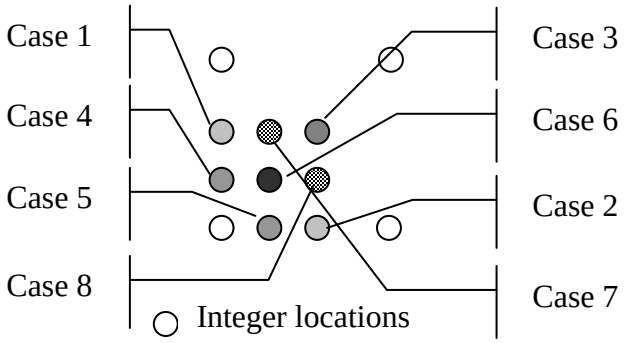

Figure 59 – Bicubic filter cases

Figure 59 shows all the possible unique interpolated positions, which are:

1. full-pel horizontal, half-pel vertical
2. half-pel horizontal, full-pel vertical
3. half-pel horizontal, half-pel vertical
4. full-pel horizontal, quarter-pel vertical
5. quarter-pel horizontal, full-pel vertical
6. quarter-pel horizontal, quarter-pel vertical
7. quarter-pel horizontal, half-pel vertical
8. half-pel horizontal, quarter-pel vertical

Note: The other possible interpolated positions are identical to one of the cases 1-8, and hence are not shown in Figure 59.

In Figure 59, the relative position of the integer pixel locations: a (bottom-left), b (top-left), c (top-right), d (bottom-right).

12.1.4.4.1 Bicubic Filter Coefficients for different shift locations

Figure 60 shows the pixel that shall be used to compute the interpolated pixels for each case. S denotes the sub-pixel position. P1, P2, P3 and P4 represent the integer pixel positions. The figure shows horizontal interpolation but the same operation shall apply to vertical interpolation.

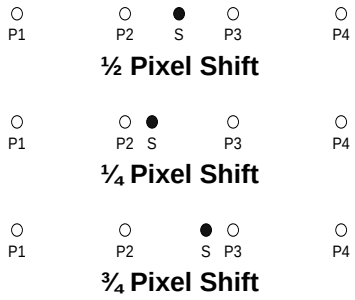

Figure 60 – Pixel shifts

The following filters shall be used for the possible shift locations of interpolated pixel:

½ pel shift F1: [-1 9 9 -1]
¼ pel shift F2: [-4 53 18 -3]
¾ pel shift F3: [-3 18 53 -4]

The following equations shall specify the filtered result, S, for the possible shift locations of the interpolated pixel:

S = (-1*P1 + 9*P2 + 9*P3 -1*P4) (1/2 pixel shift)
S = (-4*P1 + 53*P2 + 18*P3 -3*P4) (1/4 pixel shift)
S = (-3*P1 + 18*P2 + 53*P3 -4*P4) (3/4 pixel shift)

12.1.4.4.2 One-dimensional Bicubic Interpolation (Cases 1, 2, 4 and 5)

In Figure 59, cases 1, 2, 4 and 5 are where interpolation occurs in only one dimension – either horizontal or vertical

The following rounding shall be applied after the filtering operation for each case:

(S + 8) >> 4 (1/2 pixel shift)
(S + 32) >> 6 (1/4 pixel shift)
(S + 32) >> 6 (3/4 pixel shift)

where S is the filtered result after applying the filter appropriate for the location of the interpolated pixel.

12.1.4.4.3 Two-dimensional Bicubic Interpolation

In Figure 59, cases 3, 6, 7 and 8 are where interpolation occurs in both the horizontal and vertical directions.

Two-dimensionally interpolated pixel locations shall first interpolate along the vertical direction, and then along the horizontal direction using the appropriate filter among F1, F2 and F3 specified above. Rounding shall be applied after vertical filtering and after horizontal filtering.

For example, the two-dimensional interpolation for case 8, which has a quarter-pel shift vertically and a half-pel shift horizontally, uses the filter F2 for interpolation along the vertical direction and F1 for interpolation along the horizontal direction.

The rounding rule after vertical filtering shall be defined as
$(S + 2^{shiftV-1}) >> shiftV$

where

S = vertically filtered result after applying the filter appropriate for the vertical shift location of the interpolated pixel, i.e. -1*P1 + 9*P2 + 9*P3 -1*P4 for ½ pixel shift

shiftV = { 1, 5, 3, 3 } for cases 3, 6, 7 and 8 respectively.

The rounding rule after horizontal filtering shall be defined as:
(*S* + 64) >> 7.

where

S = horizontally filtered result after applying the filter appropriate for the horizontal shift location of the interpolated pixel,

All of the bicubic filtering cases potentially produce an interpolated pixel whose value is negative, or larger than the maximum range (255). In these cases, the CLIP() operator shall be applied so that the output lies within the range.

12.1.4.5 Color Difference Sample Interpolation Process

The color difference sample interpolation process employs the bi-linear interpolation. If the precision of the disparity/motion vectors is 1/4-pel, the range of xFracChr shall be from 0 to 7 when COLORDIFF_FORMAT == 0 (4:2:0) or COLORDIFF_FORMAT == 1 (4:2:2), while the range of xFracChr shall be from 0 to 3 when COLORDIFF_FORMAT == 2 (4:4:4). The range of the yFracChr shall be from 0 to 7 if COLORDIFF_FORMAT == 0 (4:2:0), while the range of the yFracChr shall be from 0 to 3 if COLORDIFF_FORMAT == 1 (4:2:2) or if COLORDIFF_FORMAT == 2 (4:4:4)

Each color difference sample at a fractional sample shall be derived as follows. A, B, C and D are the same as Figure 25.

If COLORDIFF_FORMAT == 0, the interpolated color difference sample shall be
ColorDifferenceInterpolated = ((8 – xFracChr) * (8 – yFracChar) * A + xFracChr * (8 – yFracChar) * B
+ (8 – xFracChr) * yFracChar * C + xFracChr * yFracChar * D + 32) >> 6

If COLORDIFF_FORMAT == 1, the interpolated color difference sample shall be
ColorDifferenceInterpolated = ((8 – xFracChr) * (4 – yFracChar) * A + xFracChr * (4 – yFracChar) * B
+ (8 – xFracChr) * yFracChar * C + xFracChr * yFracChar * D + 16) >> 5

If COLORDIFF_FORMAT == 2, the interpolated color difference sample shall be
ColorDifferenceInterpolated = ((4 – xFracChr) * (4 – yFracChar) * A + xFracChr * (4 – yFracChar) * B
+ (4 – xFracChr) * yFracChar * C + xFracChr * yFracChar * D + 8) >> 4

12.1.5 1-D Filtered Prediction on Reconstructed Pictures from Lower Layer

This section describes the 1-D filtered prediction on the reconstructed pictures from the lower layer. This process shall applied when LOWER_LAYER_PRED_FLAG == 1 and both resolutions of the lower layer and

the current layer are same each other. The case that both resolutions are different is described in Section 12.1.3. The equation (12-51) shall specify the filtered result, S for location n of the filtered picture.

$$S[n] = (P[n-1] + 2P[n] + P[n+1] + 2) >> 2 \quad (12\text{-}51)$$

where P[n] is the pixel of the reconstructed picture from the lower layer and the range of the input samples in the horizontal axis is 0 to N-1. This filter shall be applied in the only horizontal direction. When P[n-1] or P[n+1] is out of the picture boundary, P[0] shall be used instead of P[n-1], and P[N-1] shall be used instead of P[n+1]. This process shall be performed for the luma samples and the color difference samples in the same way.

12.2 Residual Mapping/Scaling to Up-converted Bit Depth

This section describes the residual mapping/scaling operation at the decoder side.

For each color component, three parameters *Scale*, *Shift* and *Norm* shall be signaled to indicate the residual mapping/scaling parameters such as SCALE_Y, SHIFT_Y, NORM_Y, SCALE_U, SHIFT_U, NORM_U, SCALE_V, SHIFT_V and NORM_V. Then equation (12-52) shall beperformed on the decoded residual samples $d(x, y)$ of that color component

$$r(x,y) = \left(\left(d(x,y) - Shift\right) * Scale\right) >> Norm \quad (12\text{-}52)$$

Here, the shifing by *Norm* is the arithmetic shift. In the right arithmentic shift, the sign bit of the result after scaling by *Scale* is shifted in on the left during the right-shifing by *Norm*, thus preserving the sign of the operand. If RES_SCALING_PRESENT_FLAG == 1, residual scaling parameters shall be present in the bitstream. If RES_SCALING_PRESENT_FLAG == 0, no residual scaling parameters shall not be present, and the default values shall be *Scale* = 1, *Shift* = 128, and *Norm* = 0.

The mapped and scaled residual $r(x, y)$ shall be added to the upsampled samples to reconstruct displayable pictures. The reconstructed sample $s(x, y)$ shall be in the range of the bit depth which is specified by "BIT_DEPTH_MINUS_8+8" where BIT_DEPTH_MINUS_8 is defined in the sequence header. Equation (12-53) makes the $s(x, y)$ be in the range corresponding bit depth.

$$s(x,y) = \min(\max(r(x,y) + u(x,y), \quad 0), \quad 2^{BIT\_DEPTH\_MINUS\_8+8} - 1) \quad (12\text{-}53)$$

where $u(x, y)$ is the upsampled sample.

Annex A Hypothetical Reference Decoder (Normative)

Coded video streams shall meet the constraints imposed by hypothetical reference decoder (HRD) defined in this section. Note that this is similar to the HRD model of SMPTE ST 421, Annex C.

Figure A.1 shows the conceptual components of the HRD. Figure A.1 shows the configuration of the HRD comprised of one base layer, which is decoded by an arbitrary existing video compression standard, and one residual layer as the enhancement layer. Without loss of generality, there could be more than two enhancement layers. The multiplexed bit streams are demultiplexed and put into the corresponding decoder buffers. The HRD in Figure A.1 is conceptually connected to the output of an encoder, and comprises a residual decoder buffer, a residual decoder, a residual converter, format converter, a base layer decoder buffer, and a base layer decoder. The HRD does not mandate buffering, decoding, or display mechanisms for the decoder implementations. Its purpose is to limit the encoder's bit rate fluctuations according to a basic buffering model, so that the resources necessary to decode the bitstream are predictable. Here, the base layer decoder buffer and the base layer decoder operate according the corresponding standard.

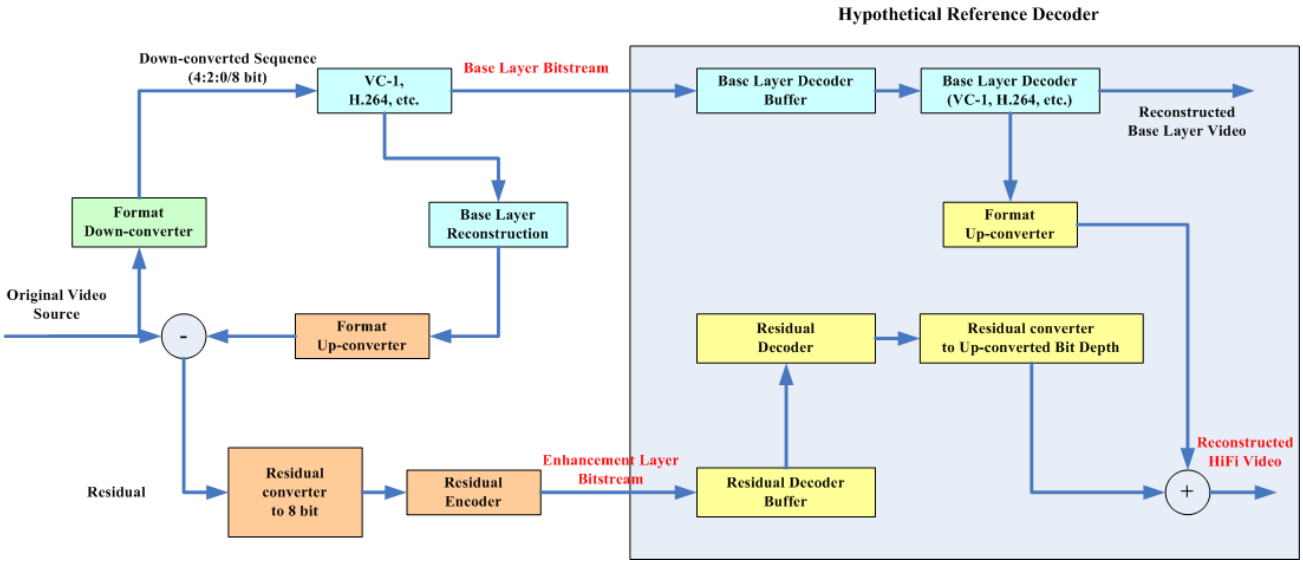

Figure A.1 – Conceptual components of an HRD

The buffering model of the residual codec in the HRD can be also described with the leaky bucket model. A leaky bucket model shall be specified with three parameters: the peak transmission bit rate R, the capacity of the coded frame buffer B, and the initial buffer fullness F_0. Here, the coded frame buffer contains the bitstream to be decoded and is corresponding to the residual decoder buffer in Figure . In the HRD, the enhancement layer bitstream shall be received at bit rate smaller than or equal to the peak transmission rate R, and it shall be stored in a residual coded frame buffer until the buffer fullness reaches the initial buffer fullness, F_0 bits. This time is referred to as the initial delay which is called "vbv_delay" in ISO/IEC 13818-2. This interval is indicated in Figure A.2 as t_1-t_0 = F_0/R. Then the decoder takes the bits for the first frame of the enhancement layer sequence from the buffer, and instantaneously decodes that frame. The bits for the following enhancement layer frames are also taken and decoded instantaneously at subsequent time intervals. Since the residual codec accesses the buffer by frame unit not field unit, if a frame is coded as two interlaced fields, the bits for both fields shall be taken as a pair and decoded instantaneously. As the HRD model of most video standards, the removal of bits from the residual coded frame buffer and decoding are treated as instantaneous.

The HRD with multiple enhancement layers has the decoder buffers as many as the number of layers. The capacity of each decoder buffer is determined based on the peak transmission rate of the layer. A set of leaky bucket values for each layer shall be signaled to the decoder.

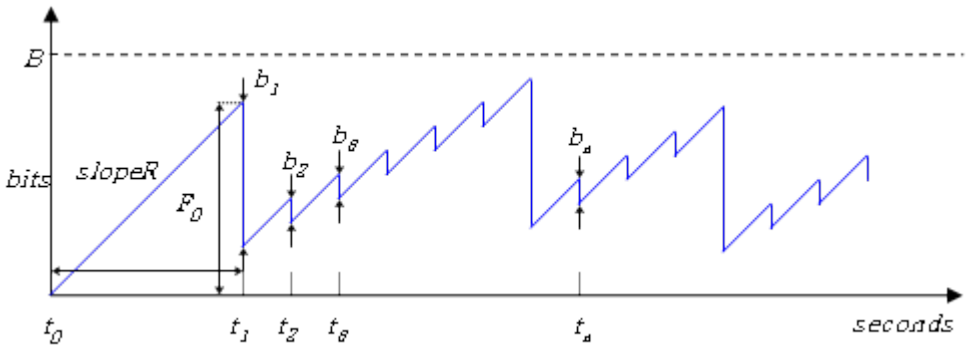

Figure A.2 – Buffer fullness of coded frame buffer at the enhancement layer

Figure A.2 shows the fullness of the residual decoder as a function of time. After removing frame *i* at time *i*, the buffer fullness of the residual coded frame buffer for the enhancement layer shall be described as follows in equation (A-1):

$$\begin{aligned} F_1 &= F_0 - b_1 && i = 1 \\ F_i &= \min(B, F_{i-1} + R_i(t_i - t_{i-1})) - b_i && i \neq 1 \end{aligned} \qquad \textbf{(A-1)}$$

where t_i is the decoding time for frame *i* and b_i is the number of bits for frame *i*. The parameter R_i is the average bit rate (in bps) that enters the buffer during the time interval (t_i, t_{i-1}) and shall be such that $R_i <= R$ for all *i*. Figure A.2 assumes that that $R_i = R$ for all *i*. This buffer model is allowed to fill up, but not to overflow so that the buffer fullness at any time instant shall be less than or equal to *B*. And thus, as a constraint on the bitstream, the decoder buffer shall not overflow.

Decoder buffer underflow could occur if an encoder produces relatively large data for frames. The buffer fullness can be reduced to the point that the bits for the next frame are not available at the nominal decoding time. As a constraint on the bitstream, the decoder buffer shall not underflow. In order to avoid buffer underflow for variable bit rate (VBR) streams, equation (A-2) shall be satisfied:

$$\begin{aligned} F_1 &= F_0 - b_1 && i = 1 \\ F_i &= \min(B, F_{i-1} + R_i(t_i - t_{i-1})) - b_i && i \neq 1 \\ R_i &<= R && all \;\; i \\ F_i &>= 0 && all \;\; i \end{aligned} \qquad \textbf{(A-2)}$$

In order to avoid buffer underflow for constant bit rate (CBR) bitstream, equation (A-3) shall be satisfied

$$\begin{aligned} F_1 &= F_0 - b_1 && i = 1 \\ F_i &= F_{i-1} + R_i(t_i - t_{i-1}) - b_i && i \neq 1 \\ R_i &= R && all \;\; i \\ F_i &>= 0 && all \;\; i \\ F_i + b_i &<= B && all \;\; i \end{aligned} \qquad \textbf{(A-3)}$$

Annex B Start Codes and Emulation Prevention (Normative)

Note: The specification in this annex is similar to, but modified from SMPTE ST 421, Annex E.

This specification defines a sequence of four bytes as the start code which shall consist of a unique three-byte Start Code Prefix (SCP), and a one-byte Start Code Suffix (SCS). The SCP shall be the unique sequence of three bytes (0x000001). The SCS shall be used to identify the type of bitstream segment that follows the start code. For example, the suffix of the start code before a frame/field is different from the suffix of the start code before a slice. Start codes shall be byte-aligned.

An Encapsulation Mechanism (EM) is described to prevent emulation of the start code prefix in the bitstream.

B.1 Extraction of Raw Bitstream Segment from Encapsulated Bitstream Segment

The decoder shall perform the extraction process of a raw bitstream segment from an encapsulated bitstream segment as described below.

Step 1: The start-code suffix shall be used to identify the type of bitstream segment. The bytes that follow the start code suffix shall then be further processed as follows.

Step 2: The decoder shall remove all zero-valued bytes at the end of encapsulated bitstream segment. After this step, if the bitstream segment payload after the start code suffix is not null, the last byte of the bitstream segment shall contain the '1' bit and any byte-aligning '0' bits that are present after the end of the raw bitstream segment.

Step 3: The bytes used for emulation prevention shall be detected and removed according to the following process:

> Whenever a string of two consecutive bytes of value 0x00 are followed by a byte equal to 0x03, the byte equal to 0x03 is understood to be an emulation prevention byte and is discarded. This process is illustrated in Table B.1Table .

Table B.1 – Decoder Removal of Emulation Prevention Data

Pattern to Replace	Replacement Pattern
0x00, 0x00, 0x03, 0x00	0x00, 0x00, 0x00
0x00, 0x00, 0x03, 0x01	0x00, 0x00, 0x01
0x00, 0x00, 0x03, 0x02	0x00, 0x00, 0x02
0x00, 0x00, 0x03, 0x03	0x00, 0x00, 0x03

Step 4: If there are bytes not removed in step 2 that follow the start code suffix in the encapsulated bitstream segment, in the last byte of the bitstream segment data processed in step 3, the last non-zero bit is identified, and that non-zero bit, and all the zero bits that follow, shall be discarded. The result is the raw bitstream segment. If there are no bytes not removed in step 2 that follow the start code suffix in the encapsulated bitstream segment, the raw bitstream segment shall be considered null.

B.2 Start-code Suffixes for Bitstream Segment Types

The start code suffixes for bitstream segment types referenced in this document shall be as defined in Table B.2.

Table B.2 – Start Code Suffixes for Various Bitstream Segment Types

Start-code Suffix	Bitstream Segment Type
0x00	SMPTE Reserved
0x01-0x09	SMPTE Reserved
0x0A	End-of-Sequence
0x0B	Slice
0x0C	Field
0x0D	Frame
0x0E	SMPTE Reserved
0x0F	Sequence Header
0x10-0x1A	SMPTE Reserved
0x1B	Slice Level User Data
0x1C	Field Level User Data
0x1D	Frame Level User Data
0x1E	SMPTE Reserved
0x1F	Sequence Level User Data
0x20-0x7F	SMPTE Reserved
0x80-0xFF	Forbidden

Sequence header bitstream segment type shall be sent to identify bitstream segments which carry sequence header. See Section 7.1 for more details on sequence headers.

Frame bitstream segment type shall be sent to identify bitstream segments which contain the frame header and the frame data. See Section 8.1 and Section 10.1 for more details on frame headers and the first field headers.

Field bitstream segment type shall be sent to identify bitstream segments which contain the second field of a picture that is coded as two separate fields. See Section 10.1 for more details on the second field headers.

Slice bitstream segment type shall be sent to identify bitstream segments which carry the slice data and the slice header. See Section 8.2 and Section 10.2 for more details on slices and slice header.

Sequence, Frame, Field, and Slice Level User data bitstream segment types shall be used to transmit any user-defined data associated with the Sequence, Frame, Field, and Slice respectively. See Annex C for more details.

End-of-sequence bitstream segment type is an optional bitstream segment type which indicates that the current sequence has ended, and no future data shall be transmitted for this sequence. When the transmission of an end-of-sequence is not present, the end of a sequence shall be inferred from the header of the next sequence.

Annex C User Data (Normative)

User Data shall be conveyed as bitstream segments and may be included in the bitstream at the sequence, frame, field, or slice layers. The user data for each of these locations shall be identified by 4 unique start codes (0x1B, 0x1C, 0x1D, and 0x1F) in Annex B . The User Data syntax shall be as defined in Table 257 in SMPTE ST 421, Annex F.

Annex D Profiles and Levels (Normative)

The classification rules for the profile and level in this specification are similar to those in other standards such as SMPTE ST 421 (VC-1) and ITU-T H.264. The lowest layer bitstream is generated by any existing standard encoder and, hence, conforming to the profile and level of the standard for the lowest layer. The bitstream of all the other layers shall be restricted by the profile and level defined in this section. Each layer has its own profile and level independently.

D.1 Profiles

There are six defined profiles: Main, High, High 4:2:2, High 4:4:4, High Intra 4:4:4 and Stereo. Table D.1 lists all the profiles and levels.

Table D.1 – List of profiles and levels

Profile	Levels	Label
Main	L0	MP@L0
	L1	MP@L1
	L2	MP@L2
	L3	MP@L3
	L4	MP@L4
	L5	MP@L5
	L6	MP@L6
	L7	MP@L7
	L8	MP@L8
High	L0	HP@L0
	L1	HP@L1
	L2	HP@L2
	L3	HP@L3
	L4	HP@L4
	L5	HP@L5
	L6	HP@L6
	L7	HP@L7
	L8	HP@L8
	L9	HP@L9
High 4:2:2	L0	HP422@L0
	L1	HP422@L1
	L2	HP422@L2
	L3	HP422@L3
	L4	HP422@L4
	L5	HP422@L5
	L6	HP422@L6
	L7	HP422@L7
	L8	HP422@L8
	L9	HP422@L9
High 4:4:4	L0	HP444@L0
	L1	HP444@L1
	L2	HP444@L2
	L3	HP444@L3
	L4	HP444@L4
	L5	HP444@L5
	L6	HP444@L6
	L7	HP444@L7
	L8	HP444@L8
	L9	HP444@L9

High Intra 4:4:4	L0	HIP444@L0
	L1	HIP444@L1
	L2	HIP444@L2
	L3	HIP444@L3
	L4	HIP444@L4
	L5	HIP444@L5
	L6	HIP444@L6
	L7	HIP444@L7
	L8	HIP444@L8
	L9	HIP444@L9
Stereo	L0	ST@L0
	L1	ST@L1
	L2	ST@L2
	L3	ST@L3
	L4	ST@L4
	L5	ST@L5
	L6	ST@L6
	L7	ST@L7
	L8	ST@L8
	L9	ST@L9

Tbale D.2 indicates the algorithmic features and limits that shall be supported by all decoders conforming to that profile. If the compression feature is marked with “O”, that feature shall be supported by the corresponding profile.

Table D.2 – Codec options in the Main, High, High 4:2:2, High 4:4:4, High Intra 4:4:4 and Stereo profile

Compression Feature	Main	High	High 4:2:2	High 4:4:4	High Intra 4:4:4	Stereo
Intra frame	O	O	O	O	O	O
Inter frame	O	O	O	O	X	O
Intra prediction	O	O	O	O	X	O
4-motion vector	O	O	O	O	X	O
Quarter-pel MC	O	O	O	O	X	O
Interlace	X	O	O	O	O	O
Picture-adaptive field/frame	X	O	O	O	O	X
4:2:0	O	O	O	O	O	O
4:2:2	X	X	O	O	O	O
4:4:4	X	X	X	O	O	O
Lossless coding	X	X	X	X	O	X
Resolution factor	any factor	any factor	any factor	any factor	any factor	1
N-bit depth (N > 8 && N <= 16)	X	O	O	O	O	X
Q-matrix	X	O	O	O	O	X
Multiple ref.	X	O	O	O	X	X
Adaptive coeff. scan	O	O	O	O	O	O
Stereo conversion	X	X	X	X	X	O
Tone-Mapping	X	O	O	O	O	X
Residual Scaling	X	O	O	O	O	X
1-D Filtered Prediction	X	O	O	O	O	O

D.2 Levels

There are several levels for each of the profiles. Each level constrains the video resolution, frame rate, HRD bit rate, HRD buffer requirements, and the motion vector range. These constraints are defined in Tbale D.3Table .

Table D.3 – Limitations of profiles and levels

Profile	Level	MB/s	MB/f	Rmax	Bmax	MV [H]x[V]	Examples
Main	L0	36,000	1,200	5,950	740	[-256, 255.75] x [-128, 127.75]	480x480 (SVCD) 30Hz 480x576 (SVCD) 30Hz 640x480 (VGA) 30Hz
	L1	56,250	1,875	9,600	1,200	[-256, 255.75] x [-128, 127.75]	720x480/576 (SD) 30Hz 800x480 (WVGA) 30Hz 800x600 (SVGA) 30Hz
	L2	115,200	3,840	16,900	2,070	[-512, 511.75] x [-256, 255.75]	1024x768 (XGA) 30Hz 1280x720 (HD) 30Hz 1280x768 (WXGA) 30Hz
	L3	225,000	7,500	33,000	4,000	[-512, 511.75] x [-256, 255.75]	1280x1024 (SXVGA) 30Hz 1400x1050 (SXGA+) 30Hz 1440x1080 (HD) 30Hz 1600x1024 (WSXGA) 30Hz 1600x1200 (GXGA) 30Hz
	L4	244,800	8,160	36,000	4,400	[-1024, 1023.75] x [-512, 511.75]	1280x720 (HD) 60Hz 1920x1080 (HD) 30Hz 1920x1088 (HD) 30Hz
	L5	270,000	9,000	40,000	4,800	[-1024, 1023.75] x [-512, 511.75]	1920x1200 (WUXGA) 30Hz 2048x1088 (2K) 30Hz
	L6	540,000	9,000	80,000	9,700	[-1024, 1023.75] x [-512, 511.75]	1920x1080 (HD) 60Hz 1920x1088 (HD) 60Hz 1920x1200 (WUXGA) 60Hz 2048x1088 (2K) 60Hz
	L7	2,073,600	34,560	153,600	21,300	[-1024, 1023.75] x [-512, 511.75]	3840x2160 (UD) 60Hz 4096x2160 (4K) 60Hz
	L8	7,776,000	129,600	576,000	80,000	[-2048, 2047.75] x [-1024, 1023.75]	7680x4320 (UD) 60Hz

High	L0	11,880	396	600	120	[-128, 127.75] x [-64, 63.75]	320x240 (QVGA) 30Hz 352x288 (CIF) 30Hz
	L1	36,000	1,200	5,950	740	[-256, 255.75] x [-128, 127.75]	480x480 (SVCD) 30Hz 480x576 (SVCD) 30Hz 640x480 (VGA) 30Hz
	L2	56,250	1,875	9,600	1,200	[-256, 255.75] x [-128, 127.75]	720x480/576 (SD) 30Hz 800x480 (WVGA) 30Hz 800x600 (SVGA) 30Hz
	L3	115,200	3,840	16,900	2,070	[-512, 511.75] x [-256, 255.75]	1024x768 (XGA) 30Hz 1280x720 (HD) 30Hz 1280x768 (WXGA) 30Hz
	L4	225,000	7,500	33,000	4,000	[-512, 511.75] x [-256, 255.75]	1280x1024 (SXVGA) 30Hz 1400x1050 (SXGA+) 30Hz 1440x1080 (HD) 30Hz 1600x1024 (WSXGA) 30Hz 1600x1200 (GXGA) 30Hz
	L5	244,800	8,160	36,000	4,400	[-1024, 1023.75] x [-512, 511.75]	1920x1080 (HD) 30Hz 1920x1088 (HD) 30Hz
	L6	270,000	9,000	40,000	4,800	[-1024, 1023.75] x [-512, 511.75]	1920x1200 (WUXGA) 30Hz 2048x1088 (2K) 30Hz
	L7	540,000	9,000	80,000	9,700	[-1024, 1023.75] x [-512, 511.75]	1920x1080 (HD) 60Hz 1920x1088 (HD) 60Hz 1920x1200 (WUXGA) 60Hz 2048x1088 (2K) 60Hz
	L8	2,073,600	34,560	153,600	21,300	[-1024, 1023.75] x [-512, 511.75]	3840x2160 60Hz 4096x2160 60Hz
	L9	7,776,000	129,600	576,000	80,000	[-2048, 2047.75] x [-1024, 1023.75]	7680x4320 60Hz
High 4:2:2	L0	11,880	396	900	180	[-128, 127.75] x [-64, 63.75]	320x240 (QVGA) 30Hz 352x288 (CIF) 30Hz
	L1	36,000	1,200	7400	930	[-256, 255.75] x [-128, 127.75]	480x480 (SVCD) 30Hz 480x576 (SVCD) 30Hz 640x480 (VGA) 30Hz

	L2	56,250	1,875	14,000	1,800	[-256, 255.75] x [-128, 127.75]	720x480/576 (SD) 30Hz 800x480 (WVGA) 30Hz 800x600 (SVGA) 30Hz
	L3	115,200	3,840	22,000	2,600	[-512, 511.75] x [-256, 255.75]	1024x768 (XGA) 30Hz 1280x720 (HD) 30Hz 1280x768 (WXGA) 30Hz
	L4	225,000	7,500	42,000	5,200	[-512, 511.75] x [-256, 255.75]	1280x1024 (SXVGA) 30Hz 1400x1050 (SXGA+) 30Hz 1440x1080 (HD) 30Hz 1600x1024 (WSXGA) 30Hz 1600x1200 (GXGA) 30Hz
	L5	244,800	8,160	45,000	5,500	[-1024, 1023.75] x [-512, 511.75]	1920x1080 (HD) 30Hz 1920x1088 (HD) 30Hz
	L6	270,000	9,000	50,000	6,000	[-1024, 1023.75] x [-512, 511.75]	1920x1200 (WUXGA) 30Hz 2048x1088 (2K) 30Hz
	L7	540,000	9,000	100,000	12,000	[-1024, 1023.75] x [-512, 511.75]	1920x1080 (HD) 60Hz 1920x1088 (HD) 60Hz 1920x1200 (WUXGA) 60Hz 2048x1088 (2K) 60Hz
	L8	2,073,600	34,560	180,000	25,000	[-1024, 1023.75] x [-512, 511.75]	3840x2160 60Hz 4096x2160 60Hz
	L9	7,776,000	129,600	720,000	100,000	[-2048, 2047.75] x [-1024, 1023.75]	7680x4320 60Hz
High 4:4:4	L0	11,880	396	1200	240	[-128, 127.75] x [-64, 63.75]	320x240 (QVGA) 30Hz 352x288 (CIF) 30Hz
	L1	36,000	1,200	11,900	1,480	[-256, 255.75] x [-128, 127.75]	480x480 (SVCD) 30Hz 480x576 (SVCD) 30Hz 640x480 (VGA) 30Hz
	L2	56,250	1,875	19,200	2,400	[-256, 255.75] x [-128, 127.75]	720x480/576 (SD) 30Hz 800x480 (WVGA) 30Hz 800x600 (SVGA) 30Hz
	L3	115,200	3,840	33,800	4,140	[-512, 511.75] x [-256, 255.75]	1024x768 (XGA) 30Hz 1280x720 (HD) 30Hz 1280x768 (WXGA) 30Hz

	L4	225,000	7,500	66,000	8,000	[-512, 511.75] x [-256, 255.75]	1280x1024 (SXVGA) 30Hz 1400x1050 (SXGA+) 30Hz 1440x1080 (HD) 30Hz 1600x1024 (WSXGA) 30Hz 1600x1200 (GXGA) 30Hz
	L5	244,800	8,160	72,000	8,800	[-1024, 1023.75] x [-512, 511.75]	1920x1080 (HD) 30Hz 1920x1088 (HD) 30Hz
	L6	270,000	9,000	80,000	9,600	[-1024, 1023.75] x [-512, 511.75]	1920x1200 (WUXGA) 30Hz 2048x1088 (2K) 30Hz
	L7	540,000	9,000	160,000	19,400	[-1024, 1023.75] x [-512, 511.75]	1920x1080 (HD) 60Hz 1920x1088 (HD) 60Hz 1920x1200 (WUXGA) 60Hz 2048x1088 (2K) 60Hz
	L8	2,073,600	34,560	307,200	42,600	[-1024, 1023.75] x [-512, 511.75]	3840x2160 60Hz 4096x2160 60Hz
	L9	7,776,000	129,600	1,152,000	160,000	[-2048, 2047.75] x [-1024, 1023.75]	7680x4320 60Hz
High Intra 4:4:4	L0	11,880	396	1200	240	N/A	320x240 (QVGA) 30Hz 352x288 (CIF) 30Hz
	L1	36,000	1,200	11,900	1,480	N/A	480x480 (SVCD) 30Hz 480x576 (SVCD) 30Hz 640x480 (VGA) 30Hz
	L2	56,250	1,875	19,200	2,400	N/A	720x480/576 (SD) 30Hz 800x480 (WVGA) 30Hz 800x600 (SVGA) 30Hz
	L3	115,200	3,840	33,800	4,140	N/A	1024x768 (XGA) 30Hz 1280x720 (HD) 30Hz 1280x768 (WXGA) 30Hz
	L4	225,000	7,500	66,000	8,000	N/A	1280x1024 (SXVGA) 30Hz 1400x1050 (SXGA+) 30Hz 1440x1080 (HD) 30Hz 1600x1024 (WSXGA) 30Hz 1600x1200 (GXGA) 30Hz
	L5	244,800	8,160	72,000	8,800	N/A	1920x1080 (HD) 30Hz 1920x1088 (HD) 30Hz

	L6	270,000	9,000	80,000	9,600	N/A	1920x1200 (WUXGA) 30Hz 2048x1088 (2K) 30Hz
	L7	540,000	9,000	160,000	19,400	N/A	1920x1080 (HD) 60Hz 1920x1088 (HD) 60Hz 1920x1200 (WUXGA) 60Hz 2048x1088 (2K) 60Hz
	L8	2,073,600	34,560	307,200	42,600	N/A	3840x2160 60Hz 4096x2160 60Hz
	L9	7,776,000	129,600	1,152,000	160,000	N/A	7680x4320 60Hz
Stereo	L0	11,880	396	1,200	240	[-128, 127.75] x [-64, 63.75]	320x240 (QVGA) 30Hz 352x288 (CIF) 30Hz
	L1	36,000	1,200	11,900	1,480	[-256, 255.75] x [-128, 127.75]	480x480 (SVCD) 30Hz 480x576 (SVCD) 30Hz 640x480 (VGA) 30Hz
	L2	56,250	1,875	19,200	2,400	[-256, 255.75] x [-128, 127.75]	720x480/576 (SD) 30Hz 800x480 (WVGA) 30Hz 800x600 (SVGA) 30Hz
	L3	115,200	3,840	33,800	4,140	[-512, 511.75] x [-256, 255.75]	1024x768 (XGA) 30Hz 1280x720 (HD) 30Hz 1280x768 (WXGA) 30Hz
	L4	225,000	7,500	66,000	8,000	[-512, 511.75] x [-256, 255.75]	1280x1024 (SXVGA) 30Hz 1400x1050 (SXGA+) 30Hz 1440x1080 (HD) 30Hz 1600x1024 (WSXGA) 30Hz 1600x1200 (GXGA) 30Hz
	L5	244,800	8,160	72,000	8,800	[-1024, 1023.75] x [-512, 511.75]	1920x1080 (HD) 30Hz 1920x1088 (HD) 30Hz
	L6	270,000	9,000	80,000	9,600	[-1024, 1023.75] x [-512, 511.75]	1920x1200 (WUXGA) 30Hz 2048x1088 (2K) 30Hz
	L7	540,000	9,000	160,000	19,400	[-1024, 1023.75] x [-512, 511.75]	1920x1080 (HD) 60Hz 1920x1088 (HD) 60Hz 1920x1200 (WUXGA) 60Hz 2048x1088 (2K) 60Hz

	L8	2,073,600	34560	307,200	42,600	[-1024, 1023.75] x [-512, 511.75]	3840x2160 (UD) 60Hz 4096x2160 (4K) 60Hz
	L9	7,776,000	129,600	1,152,000	160,000	[-2048, 2047.75] x [-1024, 1023.75]	7680x4320 (UD) 60Hz

MB/s	Maximum number of macroblocks per second
MB/f	Maximum number of macroblocks within a frame
Rmax	HRD’s maximum peak transmission bit rate in units of 1,000 bits/sec.
Bmax	HRD’s maximum buffer size in units of 16,384 bits
MV	[H]x[V] Motion vector range in full pixel units. [H] = horizontal, [V] = vertical.
Example	Example of maximum video resolution and frame rate.

Annex E Encoding Process for Adaptive Variable Length Code (Informative)

This informative annex describes the encoding process for the adaptive variable length code which matches the decoding process. All functions in Section 9.3 are used for the encoding process except for ***AdaptiveVLC3_decodeSymbol()***, ***AdaptiveVLC5_decodeSymbol()***, ***AdaptiveVLC7_decodeSymbol()***, ***AdaptiveVLC15_decodeSymbol()*** and ***AdaptiveVLC19_decodeSymbol()***. The pseudo code of Figure shows the encoding procedure of the VLC code with an alphabet code of three where the function ***AdaptiveVLC3_encodeSymbol()*** is newly defined for the encoding.

```
// AdaptiveVLC3: structure for adaptive VLC encoding with 3 alphabets
// Adapt_vlc3[19]: 19 structures for adaptive VLC encoding with 3 alphabets
//                 19 syntaxes are encoded with adaptive VLC code with 3 alphabets
// Occurrence[3]: probability models for 3 alphabets for each syntax
// Symbol[3]: storage for three alphabets
// SWAP(): the exchange function of Symbols and Occurrence
//         for encoding syntax with a probability model
//

typedef struct {
   UINT16   Occurrence[3];
   UINT16   Symbol[3];
} AdaptiveVLC3;

AdaptiveVLC3 Adapt_vlc3[19]; // declare 19 structure variables for adaptive VLC coding

SWAP(&A, &B)
{
          Temp = A;
          A = B;
          B = Temp;
}

AdaptiveVLC3_reset(idx) {
       for(i=0; i < 3; i++) {
            Adapt_vlc3[idx].Occurrence[i]=0;
            Adapt_vlc3[idx].Symbol[i]= i;
        }
}

AdaptiveVLC3_update(idx) {
       while(Adapt_vlc3[idx].Occurrence[0] > 256) {
             for(i=0; i < 3;i++)
                  Adapt_vlc3[idx].Occurrence[i] >>= 1;
       }
}

AdaptiveVLC3_encodeSymbol(idx, Symbol) {
    for(i = 0; i < 3; i++) {
        if(Adapt_vlc3[idx].Symbol[i] == Symbol) {
            Value = i;
            break;
       }
    }
    encode_vlc_code(Value); // Using Table 54
     Adapt_vlc3[idx].Occurrence[Value]++; //Update Occurrence
```

```
    // Make sure
    // Adapt_vlc3[idx].Occurrence[Value-1] >= Adapt_vlc3[idx].Occurrence[Value]
    if(Value > 0 && Adapt_vlc3[idx].Occurrence[Value] > Adapt_vlc3[idx].Occurrence[Value-1]) {
        SWAP(Adapt_vlc3[idx].Symbol[Value], Adapt_vlc3[idx].Symbol[Value-1]);
        SWAP(Adapt_vlc3[idx].Occurrence[Value], Adapt_vlc3[idx].Occurrence[Value-1]);
    }
}
```

Figure E.1 – Pseudo-code for adaptive VLC encoding with 3 alphabets

In the function ***AdaptiveVLC3_encodeSymbol()*** of Figure E.1, the function *encode_vlc_code(*Value*)* generates a codeword assigned by Value according to Table 54. A generated codeword shall be inserted into a bitstream.

The pseudo code of Figure E.2 shows the encoding procedure of the VLC code with an alphabet code of five where the function ***AdaptiveVLC5_encodeSymbol()*** is newly defined for the encoding.

```
// AdaptiveVLC5: structure for adaptive VLC encoding with 5 alphabets
// Adapt_vlc5[24]: 24 structures for adaptive VLC encoding with 5 alphabets
//                 24 syntaxes are encoded with adaptive VLC code with 5 alphabets
// Occurrence[5]: probability models for 5 alphabets for each syntax
// Symbol[5]: storage for 5 alphabets
// initSymbol5[5]: the initial symbol values for the case of using 5 alphabets when idx < 22
// Table_Index: the selected table index of 3 tables of Table 55
// SWAP(): the exchange function of Symbols and Occurrence
//          for encoding syntax with a probability model

typedef struct {
   UINT16   Occurrence[5];  //unsigned 16-bit integer variables
   UINT16   Symbol[5];
   UINT16   Table_Index;
} AdaptiveVLC5;

AdaptiveVLC5 Adapt_vlc5[24]; // declare 24 structure variables for adaptive VLC coding

SWAP(&A, &B)
{
        Temp = A;
        A = B;
        B = Temp;
}

AdaptiveVLC5_reset(idx) {
        initSymbol5[5] = {10, 9, 8, 6, 2 };
        Adapt_vlc5[idx].Table_Index = 0;
        if(idx < 22) {
            for(i=0; i < 5; i++) {
                Adapt_vlc5[idx].Occurrence[i]=0;
                Adapt_vlc5[idx].Symbol[i]= initSymbol5[i];
            }
        }
        else if (idx == 33 || idx == 34) {
            for(i=0; i < 5; i++) {
                Adapt_vlc5[idx].Occurrence[i]=0;
                Adapt_vlc5[idx].Symbol[i]= i;
```

```
                }
            }
}

AdaptiveVLC5_update(idx) {
        while(Adapt_vlc5[idx].Occurrence[0] > 256) {
            for(i=0; i < 5;i++)
                Adapt_vlc5[idx].Occurrence[i] >>= 1;
        }
        //Table Selection Logic
        MinCost = Cost = Adapt_vlc5[idx].Occurrence[0]
                    + 2 * Adapt_vlc5[idx].Occurrence[1]
                    + 3 * Adapt_vlc5[idx].Occurrence[2]
                    + 4 * Adapt_vlc5[idx].Occurrence[3]
                    + 4 * Adapt_vlc5[idx].Occurrence[4];
        Adapt_vlc5[idx].Table_Index = 0;
        Cost = Adapt_vlc5[idx].Occurrence[0]
                  +3 * Adapt_vlc5[idx].Occurrence[1]
                  +3 * Adapt_vlc5[idx].Occurrence[2]
                  +3 * Adapt_vlc5[idx].Occurrence[3]
                  +3 * Adapt_vlc5[idx].Occurrence[4];
        if(Cost < MinCost) {
              Adapt_vlc5[idx].Table_Index = 1; MinCost = Cost ;
        }
       Cost = 2 * Adapt_vlc5[idx].Occurrence[0]
                  +2 * Adapt_vlc5[idx].Occurrence[1]
                  +2 * Adapt_vlc5[idx].Occurrence[2]
                  +3 * Adapt_vlc5[idx].Occurrence[3]
                  +3 * Adapt_vlc5[idx].Occurrence[4];
        if(Cost < MinCost) {
              Adapt_vlc5[idx].Table_Index = 2; MinCost = Cost ;
        }
}

AdaptiveVLC5_encodeSymbol(idx, Symbol) {
    for(i = 0: i < 5; i++) {
        if(Adapt_vlc5[idx].Symbol[i] == Symbol) {
            Value = i;
            break;
       }
    }
    encode_vlc_code(Value); // using Table 55
    Adapt_vlc5[idx].Occurrence[Value]++; //Update Occurrence

     //Make sure
    // Adapt_vlc5[idx].Occurrence[Value-1] >= Adapt_vlc5[idx].Occurrence[Value]
     if(Value > 0 && Adapt_vlc5[idx].Occurrence[Value] > Adapt_vlc5[idx].Occurrence[Value-1]) {
          SWAP(Adapt_vlc5[idx].Symbol[Value], Adapt_vlc5[idx].Symbol[Value-1]);
          SWAP(Adapt_vlc5[idx].Symbol[Value], Adapt_vlc5[idx].Symbol[Value-1]);
    }
}
```

Figure E.2 – **Pseudo-code for adaptive VLC encoding with 5 alphabets**

In the function ***AdaptiveVLC5_encodeSymbol()*** of Figure E,2, the function *encode_vlc_code(*Value*)* generates a codeword assigned by Value according to Table 55. A generated codeword shall be inserted into a bitstream.

The pseudo code of Figure E.3 shows the encoding procedure of the VLC code with an alphabet code of seven where the function ***AdaptiveVLC7_encodeSymbol()*** is newly defined for the encoding.

```
// AdaptiveVLC7: structure for adaptive VLC encoding with 7 alphabets
// Adapt_vlc7[4]: 4 structures for adaptive VLC encoding with 7 alphabets
//                    4 syntaxes are encoded with adaptive VLC code with 7 alphabets
// Occurrence[7]: probability models for 7 alphabets for each syntax
// Symbol[7]: storage for 7 alphabets
// initSymbol7[7]: the initial symbol values for the case of using 7 alphabets
// BITS_Adapt_vlc7[3][7]: the number of bits for the selected table of Table 56 and symbol
// Table_Index: the selected table index of 3 tables of Table 56
// SWAP(): the exchange function of Symbols and Occurrence
//              for encoding syntax with a probability model
//

typedef struct {
   UINT16   Occurrence[7];
   UINT16   Symbol[7];
   UINT16   Table_Index;
} AdaptiveVLC7;

AdaptiveVLC7 Adapt_vlc7[4]; // declare 4 structure variables for adaptive VLC coding

SWAP(&A, &B)
{
        Temp = A;
        A = B;
        B = Temp;
}

AdaptiveVLC7_reset(idx) {
        initSymbol7[7] = { 1, 4, 5, 16, 17, 20, 21 };
        Adapt_vlc7[idx].Table_Index = 0;
        for(i=0; i < 7; i++) {
           Adapt_vlc7[idx].Occurrence[i]=0;
           Adapt_vlc7[idx].Symbol[i]= initSymbol7[i];
        }
}

AdaptiveVLC7_update(idx) {
      BITS_Adapt_vlc7[3][7] = {       { 2, 2, 3, 3, 3, 4, 4 },
                                      { 1, 3, 3, 4, 4, 4, 4 },
                                      { 2, 2, 2, 3, 4, 5, 5 }     };
      while(Adapt_vlc7[idx].Occurrence[0] > 256) {
          for(i=0; i < 7;i++)
              Adapt_vlc7[idx].Occurrence[i] >>= 1;
      }
      //Table Selection Logic
      MinCost = 0xffffffff;
      for(tmpTable = 0; tmpTable < 3; tmpTable++) {
                Cost = 0;
                for(i = 0; i < 7; i++) {
                        Cost += Adapt_vlc7[idx].Occurrence[i] * BITS_ Adapt_vlc7[tmpTable][i];
                }
                if(Cost < MinCost) {
```

```
                        MinCost = Cost;
                        Adapt_vlc7[idx].Table_Index = tmpTable;
                    }
                }
            }

AdaptiveVLC7_encodeSymbol(idx, Symbol) {
    for(i = 0: i < 7; i++) {
        if(Adapt_vlc7[idx].Symbol[i] == Symbol) {
            Value = i;
            break;
        }
    }
    encode_vlc_code(Value); // using Table 56
    Adapt_vlc7[idx].Occurrence[Value]++; //Update Occurrence

    //Make sure
    // Adapt_vlc7[idx].Occurrence[Value-1] >= Adapt_vlc7[idx].Occurrence[Value]
    if(Value > 0 && Adapt_vlc7[idx].Occurrence[Value] > Adapt_vlc7[idx].Occurrence[Value-1]) {
        SWAP(Adapt_vlc7[idx].Symbol[Value], Adapt_vlc7[idx].Symbol[Value-1]);
        SWAP(Adapt_vlc7[idx].Occurrence[Value], Adapt_vlc7[idx].Occurrence[Value-1]);
    }

}
```

Figure E.3 – Pseudo-code for adaptive VLC encoding with 7 alphabets

In the function ***AdaptiveVLC7_encodeSymbol()*** of Figure E.3Figure , the function *encode_vlc_code(*Value*)* generates a codeword assigned by Value according to Table 56. A generated codeword shall be inserted into a bitstream.

The pseudo code of Figure Figure E.4 shows the encoding procedure of the VLC code with an alphabet code of fifteen where the function ***AdaptiveVLC15_encodeSymbol()*** is newly defined for the encoding.

```
// AdaptiveVLC15: structure for adaptive VLC encoding with 15 alphabets
// Adapt_vlc15[5]: 5 structures for adaptive VLC encoding with 15 alphabets
//                 5 syntaxes are encoded with adaptive VLC code with 15 alphabets
// Occurrence[15]: probability models for 15 alphabets for each syntax
// Symbol[15]: storage for 15 alphabets
// initSymbol15[15]: the initial symbol values for the case of using 15 alphabets
// BITS_Adapt_vlc15[3][15]: the number of bits for the selected table of Table 57 and symbol
// Table_Index: the selected table index of 3 tables of Table 57
// SWAP(): the exchange function of Symbols and Occurrence
//         for encoding syntax with a probability model
//

typedef struct {
   UINT16   Occurrence[15];
   UINT16   Symbol[15];
   UINT16   Table_Index;
} AdaptiveVLC15;

AdaptiveVLC15 Adapt_vlc15[5]; // declare 5 structure variables for adaptive VLC coding

SWAP(&A, &B)
{
```

```
        Temp = A;
        A = B;
        B = Temp;
}

AdaptiveVLC15_reset(idx) {
        initSymbol15[15] = { 1, 4, 5, 16, 17, 20, 21, 64, 65, 68, 69, 80, 81, 84, 85 }
        Adapt_vlc15[idx].Table_Index = 0;
        for(i=0; i < 15; i++) {
          Adapt_vlc15[idx].Occurrence[i]=0;
          Adapt_vlc15[idx].Symbol[i]= initSymbol15[i];
        }
}

AdaptiveVLC15_update(idx) {
      BITS_Adapt_vlc15[3][15] = {     { 2, 3, 3, 3, 4, 4, 5, 5, 5, 5, 5, 5, 6, 6 },
                                      { 2, 2, 3, 3, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7 },
                                      { 1, 3, 3, 4, 4, 5, 6, 6, 6, 6, 7, 7, 7, 8, 8 }          };
      while(Adapt_vlc15[idx].Occurrence[0] > 256) {
          for(i=0; i < 15;i++)
              Adapt_vlc15[idx].Occurrence[i] >>= 1;
      }
      //Table Selection Logic
      MinCost = 0xffffffff;
      for(tmpTable = 0; tmpTable < 3; tmpTable++) {
              Cost = 0;
              for(i = 0; i < 15; i++) {
                      Cost += Adapt_vlc15[idx].Occurrence[i] * BITS_ Adapt_vlc15[tmpTable][i];
              }
              if(Cost < MinCost) {
                      MinCost = Cost;
                      Adapt_vlc15[idx].Table_Index = tmpTable;
              }
      }
}

AdaptiveVLC15_encodeSymbol(idx, Symbol) {
    for(i = 0: i < 15; i++) {
        if(Adapt_vlc15[idx].Symbol[i] == Symbol) {
            Value = i;
            break;
       }
    }
    encode_vlc_code(Value); // using Table 57
    Adapt_vlc15[idx].Occurrence[Value]]++; //Update Occurrence

     //Make sure
    // Adapt_vlc15[idx].Occurrence[Value-1] >= Adapt_vlc15[idx].Occurrence[Value]
     if(Value > 0 && Adapt_vlc15[idx].Occurrence[Value] > Adapt_vlc15[idx].Occurrence[Value-1]) {
          SWAP(Adapt_vlc15[idx].Symbol[Value], Adapt_vlc15[idx].Symbol[Value-1]);
          SWAP(Adapt_vlc15[idx].Occurrence[Value], Adapt_vlc15[idx].Occurrence[Value-1]);
     }

}
```

Figure E.4 – Pseudo-code for adaptive VLC encoding with 15 alphabets

In the function ***AdaptiveVLC15_encodeSymbol()*** of Figure E.4, the function *encode_vlc_code(*Value*)* generates a codeword assigned by Value according to Table 57. A generated codeword shall be inserted into a bitstream.

The pseudo code of Figure E.5 shows the encoding procedure of the VLC code with an alphabet code of nineteen where the function ***AdaptiveVLC19_encodeSymbol()*** is newly defined for the encoding.

```
// AdaptiveVLC19: structure for adaptive VLC encoding with 19 alphabets
// Adapt_vlc19[4]: 4 structures for adaptive VLC encoding with 19 alphabets
//                 4 syntaxes are encoded with adaptive VLC code with 19 alphabets
// Occurrence[19]: probability models for 19 alphabets for each syntax
// Symbol[19]: storage for 19 alphabets
// initSymbol19[19]: the initial symbol value for the case of usning the 19 alphabets
// BITS_Adapt_vlc19[2][19]: the number of bits for the selected table of Table 58 and symbol
// Table_Index: the selected table index of 2 tables of Table 58
// SWAP(): the exchange function of Symbols and Occurrence
//          for encoding syntax with a probability model
//

typedef struct {
   UINT16   Occurrence[19];
   UINT16   Symbol[19];
   UINT16   Table_Index;
} AdaptiveVLC19;

AdaptiveVLC19 Adapt_vlc19[4]; // declare 4 structure variables for adaptive VLC coding

SWAP(&A, &B)
{
        Temp = A;
        A = B;
        B = Temp;
}

AdaptiveVLC19_reset(idx) {
        initSymbol19[19] = { 2, 6, 8, 9, 10, 18, 22, 24, 25, 26, 32, 33, 34, 36, 37, 38, 40, 41, 42 };
        Adapt_vlc19[idx].Table_Index = 0;
        for(i=0; i < 19; i++) {
          Adapt_vlc19[idx].Occurrence[i]=0;
          Adapt_vlc19[idx].Symbol[i]= initSymbol19[i];
        }
}

AdaptiveVLC19_update(idx) {
      BITS_Adapt_vlc19[2][19] = {     { 2, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 6, 6, 7, 7, 7, 7 },
                                      { 3, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5, 5, 6, 6, 7, 7, 7, 7 }      };
      while(Adapt_vlc19[idx].Occurrence[0] > 256) {
          for(i=0; i < 19;i++)
              Adapt_vlc19[idx].Occurrence[i] >>= 1;
      }
      //Table Selection Logic
      MinCost = 0xffffffff;
      for(tmpTable = 0; tmpTable < 2; tmpTable++) {
                Cost = 0;
                for(i = 0; i < 19; i++) {
```

```
                        Cost += Adapt_vlc19[idx].Occurrence[i] * BITS_ Adapt_vlc19[tmpTable][i];
                }
                if(Cost < MinCost) {
                        MinCost = Cost;
                        Adapt_vlc19[idx].Table_Index = tmpTable;
                }
        }
}

AdaptiveVLC19_encodeSymbol(idx, Symbol) {
    for(i = 0: i < 19; i++) {
        if(Adapt_vlc19[idx].Symbol[i] == Symbol) {
            Value = i;
            break;
       }
    }
    encode_vlc_code(Value); // using Table 58
    Adapt_vlc19[idx].Occurrence[Value]++; //Update Occurrence

     //Make sure
    // Adapt_vlc19[idx].Occurrence[Value-1] >= Adapt_vlc19[idx].Occurrence[Value]
     if(Value > 0 && Adapt_vlc19[idx].Occurrence[Value] > Adapt_vlc19[idx].Occurrence[Value-1]) {
          SWAP(Adapt_vlc19[idx].Symbol[Value], Adapt_vlc19[idx].Symbol[Value-1]);
          SWAP(Adapt_vlc19[idx].Occurrence[Value], Adapt_vlc19[idx].Occurrence[Value-1]);
     }

}
```

Figure E.5 – Pseudo-code for adaptive VLC encoding with 19 alphabets

In the function ***AdaptiveVLC19_encodeSymbol()*** of Figure E.5, the function *encode_vlc_code(*Value*)* generates a codeword assigned by Value according to Table 58. A generated codeword shall be inserted into a bitstream.

For the initialization and adaptation of the adaptive variable length coding, ***resetModel()*** of Figure 33 and ***updateModel()*** of Figure 34 are performed at the same situation as the decoding process in the Section 9.3. After SKIP_RUN is encoded, the function ***AdaptiveVLC5_update(34)*** is performed where the function argument 34 is the probability model index for SKIP_RUN.

The pseudo-code of Figure Figure E.6 shows how these syntaxes are encoded. In this pseudo-code, the function ***AdaptiveVLC3_encodeSymbol()***, ***AdaptiveVLC5_encodeSymbol()***, ***AdaptiveVLC7_encodeSymbol()***, ***AdaptiveVLC15_encodeSymbol()*** and ***AdaptiveVLC19_encodeSymbol()*** are defined in Figure E.1, Figure E.2, Figure E.3, Figure E.4 and Figure E.5, respectively. In Figure E.6, the function ***put_bits(***Symbol, n***)*** is to write Symbol in the bitstream with n bits.

```
//type_symbol represents which alphabet code is selected
//              among 3-alphabet, 5-alphabet, 7-alphabet, 15-alphabet, or 19-alphabet when idx < 31.
//              0: No codeword (when idx < 31)
//              1: 3-alphabet code, 7-alphabet code or 15-alphabet code
//              2: 5-alphabet code or 19-alphabet code
//idx represents one among the model indices from Table 59 to Table 63.
//value1, value2, value3 and value4 are to be encoded jointly or individually.
//One Symbol values (value1) is encoded when idx == 31, idx == 32 or idx == 34
//Two symbol values (value1 and value2) are encoded when 3-alphabet or 5-alphabet code is used
//Three symbol values (value1, value2 and value3) are encoded
//                                              when 7-alphabet or 19-alphabet code is used
```

```
//Four symbol values (value1, value2 value3 and value4) are encoded when 15-alphabet code.

encodeAdaptiveVLC(idx, value1, value2, value3, value4) {
  if(idx < 5) {
       type_symbol = (value1 == 2 || value2 == 2)? 2:(!!(value1 + value2));
       if(type_symbol == 0) {
            return;
       }
      else if(type_symbol == 1) {
           Symbol = value1 - 1 + (value2 << 1);
           AdaptiveVLC3_encodeSymbol(idx, Symbol);
     }
     else if(type_symbol == 2) {
           Symbol = value1 + (value2 << 2);
           AdaptiveVLC5_encodeSymbol(idx, Symbol);
  }
   if(idx < 22) {
       type_symbol = (value1 == 2 || value2 == 2)? 2:(!!(value1 + value2));
       if(type_symbol == 0) {
            return;
       }
      else if(type_symbol == 1) {
           Symbol = (value1 << 1) + value2 - 1;
           AdaptiveVLC3_encodeSymbol(idx, Symbol);
     }
     else if(type_symbol == 2) {
           if(idx == 7 || idx == 10 || idx == 13 || idx == 16 || idx == 19) {
              // These cases use the syntaxes  L2[i]_CST and L1[i]_CST instead of L2AC16[i]_CST
              // L2[i]_CST and L1[i]_CST use 4 symbol values for encoding (See section 8.4.6 ~ section 8.4.9)
              // Therefore, this function may be recalled with idx corresponding to L2[i]_CST and L1[i]_CST.
                  return;
           }
           Symbol = (value1 << 2) + value2;
           AdaptiveVLC5_encodeSymbol(idx, Symbol);
     }
   }
   else if(idx < 26) {
          type_symbol = (value1 == 2 || value2 == 2 || value3 == 2)? 2:(!!(value1 + value2 + value3));
          if(type_symbol == 0)  {
                return;
          }
          Symbol = (value1 << 4) + (value2 << 2) + value3;
          if(type_symbol == 1) {
                AdaptiveVLC7_encodeSymbol(idx, Symbol);
          }
         else if(type_symbol == 2) {
               AdaptiveVLC19_encodeSymbol(idx, Symbol);
         }
  }
  else if(idx < 31){
          if(value1 == 2 || value2 == 2 || value3 == 2 || value4 == 2) {
                  return;
          }
          else {
                  type_symbol = !!(value1 + value2 + value3 + value4);
          }
```

```
            if(type_symbol == 0) {
                    return;
            }
            else if(type_symbol == 1) {
                    Symbol = (value1 << 6) + (value2 << 4) + (value3 << 2) + value4;
                    AdaptiveVLC15_enecodeSymbol(idx, Symbol);
            }
    }
    else {
            if(idx == 31 || idx == 32) { // IPRED_MODE or INTRA_SIG
                AdaptiveVLC3_encodeSymbol(idx, value1);
            }
            else if(idx == 34) { // SKIP_RUN (See Table 17)
                // value1 = # of skipped macroblocks
                AdaptiveVLC5_encodeSymbol(idx, value1);
            }
            else if(idx == 33) { // MB_SIG_MODE
                //value1: MB-level CSP (See Table 18)
                //value2: MB Coding Mode (See Table 18)
                if(value1 != 0 && value2 != '4-MV') {
                        Symbol = (value1 – 1) << 1 + ((value2 == '1-MV')? 1: 0);
                }
                else Symbol = 4;
                AdaptiveVLC5_encodeSymbol(idx, Symbol);
                if(Symbol == 4) { // MB_SIG_4 (See Table 19)
                        if(value2 == '4-MV') {
                                Symbol = value1;
                                put_bits(Symbol, 2);
                        }
                        else {
                                Symbol = 3; //value1 == 0
                                put_bits(Symbol, 2);
                                if(value2 == 'INTRA') //MB_MODE_1 (See section 8.3.4)
                                        put_bits(0, 1); //MB_MODE_1 == 0
                                else
                                        put_bits(1,1); //MB_MODE_1 == 1
                        }
                }
            }
    }
}
```

Figure E.6 – Pseudo-code for encoding CST or CSP syntaxes

Annex F Bitstream Construction Constraints (Normative)

This annex specifies the order constraints of encapsulated bitstream segments as defined in Annex B . Each encapsulated bitstream segment contains one start code (see Annex B) followed immediately by one bitstream segment such as sequence header (as defined in Section 7.1), frame picture header (as defined in Section 8.1 or Section 10.1), field picture header (as defined in Section 10.1), slice header (as defined in Section 8.2 or Section 10.2) and user data (see Annex C).

The encapsulated bitstream segments shall be constructed as follows:

- SEQ_HDR_n: the n-th layer sequence header which follows the startcode (0x0000010F) and consists of a sequence header.
- FRAM_DAT_n: the n-th layer frame picture data which follows the startcode (0x0000010D) and consists of a frame header and frame data
- FLD1_DAT_n: the n-th layer first field picture data which follows startcode (0x0000010C) and consists of a field header and field data.
- FLD2_DAT_n: the n-th layer second field picture data which follows startcode (0x0000010C) and consists of a field header and field data.
- SLC_DAT_n: the n-th layer slice data which follows startcode (0x0000010B) and consists of a slice header and slice data.
- SEQ_USER_DATA_n: the n-th layer sequence-level user data which follows startcode (0x0000011F).
- FRAME_USER_DATA: frame-level user data which follows startcode (0x0000011D).
- FIELD_USER_DATA: field-level user data which follows startcode (0x0000011C).
- SLICE_USER_DATA:slice-level user data which follows startcode (0x0000011B).

A conformant bitstream shall be one with any combination of the above encapsulated bitstream segments in any order which are constrained as described below.

This document supports multiple enhancement layers. In a bitstream including multiple enhancement layers, all sequence headers from all enhancement layers shall be located consecutively in lower-to-higher layer order. In other word, there shall be no any other header information between two successively located sequence headers. For example, the sequence header of the first enhancement layer shall be followed by the sequence header of the second enhancement layer. At the start of a bitstream or random access points, those sequence headers shall appear. After sequence header data, frame or field data shall appear in an enhancement bitstream. If a frame or field picture is rolled as an entry point, LAYER_ENTRY_FLAG shall be set to 1 (see Section 8.1.4 or Section 10.1.4) and the sequence headers shall be located before this picture. The coding type of the first frame or field picture after sequence headers is the intra-coded picture.

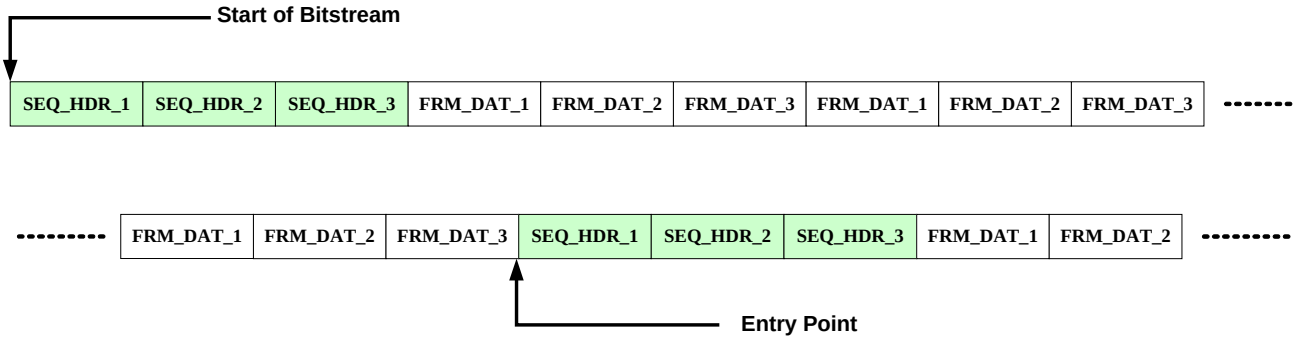

Figure F.1 – Sequence header location in a multi-layer bistream

A set of sequence headers may be regular or irregular intervals throughout the bitstream. Sequence headers shall be located in the bitstream before the entry point picture. Figure F.1 shows an example of a bitstream configuration when the number of the enhancement layers is 3 and all pictures are frame-coded picture. In Figure F.1Figure , SEQ_HDR_1, SEQ_HDR_2 and SEQ_HDR_3 are located consecutively at the start of the bitstream or those sequence headers shall be inserted before every entry points.

The decoder shall detect the number of the enhamcement layers by counting the consecutively-located sequence headers

The bitstream in Annex F shall be considerd as the input of the decoder. When the multi-layer bitstream is transmitted through the network and each enhancement layer is transmitted separately, these separated layers shall be merged into a single bitstream before decoding. For example, when this multi-layer bitstream is transmitted using the MPEG-2 Transport Stream, each enhancement layer is transmitted separately and the separation is performed by the packet identifier (PID). Figure F.2Figure shows an example of the bitstream separation in the transmission using MPEG-2 Transport Stream. These separated layers shall be merged into a single bitstream before decoding.

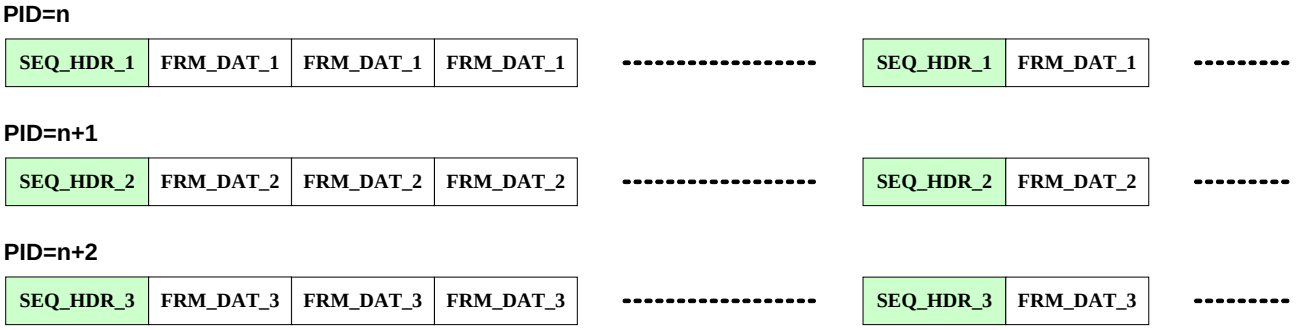

Figure F.2 – Transmission using MPEG-2 Transport Stream

When present, sequence user data of each layer shall be located in the bitstream after the sequence header of each layer and immediately before the next encapsulated bitstream segment of each layer. Sequence user data (SEQ_USER_DATA_n) shall be applicable to the entire sequence of each layer, that is until an end-of-sequence bitstream segment or another sequence header of each layer is encountered in the bitstream.

One or more frame or field picture data shall be ordered in the increasing number of LAYER_ID within one reconstructed picture when there is no any other slice from other layers between successive two slices within a picture data. A particular frame or field picture shall refer to the sequence header using the syntax element of LAYER_ID (see Section 8.1.1 or Section 10.1.1). A particular frame or field picture data shall follow the lower layer frame or field picture which is referred to by the particular frame or field picture using the syntax element of REF_LAYER_ID (see Section 7.1.2) when there is no any other slice from other layers between successive two slices within a picture in a bitstream.

When present, frame or field user data (FRAME_USER_DATA or FIELD_USER_DATA) shall be located in the bitstream after the frame or field picture header and immediately before the next encapsulated bitstream segment. Frame or field user data (FRAME_USER_DATA or FIELD_USER_DATA) shall be applicable to the frame or field pictures until another frame picture data, field picture data, or sequence header is encountered in the bitstream.

One or more slice data shall be located in the bitstream after the frame or field picture headers which are referred to by the slice data using the syntax element of LAYER_ID_SLICE (see Section 8.2.1). One or more slice data shall be ordered in the increasing number of the first macroblock locations of slices in the raster scan within the picture.

In the multi-layer bitstream configuration, the reconstruction of the current slice requires all data from lower layers. It is not necessary that all slices of the lower layer pictures referenced by the current slice are located before the current slice in a bitstream configuration when slice data from different layers are interleaved. If the current slice can be reconstructed using some slices of lower layer pictures, those slices shall be located before the current slice data and other slices data may be located after current slice. Figure F.3 shows a possible slice location when a bitstream has three layers and one picture consists of 3 slices. In Figure F.3, although the third “SLC_DAT_1” belongs to the first layer referred by the second layer picture, the second “SLC_DATA_2” can be located before the third “SLC_DAT_1” when the second “SLC_DAT_2” does not refer to the third “SLC_DAT_1”,

Figure F.3 – An example of slice arrangements

When QUALITY_REFINEMENT_FLAG == 1, the current slice whose QUALITY_LAYER_ID is zero requires at least slices of the quality base layer in the referenced picture whose LAYER_ID is the same as the current slice. In other words, the current slice whose QUALITY_LAYER_ID is zero uses lower layer slices whose LAYER_ID_SLICE is the same as REF_LAYER_ID and slices of the reference picture of the current layer where the quality base layer or the quality enhancement layer slices are referred by the current slice to reconstruct the current pictures. If more quality enhancement layer slices are used for the reconstruction, the better quality of the current picture is generated.

When QUALITY_REFINEMENT_FLAG == 1, the current slice whose QUALITY_LAYER_ID is not zero, the current slice requires slices whose QUALITY_LAYER_ID is less than QUALITY_LAYER_ID of the current slice. It is not necessary that all slices of the lower quality layer pictures referenced by the current slice are located before the current slice in a bitstream configuration when slice data from different quality layers are interleaved. If the current slice can be reconstructed using some slices of lower quality layers, those slices shall be located before the current slice data and other slices data may be located after current slice. Other slice data whose QUALITY_LAYER_ID is less than the current QUALITY_LAYER_ID shall be referred by other slices of the current quality layer.

If it is allowed to interleave slices among layers, the co-located slices of each layer shall have the same structure.

When present, slice user data (SLICE_USER_DATA) shall be located in the bitstream after the slice data and immediately before the next encapsulated bitstream segment. Slice user data (SLICE_USER_DATA) shall be applicable to the slice until another slice header, a frame picture header, a field picture header or sequence header is encountered in the bitstream.

Immediate (one after another of the same kind) duplication of sequence header, frame picture, field picture or slice start code shall not be allowed. User data start code and user bytes may be duplicated an arbitrary amount of time and at any level. Use of sequence, frame, field and slice start codes is optional.

An end-of-sequence start code (value 0x0000010A) may be inserted to indicate that the current sequence has ended and no further data shall be transmitted for this sequence.

Annex G Bibliography (Informative)

IEC 61966-2-4:2006, Multimedia Systems and Equipment — Colour Measurement and Management — Part 2-4: Colour Management — Extended-Gamut YCC Colour Space for Video Applications — xvYCC

ISO/IEC 13818-1:2000, Information Technology — Generic Coding of Moving Pictures and Associated Audio Information: Systems" (2nd Edition)

ISO/IEC 14496-1:2004, Information Technology — Coding of Audio-Visual Objects — Part 1: Systems

ISO/IEC 14496-2:2004, Information Technology — Coding of Audio-Visual Objects — Part 2: Visual

Recommendation ITU-T H.262, ISO/IEC 13818-2:2000, Information Technology — Generic Coding of Moving Pictures and Associated Audio Information: Video