ISO/IEC JTC 1/SC 29/WG 11

Coding of moving pictures and audio

Document type:	**Approved WG 11 document**
Title:	**Draft Text of ISO/IEC DIS 23094-2, Low Complexity Enhancement Video Coding**
Status:	**Approved**
Date of document:	**2020-03-03**
Source:	**Video**
Expected action:	**DIS Ballot**
No. of pages:	**112**
Email of convenor:	**leonardo@chiariglione.org**
Committee URL:	**mpeg.chiariglione.org**

INTERNATIONAL ORGANISATION FOR STANDARDISATION

ORGANISATION INTERNATIONALE DE NORMALISATION

ISO/IEC JTC 1/SC 29/WG 11

CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC 1/SC 29/WG 11 N18986

Brussels, BE – January 2020

Source:	**Video**
Status	**Draft International Standard**
Title:	**Draft Text of ISO/IEC DIS 23094-2, Low Complexity Enhancement Video Coding**
Editors:	**Brahim Allan (BT), Tarek Amara (Amazon Twitch), Stefano Battista (UNIVPM) Lorenzo Ciccarelli (V-Nova), Simone Ferrara (V-Nova), Walt Husak (Dolby), Guido Meardi (V-Nova), Alan Stein (Interdigital), Yan Ye (Alibaba)**

Draft International Standard of Low Complexity Enhancement Video Coding

Abstract

This document provides Draft International Standard of ISO/IEC DIS 23094-2 Low Complexity Enhancement Video Coding.

ISO/IEC DIS 23094-2

ISO/IEC/EC 29WG 11

Secretariat: JISC

Information technology – General Video Coding – Part 2: Low Complexity Enhancement Video Coding

Contents

Tables

Figures

Information technology – General Video Coding – Part 2: Low Complexity Enhancement Video Coding

1 Scope

This International Standard specifies ISO/IEC 23094-2 Low Complexity Enhancement Video Coding.

2 Normative references

2.1 General

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

2.2 International standards

None.

2.3 Additional references

None.

3 Definitions

For the purposes of this International Standard, the following definitions apply.
access unit
set of *NAL units* that are associated with each other according to a specified classification rule, are consecutive in *decoding order,* and contain exactly one *coded picture*

3.2
base layer
layer within the *bitstream* pertaining to a *coded base picture*

3.3
bitstream
sequence of bits, in the form of a *NAL unit stream* or a *byte stream*, that forms the representation of *coded pictures* and associated data forming one or more coded video sequences *(CVSs)*

3.4
block
MxN (M-column by N-row) array of samples, or an MxN array of *transform coefficients*

3.5
byte
sequence of 8 bits, within which, when written or read as a sequence of bit values, the left-most and right-most bits represent the most and least significant bits, respectively

3.6
byte-aligned
position in a *bitstream* in which the position is an integer multiple of 8 bits from the position of the first bit in the *bitstream*

Note 1 to entry: A bit or *byte* or *syntax element* is said to be byte-aligned when the position at which it appears in a *bitstream* is byte-aligned

3.7
byte stream
encapsulation of a *NAL unit stream* containing *start code prefixes* and *NAL units* as specified in Annex B.

3.8
can
term used to refer to behaviour that is allowed, but not necessarily required

3.9
chroma
adjective, represented by the symbols Cb and Cr, specifying that a sample array or single sample is representing one of the two colour difference signals related to the primary colours

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

3.10
chunk
entropy encoded portion of data containing the quantized *transform coefficient* belonging to a coefficient group

3.11
coded picture
coded representation of a *picture* containing all *CUs* of the *picture*

3.12
coded base picture
coded representation of a *picture* encoded using a separate *encoding process*

3.13
coded representation
data element as represented in its coded form

3.14
coding block
MxN *block* of samples for some values of M and N

3.15
coding unit
CU
coding block of *luma* samples or a *coding block* of *chroma* samples of a *picture* that has three sample arrays, or a *coding block* of samples of a monochrome *picture* or a *picture* that is coded using three separate colour planes and *syntax structures* used to code the samples

3.16
coefficient group
CG
syntactical structure containing encoded data related to a specific set of *transform coefficient*

3.17
component
array or single sample from one of the three arrays (*luma* and two *chroma*) that compose a *picture* in 4:2:0, 4:2:2, or 4:4:4 colour format or the array or a single sample of the array that compose a *picture* in monochrome format

3.18
data block
syntax structure containing *bytes* corresponding to a type of data

3.19
decoded base picture
decoded picture derived by decoding a *coded base picture*

3.20
decoded picture
picture derived by decoding a *coded picture*., and which is either a decoded *frame*, or a decoded *field*. A decoded *field* is either a decoded top field or a decoded bottom field

3.21
decoded picture buffer
DPB
buffer holding *decoded pictures* for reference or output reordering

3.22
decoder
embodiment of a *decoding process*

3.23
decoding order
order in which *syntax elements* are processed by the *decoding process*

3.24
decoding process
process specified in this document that reads a *bitstream* and derives *decoded pictures* from it

3.25
emulation prevention byte
byte equal to 0x03 that may be present within a *NAL unit*, the presence of which ensures that no sequence of consecutive *byte-aligned bytes* in the *NAL unit* contains a *start code prefix*.

3.26
encoder
embodiment of an *encoding process*

3.27
encoding process
process, not specified in this document, that produces a *bitstream* conforming to this document

3.28
enhancement layer
layer within the *bitstream* pertaining to the *residual planes*

3.29
enhancement sub-layer
layer of the *enhancement layer*

3.30
field
assembly of alternate rows of a *frame*

3.31
frame
an array of *luma* samples in monochrome format, or an array of *luma* samples and two corresponding arrays of *chroma* samples in 4:2:0, 4:2:2, and 4:4:4 colour format, and which consists of two *fields*, a top field and a bottom field

3.32
informative
term used to refer to content provided in this document that does not establish any mandatory requirements for conformance to this document and thus is not considered an integral part of this document

3.33
instantaneous decoding refresh picture
IDR picture
picture for which an *NAL unit* contains a global configuration data block as in subclause 8.2

3.34
inverse transform
part of the *decoding process* by which a set of *transform coefficients* are converted into *residuals*

3.35
layer
one of a set of syntactical structures in a non-branching hierarchical relationship

3.36
luma
adjective, represented by the symbol or subscript Y or L, specifying that a sample array or single sample represents the monochrome signal related to the primary colours

Note 1 to entry: The term *luma* is used rather than the term luminance in order to avoid the implication of the use of linear light transfer characteristics that is often associated with the term luminance. The symbol L is sometimes used instead of the symbol Y to avoid confusion with the symbol y as used for vertical location.

3.37
may
term that is used to refer to behaviour that is allowed, but not necessarily required

Note 1 to entry: In some places where the optional nature of the described behaviour is intended to be emphasized, the phrase "may or may not" is used to provide emphasis.

3.38
must
term that is used in expressing an observation about a requirement or an implication of a requirement that is specified elsewhere in this document (used exclusively in an *informative* context)

3.39
network abstraction layer unit
NAL unit
syntax structure containing an indication of the type of data to follow and *bytes* containing that data in the form of an *RBSP* interspersed as necessary with *emulation prevention bytes*

3.40
network abstraction layer unit stream
NAL unit stream
sequence of *NAL units*

3.41
note
term that is used to prefix *informative* remarks (used exclusively in an *informative* context)

3.42
output order
order in which the *decoded pictures* are output from the *decoded picture buffer* (for the *decoded pictures* that are to be output from the *decoded picture buffer*)

3.43
partitioning
The division of a set into subsets such that each element of the set is in exactly one of the subsets

3.44
plane
collection of data related to plane Y(*luma*) or C(*chroma*)

3.45
picture
collective term for a field or a frame

3.46
profile
specified subset of the syntax of this document

3.47
random access
act of starting the decoding process for a *bitstream* at a point other than the beginning of the stream

3.48
raw byte sequence payload
RBSP
syntax structure containing an integer number of *bytes* that is encapsulated in a *NAL unit*, and which is either empty or has the form of a *string of data bits* containing *syntax elements* followed by an *RBSP stop bit* and followed by zero or more subsequent bits equal to 0.

3.49
raw byte sequence payload stop bit
RBSP stop bit
bit equal to 1 present within an *RBSP* after a *string of data bits*.

Note 1 to entry: The location of the end of the string of data bits within an *RBSP* can be identified by searching from the end of the *RBSP* for the *RBSP stop bit*, which is the last non-zero bit in the *RBSP*.

3.50
reserved
term that may be used to specify that some values of a particular *syntax element* are for future use by ISO/IEC and is not to be used in *bitstreams* conforming to this version of this document, but may be used in bitstreams conforming to future extensions of this document by ISO/IEC

3.51
reserved_zeros
term that may be used to specify that some values of a particular *syntax element* are for future use by ISO/IEC and is not to be used in *bitstreams* conforming to this version of this document, but may be used in bitstreams conforming to future extensions of this document by ISO/IEC

Note 1 to entry: In this document the value of any reserved_zeros bit is zero

3.52
residual
difference between a prediction of a sample or data element and a reference of that same sample or data element

3.53
residual plane
collection of *residuals*

3.54
run length encoding
RLE
method for encoding a sequence of values in which consecutive occurrences of the same value are represented as a single value together with its number of occurrences

3.55
shall
term used to express mandatory requirements for conformance to this document

Note 1 to entry: When used to express a mandatory constraint on the values of syntax elements or on the results obtained by operation of the specified decoding process, it is the responsibility of the encoder to ensure that the constraint is fulfilled. When used in reference to operations performed by the decoding process, any decoding process that produces identical cropped decoded pictures to those output from the decoding process described in this document conforms to the decoding process requirements of this document.

3.56
should
term used to refer to behaviour of an implementation that is encouraged to be followed under anticipated ordinary circumstances but is not a mandatory requirement for conformance to this document

3.57
source
term used to describe the video material or some of its attributes before encoding

3.58
start code prefix
unique sequence of three *bytes* equal to 0x000001 embedded in the *byte stream* as a prefix to each *NAL unit*

Note 1 to entry: The location of a start code prefix can be used by a decoder to identify the beginning of a new NAL unit and the end of a previous NAL unit. Emulation of start code prefixes is prevented within NAL units by the inclusion of emulation prevention bytes.

3.59
string of data bits
SODB
sequence of some number of bits representing *syntax elements* present within a *raw byte sequence payload* prior to the *raw byte sequence payload stop bit*, and within which the left-most bit is considered to be the first and most significant bit, and the right-most bit is considered to be the last and least significant bit.

3.60
syntax element
element of data represented in the *bitstream*

3.61
syntax structure
zero or more *syntax elements* present together in the *bitstream* in a specified order

3.62
tile
rectangular region of *CU*s within a particular *picture*

3.63
transform coefficient
scalar quantity, considered to be in a transformed domain, that is associated with a particular index in an *inverse transform* part of the *decoding process*

3.64
unspecified
term that may be used to specify some values of a particular *syntax element* to indicate that the values have no specified meaning in this document and will not have a specified meaning in the future as an integral part of future versions of this document

3.65
video coding layer NAL unit
VCL NAL unit
collective term for *NAL units* that have *reserved* values of NalUnitType that are classified as VCL NAL units in this document

4 Abbreviations

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

CG	Coefficient Group
CPB	Coded Picture Buffer
CPBB	Coded Picture Buffer of the Base
CPBL	Coded Picture Buffer LCEVC
CU	Coding Unit
DPB	Decoded Picture Buffer
DPBB	Decoded Picture Buffer of the Base
DUT	Decoder Under Test
GBR	Green, Blue, and Red; same as RGB
HBD	Hypothetical Base Decoder
HDM	Hypothetical Demuxer
HRD	Hypothetical Reference Decoder
HSS	Hypothetical Stream Scheduler
I	Intra
IDR	Instantaneous Decoding Refresh
LCEVC	Low Complexity Enhancement Video Coding
LSB	Least Significant Bit
MSB	Most Significant Bit
NAL	Network Abstraction Layer
RBSP	Raw Byte Sequence Payload
RGB	Red, Green and, Blue; same as GBR
RLE	Run length encoding
SEI	Supplemental Enhancement Information
SODB	String of data bits
VCL	Video Coding Layer

5 Conventions

5.1 General (Informational)

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

5.2 Arithmetic operators

The following arithmetic operators are defined as follows:

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

5.3 Logical operators

The following logical operators are defined as follows:

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

5.4 Relational operators

The following relational operators are defined as follows:

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

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

5.5 Bit-wise operators

The following bit-wise operators are defined as follows:

&	Bit-wise “and”. When operating on integer arguments, operates on a two’s complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.
|	Bit-wise “or”. When operating on integer arguments, operates on a two’s complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.
^	Bit-wise “exclusive or”. When operating on integer arguments, operates on a two’s complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.
x >> y	Arithmetic right shift of a two’s complement integer representation of x by y binary digits. This function is defined only for non-negative integer values of y. Bits shifted into the most significant bits (MSBs) as a result of the right shift have a value equal to the MSB of x prior to the shift operation.
x << y	Arithmetic left shift of a two’s complement integer representation of x by y binary digits. This function is defined only for non-negative integer values of y. Bits shifted into the least significant bits (LSBs) as a result of the left shift have a value equal to 0.

5.6 Assignment operators

The following arithmetic operators are defined as follows:

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

5.7 Range notation

The following notations are used to specify a range of values:

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

5.8 Mathematical functions

The following mathematical functions are defined:

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

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

$$\text{Clip3}(x, y, z) = \begin{cases} x & ; \quad z < x \\ y & ; \quad z > y \\ z & ; \quad \text{otherwise} \end{cases} \tag{3}$$

Floor(x) the largest integer less than or equal to x. (4)

Ln(x) the natural logarithm of x (the base-e logarithm, where e is the natural logarithm base constant 2.718 281 828...). (5)

Log10(x) the base-10 logarithm of x. (6)

$$\text{Min}(x, y) = \begin{cases} x & ; \quad x <= y \\ y & ; \quad x > y \end{cases} \tag{7}$$

$$\text{Max}(x, y) = \begin{cases} “x” & ; \quad ”x >= y” \\ ”y” & ; \quad ”x < y” \end{cases} \tag{8}$$

Round(x) = Sign(x) * Floor(Abs(x) + 0.5) (9)

$$\text{Sign}(x) = \begin{cases} “1” & ; \quad ”x > 0” \\ ”0” & ; \quad ”x == 0” \\ -“1” & ; \quad ”x < 0” \end{cases} \tag{10}$$

$$\text{Sqrt}(x) = >\sqrt{x} \tag{11}$$

5.9 Order of operation precedence

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

- Operations of a higher precedence are evaluated before any operation of a lower precedence.
- Operations of the same precedence are evaluated sequentially from left to right.

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

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

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

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

5.10 Variables, syntax elements and tables

Syntax elements in the bitstream are represented in **bold** type. Each syntax element is described by its name (all lower-case letters with underscore characters), and one descriptor for its method of coded representation. The decoding process behaves according to the value of the syntax element and to the values of previously decoded syntax elements. When a value of a syntax element is used in the syntax tables or the text, it appears in regular (i.e., not bold) type.

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

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

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

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

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

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

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

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

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

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

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

5.11 Text description of logical operations

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

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

may be described in the following manner:

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

- If condition 0, statement 0
- Otherwise, if condition 1, statement 1
- ...
- Otherwise (informative remark on remaining condition), statement n

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

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

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

may be described in the following manner:

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

- If all of the following conditions are true, statement 0:
  - condition 0a
  - condition 0b
- Otherwise, if one or more of the following conditions are true, statement 1:
  - condition 1a
  - condition 1b
- ...

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

```
if(condition 0)
    statement 0
if(condition 1)
    statement 1
```

may be described in the following manner:

- When condition 0, statement 0
- When condition 1, statement 1

5.12 Processes

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

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

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

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

6 Bitstream and picture formats, partitioning, scanning processes and neighbouring relationships

6.1 Bitstream formats

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

The bitstream can be in one of two formats: the NAL unit stream format or the byte stream format. The NAL unit stream format is conceptually the more “basic” type. It consists of a sequence of syntax structures called NAL units. This sequence is ordered in decoding order. There are constraints imposed on the decoding order (and contents) of the NAL units in the NAL unit stream. The byte stream format can be constructed from the NAL unit stream format by ordering the NAL units in decoding order and prefixing each NAL unit with a start code prefix and zero or more zero-valued bytes to form a stream of bytes. The NAL unit stream format can be extracted from the byte stream format by searching for the location of the unique start code prefix pattern within this stream of bytes. Methods of framing the NAL units in a manner other than use of the byte stream format are outside the scope of this document. The byte stream format is specified in Annex B.

6.2 Source, decoded and ouptput picture formats

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

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

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

- Luma (Y) only (monochrome).
- Luma and two chroma (YCbCr or YCgCo).
- Green, blue, and red (GBR, also known as RGB).
- Arrays representing other unspecified monochrome or tri-stimulus colour samplings (for example, YZX, also known as XYZ).

For convenience of notation and terminology in this document, the variables and terms associated with these arrays are referred to as luma (or L or Y) and chroma, where the two chroma arrays are referred to as Cb and Cr; regardless of the actual colour representation method in use. The actual colour representation method in use can be indicated in syntax that is specified in Annex B.

The variables SubWidthC and SubHeightC are specified in Table 2, depending on the chroma format sampling structure, which is specified through chroma_format_idc and separate_colour_plane_flag. Other values of chroma_format_idc, SubWidthC and SubHeightC may be specified in the future by ITU-T | ISO/IEC.

Table 2 — ShiftWidthC and ShiftHeight values derived from chroma_sampling_type (subclause 7.3.4)

chroma_sampling_type	Chroma format	SubWidthC	SubHeightC
0	Monochrome	1	1
1	4:2:0	2	2

chroma_sampling_type	Chroma format	SubWidthC	SubHeightC
2	4:2:2	2	1
3	4:4:4	1	1

In monochrome sampling there is only one sample array, which is nominally considered the luma array.

In 4:2:0 sampling, each of the two chroma arrays has half the height and half the width of the luma array.

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

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

The number of bits necessary for the representation of each of the samples in the luma and chroma arrays in a video sequence is in the range of 8 to 16, inclusive, and the number of bits used in the luma array may differ from the number of bits used in the chroma arrays.

When the value of chroma_sampling_type is equal to 0, the nominal vertical and horizontal relative locations of luma and chroma samples in pictures are shown in Figure 1.

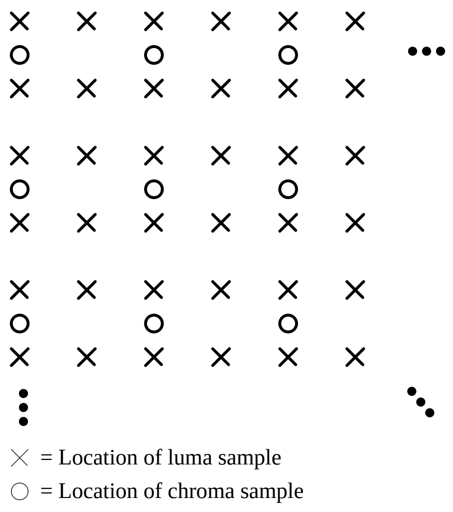

Figure 1 — Nominal vertical and horizontal locations of 4:2:0 luma and chroma samples in a picture

When the value of chroma_sampling_type is equal to 1, the chroma samples are co-sited with the corresponding luma samples and the nominal locations in a picture are as shown in Figure 2.

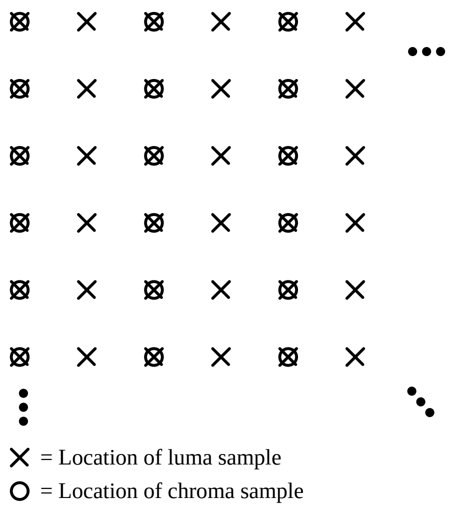

Figure 2 — Nominal vertical and horizontal locations of 4:2:2 luma and chroma samples in a picture

When the value of chroma_sampling_type is equal to 2, all array samples are co-sited for all cases of pictures and the nominal locations in a picture are as shown in Figure 3.

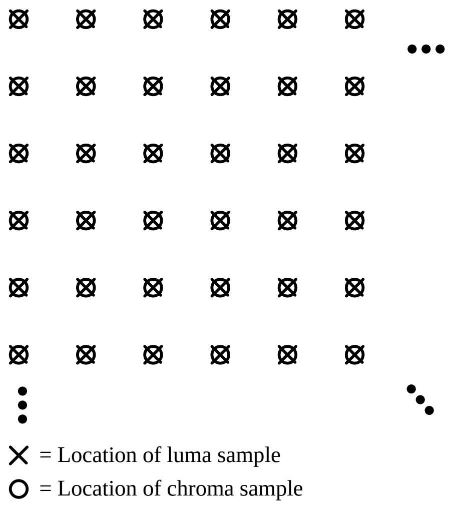

Figure 3 — Nominal vertical and horizontal locations of 4:4:4 luma and chroma samples in a picture

6.3 Partitioning of pictures

6.3.1 Organization of the hierarchical structure

Each picture is composed of three different planes, organized in a hierarchical structure: the decoded base picture planes and the residuals planes. The following sections specify how the different planes are organized.

The decoded base picture corresponds to the decoded output of a base decoder (Note: the bitstream syntax and decoding process for the base decoder is not part of this specification). Residuals planes are partitioned as described below.

6.3.2 Partitioning of residuals plane

A residuals plane is divided into CUs whose size depends on the size of the transform used. The CUs have either dimension 2x2 if a 2x2 directional decomposition transform is used or dimension 4x4 if a 4x4 directional decomposition transform is used.

7 Syntax and semantics

7.1 Method of specifying syntax in tabular form

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

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

Table 3 lists examples of the syntax specification format. When **syntax_element** appears, it specifies that a syntax element is parsed from the bitstream and the bitstream pointer is advanced to the next position beyond the syntax element in the bitstream parsing process.

Table 3 — Syntax Specification Format Examples

Syntax Specification	Descriptor
/* A statement can be a syntax element with an associated descriptor or can be an expression used to specify conditions for the existence, type and quantity of syntax elements, as in the following two examples */	
syntax_element	u(n)
conditioning statement	

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

7.2 Specification of syntax functions and descriptors

The functions presented in Table 4 are used in the syntactical description. These functions are expressed in terms of the value of a bitstream pointer that indicates the position of the next bit to be read by the decoding process from the bitstream.

Table 4 — Syntax Functions and Descriptors

Syntax function	Use
byte_stream_has_data( )	If the byte-stream has more data, then returns TRUE; otherwise returns FALSE.
process_payload_function(payload_type, payload_byte_size)	Behaves like a function lookup table, by selecting and invoking the process payload function relating to the payload_type as outlined in 7.3.4 below.
read_bits(n)	Reads the next n bits from the bitstream. Following the read operation, the bitstream pointer is advanced by n bit positions. When n is equal to 0, read_bits(n) returns a value equal to 0 and the bitstream pointer is not advanced.
read_byte(bitstream)	Reads a byte in the bitstream returning its value. Following the return of the value, the bitstream pointer is advanced by a byte.

Syntax function	Use
read_multibyte(bitstream)	Executes a read_byte(bitstream) until the MSB of the read byte is equal to zero.
bytestream_current(bitstream)	Returns the current bitstream pointer.
bytestream_seek(bitstream, n)	Returns the current bitstream pointer at the position in the bitstream corresponding to n bytes.

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

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

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

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

– ue(v): unsigned integer 0-th order Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in clause 9.4.

– mb: read multiple bytes. The parsing process for this descriptor is specified by the return value of the function read_multibyte(bitstream) interpreted as a binary representation of multiple unsigned char with most significant bit written first, and most significant byte of the sequence of unsigned char written first.

7.3 Syntax in tabular form

7.3.1 Syntax order

The order in which the syntax is presented is from MSB to LSB.

7.3.2 NAL unit and NAL unit header syntax

NAL unit and NAL unit header syntax shall be as specified in Table 5 and Table 6, respectively.

Table 5 — NAL unit

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

Table 6 — NAL unit header

Syntax	Descriptor
nal_unit_header( ) {	
forbidden_zero_bit	u(1)
forbidden_one_bit	u(1)

Syntax	Descriptor
nal_unit_type	u(5)
reserved_flag	u(9)
}	

7.3.3 Process block syntax

Process block syntax shall be as specified in Table 7.

Table 7 — Process block syntax

Syntax	Descriptor
process_block( ) {	
payload_size_type	u(3)
payload_type	u(5)
payload_size = 0	
if (payload_size_type == 7) {	
custom_byte_size	mb
payload_size = custom_byte_size	
} else {	
if (payload_size_type == 0) payload_size = 0	
if (payload_size_type == 1) payload_size = 1	
if (payload_size_type == 2) payload_size = 2	
if (payload_size_type == 3) payload_size = 3	
if (payload_size_type == 4) payload_size = 4	
if (payload_size_type == 5) payload_size = 5	
}	
if (payload_type == 0)	
process_payload_sequence_config(payload_size)	
else if (payload_type == 1)	
process_payload_global_config(payload_size)	
else if (payload_type == 2)	
process_payload_picture_config(payload_size)	
else if (payload_type == 3)	
process_payload_encoded_data(payload_size)	
else if (payload_type == 4)	
process_payload_encoded_data_tiled(payload_size)	
else if (payload_type == 5)	
process_payload_additional_info(payload_size)	
else if (payload_type == 6)	
process_payload_filler(payload_size)	
}	

7.3.4 Process payload – sequence configuration

Process payload global configuration syntax shall be as specified in Table 8.

Table 8 — Process payload – sequence configuration

Syntax	Descriptor
process_payload_sequence_config(payload_size) {	
profile_idc	u(4)
level_idc	u(4)
sublevel_idc	u(2)
conformance_window_flag	u(1)
reserved_zeros_5bit	u(5)
if (profile_idc == 16 || level_idc == 16) {	
extended_profile_idc	u(3)

Syntax	Descriptor
extended_level_idc	u(4)
reserved_zeros_1bit	u(1)
}	
if (conformance_window_flag == 1) {	
conf_win_left_offset	mb
conf_win_right_offset	mb
conf_win_top_offset	mb
conf_win_bottom_offset	mb
}	
}	

7.3.5 Process payload – global configuration

Process payload global configuration syntax shall be as specified in Table 9.

Table 9 — Process payload – global configuration

Syntax	Descriptor
process_payload_global_config(payload_size) {	
processed_planes_type_flag	u(1)
resolution_type	u(6)
transform_type	u(1)
chroma_sampling_type	u(2)
base_depth_type	u(2)
enhancement_depth_type	u(2)
temporal_step_width_modifier_signalled_flag	u(1)
predicted_residual_mode_flag	u(1)
temporal_tile_intra_signalling_enabled_flag	u(1)
temporal_enabled_flag	u(1)
upsample_type	u(3)
level_1_filtering_signalled_flag	u(1)
scaling_mode_level1	u(2)
scaling_mode_level2	u(2)
tile_dimensions_type	u(2)
user_data_enabled	u(2)
level1_depth_flags	u(1)
reserved_zeros_1bit	u(1)
if (temporal_step_width_modifier_signalled_flag == 1) {	
temporal_step_width_modifier	u(8)
} else {	
temporal_step_width_modifier = 48	
}	
if (level_1_filtering_signalled_flag) {	
level_1_filtering_first_coefficient	u(4)
level_1_filtering_second_coefficient	u(4)
}	
if (tile_dimensions_type > 0) {	
if (tile_dimensions_type == 3) {	
custom_tile_width	u(16)
custom_tile_height	u(16)
}	
reserved_zeros_5bit	u(5)
compression_type_entropy_enabled_per_tile_flag	u(1)
compression_type_size_per_tile	u(2)
}	
if (resolution_type == 63) {	
custom_resolution_width	u(16)

Syntax	Descriptor
custom_resolution_height	u(16)
}	
}	

7.3.6 Process payload – picture configuration

Process payload picture configuration syntax shall be as specified in Table 10.

Table 10 — Proccess payload – picture configuration

Syntax	Descriptor
process_payload_picture_config(payload_size) {	
no_enhancement_bit_flag	u(1)
if (no_enhancement_bit_flag == 0) {	
quant_matrix_mode	u(3)
dequant_offset_signalled_flag	u(1)
picture_type_bit_flag	u(1)
temporal_refresh_bit_flag	u(1)
step_width_level1_enabled_flag	u(1)
step_width_level2	u(15)
dithering_control_flag	u(1)
} else {	
reserved_zeros_4bit	u(4)
picture_type_bit_flag	u(1)
temporal_refresh_bit_flag	u(1)
temporal_signalling_present_flag	u(1)
}	
if (picture_type_bit_flag == 1) {	
field_type_bit_flag	u(1)
reserved_zeros_7bit	u(7)
}	
if (step_width_level1_enabled_flag == 1) {	
step_width_level1	u(15)
level_1_filtering_enabled_flag	u(1)
}	
if (quant_matrix_mode == 2 || quant_matrix_mode == 3 || quant_matrix_mode == 5) {	
for(layerIdx = 0; layerIdx < nLayers; layerIdx++) {	
qm_coefficient_0[layerIdx]	u(8)
}	
}	
if (quant_matrix_mode == 4 || quant_matrix_mode == 5) {	
for(layerIdx = 0; layerIdx < nLayers; layerIdx++) {	
qm_coefficient_1[layerIdx]	u(8)
}	
}	
if (dequant_offset_signalled_flag) {	
dequant_offset_mode_flag	u(1)
dequant_offset	u(7)
}	
if (dithering_control_flag == 1) {	
dithering_type	u(2)
reserverd_zero	u(1)
if (dithering_type != 0) {	
dithering_strength	u(5)
} else {	
reserved_zeros_5bit	u(5)
}	

Syntax	**Descriptor**
}	
}	

7.3.7 Process payload – encoded data

Process payload encoded data syntax shall be as specified in Table 11.

Table 11 — Process payload – encoded data

Syntax	**Descriptor**
process_payload_encoded_data(payload_size) {	
if (tile_dimensions_type == 0) {	
for (planeIdx = 0; planeIdx < nPlanes; planeIdx++) {	
if (no_enhancement_bit_flag == 0) {	
for (levelIdx = 1; levelIdx <= 2; levelIdx++) {	
for (layerIdx = 0; layerIdx < nLayers;layerIdx++) {	
surfaces[planeIdx][levelIdx][layerIdx]. **entropy_enabled_flag**	u(1)
surfaces[planeIdx][levelIdx][layerIdx]**.rle_only_flag**	u(1)
}	
}	
}	
if (temporal_signalling_present_flag == 1){	
temporal_surfaces[planeIdx]**.entropy_enabled_flag**	u(1)
temporal_surfaces[planeIdx]**.rle_only_flag**	u(1)
}	
}	
byte_alignment( )	
for (planeIdx = 0; planeIdx < nPlanes; planeIdx++) {	
for (levelIdx = 1; levelIdx <= 2; levelIdx++) {	
for (layerIdx = 0; layerIdx < nLayers; layerIdx++)	
process_surface(surfaces[planeIdx][levelIdx][layerIdx])	
}	
if (temporal_signalling_present_flag == 1)	
process_surface(temporal_surfaces[planeIdx])	
}	
} else {	
process_payload_encoded_data_tiled(payload_size)	
}	
}	

7.3.8 Process payload – encoded tiled data

Process payload encoded data syntax shall be as specified in Table 12.

Table 12 — Process payload – encoded tiled data

Syntax	**Descriptor**
process_payload_encoded_data_tiled(payload_size) {	
for (planeIdx = 0; planeIdx < nPlanes; planeIdx++) {	
for (levelIdx = 1; levelIdx <= 2; levelIdx++) {	
if (no_enhancement_bit_flag == 0) {	
for (layerIdx = 0; layerIdx < nLayers;layerIdx++)	
surfaces[planeIdx][levelIdx][layerIdx]**.rle_only_flag**	u(1)
}	
}	
if (temporal_signalling_present_flag == 1)	
temporal_surfaces[planeIdx]**.rle_only_flag**	u(1)

Syntax	Descriptor
}	
byte_alignment( )	
if (compression_type_entropy_enabled_per_tile_flag == 0) {	
for (planeIdx = 0; planeIdx < nPlanes; planeIdx++) {	
if (no_enhancement_bit_flag == 0) {	
for (levelIdx = 1; levelIdx <= 2; levelIdx++) {	
if (levelIdx == 1)	
nTiles = nTilesL1	
else	
nTiles = nTilesL2	
for (layerIdx = 0; layerIdx < nLayers; layerIdx++) {	
for (tileIdx = 0; tileIdx < nTiles; tileIdx++)	
surfaces[planeIdx][levelIdx][layerIdx]**.tiles**[tileIdx]**.** **entropy_enabled_flag**	u(1)
}	
}	
}	
if (temporal_signalling_present_flag == 1) {	
for (tileIdx = 0; tileIdx < nTilesL2; tileIdx++)	
temporal_surfaces[planeIdx]**.tiles**[tileIdx]**.entropy_enabled_flag**	u(1)
}	
}	
} else {	
entropy_enabled_per_tile_compressed_data_rle	mb
}	
byte_alignment( )	
if (compression_type_size_per_tile == 0) {	
for (planeIdx = 0; planeIdx < nPlanes; planeIdx++) {	
for (levelIdx = 1; levelIdx <= 2; levelIdx++) {	
if (levelIdx == 1)	
nTiles = nTilesL1	
else	
nTiles = nTilesL2	
for (layerIdx = 0; layerIdx < nLayers; layerIdx++) {	
for (tileIdx = 0; tileIdx < nTiles; tileIdx++)	
process_surface(surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx])	
}	
}	
if (temporal_signalling_present_flag == 1) {	
for (tileIdx = 0; tileIdx < nTilesL2; tileIdx++)	
process_surface(temporal_surfaces[planeIdx].tiles[tileIdx])	
}	
}	
} else {	
for (planeIdx = 0; planeIdx < nPlanes; planeIdx++) {	
for (levelIdx = 1; levelIdx <= 2; levelIdx++) {	
if (levelIdx == 1)	
nTiles = nTilesL1	
else	
nTiles = nTilesL2	
for (layerIdx = 0; layerIdx < nLayers; layerIdx++) {	
if(surfaces[planeIdx][levelIdx][layerIdx].rle_only_flag) {	
compressed_size_per_tile_prefix	mb
} else {	
compressed_prefix_last_symbol_bit_offset_per_tile_prefix	mb
compressed_size_per_tile_prefix	mb

Syntax	Descriptor
}	
for (tileIdx=0; tileIdx < nTiles; tileIdx++)	
process_surface(surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx])	
}	
}	
if (temporal_signalling_present_flag == 1) {	
if(temporal_surfaces[planeIdx].rle_only_flag) {	
compressed_size_per_tile_prefix	mb
} else {	
compressed_prefix_last_symbol_bit_offset_per_tile_prefix	mb
compressed_size_per_tile_prefix	mb
}	
for (tileIdx = 0; tileIdx < nTilesL2; tileIdx++)	
process_surface(temporal_surfaces[planeIdx].tiles[tileIdx])	
}	
}	
}	
}	

7.3.9 Process payload – surface

Process payload surface syntax shall be as specified in Table 13.

Table 13 — Process payload – surface

Syntax	Descriptor
process_surface(surface) {	
if (compression_type_size_per_tile == 0) {	
if (surface.entropy_enabled_flag) {	
if (surface.rle_only_flag) {	
surface.size	mb
surface.data	surface.size
} else {	
reserved_zeros_3bit	u(3)
surface.prefix_last_symbol_bit_offset	u(5)
surface.size	mb
surface.data	surface.size
}	
}	
} else {	
if (surface.entropy_enabled_flag) {	
surface.data	surface.size
}	
}	
}	

7.3.10 Process payload – additional info

Process payload additional info syntax shall be as specified in Table 14.

Table 14 — Process payload – additional info

Syntax	Descriptor
additional_info(payload_size) {	
additional_info_type	u(8)
if (additional_info_type == 0) {	
payload_type	u(8)
sei_payload(payload_type, payload_size − 2)	

Syntax	Descriptor
} else if (additional_info_type == 1)	
vui_parameters (payload_size − 1)	
else // (additional_info_type >= 2)	
// reserved for future use	
}	

7.3.11 Process payload – filler

Process payload filler syntax shall be as specified in Table 15.

Table 15 — Process payload – filler

Syntax	Descriptor
process_payload_filler(payload_size) {	
for(x = 0; x < payload_size; x++) {	
filler_byte // equal to 0xAA	u(8)
}	
}	

7.3.12 Byte alignment syntax

Byte alignment syntax shall be as specified in Table 16.

Table 16 — Byte alignment syntax

Syntax	Descriptor
byte_alignment( ) {	
alignment_bit_equal_to_one /* equal to 1 */	f(1)
while(!byte_aligned( ))	
alignment_bit_equal_to_zero /* equal to 0 */	f(1)
}	

7.4 Semantics

7.4.1 General

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

7.4.2 NAL unit semantics

7.4.2.1 General NAL unit semantics

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

rbsp_byte[i] is the i-th byte of an RBSP. An RBSP is specified as an ordered sequence of bytes as follows:
The RBSP contains a SODB as follows:

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

Otherwise, the RBSP contains the SODB as follows:

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

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

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

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

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

Syntax structures having these RBSP properties are denoted in the syntax tables using an "_rbsp" suffix. These structures are carried within NAL units as the content of the rbsp_byte[i] data bytes. The association of the RBSP syntax structures to the NAL units is as specified in Table 17.

NOTE When the boundaries of the RBSP are known, the decoder can extract the SODB from the RBSP by concatenating the bits of the bytes of the RBSP and discarding the rbsp_stop_one_bit, which is the last (least significant, right-most) bit equal to 1, and discarding any following (less significant, farther to the right) bits that follow it, which are equal to 0. The data necessary for the decoding process is contained in the SODB part of the RBSP.

emulation_prevention_three_byte is a byte equal to 0x03. When an emulation_prevention_three_byte is present in the NAL unit, it shall be discarded by the decoding process.

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

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

0x000000

0x000001

0x000002

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

0x00000300

0x00000301

0x00000302

0x00000303

7.4.2.2 NAL unit header semantics

forbidden_zero_bit shall be equal to 0.

forbidden_one_bit shall be equal to 1.

nal_unit_type specifies the type of RBSP data structure contained in the NAL unit as specified in Table 17. NAL units that have nal_unit_type in the range of UNSPEC0…UNSPEC27, inclusive, and UNSPEC31 for which semantics are not specified, shall not affect the decoding process specified in this specification.

reserved_flag shall be equal to the bit sequence 111111111.

Table 17 — NAL unit type codes and NAL unit type classes

nal_unit_type	**Name of nal_unit_type**	**Content of NAL unit and RBSP syntax structure**	**NAL unit type class**
0...27	UNSPEC0…UNSPEC27	unspecified	Non-VCL
28	LCEVC_LEVEL	segment of a Low Complexity Enhancement Level	VCL/Non-VCL

29-30	RSV_LEVEL	reserved	Non-VCL
31	UNSPEC31	unspecified	Non-VCL

NOTE 1 NAL unit types in the range of UNSPEC0…UNSPEC27 and UNSPEC31 may be used as determined by the application. No decoding process for these values of nal_unit_type is specified in this Specification. Since different applications might use these NAL unit types for different purposes, particular care must be exercised in the design of encoders that generate NAL units with these nal_unit_type values, and in the design of decoders that interpret the content of NAL units with these nal_unit_type values.

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

NOTE 2 This requirement allows future definition of compatible extensions to this Specification.

7.4.2.3 Data block unit general semantics

payload_size_type specifies the size of the payload, and it shall take a value between 0 and 7, as specified by Table 18.

Table 18 — Payload sizes

payload_size_type	Size (bytes)
0	0
1	1
2	2
3	3
4	4
5	5
6	Reserved
7	Custom

payload_type specifies the type of the payload used, and it shall take a value between 0 and 31, as specified by Table 19.

Table 19 — Content of payload and minimum frequency of appearance of such content within a bitstream

payload_type	Content of payload	Minimum frequency
0	process_payload_sequence_config( )	at least once
1	process_payload_global_config( )	at least every IDR
2	process_payload_picture_config( )	picture
3	process_payload_encoded_data( )	picture
4	process_payload_encoded_data_tiled( )	picture
5	process_payload_additional_info( )	picture
6	process_payload_filler( )	picture
7-30	Reserved	
31	Custom	

7.4.3 Data block unit configuration semantics

7.4.3.1 Data block semantics

The following describes the semantics for each of the data block units.

7.4.3.2 Data block unit sequence configuration semantics

The following describes the semantics for each of the data block units.

profile_idc indicates a profile as specified in Annex A. Bitstreams shall not contain values of profile_idc other than those specified in Annex A. Other values of profile_idc are reserved for future use by ITU-T | ISO/IEC

level_idc indicates a level as specified in Annex A. Bitstreams shall not contain values of level_idc other than those specified in Annex A. Other values of level_idc are reserved for future use by ITU-T | ISO/IEC.

sublevel_idc indicates a sublevel as specified in Annex A.

conformance_window_flag equal to 1 indicates that the conformance cropping window offset parameters are present in the sequence configuration data block. conformance_window_flag equal to 0 indicates that the conformance cropping window offset parameters are not present.

extended_profile_idc indicates an extended profile as specified in Annex A. Bitstreams shall not contain values of extended profile_idc other than those specified in Annex A. Other values of extended_profile_idc are reserved for future use by ITU-T | ISO/IEC

extended_level_idc indicates an extended level as specified in Annex A. Bitstreams shall not contain values of extended_level_idc other than those specified in Annex A. Other values of extended_level_idc are reserved for future use by ITU-T | ISO/IEC.

conf_win_left_offset, **conf_win_right_offset**, **conf_win_top_offset** and **conf_win_bottom_offset** specify the samples of the pictures in the CVS that are output from the decoding process, in terms of a rectangular region specified in picture coordinates for output. When conformance_window_flag is equal to 0, the values of conf_win_left_offset, conf_win_right_offset, conf_win_top_offset and conf_win_bottom_offset are inferred to be equal to 0.

The conformance cropping window contains the luma samples with horizontal picture coordinates from SubWidthC * conf_win_left_offset to width − (SubWidthC * conf_win_right_offset + 1) and vertical picture coordinates from SubHeightC * conf_win_top_offset to height − (SubHeightC * conf_win_bottom_offset + 1), inclusive.

The value of SubWidthC * (conf_win_left_offset + conf_win_right_offset) shall be less than width, and the value of SubHeightC * (conf_win_top_offset + conf_win_bottom_offset) shall be less than height.

The corresponding specified samples of the two chroma arrays are the samples having picture coordinates (x / SubWidthC, y / SubHeightC), where (x, y) are the picture coordinates of the specified luma samples.

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

7.4.3.3 Data block unit global configuration semantics

processed_planes_type_flag specifies the plane to be processed by the decoder. It shall be equal to 0 or 1. If equal to 0, only the Luma (Y) plane shall be processed. If equal to 1, all planes (Luma and two chroma) shall be processed. If processed_planes_type_flag is equal to 0 nPlanes shall be equal to 1 and if processed_planes_type_flag is equal to 1 nPlanes shall be equal to 3.

resolution_type specifies the resolution of the Luma (Y) plane of the enhanced decoded picture, and it shall take a value between 0 and 63, as specified by Table 20. The value of the type is expressed as NxM, where N is the width of the Luma (Y) plane of the enhanced decoded picture and M is height of the Luma (Y) plane of the enhanced decoded picture.

Table 20 — Resolution of the decoded picture

resolution_type	**Value of type**
0	unused /* Escape code prevention */
1	360x200
2	400x240

resolution_type	**Value of type**
3	480x320
4	640x360
5	640x480
6	768x480
7	800x600
8	852x480
9	854x480
10	856x480
11	960x540
12	960x640
13	1024x576
14	1024x600
15	1024x768
16	1152x864
17	1280x720
18	1280x800
19	1280x1024
20	1360x768
21	1366x768
22	1400x1050
23	1440x900
24	1600x1200
25	1680x1050
26	1920x1080
27	1920x1200
28	2048x1080
29	2048x1152
30	2048x1536
31	2160x1440
32	2560x1440
33	2560x1600
34	2560x2048
35	3200x1800
36	3200x2048
37	3200x2400
38	3440x1440
39	3840x1600
40	3840x2160
41	3840x2400
42	4096x2160
43	4096x3072
44	5120x2880
45	5120x3200
46	5120x4096
47	6400x4096
48	6400x4800
49	7680x4320
50	7680x4800
51-62	Reserved
63	Custom

chroma_sampling_type defines the colour format for the enhanced decoded picture (see Section 6.2) in accordance with Table 21.

Table 21 — Colour format for the decoded picture

chroma_sampling_type	Value of type
0	Monochrome
1	4:2:0
2	4:2:2
3	4:4:4

transform_type defines the type of transform to be used in accordance with Table 22. If transform_type is equal to 0 nLayers shall be equal to 4 and if transform_type is equal to 1 nLayers shall be equal to 16.

Table 22 — Transform used for decoding

transform_type	Value of type
0	2x2 directional decomposition transform
1	4x4 directional decomposition transform

base_depth_type defines the bit depth of the decoded base picture in accordance with Table 23.

Table 23 — Bit depth of the decoded base picture

base_depth_type	Value of type
0	8
1	10
2	12
3	14

enhancement_depth_type defines the bit depth of the enhanced decoded picture in accordance with Table 24.

Table 24 — Bit depth of the decoded picture

enhancement_depth_type	Value of type
0	8
1	10
2	12
3	14

temporal_step_width_modifier_signalled_flag specifies if the value of the temporal_step_width_modifier parameter is signalled. It shall be equal to 0 or 1. If equal to 0, the temporal_step_width_modifier parameter is not signalled.

predicted_residual_mode_flag specifies whether the decoder should activate the predicted residual process during the decoding process. If the value is 0, the predicted residual process shall be disabled.

temporal_tile_intra_signalling_enabled_flag specifies whether temporal tile prediction should be used when decoding a 32x32 tile. If the value is 1, the temporal tile prediction process shall be enabled.

upsample_type specifies the type of upsampler to be used in the decoding process in accordance with Table 25.

Table 25 — Upsampler type

upsample_type	Value of type
0	Nearest
1	Linear
2	Cubic
3	Modified Cubic
4-6	Reserved
7	Custom

level_1_filtering_signalled specifies whether deblocking filter should use the signalled parameters instead of default parameters. If equal to 1, the values of the deblocking coefficients are signalled.

temporal_step_width_modifier specifies the value used to calculate a variable step width modifier for transforms that use temporal prediction. If temporal_step_width_modifier_signalled_flag is equal to 0, temporal_step_width_modifier is set to 48.

level_1_filtering_first_coefficient specifies the value of the first coefficient in the deblocking mask namely 4x4 block corner residual weight. The value of the first coefficient shall be between 0 and 15.

level_1_filtering_second_coefficient specifies the value of the second coefficient in the deblocking mask namely 4x4 block side residual weight. The value of the second coefficient shall be between 0 and 15.

scaling_mode_level1 specifies whether and how the upsampling process should be performed between decoded base picture and preliminary intermediate picture in accordance with Table 26. The preliminary intermediate picture corresponds to the output of process of 8.8.1.1.

Table 26 — Scaling mode level1 values

scaling_mode_level1	Value of type
0	no scaling
1	one-dimensional 2:1 scaling only across the horizontal dimension
2	two-dimensional 2:1 scaling across both dimensions
3	Reserved

scaling_mode_level2 specifies whether and how the upsampling process should be performed between combined intermediate picture and preliminary output picture in accordance with Table 27. The combined intermediate picture corresponds to the output of process 8.9.1. The preliminary output picture corresponds to the output of process 8.8.1.2. Scaling mode level2values are specified in Table 27.

Table 27 — Scaling mode level2values

scaling_mode_level2	Value of type
0	no scaling
1	one-dimensional 2:1 scaling only across the horizontal dimension
2	two-dimensional 2:1 scaling across both dimensions
3	Reserved

user_data_enabled specifies whether user data are included in the bitstream and the size of the user data, as specified in Table 28.

Table 28 — User data

user_data_enabled	**Value of type**
0	disabled
1	enabled 2-bits
2	enabled 6-bits
3	reserved

level1_depth_flag specifies whether level1 is processed using the base depth type or the enhancement depth type. If the value of the flag is 0, level1 sub-layer shall be processed using the base depth type. If the value of the flag is 1, level 1 sub-layer shall be processed using the enhancement depth type.

tile_dimensions_type specifies the resolution of the picture tiles, and it shall take a value between 0 and 3 according to Table 29. The value of the type is expressed as NxM, where N is the width of the picture tile and M is height of the picture tile.

Table 29 — Resolution of the decoded picture tiles

tile_dimensions_type	**Value of type**
0	no tiling
1	512x256
2	1024x512
3	Custom

custom_tile_width specifies a custom width for the tile.

custom_tile_height specifies a custom height for the tile.

compression_type_entropy_enabled_per_tile_flag specifies the compression method used to encode the entropy_enabled_flag field of each picture tile, and it shall take a value between 0 and 1 according to Table 30.

Table 30 — Compression method for entropy_enabled_flag of each picture tile

compression_type_entropy_enabled_per_tile_flag	**Value of type**
0	No compression used
1	Run length encoding

compression_type_size_per_tile specifies the compression method used to encode the size field of each picture tile, and it shall take a value between 0 and 3 according to Table 31.

Table 31 — Compression method for size of each picture tile

compression_type_size_per_tile	**Value of type**
0	No compression used
1	Prefix Coding encoding
2	Prefix Coding encoding on differences
3	Reserved

custom_resolution_width specifies the width of a custom resolution.

custom_resolution_height specifies the height of a custom resolution.

7.4.3.4 Data block unit picture configuration semantics

no_enhancement_bit_flag specifies that there are no enhancement data for all layerIdx < nLayers in the picture.

quant_matrix_mode specifies which quantization matrix to be used in the decoding process in accordance with Table 32. When quant_matrix_mode is not present, it is inferred to be equal to 0.

Table 32 — Quantization matrix

quant_matrix_mode	Value of type
0	each enhancement sub-layer uses the matrices used for the previous frame, unless the current picture is an IDR picture, in which case both enhancement sub-layers use default matrices
1	both enhancement sub-layers use default matrices
2	one matrix of modifiers is signalled and should be used on both residual plane
3	one matrix of modifiers is signalled and should be used on enhancement sub-layer 2 residual plane
4	one matrix of modifiers is signalled and should be used on enhancement sub-layer 1 residual plane
5	two matrices of modifiers are signalled – the first one for enhancement sub-layer 2 residual plane, the second for enhancement sub-layer 1 residual plane
6-7	Reserved

dequant_offset_signalled_flag specifies if the offset method and the value of the offset parameter to be applied when dequantizing is signalled. If equal to 1, the method for dequantization offset and the value of the dequantization offset parameter are signalled. When dequant_offset_signalled_flag is not present, it is inferred to be equal to 0.

picture_type_bit_flag specifies whether the encoded data are sent on a frame basis (e.g., progressive mode or interlaced mode) or on a field basis (e.g., interlaced mode) in accordance with Table 33.

Table 33 — Picture type

picture_type_bit_flag	Value of type
0	Frame
1	Field

field_type_bit_flag specifies, if picture_type is equal to 1, whether the data sent are for top or bottom field in accordance with Table 34.

Table 34 — Field type

field_type_bit_flag	Value of type
0	Top
1	Bottom

temporal_refresh_bit_flag specifies whether the temporal buffer should be refreshed for the picture. If equal to 1, the temporal buffer should be refreshed.

temporal_signalling_present_flag specifies whether the temporal signalling coefficient group is present in the bitstream. When temporal_signalling_present_flag is not present, it is inferred to be equal to 1 if temporal_enabled_flag is equal to 1 and temporal_refresh_bit_flag is equal to 0, otherwise it is inferred to be equal to 0.

step_width_level2 specifies the value of the stepwidth value to be used when decoding the encoded residuals in enhancement sub-layer 2.

step_width_level1_enabled_flag specifies whether the value of the stepwidth to be used when decoding the encoded residuals in the enhancement sub-layer 1 is a default value or is signalled. It shall be either 0 (default value) or 1 (value signalled by step_width_level1). The default value is 32,767. When step_width_level1_enabled_flag is not present, it is inferred to be equal to 0.

dithering_control_flag specifies whether dithering should be applied. It shall be either 0 (dithering disabled) or 1 (dithering enabled). When dithering_control_flag is not present, it is inferred to be equal to 0.

step_width_level1 specifies the value of the stepwidth value to be used when decoding the encoded residuals in enhancement sub-layer 1.

level1_filtering_enabled_flag specifies whether the level1 deblocking filter should be used. It shall be either 0 (filtering disabled) or 1 (filtering enabled). When level1_filtering_enabled_flag is not present, it is inferred to be equal to 0.

qm_coefficient_0[layerIdx] specifies the values of the quantization matrix scaling parameter when quant_matrix_mode is equal to 2, 3 or 5.

qm_coefficient_1[layerIdx] specifies the values of the quantization matrix scaling parameter for when quant_matrix_mode is equal to 4 or 5.

dequant_offset_mode_flag specifies the method for applying dequantization offset. If equal to 0, the default method applies, using the signaled dequant_offset as parameter. If equal to 1, the constant-offset method applies, using the signaled dequant_offset parameter.

dequant_offset specifies the value of the dequantization offset parameter to be applied. The value of the dequantization offset parameter shall be between 0 and 127, inclusive.

dithering_type specifies what type of dithering is applied to the final reconstructed picture according to Table 35.

Table 35 — Dithering

dithering_type	**Value of type**
0	None
1	Uniform
2-3	Reserved

dithering_strength specifies a value between 0 and 31.

7.4.3.5 Data block unit encoded data semantics

surfaces[planeIdx][levelIdx][layerIdx].**entropy_enabled_flag** indicates whether there are encoded data in surfaces[planeIdx][levelIdx][layerIdx].

surfaces[planeIdx][levelIdx][layerIdx]**.rle_only_flag** indicates whether the data in surfaces[planeIdx][levelIdx][layerIdx].are encoded using only RLE or using RLE and Prefix Coding.

temporal_surfaces[planeIdx].**entropy_enabled_flag** indicates whether there are encoded data in temporal_surfaces[planeIdx].

temporal_surfaces[planeIdx]**.rle_only_flag** indicates whether the data in temporal_surfaces[planeIdx] are encoded using only RLE or using RLE and Prefix Coding.

7.4.3.6 Data block unit encoded tiled data semantics

surfaces[planeIdx][levelIdx][layerIdx]**.rle_only_flag** indicates whether the data in surfaces[planeIdx][levelIdx][layerIdx] are encoded using only RLE or using RLE and Prefix Coding.

surfaces[planeIdx][levelIdx][layerIdx]**.tiles**[tileIdx]**.entropy_enabled_flag** indicates whether there are encoded data in surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].

temporal_surfaces[planeIdx]**.rle_only_flag** indicates whether the data in temporal_surfaces[planeIdx] are encoded using only RLE or using RLE and Prefix Coding.

temporal_surfaces[planeIdx]**.tiles**[tileIdx]**.entropy_enabled_flag** indicates whether there are encoded data in temporal_surfaces[planeIdx].tiles[tileIdx].

entropy_enabled_per_tile_compressed_data_rle contains the RLE-encoded signalling for each picture tile and shall be decoded based on subclause 9.3.5.

compressed_size_per_tile_prefix specifies the compressed size of the encoded data for each picture tile, and shall be decoded based on subclause 9.2.3.

compressed_prefix_last_symbol_bit_offset_per_tile_prefix specifies the last symbol bit offset of the Prefix Coding encoded data, and shall be decoded based on subclause 9.2.4.

7.4.3.7 Data block unit surface semantics

surface.size specifies the size of the entropy encoded data (see subclause 9.1).

surface.data contains the entropy encoded data (see subclause 9.1).

surface.prefix_last_symbol_bit_offset specifies the last symbol bit offset of the Prefix Coding encoded data (see subclause 9.2)

7.4.3.8 Data block unit additional info semantics

additional_info_type contains additional information in the form of either SEI messages (see Annex D) or VUI messages (see Annex E).

payload_type specifies the payload type of an SEI message (see Annex D).

7.4.3.9 Data block unit filler semantics

filler byte shall be a byte equal to 0xAA.

8 Decoding process

8.1 General decoding process

The input to this process is a LCEVC bitstream that contains an enhancement layer consisting of up to two sub-layers.

The outputs of this process are:

- the enhancement residuals planes (sub-layer 1 residual planes) to be added to the preliminary pictures 1, obtained from the base decoder reconstructed pictures; and
- the enhancement residuals planes (sub-layer 2 residual planes) to be added to the preliminary output pictures resulting from upscaling and modifying via predicted residuals the combination of the preliminary pictures 1 and the sub-layer 1 residual planes.

The decoding process operates on data blocks and it is described as follows:

1. The decoding of Payload data block units is specified in clause 8.2.
2. The decoding process for the picture is specified in clause 8.3 using syntax elements in clause 7.
3. The decoding process for temporal prediction is specified in clause 8.4.
4. The decoding process for the dequantization process is specified in clause 8.5.
5. The decoding process for the transform is specified in clause 8.6.

6. The decoding process for the upscaler is specified in clause 8.7.
7. The decoding process for the predicted residual is specified in clause 8.7.4.
8. The decoding process for the residual reconstruction is specified in clause 8.8.
9. The decoding process for the L-1 filter is specified in clause 8.9.
10. The decoding process for the base encoding data extraction is specified in clause 8.10.
11. The decoding process for the dithering filter is specified in clause 8.11.

8.2 Payload data block unit process

Input to this process is the enhancement layer bitstream. The enhancement layer bitstream is encapsulated in NAL units specified in subclause 7.2.1. A NAL unit is used to synchronize the enhancement layer information with the base layer decoded information.

The bitstream is organized in NAL units, with each NAL unit including one or more data blocks. For each data block, the process_block( ) syntax structure (subclause 7.2.3) is used to parse a block header (and only the block header) and invokes the relevant process_block_( ) syntax element based upon the information in the block header. A NAL unit which include encoded data comprises at least two data blocks, a picture configuration data block and an encoded (tiled) data block. The possible different data blocks are indicated in Table 19.

A sequence configuration data block shall occur at least once at the beginning of the bitstream. A global configuration data block shall occur at least for every IDR picture. Every encoded (tiled) data block shall be preceded by a picture configuration data block. When present in a NAL unit, a global configuration data block shall be the first data block in the NAL unit.

8.3 Picture enhancement decoding process

8.3.1 General enhancement decoding process

Input of this process is the portion of the bitstream following the headers decoding process described in subclause 7.3.3. Outputs are the entropy encoded transform coefficients belonging to the picture enhancement being decoded. Every encoded picture is preceded by the picture configuration payload described in subclause 7.3.6 and 7.4.3.4.

8.3.2 Decoding process for picture enhancement encoded data (payload_encoded_data)

Syntax of this process is described in subclause 7.3.7. Inputs to this process are:

- A variable nPlanes containing the number of plane (subclause 7.4.3.2 depending on the value of the variable processed_planes_type_flag),
- A variable nLayers (subclause 7.4.3.2 depending on the value of transform_type),
- A constant nLevel equal to 2, since 2 enhancement sub-layers are processed.

Output of this process is the (nPlanes)x(nLevels)x(nLayers) array surfaces with elements surfaces[nPlanes][nLevels][nLayers] and, if temporal_signalling_present_flag is equal to 1, an additional temporal surface of a size nPlanes with elements temporal_surface[nPlanes].

- variable nPlanes is derived as follows:
```
if (processed_planes_type_flag == 0)
    nPlanes = 1
else
    nPlanes = 3
```
- variable nLayers is derived as follows:
```
if (transform_type == 0)
    nLayers = 4
else
    nLayers = 16
```

The encoded data is organized in chunks. The total number of chunks is calculated as follows:

total_chunk_count = nPlanes * nLevels * nLayers * (no_enhancement_bit_flag == 0) + nPlanes * (temporal_signalling_present_flag == 1) (12)

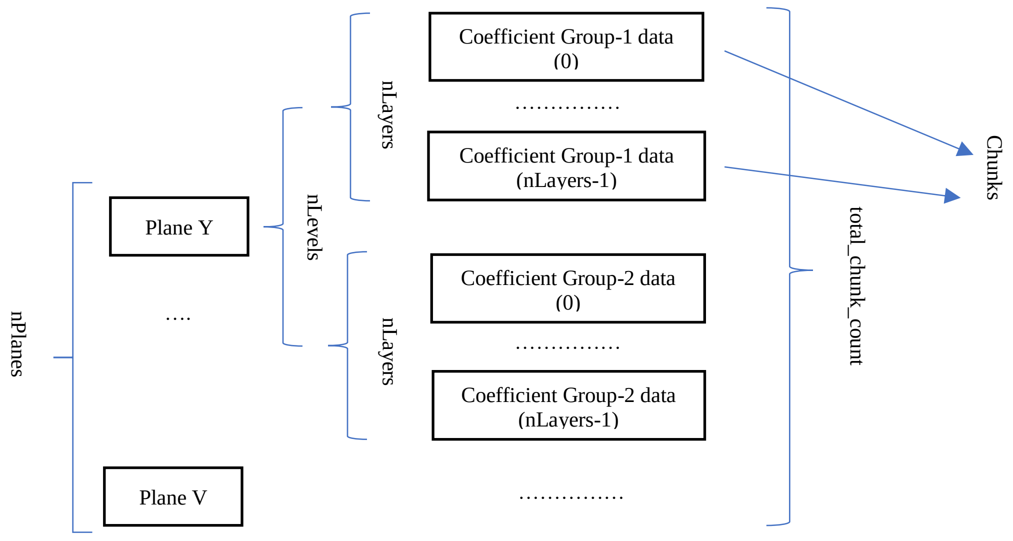
Coefficient Group-1 data (0)
……………
Coefficient Group-1 data (nLayers-1)
Coefficient Group-2 data (0)
……………
Coefficient Group-2 data (nLayers-1)
……………
Plane Y
….
Plane V
nPlanes
nLevels
nLayers
nLayers
total_chunk_count
Chunks

Figure 4 — Encoded enhancement picture data chunk structure

The enhancement picture data chunks are hierarchically organized as shown in Figure 4. For each plane, up to 2 enhancement sub-layers are extracted. For each enhancement sub-layer, up to 16 coefficient groups of transform coefficients can be extracted. Additionally, if temporal_signalling_present_flag is equal to 1, an additional chunk with temporal data for enhancement sub-layer 2 is extracted.

The variable levelIdx 1 refers to enhancement sub-layer 1 and levelIdx 2 refers to enhancement sub-layer 2.

The decoding process shall be the following:

1. For each chunk 2 bits at time shall be read.

2. surfaces[planeIdx][levelIdx][layerIdx].entropy_enabled_flag, surfaces[planeIdx][levelIdx][layerIdx].rle_only_flag, temporal_surfaces[planeIdx].entropy_enabled_flag and temporal_surfaces[planeIdx].rle_only_flag are derived as follows:

```
shift_size = −1
for (planeIdx = 0; planeIdx < nPlanes; ++planeIdx) {
    if (no_enhancement_bit_flag == 0) {
        for (levelIdx = 1; levelIdx <= nLevels; ++levelIdx) {
            for (layerIdx = 0; layer < nLayers; ++layerIdx) {
                if (shift_size < 0) {
                    data = read_byte(bitstream)
                    shift_size = 8 – 1
                }
                surfaces [planeIdx][levelIdx][layerIdx].entropy_enabled_flag = ((data >> shift_size) & 0x1)
                surfaces [planeIdx][levelIdx][layerIdx].rle_only_flag = ((data >> (shift_size − 1)) & 0x1)
                shift_size −= 2
            }
        }
    } else {
        for (layerIdx = 0; layer < nLayers; ++layerIdx)
            surfaces [planeIdx][levelIdx][layerIdx].entropy_enabled_flag = 0
    }
    if (temporal_signalling_present_flag == 1) {
        if (shift_size < 0) {
                data = read_byte(bitstream)
                shift_size = 8 – 1
            }
            temporal_surfaces[planeIdx].entropy_enabled_flag = ((data >> shift_size) & 0x1)
            temporal_surfaces[planeIdx].rle_only_flag = ((data >> (shift_size − 1)) & 0x1)
            shift_size −= 2
        }
    }
```

3. According to the values of the entropy_enabled_flag and rle_only_flag fields the content for the surfaces[planeIdx][levelIdx][layerIdx].data indicating the beginning of the RLE only or Prefix Coding and RLE encoded transform coefficients related to the specific chunk of data and if temporal_signalling present_flag is equal to 1, according to the values of the entropy_enabled_flag and rle_only_flag fields the content for the temporal_surfaces[planeIdx].data indicating the beginning of the RLE only or Prefix Coding and RLE encoded temporal signal coefficient group related to the specific chunk of data are derived as follows:

```
for (planeIdx = 0; planeIdx < nPlanes; ++planeIdx) {
    for (levelIdx = 1; levelIdx <= nLevels; ++levelIdx) {
        for (layerIdx = 0; layer < nLayers; ++layerIdx) {
            if (surfaces [planeIdx][levelIdx][layerIdx].entropy_enabled_flag) {
                if (surfaces [planeIdx][levelIdx][layerIdx].rle_only_flag) {
                    multibyte = read_multibyte(bitstream)
                    surfaces[planeIdx][levelIdx][layerIdx].size = multibyte
                    surfaces[planeIdx][levelIdx][layerIdx].data = bytestream_current(bitstream)
                } else {
                    data = read_byte(bitstream)
                    surfaces[planeIdx][levelIdx][layerIdx].prefix_last_symbol_bit_offset = (data&0x1F)
                    multibyte = read_multibyte(bitstream)
                    surfaces[planeIdx][levelIdx][layerIdx].size = multibyte
                    surfaces[planeIdx][levelIdx][layerIdx].data = bytestream_current(bitstream)
                    bytestream_seek(bitstream, surfaces[planeIdx][levelIdx][layerIdx].size)
                }
            }
        }
    }
    if (temporal_signalling_present_flag == 1) {
        if (temporal_surfaces[planeIdx].entropy_enabled_flag) {
            if (temporal_surfaces[planeIdx].rle_only_flag) {
                multibyte = read_multibyte(bitstream)
                temporal_surfaces[planeIdx].size = multibyte
                temporal_surfaces[planeIdx].data = bytestream_current(bitstream)
            } else {
                data = read_byte(bitstream)
                temporal_surfaces[planeIdx].prefix_last_symbol_bit_offset = (data&0x1F)
                multibyte = read_multibyte(bitstream)
                temporal_surfaces[planeIdx].size = multibyte
                temporal_surfaces[planeIdx].data = bytestream_current(bitstream)
                bytestream_seek(bitstream, temporal_surfaces[planeIdx].size)
            }
        }
    }
}
```

The transform coefficients contained in the block of bytes of length surfaces[planeIdx][levelIdx][layerIdx].size and starting from surfaces[planeIdx][levelIdx][layerIdx].data address are then passed to the entropy decoding process described in subclause 9.1.1.

If temporal_signalling_present_flag is set to 1, the temporal signal coefficient group contained in the block of bytes of length temporal_surfaces[planeIdx].size and starting from temporal_surfaces[planeIdx].data address are then passed to the entropy decoding process described in subclause 9.1.2.

8.3.3 Decoding process for picture enhancement encoded tiled data (payload_encoded_tiled_data)

Syntax of this process is described in subclause 7.3.8. Inputs to this process are:

- A variable nPlanes containing the number of planes (subclause 7.4.3.2 depending on the value of the variable processed_planes_type_flag),
- A variable nLayers (subclause 7.4.3.2 depending on the value of transform_type),
- A constant nLevels equal to 2, since there are 2 enhancement sub-layers,
- A variable nTilesL2, which equals to Ceil(Picture_Width / Tile_Width)xCeil(Picture_Height / Tile_Height) and refers to the number of tiles in L-2.
- A variable nTilesL1, which equals to: (a) nTilesL2 if the variable scaling_mode_level2 is equal to 0, (b) Ceil(Ceil(Picture_Width / 2) / Tile_Width)xCeil(Ceil(Picture_Height) / Tile_Height) if the variable scaling_mode_level2 is equal to 1, and (c) Ceil(Ceil(Picture_Width / 2) / Tile_Width)xCeil(Ceil(Picture_Height / 2) / Tile_Height) if the variable scaling_mode_level2 is equal to 2, and refers to the number of tiles in L-1.

Picture_Width and Picture_Height refer to the picture width and height as derived from the value of the variable resolution_type (subclause 7.4.3.2). Tile_Width and Tile_Height refer to the tile width and height as derived from the value of the variable tile_dimensions_type (subclause 7.4.3.1).

Output of this process is the (nPlanes)x(nLevels)x(nLayer) array surfaces with elements surfaces[nPlanes][nLevels][nLayer] and, if temporal_signalling_present_flag is set to 1, an additional temporal surface of a size nPlanes with elements temporal_surface[nPlanes].

- variable nPlanes is derived as follows:
```
if (processed_planes_type_flag == 0)
    nPlanes = 1
else
    nPlanes = 3
```
- variable nLayers is derived as follows:
```
if (transform_type == 0)
    nLayers = 4
else
    nLayers = 16
```

The encoded data is organized in chunks. The total number of chunks is calculated as following:

total_chunk_count = nPlanes * nLevels * nLayers * (nTilesL1 + nTilesL2) * (no_enhancement_bit_flag == 0) + nPlanes * nTilesL2 * (temporal_signalling_present_flag == 1) (13)

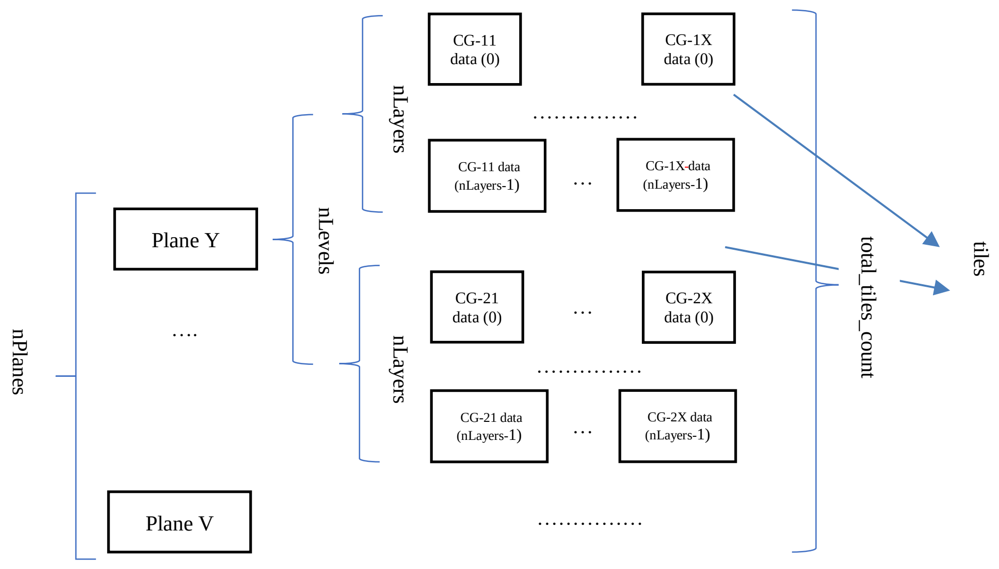

Figure 5 — Encoded chunk tile structure

The enhancement picture data chunks are hierarchically organized as shown in Figure 5. For each plane up to 2 layers of enhancement sub-layers are extracted. For each sub-layer of enhancement, up to 16 coefficient groups of transform coefficients can be extracted. Additionally, if temporal_signalling_present_flag is set to 1, an additional chunk with temporal data for enhancement sub-layer 2 is extracted.

The variable levelIdx 1 refers to enhancement sub-layer 1 and levelIdx 2 refers to enhancement sub-layer 2.

The decoding process shall be the following:

1. For each chunk 1 bit at time shall be read.

2. surfaces[planeIdx][levelIdx][layerIdx].rle_only_flag and, if temporal_signalling_present_flag is set to 1, temporal_surfaces[planeIdx].rle_only_flag are derived as follows:

```
shift_size = −1
for (planeIdx = 0; planeIdx < nPlanes; ++ planeIdx) {
    if (no_enhancement_bit_flag == 0) {
        for (levelIdx = 1; levelIdx <= nLevels; ++levelIdx) {
            for (layerIdx = 0; layer < nLayers; ++layerIdx) {
                if (shift_size < 0) {
                    data = read_byte(bitstream)
                    shift_size = 8 – 1
                }
                surfaces [planeIdx][levelIdx][layerIdx].rle_only_flag = ((data >> (shift_size − 1)) & 0x1)
                shift_size −= 1
            }
        }
    }
    if (temporal_signalling_present_flag == 1) {
        if (shift_size < 0) {
            data = read_byte(bitstream)
            shift_size = 8 – 1
        }
        temporal_surfaces[planeIdx].rle_only_flag = ((data >> (shift_size − 1)) & 0x1)
        shift_size −= 1
    }
}
```

3. surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].entropy_enabled_flag and, if temporal_signalling_present_flag is set to 1, temporal_surfaces[planeIdx].tiles[tileIdx].entropy_enabled_flag are derived as follows:

```
if (compression_type_entropy_enabled_per_tile_flag == 0) {
    shift_size = −1
    for (planeIdx = 0; planeIdx < nPlanes; ++planeIdx) {
        if (no_enhancement_bit_flag == 0) {
            for (levelIdx = 1; levelIdx <= nLevels; ++levelIdx) {
                if (levelIdx == 1)
                    nTiles = nTilesL1
                else
                    nTiles = nTilesL2
                for (layerIdx = 0; layer < nLayers; ++layerIdx) {
                    for (tileIdx=0; tileIdx < nTiles; tileIdx ++) {
                        if (shift_size < 0) {
                            data = read_byte(bitstream)
                            shift_size = 8 – 1
                        }
                        surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].entropy_enabled_flag =
                            ((data >> (shift_size − 1)) & 0x1)
                        shift_size −= 1
                    }
                }
            }
        } else {
            for (levelIdx = 1; levelIdx <= nLevels; ++levelIdx) {
                if (levelIdx == 1)
                    nTiles = nTilesL1
                else
                    nTiles = nTilesL2
                for (layerIdx = 0; layer < nLayers; ++layerIdx) {
                    for (tileIdx = 0; tileIdx < nTiles; tileIdx++)
                        surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].entropy_enabled_flag = 0
                }
            }
        }
        if (temporal_signalling_present_flag == 1) {
            for (tileIdx = 0; tileIdx < nTilesL2; tileIdx++) {
                if (shift_size < 0) {
                    data = read_byte(bitstream)
                    shift_size = 8 – 1
                }
                temporal_surfaces[planeIdx].tiles[tileIdx].entropy_enabled_flag =
                    ((data >> (shift_size − 1)) & 0x1)
                shift_size = 1
            }
        }
    }
} else {
    RLE decoding process as defined in subclause 9.3.5.
}
```

4. According to the value of the entropy_enabled_flag and rle_only_flag fields the content for the surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].data indicating the beginning of the RLE only or Prefix Coding and RLE encoded coefficients related to the specific chunk of data and, if temporal_signalling_present_flag is set to 1, according to the value of the entropy_enabled_flag and rle_only_flag fields the content for the temporal_surfaces[planeIdx].tiles[tileIdx].data indicating the beginning of the RLE only or Prefix Coding and RLE encoded temporal signal coefficient group related to the specific chunk of data are derived as follows:

```
if (compression_type_size_per_tile == 0) {
        for (planeIdx = 0; planeIdx < nPlanes; ++planeIdx) {
            for (levelIdx = 1; levelIdx <= nLevels; ++levelIdx) {
                if (levelIdx == 1)
                    nTiles = nTilesL1
                else
                    nTiles = nTilesL2
                for (layerIdx = 0; layer < nLayers; ++layerIdx) {
                    for (tileIdx = 0; tileIdx < nTiles; tileIdx++) {
                        if (surfaces [planeIdx][levelIdx][layerIdx].tiles[tileIdx].entropy_enabled_flag) {
                            if (surfaces [planeIdx][levelIdx][layerIdx].rle_only_flag) {
                                multibyte = read_multibyte(bitstream)
                                surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].size = multibyte
                                surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx]data =
                                    bytestream_current(bitstream)
                            } else {
                                data = read_byte(bitstream)
                                surfaces[planeIdx][levelIdx][layerIdx].
                                    tiles[tileIdx].prefix_last_symbol_bit_offset = (data&0x1F)
                                multibyte = read_multibyte(bitstream)
                                surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].size = multibyte
                                surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].data =
                                    bytestream_current(bitstream)
                                bytestream_seek(bitstream,
                                    surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].size)
                            }
                        }
                    }
                }
            }
            if (temporal_signalling_present_flag == 1) {
                for (tileIdx = 0; tileIdx < nTilesL2; tileIdx++) {
                    if (temporal_surfaces[planeIdx].tiles[tileIdx].entropy_enabled_flag) {
                        if (temporal_surfaces[planeIdx].rle_only_flag) {
                            multibyte = read_multibyte(bitstream)
                            temporal_surfaces[planeIdx].tiles[tileIdx].size = multibyte
                            temporal_surfaces[planeIdx].tiles[tileIdx].data = bytestream_current(bitstream)
                        } else {
                            data = read_byte(bitstream)
                            temporal_surfaces[planeIdx].tiles[tileIdx].
                                prefix_last_symbol_bit_offset = (data&0x1F)
                            multibyte = read_multibyte(bitstream)
                            temporal_surfaces[planeIdx].tiles[tileIdx].size = multibyte
                            temporal_surfaces[planeIdx].tiles[tileIdx].data = bytestream_current(bitstream)
                            bytestream_seek(bitstream, temporal_surfaces[planeIdx].tiles[tileIdx].size)
                        }
                    }
                }
            }
        }
    } else {
        for (planeIdx = 0; planeIdx < nPlanes; ++planeIdx) {
            for (levelIdx = 1; levelIdx <= nLevels; ++levelIdx) {
                if (levelIdx == 1)
                    nTiles = nTilesL1
                else
                    nTiles = nTilesL2
                for (layerIdx = 0; layer < nLayers; ++layerIdx) {
                    for (tileIdx = 0; tileIdx < nTiles; tileIdx ++) {
                        if (surfaces [planeIdx][levelIdx][layerIdx].tiles[tileIdx].entropy_enabled_flag) {
                            if (surfaces[planeIdx][levelIdx][layerIdx].rle_only_flag) {
```

```
                            Prefix Coding decoding process as defined in subclause 9.2.3 to fill
                                surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].size
                            surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].data =
                                bytestream_current(bitstream)
                        } else {
                            Prefix Coding decoding process as defined in subclause 9.2.4 to fill
                                surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].
                                prefix_last_symbol_bit_offset
                            Prefix Coding decoding process as defined in subclause 9.2.3 to fill
                                surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].size
                            surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].data =
                                bytestream_current(bitstream)
                            bytestream_seek(bitstream,surfaces[planeIdx][levelIdx][layerIdx].
                                tiles[tileIdx].size)
                        }
                    }
                }
            }
        }
        if (temporal_signalling_present_flag == 1) {
            for (tileIdx = 0; tileIdx < nTilesL2; tileIdx++) {
                if (temporal_surfaces [planeIdx].tiles[tileIdx].entropy_enabled_flag) {
                    if (temporal_surfaces [planeIdx].rle_only_flag) {
                        Prefix Coding decoding process as defined in subclause 9.2.3 to fill
                            temporal_surfaces[planeIdx].tiles[tileIdx].size
                        temporal_surfaces[planeIdx].tiles[tileIdx].data = bytestream_current(bitstream)
                    } else {
                        Prefix Coding decoding process as defined in subclause 9.2.4 to fill
                            temporal_surfaces[planeIdx].tiles[tileIdx].prefix_last_symbol_bit_offset
                        Prefix Coding decoding process as defined in subclause 9.2.3 to fill
                            temporal_surfaces[planeIdx].tiles[tileIdx].size
                        temporal_surfaces[planeIdx].tiles[tileIdx].data = bytestream_current(bitstream)
                        bytestream_seek(bitstream, temporal_surfaces[planeIdx].tiles[tileIdx].size)
                    }
                }
            }
        }
    }
}
```

The coefficients contained in the block of bytes of length surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].size and starting from surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].data address are then passed to the entropy decoding process described in subclause 9.1.1.

If temporal_signalling_enabled_flag is set to 1, the temporal signal coefficient group contained in the block of bytes of length temporal_surfaces[planeIdx].tiles[tileIdx].size and starting from temporal_surfaces[planeIdx].tiles[tileIdx].data address are then passed to the entropy decoding process described in subclause 9.1.2.

8.3.4 Decoding process for enhancement sub-layer 1 (L-1) encoded data

8.3.4.1 Derivation of dimensions of L-1 picture

The result of this process is the L-1 enhancement residual surface to be added to the preliminary intermediate picture. The L-1 dimensions of the residuals surface are the same as the preliminary intermediate picture and they are derived as follows:

- If scaling_mode_level2 (specified in subclause 7.3.3.1) is equal to 0:

  The L-1 dimensions shall be the same as the L-2 dimensions derived from resolution_type (specified in subclause 7.4.3.2).

- If scaling_mode_level2 (specified in subclause 7.4.3.2) is equal to 1:

  The L-1 length shall be the same as the L-2 length as derived from resolution_type (specified in subclause 7.4.3.2), whereas the L-1 width shall be computed by halving the L-2 width as derived from resolution_type (specified in subclause 7.4.3.2).

- If scaling_mode_level2 (specified in subclause 7.4.3.2) is equal to 2:

  The L-2 dimensions shall be computed by halving the L-1 dimensions as derived from resolution_type (specified in subclause 7.4.3.2).

8.3.4.2 General decoding process for an L-1 encoded data block

Inputs to this process are:

- a sample location (xTb0, yTb0) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture,
- a variable nTbS specifying the size of the current transform block derived in subclause 7.4.3.2 from the value of variable transform_type (nTbS = 2 if transform_type is equal to 0 and nTbS = 4 if transform_type is equal to 1)
- an array TransformCoeffQ of a size (nTbS)x(nTbS) specifying L-1 entropy decoded quantized transform coefficients,
- an array recL1BaseSamples of a size (nTbS)x(nTbS) specifying the preliminary intermediate picture reconstructed samples of the current block resulting from the process described in subclause 8.10,
- stepWidth value derived in subclause 7.3.3.2 from the value of variable step_width_level1 obtained by shifting step_width_level1 to the left by one bit (i.e., step_width_level1 << 1),
- a variable IdxPlanes specifing to which plane the transform coefficients belong,
- a variable userDataEnabled derived from the value of variable user_data_enabled,

Output of this process is the (nTbS)x(nTbS) array of the residual resL1FilteredResiduals with elements resL1FilteredResiduals[x][y].

The sample location (xTbP, yTbP) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture is derived as follows:

(xTbP, yTbP) = (IdxPlanes == 0) ? (xTb0, yTb0) : (xTb0 >> ShiftWidthC, yTb0 >> ShiftHeightC) (14)

P can be related to either luma or chroma plane depending to which plane the transform coefficients belong.Where ShiftWidthC and ShiftHeightC are specified in subclause 6.2.

If no_enhancement_bit_flag is equal to 0, then the following ordered steps apply:

1. If nTbs is equal to 4, TransCoeffQ(1)(1) is shifted to the right either by two bits (>>2) if the variable userDataEnabled is set to 1, or by six bits (>>6) if userDataEnabled is set to 2. And in addition, if the last bit of TransCoeffQ(1)(1) is set to 0, TransCoeffQ(1)(1) is shifted to the right by one bit (>>1); otherwise (last bit of TransCoeffQ(1)(1) is set to 1), TransCoeffQ(1)(1) is shifted to the right by one bit (>>1) and TransCoeffQ(1)(1) is set to have a negative value.
2. If nTbs is equal to 2, TransCoeffQ(0)(1) is shifted to the right either by two bits (>>2) if the variable userDataEnabled is set to 1, or by six bits (>>6) if userDataEnabled is set to 2. And in addition, if the last bit of TransCoeffQ(0)(1) is set to 0, TransCoeffQ(0)(1) is shifted to the right by one bit (>>1); otherwise (last bit of TransCoeffQ(0)(1) is set to 1), TransCoeffQ(0)(1) is shifted to the right by one bit (>>1) and TransCoeffQ(0)(1) is set to have a negative value.
3. The dequantization process as specified in subclause 8.5 is invoked with the transform size set equal to nTbS, the array TransformCoeffQ of a size (nTbS)x(nTbS), and the variable stepWidth as inputs, and the output is an (nTbS)x(nTbS) array dequantCoeff.

4. The transformation process as specified in subclause 8.6 is invoked with the luma location (xTbX, yTbX), the transform size set equal to nTbS, the array dequantCoeff of size (nTbS)x(nTbS) as inputs, and the output is an (nTbS)x(nTbS) array resL1Residuals.

5. The L1 filter process as specified in subclause 8.9 is invoked with the luma location (xTbX, yTbX), the array resL1Residuals of a size (nTbS)x(nTbS) as inputs, and the output is an (nTbS)x(nTbS) array resL1FilteredResiduals.

If no_enhancement_bit_flag is equal to 1, the array resL1FilteredResiduals of size (nTbS)x(nTbS) is set to contain only zeros.

The picture reconstruction process for each plane as specified in subclause 8.8.2 is invoked with the transform block location (xTb0, yTb0), the transform block size nTbS, the variable IdxPlanes, the (nTbS)x(nTbS) array resL1FilteredResiduals, and the (nTbS)x(nTbS) array recL1BaseSamples as inputs.

8.3.5 Decoding process for enhancement sub-layer 2 (L-2) encoded data

8.3.5.1 Derivation of dimensions of L-2 picture

The result of this process is the L-2 enhancement residuals plane to be added to the upscaled L-1 enhanced reconstructed picture. The dimensions of the residuals plane are the same of the value derived by the variable resolution_type described in clause. 7.4.3.3

8.3.5.2 General decoding process for an L-2 encoded data block

Inputs to this process are:

− a sample location (xTb0, yTb0) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture, a variable nTbS specifying the size of the current transform block derived in subclause 7.4.3.2 from the value of variable transform_type (nTbS = 2 if transform_type is equal to 0 and nTbS = 4 if transform_type is equal to 1)**,**

− a variable temporal_enabled_flag as derived in subclause 7.3.3.1 and a variable temporal_refresh_bit_flag as derived in subclause 7.4.3.4, a variable temporal_signalling_present_flag as derived in subclause 7.4.3.4 and temporal_step_width_modifier as specified in subclause 7.3.3.1

− an array recL0ModifiedUpsampledSamples of a size (nTbS)x(nTbS) specifying the upsampled reconstructed samples resulting from process specified in subclause 8.8.2 of the current block,

− an array TransformCoeffQ of a size (nTbS)x(nTbS) specifying L-2 entropy decoded quantized transform coefficient,

− if variable temporal_signalling_present_flag is equal to 1 and temporal_tile_intra_signalling_enabled_flag is equal to 1, a variable TransformTempSig corresponding to the value in TempSigSurface (subclause 9.3.3.1) at the position (xTb0 >> nTbs, yTb0 >> nTbs) ; and if in addition temporal_tile_intra_signalling_enabled_flag is set to 1, a variable TileTempSig corresponding to the value in TempSigSurface (subclause 9.3.3.1) at the position ((xTb0%32) * 32, (yTb0%32) * 32),

− stepWidth value derived in subclause 7.4.3.4 from the value of variable step_width_level2,

− a variable IdxPlanes specifing to which plane the transform coeffiencients are belonging to.

Output to this process is the (nTbS)x(nTbS) array of L-2 residuals resL0Residuals with elements resL0Residuals[x][y].

The sample location (xTbP, yTbP) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture is derived as follows:

(xTbP, yTbP) = (IdxPlanes == 0) ? (xTb0, yTb0) : (xTb0 >> ShiftWidthC, yTb0 >> ShiftHeightC) (15)

P can be related to either luma or chroma plane depending to which plane the transform coefficients belong. Where ShiftWidthC and ShiftHeightC are specified in subclause 6.2.

If no_enhancement_bit_flag is set to 0, then the following ordered steps apply:

1. If variable temporal_enabled_flag is equal to 1 and temporal_refresh_bit_flag is equal to 0 (subclause 7.4.3.4) the temporal prediction process as specified in clause 8.4 is invoked with the luma location (xTbY, yTbY), the transform size set equal to nTbS, a variable TransformTempSig and a variable TileTempSig as inputs and the output is an array tempPredL0Residuals of a size (nTbS)x(nTbS).

   If variable temporal_enabled_flag is equal to 1 and temporal_refresh_bit_flag is equal to 1 (subclause 7.4.3.4) the array tempPredL0Residuals of a size (nTbS)x(nTbS) is set to contain only zeros.

2. If variable temporal_enabled_flag is equal to 1, temporal_refresh_bit_flag is equal to 0 and temporal_tile_intra_signalling_enabled_flag is equal to 1 (subclause 7.4.3.4) and TransformTempSig is equal to 0 the variable stepWidth is modified to Floor(stepWidth * (1 − (Clip3(0, 0.5, temporal_step_width_modifier) / 255))).

3. The dequantization process as specified in subclause 8.5 is invoked with the transform size set equal to nTbS, the array TransformCoeffQ of a size (nTbS)x(nTbS), and the variable stepWidth as inputs, and the output is an (nTbS)x(nTbS) array dequantCoeff.

4. The transformation process as specified in subclause 8.6 is invoked with the luma location (xTbY, yTbY), the transform size set equal to nTbS, the array dequantCoeff of a size (nTbS)x(nTbS) as inputs, and the output is an (nTbS)x(nTbS) array resL0Residuals.

5. If variable temporal_enabled_flag is equal to 1 the array of tempPredL0Residuals of a size (nTbS)x(nTbS) is added to the (nTbS)x(nTbS) array resL0Residuals and resL0Residuals array is stored to the temporalBuffer at the luma location (xTbY, yTbY).

If no_enhancement_bit_flag is set to 1, the following ordered steps apply:

1. If variable temporal_enabled_flag is equal to 1, temporal_refresh_bit_flag is equal to 0 and variable temporal_signalling_present_flag is equal to 1 (subclause 7.4.3.4), the temporal prediction process as specified in clause 8.4 is invoked with the luma location (xTbY, yTbY), the transform size set equal to nTbS, a variable TransformTempSig and a variable TileTempSig as inputs and the output is an array tempPredL0Residuals of a size (nTbS)x(nTbS).

   If variable temporal_enabled_flag is equal to 1, temporal_refresh_bit_flag is equal to 0 and variable temporal_signalling_present_flag is equal to 0 (subclause 7.4.3.4), the temporal prediction process as specified in clause 8.4 is invoked with the luma location (xTbY, yTbY), the transform size set equal to nTbS, a variable TransformTempSig set equal to 0 and a variable TileTempSig set equal to 0 as inputs and the output is an array tempPredL0Residuals of a size (nTbS)x(nTbS).

   If variable temporal_enabled_flag is equal to 1 and temporal_refresh_bit_flag is equal to 1 (subclause 7.4.3.4) the array tempPredL0Residuals of a size (nTbS)x(nTbS) is set to contain only zeros.

2. If variable temporal_enabled_flag is equal to 1 the array of tempPredL0Residuals of a size (nTbS)x(nTbS) is stored in the (nTbS)x(nTbS) array resL0Residuals and resL0Residuals array is stored to the temporalBuffer at the luma location (xTbY, yTbY).
   Else, the array resL0Residuals of a size (nTbS)x(nTbS) is set to contain only zeros.

The picture reconstruction process for each plane as specified in subclause 8.8.2 is invoked with the transform block location (xTb0, yTb0), the transform block size nTbS, the variable IdxPlanes, the (nTbS)x(nTbS) array resL0Residuals, and the (xTbY)x(yTbY) recL0ModifiedUpsampledSamples as inputs.

8.4 Decoding process for the temporal prediction

8.4.1 General decoding process for temporal prediction

Inputs to this process are:

- a location (xTbP, yTbP) specifying the top-left sample of the current luma or chroma transform block relative to the top-left luma or chroma sample of the current picture. P can be related to either luma or chroma plane depending to which plane the transform coefficients belong,

- a variable nTbS specifying the size of the current transform block (nTbS = 2 if transform_type is equal to 0 and nTbS = 4 if transform_type is equal to 1),
- a variable TransformTempSig,
- a variable TileTempSig,

Output to this process is the (nTbS)x(nTbS) array of the tempPredL0Residuals with elements tempPredL0Residuals[x][y].

The following ordered steps apply:

1. If variable temporal_tile_intra_signalling is equal to 1 and xTbP >>5 is equal to 0 and yTbP >>5 is equal to 0 and TileTempSig is equal to 1, tiled temporal refresh process as specified in subclause 8.4.2 is invoked with the location (xTbP, yTbP).
2. If variable TransformTempSig is equal to 0, then tempPredL0Residuals[x][y] = temporalBuffer[xTbP + x][yTbP + y] where x and y are in the range [0, (nTbS − 1)]. Otherwise, tempPredL0Residuals[x][y] are all set to 0.

8.4.2 Tiled temporal refresh

Input to this process is:

- a location (xTbP, yTbP) specifying the top-left sample of the current luma or chroma transform block relative to the top-left luma or chroma sample of the current picture. P can be related to either luma or chroma plane depending to which plane the transform coefficients belong.

Output of this process is that the samples of the area of the size 32x32 of temporalBuffer at the location (xTbP, yTbP) are set to zero.

8.5 Decoding process for the dequantization

8.5.1 Decoding process for the dequantization overview

Every group of transform coefficient passed to this process belongs to a specific plane and enhancement sub-layer. They have been scaled using a uniform quantizer with deadzone. The quantizer can use a non-centered dequantization offset.

8.5.2 Scaling process for transform coefficients

Inputs to this process are:

- a variable nTbS specifying the size of the current transform block (nTbS = 2 if transform_type is equal to zero and nTbS = 4 if transform_type is equal to 1),
- an array TransformCoeffQ of size (nTbS)x(nTbS) containing entropy decoded quantized transform coefficient,
- a variable stepWidth specifying the step width value parameter,
- a variable levelIdx specifing the index of the enhancement sub-layer (with levelIdx = 1 for enhancement sub-layer 1 and levelIdx = 2 for enhancement sub-layer 2),
- a variable dQuantOffset specifying the dequantization offset (derived from process 7.4.3.4 and variable dequant_offset),
- if quant_matrix_mode is different from 0, an array QmCoeff0 of size 1 x $nTbS^2$ (derived from variable qm_coefficient_0) and further, if quant_matrix_mode is equal to 4, an array QmCoeff1 of size 1 x $nTbS^2$ (derived from variable qm_coefficient_1),
- if nTbS == 2, an array QuantScalerDDBuffer of size (3 * nTbS)x(nTbS) containing the scaling parameters array used in the previous picture;

- if nTbS == 4, an array QuantScalerDDSBuffer of size (3 * nTbS)x(nTbS) containing the scaling parameters array used in the previous picture.

Output of this process is the (nTbS)x(nTbS) array d of dequantized transform coefficients with elements d[x][y] and the updated array QuantMatrixBuffer.

For the derivation of the scaled transform coefficients d[x][y] with x = 0...nTbS − 1, y = 0...nTbS − 1, and given matrix qm[x][y] as specified in subclause 8.6.2, the following formula is used:

d[x][y] = (TransformCoeffQ[x][y] * ((qm[x + (levelIdxSwap * nTbS)][y] + stepWidthModifier[x][y]) + appliedOffset [x][y] (16)

8.5.3 Derivation of dequantization offset and stepwidth modifier

The variables appliedOffset [x][y] and stepWidthModifier [x][y] are derived as follows:

```
if (dequant_offset_signalled_flag == 0) {
    stepWidthModifier [x][y] = ((((Floor(−Cconst * Ln (qm[x + (levelIdxSwap * nTbS)][y]))) + Dconst) *
    (qm[x + (levelIdxSwap * nTbS)][y]$^2$))) / 32768) >> 16
    if (TransformCoeffQ[x][y] < 0)
        appliedOffset = (−1 * (−deadZoneWidthOffset [x][y]))
    else if (TransformCoeffQ [x][y] > 0)
         appliedOffset [x][y] = −deadZoneWidthOffset [x][y]
    else
        appliedOffset [x][y] = 0
} else if (dequant_offset_signalled_flag == 1) && (dequant_offset_mode_flag ==1) {
    stepWidthModifier [x][y] = 0
    if (TransformCoeffQ[x][y] < 0)
        appliedOffset = (−1 * (dQuantOffsetActual [x][y] − deadZoneWidthOffset [x][y]))
    else if (TransformCoeffQ [x][y] > 0)
        appliedOffset [x][y] = dQuantOffsetActual [x][y] − deadZoneWidthOffset [x][y]
    else
        appliedOffset [x][y] = 0}
} else if (dequant_offset_signalled_flag == 1) && (dequant_offset_mode_flag == 0) {
    stepWidthModifier [x][y] = (Floor((dQuantOffsetActual [x][y]) * (qm[x + (levelIdxSwap * nTbS)][y]))
        / 32768)
    if (TransformCoeffQ[x][y] < 0)
        appliedOffset = (−1 * (−deadZoneWidthOffset [x][y]))
    else if (TransformCoeffQ [x][y] > 0)
        appliedOffset [x][y] = −deadZoneWidthOffset [x][y]
    else
        appliedOffset [x][y] = 0
}
```

- Where, if stepWidth > 16, deadZoneWidthOffset is derived as follows:

deadZoneWidthOffset [x][y] = ((1 << 16) − ((Aconst * (qm[x + (levelIdxSwap * nTbs)][y] + stepWidthModifier [x][y])) + Bconst) >> 1) * (qm[x + (levelIdxSwap * nTbs)][y] + stepWidthModifier [x][y]))) >> 16

- Where, if stepWidth <= 16, deadZoneWidthOffset is derived as follows:

deadZoneWidthOffset [x][y] = stepWidth >> 1

- Where:

Aconst = 39
Bconst = 126484
Cconst = 9175
Dconst = 79953

- Where dQuantOffsetActual [x][y] is computed as follows:

```
if (dequant_offset == 0)
    dQuantOffsetActual [x][y] = dQuantOffset
else {
    if (dequant_offset_mode_flag == 1)
        dQuantOffsetActual [x][y] = ((Floor(−Cconst * Ln(qm[x + (levelIdxSwap * nTbs)][y]) +
            (dQuantOffset << 9) + Floor(Cconst * Ln(StepWidth)))) * (qm[x + (levelIdxSwap * nTbs)][y])) >> 16
    else if (dequant_offset_mode_flag == 0)
        dQuantOffsetActual [x][y] = ((Floor(−Cconst * Ln(qm[x + (levelIdxSwap * nTbs)][y]) +
            (dQuantOffset << 11) + Floor(Cconst * Ln(StepWidth)))) * (qm[x + (levelIdxSwap * nTbs)][y]))
            >>16
}
```

− Where levelIdxSwap is derived as follows:

```
if (levelIdx == 2)
    levelIdxSwap = 0
else
    levelIdxSwap = 1
```

8.5.4 Derivation of quantization matrix

8.5.4.1 The quantization matrix

The quantization matrix qm [x][y] contains the actual quantization step widths to be used to decode each coefficient group.

```
if (levelIdx == 2) {
    if (scaling_mode_level2 == 1) {
        for (x = 0; x < nTbS; x++) {
            for (y = 0; y < nTbs; y++)
                qm [x][y] = qm_p [x][y]
        }
    } else {
        for (x = 0; x < nTbS; x++) {
            for (y = 0; y < nTbS; y++)
                qm [x][y] = qm_p [x + nTbS][y]
        }
    }
} else {
    for (x = 0; x < nTbS; x++) {
        for (y = 0; y < nTbs; y++)
            qm [x][y] = qm_p [x + (2 * nTbS)][y]
    }
}
```

Where qm_p[x][y] is computed as follows:

```
if (nTbs == 2) {
    for (x = 0; x < 6; x++) {
        for (y = 0; y < nTbs; y++)
            qm_p[x][y] = (Clip3 (0, (3 << 16),[(QuantScalerDDBuffer [x][y] * stepWidth) + (1 << 16)]) *
                stepWidth) >> 16
    }
} else {
    for (y = 0; y < 12; y++) {
        for (x = 0; x < nTbs; x++)
            qm_p[x][y] = (Clip3 (0, (3 << 16),[(QuantScalerDDSBuffer [x][y] * stepWidth) + (1 << 16)]) *
            stepWidth) >> 16
    }
}
```

And where QuantScalerDDBuffer [x][y] is derived in sub-clause 8.6.2.1 and QuantScalerDDSBuffer [x][y] is derived in sub-clause 8.6.2.2.

8.5.4.2 Derivation of scaling parameters for 2x2 transform

If the variable nTbS is equal to 2, the default scaling parameters are as follows:

```
default_scaling_dd[x][y] =
    {
    {  0,  2 }
    {  0,  0 }
    { 32,  3 )
    {  0, 32 }
    {  0,  3 }
    {  0, 32 }
    }                                                        (17)
```

As a first step, the array QuantScalerDDBuffer[][] shall be initialized as follows:

If the current picture is an IDR picture, QuantScalerDDBuffer[][] shall be initialized to be equal to default_scaling_dd[][]. If the current picture is not an IDR picture, the QuantScalerDDBuffer[][] matrix shall be left unchanged.

Following initialization, based on the value of quant_matrix_mode the array QuantScalerDDBuffer[][] shall be processed as follows:

– If the quant_matrix_mode is equal to 0 and the current picture is not an IDR picture, the QuantScalerDDBuffer[][] shall be left unchanged.

– If quant_matrix_mode is equal to 1, the QuantScalerDDBuffer[][] shall be equal to the default_scaling_dd[][].

– If quant_matrix_mode is equal to 2, the QuantScalerDDBuffer[][] shall be modified as follows:

```
for (MIdx = 0; MIdx < 3; MIdx++)
    for (x = 0; x < 2; x++)
        for (y = 0; y < 2; y++)
            QuantScalerDDBuffer [x + (MIdx * 2)][y] = QmCoeff0[(x * 2) + y]
```

– If quant_matrix_mode is equal to 3, the QuantScalerDDBuffer[][] shall be modified as follows:

```
for (MIdx = 0; MIdx < 2; MIdx++)
    for (x = 0; x < 2; x++)
        for (y = 0; y < 2; y++)
            QuantScalerDDBuffer [x + (MIdx * 2)][y] = QmCoeff0 [(x * 2) + y]
```

– If quant_matrix_mode is equal to 4, the QuantScalerDDBuffer[][] shall be modified as follows:

```
for (x = 0; x < 2; x++)
    for (y = 0; y < 2; y++)
        QuantScalerDDBuffer [x + 4][y] = QmCoeff1 [(x * 2) + y]
```

– If quant_matrix_mode is equal to 5, the QuantScalerDDBuffer shall be modified as follows:

```
for (MIdx = 0; MIdx < 2; MIdx ++)
    for (x = 0; x < 2; x++)
        for (y = 0; y < 2; y++)
            QuantScalerDDBuffer [x + (MIdx * 2)][y] = QmCoeff0[(x * 2) + y]
    for (x = 4, x< 6; x++)
        for (y = 0; y < 2; y++)
            QuantScalerDDBuffer [y][x] = QmCoeff1[(x * 2) + y]
```

Derivation of scaling parameters for 4x4 transform

If the variable nTbS is equal to 4, the default scaling parameters are as follows:

default_scaling_dds[x][y] =

```
{
{  13, 26, 19, 32 }
{  52,  1, 78,  9 }
{  13, 26, 19, 32 }
{ 150, 91, 91, 19 }
{  13, 26, 19, 32 }
{  52,  1, 78,  9 }
{  26, 72,  0,  3 }
{ 150, 91, 91, 19 }
{   0,  0,  0,  2 }
{  52,  1, 78,  9 }
{  26, 72,  0,  3 }
{ 150, 91, 91, 19 }
}
```
(18)

As a first step, the array QuantScalerDDSBuffer[][] shall be initialized as follows:

– If the current picture is an IDR picture, QuantScalerDDSBuffer[][] shall be initialized to be equal to default_scaling_dds[][]. If the current picture is not an IDR picture, the QuantScalerDDSBuffer[][] matrix shall be left unchanged.

Following initialization, based on the value of quant_matrix_mode the array QuantScalerDDSBuffer[][] shall be processed as follows:

– If the quant_matrix_mode is equal to 0 and the current picture is not an IDR picture, the QuantScalerDDSBuffer shall be left unchanged.

– If quant_matrix_mode is equal to 1, the QuantScalerDDSBuffer shall be equal to the default_scaling_dds[x][y].

– If quant_matrix_mode is equal to 2, the QuantScalerDDSBuffer shall be modified as follows:

```
for (MIdx = 0; MIdx < 3; MIdx++)
    for (x = 0; x < 4; x++)
        for (y = 0; y < 4; y++)
            QuantScalerDDSBuffer [x + (MIdx * 4)][y] = QmCoeff0[(x * 4) + y]
```

– If quant_matrix_mode is equal to 3, the QuantScalerDDSBuffer shall be modified as follows:

```
for (MIdx = 0; MIdx < 2; MIdx++)
    for (x = 0; x < 4; x++)
        for (y = 0; y < 4; y++)
            QuantScalerDDSBuffer [x + (MIdx * 4)][y] = QmCoeff0[(x * 4) + y]
```

– If quant_matrix_mode is equal to 4, the QuantScalerDDSBuffer shall be modified as follows:

```
for (x = 0; x < 4; x++)
    for (y = 0; y < 4; y++)
        QuantScalerDDSBuffer [x + 8][y] = QmCoeff1[(x * 4) + y]
```

– If quant_matrix_mode is equal to 5, the QuantScalerDDSBuffer shall be modified as follows:

```
for (MIdx = 0; MIdx < 2; MIdx++)
    for (x = 0; x < 4; x++)
        for (y = 0; y < 4; y++)
            QuantScalerDDSBuffer [x + (MIdx * 4)][y] = QmCoeff0[(x * 4) + y]
    for (x = 8, x < 12; x++)
        for (y = 0; y < 4; y++)
```

QuantScalerDDSBuffer [x][y] = qm_coefficient_1[(x * 4) + y]

8.6 Decoding process for the transform

8.6.1 General upscaling process description

8.6.1.1 Upscaling processes overview

Upscaling processes are applied to the decoded base picture based on the indications of scaling_mode_level1 and to the combined intermediate picture based on scaling_mode_level2.

8.6.1.2 Upscaling from decoded base picture to preliminary intermediate picture

Inputs to this process are the following:

− a location (xCurr, yCurr) specifying the top-left sample of the current block relative to the top-left sample of the current picture component,

− a variable IdxPlanes specifying the colour component of the current block,

− a variable nCurrS specifing the size of the residual block,

− an (nCurrS)x(nCurrS) array recDecodedBaseSamples specifying the decoded base samples of the current block,

− a variable srcWidth and srcHeight specifying the width and the height of the decoded base picture,

− a variable dstWidth and dstHeight specifying the with and the height of the resulting upscaled picture,

− a variable is8Bit used to select the kernel coefficients for the scaling to be applied. If the samples are 8-bit, then variable is8Bit shall be equal to 0, if the samples are 16-bit, then variable is8Bit shall be equal to 1.

Output of this process is the (nCurrX)x(nCurrY) array recL1ModifiedUpsampledBaseSamples of residuals with elements recL1ModifiedUpsampledBaseSamples [x][y] where nCurrX and nCurrY are derived as follows:

− If scaling_mode_level1 (specified in subclause 7.4.3.3) is equal to 0, no upscaling is performed, and recL1ModifiedUpsampledBaseSamples [x][y] are set to be equal to recDecodedBaseSamples [x][y]

− If scaling_mode_level1 (specified in subclause7.4.3.3) is equal to 1:

$$nCurrX = nCurrS << 1 \quad (19)$$

$$nCurrY = nCurrS \quad (20)$$

− If scaling_mode_level1 (specified in subclause 7.4.3.2) is equal to 2:

$$nCurrX = nCurrS << 1 \quad (21)$$

$$nCurrY = nCurrS << 1 \quad (22)$$

The result of the process described in subclause 8.8.2 shall be processed by an upscaling filter of the type signalled in the bitstream. The type of upscaler is derived from the process described in subclause 7.4.3.3. Depending on the value of the variable upsample_type the following kernel types will be selected, providing recDecodedBaseSamples as input and producing recL1UpsampledBaseSamples as output:

− If upsample_type is equal to 0 the Nearest sample upscaler described in subclause 8.7.1 shall be selected.

− If upsample_type is equal to 1 the Bilinear upscaler described in subclause 8.7.2 shall be selected.

− If upsample_type is equal to 2 the Bicubic upscaler described in subclause 8.7.3 shall be selected.

- If upsample_type is equal to 3 the Modified Cubic upscaler described in subclause 8.7.4 shall be selected.

The general upscaler divide the picture to upscale in 2 areas: center area and border areas. For the Bilinear and Bicubic kernel the border area consists of four segments, Top, Left, Right and Bottom segments, while for the Nearest kernel consists of 2 segments; Right and Bottom. These segments are defined by the border-size parameter which is usually set to 2 samples (1 sample for nearest method). Figure 6 is describing the upscaler areas.

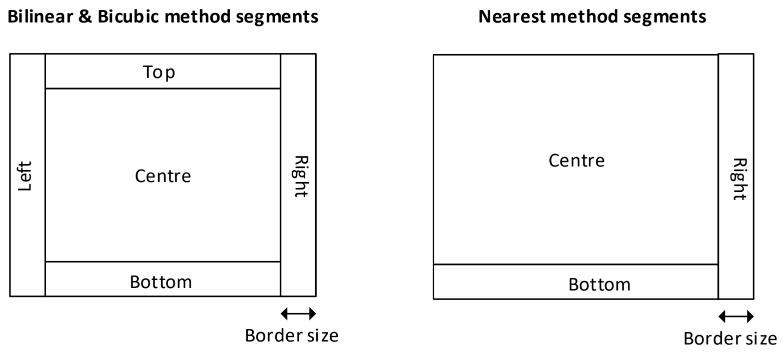

Figure 6 — Diagram describing upscaler areas

Following the upscaling, if predicted_residual_mode_flag as derived in subclause 7.3.3.1 is equal to 1, then the process described in subclause 8.7.5 is invoked with inputs the (nCurrX)x(nCurrY) array recL1UpsampledBaseSamples and the (nCurrS)x(nCurrS) array recDecodedBaseSamples specifying the decoded base samples of the current block and output is the (nCurrX)x(nCurrY) array recL1ModifiedUpsampledBaseSamples of preliminary intermediate picture samples with elements recL1ModifiedUpsampledBaseSamples [x][y]. Otherwise, if predicted_residual_mode_flag as derived in subclause 7.3.3.1 is equal to 0, recL1ModifiedUpsampledBaseSamples [x][y] are set to be equal to recL1UpsampledSamples [x][y].

Depending on the values of the bitstream fields in global configuration as specified in 7.3.5, Table 9, the sample bit depth for L-1 is derived as follows:

If level1_depth_flag is equal to 0, the preliminary intermediate picture samples are processed at the same bit depth as they are represented for the decoded base picture, following the processes described in clauses 8.8 and 8.9.

If level1_depth_flag is equal to 1, the preliminary intermediate picture samples are converted depending on the value of a variable base_depth and a variable enhancement_depth, dervied as follows:

base_depth is assigned a value between 8 and 14, depending on the value of field base_depth_type as specified in Table 23;

enhancement_depth is assigned a value between 8 and 14, depending on the value of field base_depth_type as specified in Table 24;

If base_depth is equal to enhancement_depth, no further processing is required.

If enhancement_depth is greater than base_depth, the array recL1ModifiedUpsampledBaseSamples is modified as follows:

recL1ModifiedUpsampledBaseSamples [x][y] =

recL1ModifiedUpsampledBaseSamples [x][y] << (enhancement_depth - base_depth)

If base_depth is greater than enhancement_depth, the array recL1ModifiedUpsampledBaseSamples is modified as follows:

recL1ModifiedUpsampledBaseSamples [x][y] =

recL1ModifiedUpsampledBaseSamples [x][y] >> (base_depth - enhancement_depth)

8.6.1.3 Upscaling from combined intermediate picture to preliminary output picture

Inputs to this process are the following:

- a location (xCurr, yCurr) specifying the top-left sample of the current block relative to the top-left sample of the current picture component,
- a variable IdxPlanes specifying the colour component of the current block,
- a variable nCurrS specifing the size of the residual block,
- an (nCurrS)x(nCurrS) array recL1PictureSamples specifying the combined intermediate picture samples of the current block,
- a variable srcWidth and srcHeight specifying the width and the height of the reconstructed base picture,
- a variable dstWidth and dstHeight specifying the with and the height of the resulting upscaled picture,
- a variable is8Bit used to select the kernel coefficients for the scaling to be applied. If the samples are 8-bit, then variable is8Bit shall be equal to 0, if the samples are 16-bit, then variable is8Bit shall be equal to 1.

Output of this process is the (nCurrX)x(nCurrY) array recL0ModifiedUpsampledSamples of preliminary output picture samples with elements recL0ModifiedUpsampledSamples [x][y] where nCurrX and nCurrY are derived as follows:

- If scaling_mode_level2 (specified in subclause 7.3.3.1) is equal to 0, no upscaling is performed, and recL0ModifiedUpsampledSamples [x][y] are set to be equal to recL1PictureSamples [x][y]
- If scaling_mode_level2 (specified in subclause 7.4.3.2) is equal to 1:

  nCurrX = nCurrS << 1 (23)

  nCurrY = nCurrS (24)

- If scaling_mode_level2 (specified in subclause 7.4.3.2) is equal to 2:

  nCurrX = nCurrS << 1 (25)

  nCurrY = nCurrS << 1 (26)

The result of the process described in subclause 8.8.2 shall be processed by an upscaling filter of the type signalled in the bitstream. The type of upscaler is derived from the process described in subclause 7.4.3.3. Depending on the value of the variable upsample_type the following kernel types will be selected, providing recL1PictureSamples as input and producing recL0UpsampledSamples as output:

- If upsample_type is equal to 0 the Nearest sample upscaler described in subclause 8.7.1 shall be selected.
- If upsample_type is equal to 1 the Bilinear upscaler described in subclause 8.7.2 shall be selected.
- If upsample_type is equal to 2 the Bicubic upscaler described in subclause 8.7.3 shall be selected.
- If upsample_type is equal to 3 the Modified Cubic upscaler described in subclause 8.7.4 shall be selected.

The general upscaler divide the picture to upscale in 2 areas: center area and border areas. For the Bilinear and Bicubic kernel the border area consists of four segments, Top, Left, Right and Bottom segments, while for the Nearest kernel consists of 2 segments; Right and Bottom. These segments are defined by the border-size parameter which is usually set to 2 samples (1 sample for nearest method). Figure 7 is describing the upscaler areas.

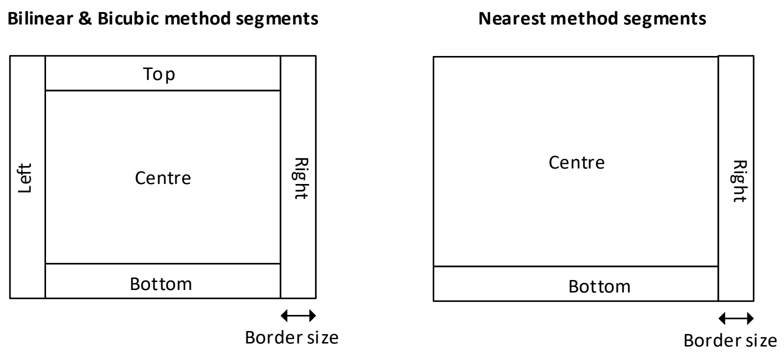

Figure 7 — Diagram describing upsampler areas

Following the upscaling, if predicted_residual_mode_flag as derived in subclause 7.3.3.1 is equal to 1 process described in subclause 8.7.5 is invoked with inputs the (nCurrX)x(nCurrY) array recL0UpsampledSamples and the (nCurrS)x(nCurrS) array recL1PictureSamples specifying the combined intermediate picture samples of the current block and output is the (nCurrX)x(nCurrY) array recL0ModifiedUpsampledSamples of residuals with elements recModifiedUpsampledL0Samples [x][y]. Otherwise, if predicted_residual_mode_flag as derived in subclause 7.3.3.1 is equal to 0, recL0ModifiedUpsampledSamples [x][y] are set to be equal to recL0UpsampledSamples [x][y].

Depending on the values of the bitstream fields in global configuration as specified in 7.3.5, Table 9, the sample bit depth for L-2 is derived as follows:

If level1_depth_flag is equal to 1, the preliminary output picture samples are processed at the same bit depth as they are represented for the preliminary intermediate picture, following the processes described in clauses 8.8 and 8.9.

If level1_depth_flag is equal to 0, the output intermediate picture samples are converted depending on the value of a variable base_depth and a variable enhancement_depth, dervied as follows:

base_depth is assigned a value between 8 and 14, depending on the value of field base_depth_type as specified in Table 23;

enhancement_depth is assigned a value between 8 and 14, depending on the value of field base_depth_type as specified in Table 24;

If base_depth is equal to enhancement_depth, no further processing is required.

If enhancement_depth is greater than base_depth, the array recL0ModifiedUpsampledSamples is modified as follows:

recL0ModifiedUpsampledSamples [x][y] =

recL0ModifiedUpsampledSamples [x][y] << (enhancement_depth - base_depth)

If base_depth is greater than enhancement_depth, the array recL0ModifiedUpsampledSamples is modified as follows:

recL0ModifiedUpsampledSamples [x][y] =

recL0ModifiedUpsampledSamples [x][y] >> (base_depth - enhancement_depth)

8.6.2 Transform inputs and outputs, transform types, and residual samples derivation

Inputs to this process are:

- a location (xTbP, yTbP) specifying the top-left sample of the current luma or chroma transform block relative to the top-left luma or chroma sample of the current picture. P can be related to either luma or chroma plane depending to which plane the transform coefficients belong,
- a variable nTbS specifying the size of the current transform block (nTbS = 2 if transform_type is equal to zero and nTbS = 4 if transform_type is equal to 1),
- an (nTbS)x(nTbS) array d of dequantized transform coefficients with elements d[x][y].

Output of this process is the (nTbS)x(nTbS) array R of residuals with elements R[x][y].

There are two types of transforms that can be used in the encoding process. Both leverage small kernels which are applied directly to the residuals that remain after the stage of applying Predicted Residuals (PRs). Figure 8 is a representation of how the residuals are transformed.

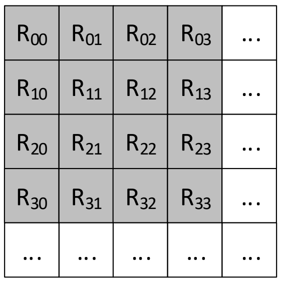
R_{00} R_{01} R_{02} R_{03} …
R_{10} R_{11} R_{12} R_{13} …
R_{20} R_{21} R_{22} R_{23} …
R_{30} R_{31} R_{32} R_{33} …
… … … … …

Figure 8 — Representation of how residuals are transformed

The (nTbS)x(nTbS) array R of residual samples is derived as follows:

Each (vertical) column of dequantized transform coefficients d[x][y] with x = 0…nTbS − 1, y = 0...nTbS − 1 is transformed to R[x][y] with x = 0...nTbS − 1, y = 0...nTbS − 1 by invoking the two-dimensional transformation process as specified in subclause 8.6.3 if nTbS is equal to 2

Or

Each (vertical) column of dequantized transform coefficients d[x][y] with x = 0...nTbS − 1, y = 0...nTbS − 1 is transformed to R[x][y] with x = 0...nTbS − 1, y = 0...nTbS − 1 by invoking the two-dimensional transformation process as specified in subclause 8.6.4 if nTbS is equal to 4.

8.6.3 2x2 directional decomposition transform

If nTbS is equal to 2 the transform has a 2x2 kernel which is applied to each 2x2 block of transform coefficients. The resulting residuals are derived as follows:

- If scaling_mode_levelX (specified in subclause 7.4.3.3) for the corresponding enhancement sub-layer is equal to 0 or 2:

$$\begin{Bmatrix} R_{00} \\ R_{01} \\ R_{10} \\ R_{11} \end{Bmatrix} = \begin{Bmatrix} 1, & 1, & 1, & 1 \\ 1, & -1, & 1, & -1 \\ 1, & 1, & -1, & -1 \\ 1, & -1, & -1, & 1 \end{Bmatrix} * \begin{Bmatrix} C_{00} \\ C_{01} \\ C_{10} \\ C_{11} \end{Bmatrix} \qquad (27)$$

- If scaling_mode_levelX (specified in subclause 7.4.3.3) for the corresponding enhancement sub-layer is equal to 1:

$$\begin{Bmatrix} R_{00} \\ R_{01} \\ R_{10} \\ R_{11} \end{Bmatrix} = \begin{Bmatrix} 1, & 1, & 1, & 0 \\ 1, & -1, & -1, & 0 \\ 0, & 1, & -1, & 1 \\ 0, & -1, & -1, & 1 \end{Bmatrix} * \begin{Bmatrix} C_{00} \\ C_{01} \\ C_{10} \\ C_{11} \end{Bmatrix} \qquad (28)$$

8.6.4 4x4 directional decomposition transform

If nTbS is equal to 4 the transform has a 4x4 kernel which is applied to a 4x4 block of transform coefficients. The resulting residuals are derived as follows:

- If scaling_mode_levelX (specified in subclause 7.4.3.3) for the corresponding enhancement sub-layer is equal to 0 or 2:

$$\begin{Bmatrix} R_{00} \\ R_{01} \\ R_{02} \\ R_{03} \\ R_{10} \\ R_{11} \\ R_{12} \\ R_{13} \\ R_{20} \\ R_{21} \\ R_{22} \\ R_{23} \\ R_{30} \\ R_{31} \\ R_{32} \\ R_{33} \end{Bmatrix} = \begin{Bmatrix}
1, & 1, & 1, & 1, & 1, & 1, & 1, & 1, & 1, & 1, & 1, & 1, & 1, & 1, & 1, & 1 \\
1, & 1, & -1, & -1, & 1, & 1, & -1, & -1, & 1, & 1, & -1, & -1, & 1, & 1, & -1, & -1 \\
1, & -1, & 1, & -1, & 1, & -1, & 1, & -1, & 1, & -1, & 1, & -1, & 1, & -1, & 1, & -1 \\
1, & -1, & -1, & 1, & 1, & -1, & -1, & 1, & 1, & -1, & -1, & 1, & 1, & -1, & -1, & 1 \\
1, & 1, & 1, & 1, & 1, & 1, & 1, & 1, & -1, & -1, & -1, & -1, & -1, & -1, & -1, & -1 \\
1, & 1, & -1, & -1, & 1, & 1, & -1, & -1, & -1, & -1, & 1, & 1, & -1, & -1, & 1, & 1 \\
1, & -1, & 1, & -1, & 1, & -1, & 1, & -1, & -1, & 1, & -1, & 1, & -1, & 1, & -1, & 1 \\
1, & -1, & -1, & 1, & 1, & -1, & -1, & 1, & -1, & 1, & 1, & -1, & -1, & 1, & 1, & -1 \\
1, & 1, & 1, & 1, & -1, & -1, & -1, & -1, & 1, & 1, & 1, & 1, & -1, & -1, & -1, & -1 \\
1, & 1, & -1, & -1, & -1, & -1, & 1, & 1, & 1, & 1, & -1, & -1, & -1, & -1, & 1, & 1 \\
1, & -1, & 1, & -1, & -1, & 1, & -1, & 1, & 1, & -1, & 1, & -1, & -1, & 1, & -1, & 1 \\
1, & -1, & -1, & 1, & -1, & 1, & 1, & -1, & 1, & -1, & -1, & 1, & -1, & 1, & 1, & -1 \\
1, & 1, & 1, & 1, & -1, & -1, & -1, & -1, & -1, & -1, & -1, & -1, & 1, & 1, & 1, & 1 \\
1, & 1, & -1, & -1, & -1, & -1, & 1, & 1, & -1, & -1, & 1, & 1, & 1, & 1, & -1, & -1 \\
1, & -1, & 1, & -1, & -1, & 1, & -1, & 1, & -1, & 1, & -1, & 1, & 1, & -1, & 1, & -1 \\
1, & -1, & -1, & 1, & -1, & 1, & 1, & -1, & -1, & 1, & 1, & -1, & 1, & -1, & -1, & 1
\end{Bmatrix} * \begin{Bmatrix} C_{00} \\ C_{01} \\ C_{02} \\ C_{03} \\ C_{10} \\ C_{11} \\ C_{12} \\ C_{13} \\ C_{20} \\ C_{21} \\ C_{22} \\ C_{23} \\ C_{30} \\ C_{31} \\ C_{32} \\ C_{33} \end{Bmatrix} \qquad (29)$$

– If scaling_mode_levelX (specified in subclause 7.4.3.3) for the corresponding enhancement sub-layer is equal to 1:

$$\begin{Bmatrix} R_{00} \\ R_{01} \\ R_{02} \\ R_{03} \\ R_{10} \\ R_{11} \\ R_{12} \\ R_{13} \\ R_{20} \\ R_{21} \\ R_{22} \\ R_{23} \\ R_{30} \\ R_{31} \\ R_{32} \\ R_{33} \end{Bmatrix} = \begin{Bmatrix} 1, & 1, & 1, & 1, & 1, & 1, & 1, & 1, & 0, & 0, & 1, & 1, & 0, & 0, & 1, & 1 \\ 1, & 1, & -1, & -1, & 1, & 1, & -1, & -1, & 0, & 0, & -1, & -1, & 0, & 0, & -1, & -1 \\ 1, & -1, & 1, & -1, & 1, & -1, & 1, & -1, & 0, & 0, & 1, & -1, & 0, & 0, & 1, & -1 \\ 1, & -1, & -1, & 1, & 1, & -1, & -1, & 1, & 0, & 0, & -1, & 1, & 0, & 0, & -1, & 1 \\ 0, & 0, & 1, & 1, & 0, & 0, & 1, & 1, & 1, & 1, & -1, & -1, & 1, & 1, & -1, & -1 \\ 0, & 0, & -1, & -1, & 0, & 0, & -1, & -1, & 1, & 1, & 1, & 1, & 1, & 1, & 1, & 1 \\ 0, & 0, & 1, & -1, & 0, & 0, & 1, & -1, & 1, & -1, & -1, & 1, & 1, & -1, & -1, & 1 \\ 0, & 0, & -1, & 1, & 0, & 0, & -1, & 1, & 1, & -1, & 1, & -1, & 1, & -1, & 1, & -1 \\ 1, & 1, & 1, & 1, & -1, & -1, & -1, & -1, & 0, & 0, & 1, & 1, & 0, & 0, & -1, & -1 \\ 1, & 1, & -1, & -1, & -1, & -1, & 1, & 1, & 0, & 0, & -1, & -1, & 0, & 0, & 1, & 1 \\ 1, & -1, & 1, & -1, & -1, & 1, & -1, & 1, & 0, & 0, & 1, & -1, & 0, & 0, & -1, & 1 \\ 1, & -1, & -1, & 1, & -1, & 1, & 1, & -1, & 0, & 0, & -1, & 1, & 0, & 0, & 1, & -1 \\ 0, & 0, & 1, & 1, & 0, & 0, & -1, & -1, & 1, & 1, & -1, & -1, & -1, & -1, & 1, & 1 \\ 0, & 0, & -1, & -1, & 0, & 0, & 1, & 1, & 1, & 1, & 1, & 1, & -1, & -1, & -1, & -1 \\ 0, & 0, & 1, & -1, & 0, & 0, & -1, & 1, & 1, & -1, & -1, & 1, & -1, & 1, & 1, & -1 \\ 0, & 0, & -1, & 1, & 0, & 0, & 1, & -1, & 1, & -1, & 1, & -1, & -1, & 1, & -1, & 1 \end{Bmatrix} * \begin{Bmatrix} C_{01} \\ C_{02} \\ C_{03} \\ C_{04} \\ C_{10} \\ C_{11} \\ C_{12} \\ C_{13} \\ C_{20} \\ C_{21} \\ C_{22} \\ C_{23} \\ C_{30} \\ C_{31} \\ C_{32} \\ C_{33} \end{Bmatrix} \tag{30}$$

8.7 Decoding process for the upscaling

8.7.1 Nearest sample upsampler kernel description

Inputs to this process are the following:

– variables srcX and srcY specifying the width and the height of the input array,

– variables dstX and dstY specifying the width and the height of the output array,

a (srcX)x(srcY) array recInputSamples [x][y] of input samples,Outputs to this process are the following:

– a (dstX)x(dstY) array recUpsampledSamples [x][y] of output samples,

The Nearest kernel performs upscaling by copying the current source sample onto the destination 2x2 grid. This is shown in Figure 9. The destination sample positions are calculated by doubling the index of the source sample on both axes and adding +1 to extend the range to cover 4 samples as shown in Figure 9.

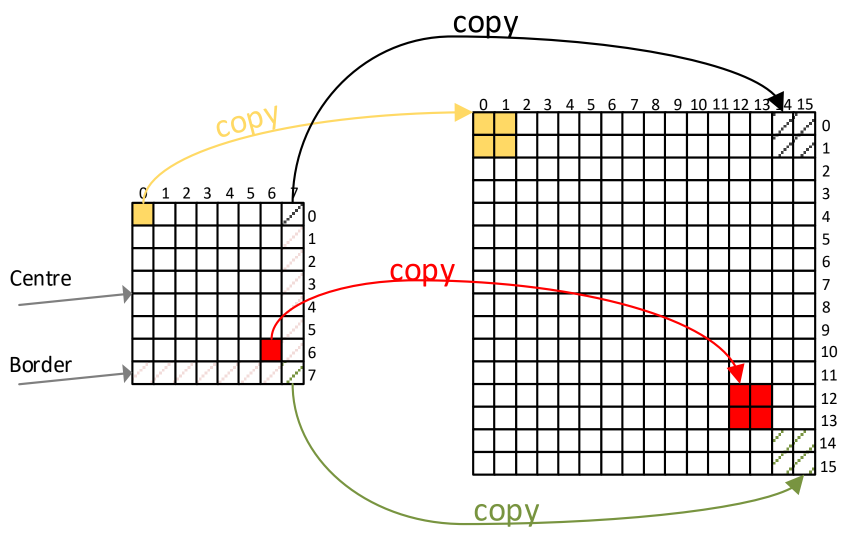

Figure 9 — Nearest upsample kernel

The nearest sample kernel upscaler is applied as specified by the following ordered steps whenever (xCurr, yCurr) block belongs to the picture or to the the border area as specified in Figure 9.

− If scaling_mode_levelX (specified in subclause 7.4.3.3) is equal to 1:

```
for (ySrc = 0; ySrc < nCurrS; ++ySrc)
    yDst = ySrc
    for (xSrc = 0; xSrc < nCurrS; ++xSrc)
        xDst = xSrc << 1
        recUpsampledSamples[xDst][yDst] = recInputSamples[xSrc][ySrc]
        recUpsampledSamples[xDst + 1][yDst] = recInputSamples[xSrc][ySrc]
```

− If scaling_mode_levelX (specified in subclause 7.4.3.3) is equal to 2:

```
for (ySrc = 0; ySrc < nCurrS; ++ySrc)
    yDst = ySrc << 1
    for (xSrc = 0; xSrc < nCurrS; ++xSrc)
        xDst = xSrc << 1
        recUpsampledSamples[xDst][yDst] = recInputSamples[xSrc][ySrc]
        recUpsampledSamples[xDst][yDst + 1] = recInputSamples[xSrc][ySrc]
        recUpsampledSamples[xDst + 1][yDst] = recInputSamples[xSrc][ySrc]
        recUpsampledSamples[xDst + 1][yDst + 1] = recInputSamples[xSrc][ySrc]
```

8.7.2 Bilinear upsampler kernel description

8.7.2.1 Bilinear upsampler kernel process inputs and outputs, process overview

Inputs to this process are the following:

− variables srcX and srcY specifying the width and the height of the input array,

− variables dstX and dstY specifying the width and the height of the output array,

− a (srcX)x(srcY) array recInputSamples [x][y] of input samples.

Outputs to this process are the following:

− a (dstX)x(dstY) array recUpsampledSamples [x][y] of output samples.

The Bilinear upsampling kernel consists of three main steps. The first step involves constructing a 2x2 grid of source samples with the base sample positioned at the bottom right corner. The second step involves performing the bilinear interpolation and the third step involves writing the interpolation result to the destination samples.

8.7.2.2 Source sample grid

The bilinear method performs the upsampling by considering the values of the nearest 3 samples to the base sample. The base sample is the source sample from which the address of the destination sample is derived. Figure 10 shows the source grid used in the kernel.

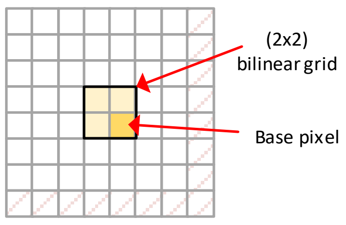

Figure 10 — Example of bilinear 2x2 source grid

8.7.2.3 Bilinear interpolation

The bilinear interpolation is a weighted summation of all the samples in the source grid. The weights employed are dependent on the destination sample being derived. The algorithm applies weights which are relative to the position of the source samples with respect to the position of the destination samples. If calculating the value for the top left destination sample, then the top left source sample will receive the largest weighting coefficient while the bottom right sample (diagonally opposite) will receive the smallest weighting coefficient, and the remaining two samples will receive an intermediate weighting coefficient. This is visualized in Figure 11.

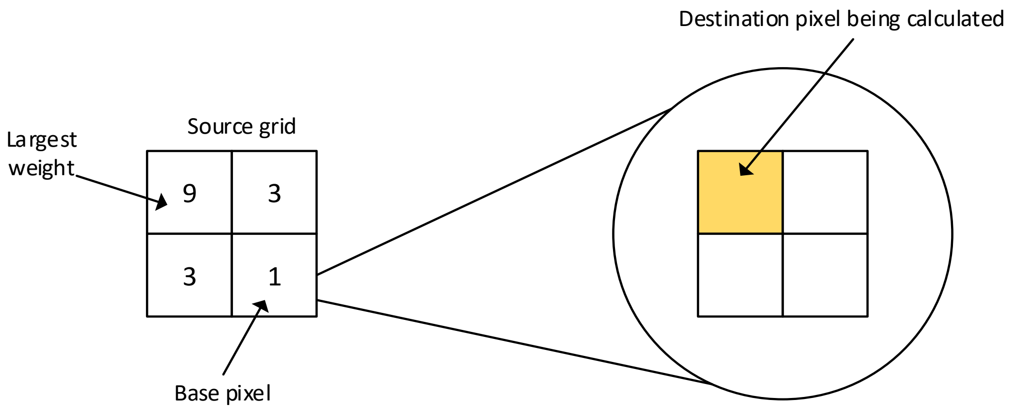

Figure 11 — Bilinear coefficient derivation

8.7.2.4 Bilinear interpolation upsampler kernel description

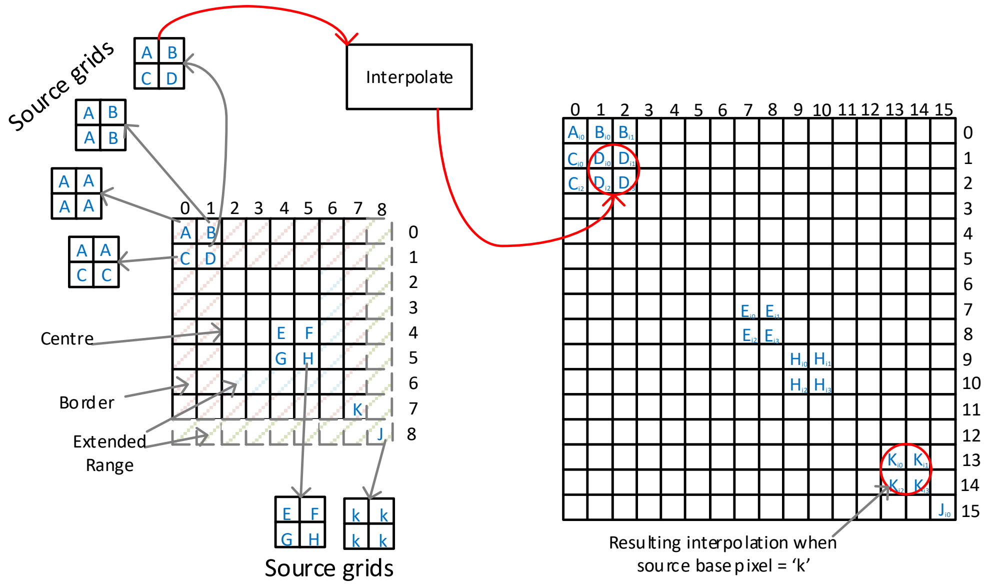

Figure 12 — Bilinear upsample kernel

The bilinear kernel upscaler, as illustrated in Figure 12, is applied as specified by the following ordered steps below when (xCurr, yCurr) block does not belong to the border area as specified in Figure 7:

− If scaling_mode_levelX (specified in subclause 7.4.3.3) is equal to 1:

```
for (ySrc = 0; ySrc < nCurrS + 1; ++ySrc)
    for (xSrc = 0; xSrc < nCurrS + 1; ++xSrc)
        xDst = (xSrc << 1) − 1
        bilinear1D(recInputSamples[xSrc − 1][ySrc], recInputSamples[xSrc1][ySrc],
            recUpsampledSamples [xDst][ySrc], recUpsampledSamples [xDst + 1][ySrc])
```

− If scaling_mode_levelX (specified in subclause 7.4.3.3) is equal to 2

```
for (ySrc = 0; ySrc < nCurrS + 1; ++ySrc)
    yDst = (ySrc << 1) − 1
    for (xSrc = 0; xSrc < nCurrS + 1; ++xSrc)
        xDst = (xSrc << 1) − 1
        bilinear2D(recInputSamples[xSrc − 1][ySrc − 1], recInputSamples[xSrc][ySrc − 1],
            recInputSamples[xSrc − 1][ySrc], recInputSamples[xSrc][ySrc],
            recUpsampledSamples[xDst][ySrc], recUpsampledSamples [xDst + 1][ySrc],
            recUpsampledSamples[xDst][ySrc + 1], recUpsampledSamples[xDst + 1][ySrc + 1])
```

The bilinear kernel upscaler (described in Figure 12 is applied as specified by the following ordered steps below when (xCurr, yCurr) block belongs to the border area as specified in Figure 7:

− If scaling_mode_levelX (specified in subclause 7.4.3.3) is equal to 1:

```
for (ySrc = 0; ySrc < nCurrS + 1; ++ySrc)
    for (xSrc = 0; xSrc < nCurrS + 1; ++xSrc)
        xDst = (xSrc << 1) − 1
        xSrc0 = Max(xSrc − 1, 0);
        xSrc1 = Min(xSrc, srcWidth − 1)
        bilinear1D(recInputSamples[xSrc0][ySrc], recInputSamples[xSrc1][ySrc], dst00, dst10)
        if (xDst >= 0)
            recUpsampledSamples[xDst][ySrc] = dst00
        if (xDst < (dstWidth–1))
            recUpsampledSamples[xDst + 1][ySrc] = dst10
```

− If scaling_mode_levelX (specified in subclause 7.4.3.3) is equal to 2:

```
for (ySrc = 0; ySrc < nCurrS + 1; ++ySrc)
    yDst = (ySrc << 1) − 1
    ySrc0 = Max(ySrc − 1, 0)));
    ySrc1 = Min (ySrc, srcHeight − 1)))
    for (xSrc = 0; xSrc < nCurrS + 1; ++xSrc)
        xDst = (xSrc << 1) −1
        xSrc0 = Max(xSrc − 1, 0);
        xSrc1 = Min(xSrc, srcWidth − 1)
        bilinear2D(recInputSamples[xSrc0][ySrc0], recInputSamples[xSrc1][xSrc0],
            recInputSamples[xSrc0][ySrc1], recInputSamples[xSrc1][ySrc1], dst00, dst10, dst01, dst11)
```

The function bilinear1D (in00, in10, out00, out10) is applied as specified by the following ordered steps below:

```
in00x3 = in00 * 3
in10x3 = in10 * 3
out00 = ((in00x3 + in10 + 2) >> 2)
out10 = ((in00 + in10x3 + 2) >> 2)
```

The function bilinear2D (in00, in10, in01, in11, out00, out10, out01, out11) is applied as specified by the following ordered steps below:

```
in00x3 = in00 * 3
in10x3 = in10 * 3
in01x3 = in01 * 3
in11x3 = in11 * 3
in00x9 = in00x3 * 3
in10x9 = in10x3 * 3
in01x9 = in01x3 * 3
in11x9 = in11x3 * 3
out00 = ((in00x9 + in10x3 + in01x3 + in11 + 8) >> 4))
out10 = ((in00x3 + in10x9 + in01+ in11x3 + 8) >> 4))
out01 = ((in00x3 + in10 + in01x9 + in11x3 + 8) >> 4))
```

out11 = ((in00 + in10x3 + in01x3 + in11x9 + 8) >> 4))

8.7.3 Cubic upsampler kernel description

8.7.3.1 Cubic upsampler kernel process inputs and outputs, process overview

Inputs to this process are the following:

- variables srcX and srcY specifying the width and the height of the input array,
- variables dstX and dstY specifying the width and the height of the output array,
- a (srcX)x(srcY) array recInputSamples [x][y] of input samples.Outputs to this process are a (dstX)x(dstY) array recUpsampledSamples [x][y] of output samples.

The Cubic upsampling kernel can be divided into three main steps. The first step involves constructing a 4x4 grid of source samples with the base sample positioned at the local index (2,2). The second step involves performing the bicubic interpolation and the third step involves writing the interpolation result to the destination samples.

8.7.3.2 Source sample grid

The Cubic upsampling kernel is performed by using a 4x4 source grid which is subsequently multiplied by a 4x4 kernel. During the generation of the source grid, any samples which fall outside the frame limits of the source frame are replaced with the value of the source samples at the boundary of the frame. This is visualized in Figure 13.

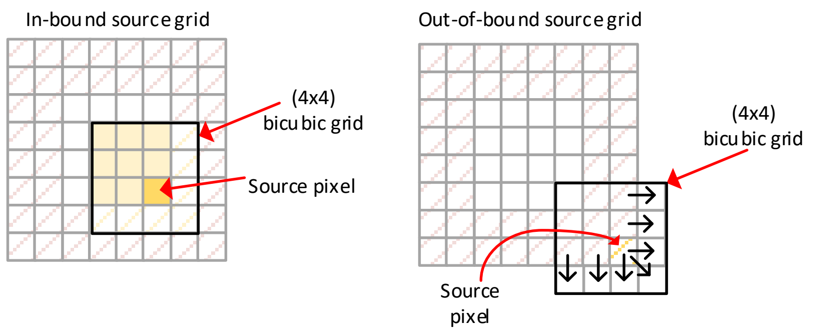

Figure 13 — Source grid for the cubic upsampler

8.7.3.3 Cubic interpolation

The kernel used for the Bicubic upsampling process typically have a 4x4 coefficient grid. However, the relative position of the destination sample with regards to the source sample will yield a different coefficient set, and since the upsampling is a factor of two, there will be 4 sets of 4x4 kernels used in the upsampling process. These sets are represented by a 4-dimensional grid of coefficients (2 x 2 x 4 x 4). The Bicubic coefficients are calculated from a fixed set of parameters; a core parameter (Bicubic parameter) of −0.6 and four spline creation parameters of [1.25, 0.25, −0.75 & −1.75]. The implementation of the filter is using fixed point math.

8.7.3.4 Cubic interpolation kernel description

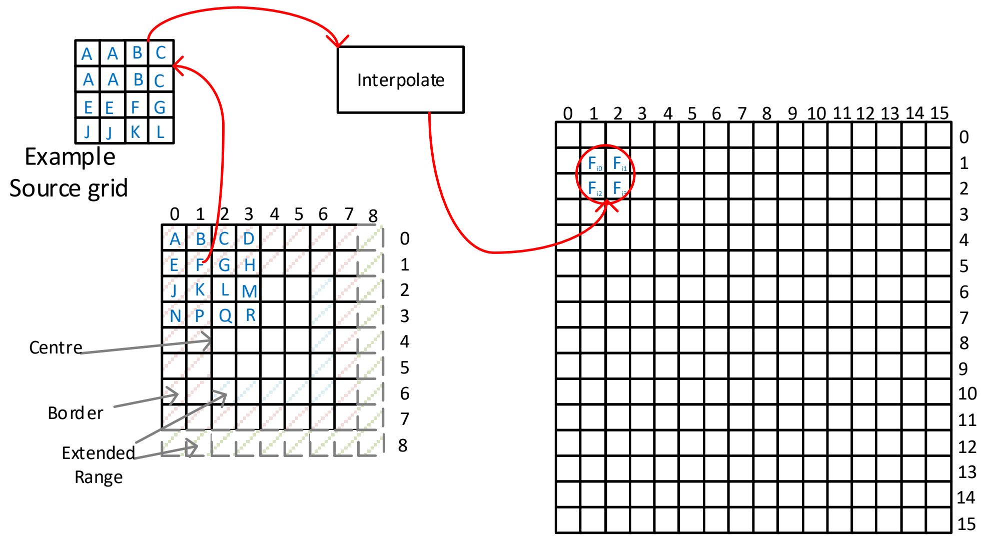

Figure 14 — Cubic upsampling algorithm

The cubic kernel upscaler (described in Figure 14) is applied on one direction (vertical and horizontal) at time and follows different steps if (xCurr, yCurr) block belongs to the border as specified in Figure 7.

Given a set of coefficients as follows:

kernel[y][x] =

```
{
{ -1382, 14285,  3942,  -461 }
{  -461,  3942, 14285, -1382 }
{ -1280, 14208,  3840,  -384 }
{  -384,  3840, 14208, -1280 }
}
```
(31)

- where y = 0...1 are coefficients to be used with 10 bit samples and y = 2...3 to be used with 8 bits samples.
- kernelOffset is equal to 4.
- kernelSize is equal to 4.

```
if (Horizontal) {
    for (y = 0; y < nCurrS; y++)
        for (xSrc = 0; xSrc < nCurrS + 1; xSrc++)
            ConvolveHorizontal(recInputSamples, recUpsampledSamples, xSrc, y, kernel[is8Bit * 2])
} else if (Vertical) {
    dstHeightM1 = dstHeight – 1
    for (ySrc = 0; ySrc < nCurrS + 1; ySrc++)
        yDst = (ySrc << 1) − 1
        if (border) {
            yDst0 = ((yDst > 0)&&(yDst < dstHeight)) ? yDst : −1
            yDst1 = ((yDst + 1) < dstHeightM1) ? yDst + 1 : −1
        } else {
            yDst0 = yDst
            yDst1 = (yDst + 1)
        }
        for (x = 0; x < nCurrS; x++)
            ConvolveVertical(recInputSamples, recUpsampledSamples, yDst0, yDst1, x, ySrc,
                kernel[is8Bit * 2])
}
```

The function ConvolveHorizontal(input, output, x, y, kernel, border) is applied as specified by the following ordered steps below:

```
xDst = (x << 1) − 1;
if (border)
    dstWidthM1 = dstWidth − 1
if (xDst >= 0 && xDst < dstWidth)
    output[xDst][y] = ConvolveHorizontal (kernel[0], input[x + kernelOffset][y] , 14);
if (xDst < dstWidthM1)
    output [xDst + 1][y] = ConvolveHorizontal(kernel[1], input[x + kernelOffset][y]) , 14)
else
    output [xDst][y] = ConvolveHorizontal (kernel[0], input[x + kernelOffset][y] , 14)
    output [xDst + 1][y] = ConvolveHorizontal (kernel[1], input[x + kernelOffset][y] , 14)
```

The function ConvolveVertical (input, output, yDst0, yDst1, x, ySrc, kernel) is applied as specified by the following ordered steps below:

```
if (border)
    dstWidthM1 = dstWidth − 1
if (yDst0 >= 0)
    output[x][yDst0] = ConvolveHorizontal (kernel[0], input[x][y + kernelOffset] , 14)
if (yDst0 >= 0)
    output [x][yDst] = ConvolveHorizontal(kernel[1], input[x][y + kernelOffset]) , 14)
else
    output [x][yDst0] = ConvolveHorizontal (kernel[0], input[x + kernelOffset][y] , 14)
    output [x][yDst1] = ConvolveHorizontal (kernel[1], input[x + kernelOffset][y], 14)
```

The function output = ConvolveHorizontal (kernel, input, shift) is applied as specified by the following ordered steps below:

```
accumulator = 0
for (int32_t x = 0; x < kernelSize; x++)
accumulator += input[x] * kernel[x]
offset = 1 << (shift − 1)
output = ((accumulator + offset) >> shift)
```

8.7.4 Modified Cubic upsampler kernel description

Inputs to this process are the following:

- variables srcX and srcY specifying the width and the height of the input array,
- variables dstX and dstY specifying the width and the height of the output array,
- a (srcX)x(srcY) array recInputSamples [x][y] of input sample

Outputs to this process are the following:

- a (dstX)x(dstY) array recUpsampledSamples [x][y] of output samples,

The implementation of the Modified Cubic filter is using fixed point math and the process is the same described in subclause 8.7.3.3, but with the following kernel coefficients:

kernel[y][x] =

```
{
{ -2360, 15855,  4165, -1276 }
{ -1276,  4165, 15855, -2360 }
{ -2360, 15855,  4165, -1276 }
{ -1276,  4165, 15855, -2360 }
}
```
(32)

- where y = 0...1 are coefficients to be used with 10 bit samples and y = 2...3 to be used with 8 bits samples.
- kernelOffset is equal to 4.
- kernelSize is equal to 4.

8.7.5 Predicted residual process description

Inputs to this process are the following:

- variables srcX and srcY specifying the width and the height of the lower resolution array,
- variables dstX and dstY specifying the width and the height of the upsampled arrays,
- a (srcX)x(srcY) array recLowerResSamples [x][y] of samples that were provided as input to the upscaling process,
- a (dstX)x(dstY) array recUpsampledSamples [x][y] of samples that were output of the uscaling process,

Outputs to this process are the following:

- a (dstX)x(dstY) array recUpsampledModifiedSamples [x][y] of output samples,

The predicted residual process is modifying recUpsampledSamples using a 2x2 grid if scaling_mode_levelX is equal to 2 and using a 2x1 grid if scaling_mode_levelX is equal to 1. The predicted residual process is not applied if scaling_mode_levelX is equal to 0.

The predicted residual process is applied as specified by the following ordered steps whenever (xCurr, yCurr) block belongs to the picture or to the the border area as specified in Figure 7:

- If scaling_mode_levelX (specified in subclause 7.4.3.3) is equal to 1:

```
for (ySrc = 0; ySrc < srcY; ySrc++)
    yDst = ySrc
    for (xSrc = 0; xSrc < srcX; xSrc++)
        xDst = xSrc << 1
        modifier = recLowerResSamples[xSrc][ySrc] – (recUpsampledSamples[xDst][yDst] +
            recUpsampledSamples[xDst + 1][yDst]) >> 1
        recModifiedUpsampledSamples[xDst][yDst] = recUpsampledSamples[xDst][yDst] + modifier
        recModifiedUpsampledSamples[xDst + 1][yDst] = recUpsampledSamples[xDst + 1][yDst] +
            modifier
```

- If scaling_mode_levelX (specified in subclause 7.4.3.3) is equal to 2

```
for (ySrc = 0; ySrc < srcY; ySrc++)
    yDst = ySrc << 1
    for (xSrc = 0; xSrc < srcX; xSrc++)
        xDst = xSrc << 1
        modifier = recLowerResSamples[xSrc][ySrc] –
            (recUpsampledSamples[xDst][yDst] +
            recUpsampledSamples[xDst + 1][yDst] +
            recUpsampledSamples[xDst][yDst + 1] +
            recUpsampledSamples[xDst + 1][yDst + 1]) >> 2
        recModifiedUpsampledSamples [xDst][yDst] = recUpsampledSamples[xDst][yDst] + modifier
        recModifiedUpsampledSamples [xDst][yDst + 1] = recUpsampledSamples[xDst + 1][yDst] +
            modifier
        recModifiedUpsampledSamples [xDst + 1][yDst] = recUpsampledSamples[xDst][yDst + 1] +
            modifier
        recModifiedUpsampledSamples [xDst + 1][yDst + 1] = recUpsampledSamples[xDst + 1][yDst + 1] +
            modifier
```

8.8 Decoding process for the residual reconstruction

8.8.1 Reconstructed residual of each block derivation

The reconstructed residual of each block is derived as follows:

The variable $nCbS_L$ is set equal to 2 if transform_type or 4 if transform_type is equal to 1 (subclause 7.3.3.1). The variable $nCbS_C$ is set equal to $nCbS_L$ >> 1.

If IdxPlanes is equal to 0 the residual reconstruction process for a colour component as specified in subclause 8.8.2 and subclause 8.8.3 is invoked with the luma coding block location (xCb, yCb), the variable nCurrS set equal to $nCbS_L$, and the variable IdxPlanes set equal to 0 as input.

If IdxPlanes is equal to 1 the residual reconstruction process for a colour component as specified in subclause 8.8.2 and 8.8.3 is invoked with the the chroma coding block location (xCb >> ShiftWidthC, yCb >> ShiftHeightC), the variable nCurrS set equal to $nCbS_C$, and the variable IdxPlanes set equal to as inputs.

Where ShiftWidthC and ShiftHeightC are specified in subclause 6.2.

8.8.2 Residual reconstruction for L-1 block

Inputs to this process are:

- a location (xCurr, yCurr) specifying the top-left sample of the current block relative to the top-left sample of the current picture component,
- a variable IdxPlanes specifying the colour component of the current block,
- a variable nCurrS specifying the size of the residual block,
- an (nCurrS)x(nCurrS) array recL1BaseSamples specifying the preliminary intermediate picture reconstructed samples of the current block as specifyied in 8.10,
- an (nCurrS)x(nCurrS) array resL1FilteredResiduals specifying the L-1 filtered residuals of the current block.

Output of this process is the combined intermediate picture (nCurrS)x(nCurrS) array recL1Samples with elements recL1Samples [x][y].

The (nCurrS)x(nCurrS) block of the reconstructed sample array recL1Samples at location (xCurr, yCurr) is derived as follows:

$$recL1Samples[xCurr + i][yCurr + j] = recL1BaseSamples[i][j] + resL1FilteredResiduals[i][j] \quad (33)$$

with i = 0 ... nCurrS − 1, j = 0 ... nCurrS – 1

The upscaling process for a colour component as specified in subclause 8.7 is invoked with the location (xCurr, yCurr), the transform block size nTbS, the (nCurrS)x(nCurrS) array recL1Samples, the variables srcWidth and srcHeight specifying the size of the reconstructed base picture, the variables dstWidth and dstHeight specifying the width and the height of the upscaled resulting picture, and the variable is8Bit equal to 1 if enhancement_depth_type (subclause 7.4.3.3) is equal to 0 as inputs.

8.8.3 Residual reconstruction for L-2 block

Inputs to this process are:

- a location (xCurr, yCurr) specifying the top-left sample of the current block relative to the top-left sample of the current picture component,
- a variable IdxPlanes specifying the colour component of the current block,
- an (nCurrS)x(nCurrS) array recL0ModifiedUpscaledSamples specifying the preliminary output picture samples of the current block,

- an (nCurrS)x(nCurrS) array resL0Residuals specifying the L-2 residuals of the current block.

Output of this process is the (nCurrS)x(nCurrS) array recL0PictureSamples of combined output picture samples with elements recL0PictureSamples [x][y].

The (nCurrS)x(nCurrS) block of the reconstructed sample array recL0PictureSamples at location (xCurr, yCurr) is derived as follows:

recL0PictureSamples [xCurr + i][yCurr + j] = recL0ModifiedUpscaledSamples [i][j] + resL0Residuals [i][j] (34)

with i = 0 ... nCurrS − 1, j = 0 ... nCurrS – 1

If dithering_type as derived in subclause 7.4.3.4 is not equal to 0, the process as specified in sub-clause 8.12 is invoked with the location (xCurr, yCurr), the (nCurrS)x(nCurrS) array recL0PictureSamples.

8.9 Decoding process for the L-1 filter

8.9.1 L-1 residual filter overview

One in-loop filter, namely L-1 residual filter, is applied on the L-1 residual surface block before they are being added to the base reconstructed picture. The L-1 filter operates only if the variable transform_type specified in subclause 7.4.3.2 is equal to 1. The L-1 filter operates on each 4x4 block of transformed residuals by applying a mask whose weights are structured as follows:

```
{
{ α, β, β, α }
{ β, 1, 1, β }
{ β, 1, 1, β }
{ α, β, β, α }
}
```
(35)

8.9.2 Decoding process for filtering L-1 block

Inputs to this process are:

- a sample location (xTb0, yTb0) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture,
- an array resL1Residuals of a size 4x4 specifying residuals for enhancement sub-layer 1,

Output to this process is the 4x4 array of the residual resL1FilteredResiduals with elements resL1FilteredResiduals[x][y].

In-loop filter L-1 residual filter is applied as specified by the following ordered steps:

1. A variable deblockEnabled, α and β are derived as follows (as described in subclause 7.3.5):

```
deblockEnabled = level1_filtering_enabled_flag
if (level_1_filtering_signalled_flag)
    $\alpha$ = 16 − level_1_filtering_first_coefficient        (36)
    $\beta$ = 16 − level_1_filtering_second_coefficient       (37)

else
    $\alpha$ = 16
    $\beta$ = 16
```

2. If deblockEnabled is true, the following steps are applied given the residual representation in Figure 8:

```
    resL1FilteredResiduals[0][0] = (resL1Residuals[0][0] * $\alpha$) >> 4
    resL1FilteredResiduals[0][3] = (resL1Residuals[0][3] * $\alpha$) >> 4
    resL1FilteredResiduals[3][0] = (resL1Residuals[3][0] * $\alpha$) >> 4
    resL1FilteredResiduals[3][3] = (resL1Residuals[3][3] * $\alpha$) >> 4
```

resL1FilteredResiduals[0][1] = (resL1Residuals[0][1] * β) >> 4
resL1FilteredResiduals[0][2] = (resL1Residuals[0][2] * β) >> 4
resL1FilteredResiduals[1][0] = (resL1Residuals[1][0] * β) >> 4
resL1FilteredResiduals[2][0] = (resL1Residuals[2][0] * β) >> 4
resL1FilteredResiduals[1][3] = (resL1Residuals[1][3] * β) >> 4
resL1FilteredResiduals[2][3] = (resL1Residuals[2][3] * β) >> 4
resL1FilteredResiduals[3][1] = (resL1Residuals[3][1] * β) >> 4
resL1FilteredResiduals[3][2] = (resL1Residuals[3][2] * β) >> 4
resL1FilteredResiduals[1][1] = resL1Residuals[1][1]
resL1FilteredResiduals[1][2] = resL1Residuals[1][2]
resL1FilteredResiduals[2][1] = resL1Residuals[2][1]
resL1FilteredResiduals[2][2] = resL1Residuals[2][2] (38)

otherwise:
resL1FilteredResiduals[0][0] = resL1Residuals[0][0]
resL1FilteredResiduals[0][3] = resL1Residuals[0][3]
resL1FilteredResiduals[3][0] = resL1Residuals[3][0]
resL1FilteredResiduals[3][3] = resL1Residuals[3][3]

resL1FilteredResiduals[0][1] = resL1Residuals[0][1]
resL1FilteredResiduals[0][2] = resL1Residuals[0][2]
resL1FilteredResiduals[1][0] = resL1Residuals[1][0]
resL1FilteredResiduals[2][0] = resL1Residuals[2][0]
resL1FilteredResiduals[1][3] = resL1Residuals[1][3]
resL1FilteredResiduals[2][3] = resL1Residuals[2][3]
resL1FilteredResiduals[3][1] = resL1Residuals[3][1]
resL1FilteredResiduals[3][2] = resL1Residuals[3][2]

resL1FilteredResiduals[1][1] = resL1Residuals[1][1]
resL1FilteredResiduals[1][2] = resL1Residuals[1][2]
resL1FilteredResiduals[2][1] = resL1Residuals[2][1]
resL1FilteredResiduals[2][2] = resL1Residuals[2][2] (39)

8.10 Decoding process for base decoder data extraction

Inputs to this process are:

- a location (xCurr, yCurr) specifying the top-left sample of the current block relative to the top-left sample of the current picture component,
- a variable IdxBaseFrame specifying the base decoder picture buffer frame from which to read the samples,
- a variable IdxPlanes specifying the colour component of the current block.

Output of this process is the (nCurrX)x(nCurrY) array recDecodedBaseSamples of picture samples with elements recDecodedBaseSamples [x][y].

The process is reading the block of sample (nCurrS)x(nCurrS) from the location (xCurr, yCurr) and the frame pointed by the variable IdxBaseFrame. The blocks are read in raster order.

The sample block size nCurrX and nCurrY are derived as follows:

nCurrX = (IdxPlanes == 0) ? nCurrX : nCurrX >> ShiftWidthC (40)

nCurrY = (IdxPlanes == 0) ? nCurrY : nCurrY >> ShiftHeightC (41)

8.11 Decoding process for dither filter

Inputs to this process are:

- a location (xCurr, yCurr) specifying the top-left sample of the current block relative to the top-left sample of the current picture component,

- an (nCurrS)x(nCurrS) array recL0PictureSamples specifying the reconstructed combined output picture samples,

Output of this process is the (nCurrS)x(nCurrS) array recL0DitheredPictureSamples of residuals with elements recL0DitheredPictureSamples [x][y].

If dithering_type is equal 1 (uniform dither), the (nCurrS)x(nCurrS) block of the reconstructed sample array recL0DitheredPictureSamples at location (xCurr, yCurr) is derived as follows:

recL0DitheredPictureSamples [xCurr + i][yCurr + j] = recL0PictureSamples [i][j] + rand(i,j) (42)

with i = 0…nCurrS − 1, j = 0…nCurrS – 1. rand(i,j) is pseudo_random number in the range [−dithering_strength,+dithering_strength] with dithering_strength as derived in subclause 7.4.5.

9 Parsing process

9.1 Parsing process inputs and outputs, process overview

9.1.1 Parsing process for entropy encoded transform coefficients

Inputs to this process are the bits belonging to chunks of data containing the entropy encoded transform coefficients derived from the process described in clause 8.3.

If tile_dimensions_type is equal to 0, for each chunk the following information is provided:

- a variable surfaces[planeIdx][levelIdx][layerIdx].rle_only_flag specifying if the Prefix Coding decoder is needed,
- a variable surfaces[planeIdx][levelIdx][layerIdx].size specifying the size of the chunk of data,
- a variable surfaces[planeIdx][levelIdx][layerIdx].data specifying the beginning of the chunk.

Where planeIdx, levelIdx and layerIdx indicate the plane, enhancement sub-layer and coefficient group to which the chunk belongs.

Outputs of this process are entropy decoded quantized transform coefficients to be used as input for processes described in subclause 8.3.4 and in subclause 8.3.5, in the order described in Figure 4 and the clause 8.3.

If tile_dimensions_type is not equal to 0, and the following information is provided for each chunk:

- a variable surfaces[planeIdx][levelIdx][layerIdx].tiles pointing to the tiles of the decoded picture.
- a variable surfaces[planeIdx][levelIdx][layerIdx].rle_only_flag specifying if the Prefix Coding decoder is needed for all tiles.

In this case, a chunk of data is further split to smaller chunks of data, which are termed as tiles. For each tile the following information are provided:

- a variable surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].size specifying the size of the chunk of tile data;
- a variable surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].data specifying the beginning of the chunk.

Where planeIdx, levelIdx, layerIdx and tileIdx indicate the plane, enhancement sub-layer, coefficient group and tile to which the chunk belongs.

Outputs of this process are entropy decoded quantized transform coefficients to be used as input for processes described in subclause 8.3.4 and in subclause 8.3.5, in the order described in Figure 5 and the clause 8.3.

The entropy decoder consists of two components:

- Prefix Coding decoder.

– Run Length decoder.

The general process is described in Figure 15.

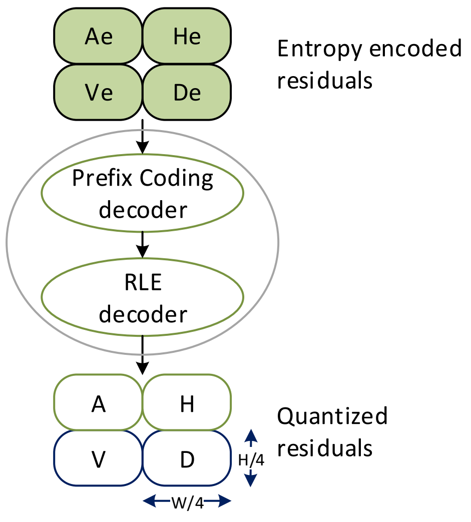

Figure 15 — Generic entropy decoder

9.1.2 Parsing process for entropy encoded temporal signal coefficient group

Inputs to this process are the bits belonging to chunks of data containing the entropy encoded temporal signal coefficient group derived from the process described in clause 8.3.

If tile_dimensions_type is equal to 0, for each chunk the following information are provided:

– a variable temporal_surfaces[planeIdx].rle_only_flag specifying if the Prefix Coding decoder is needed,

– a variable temporal_surfaces[planeIdx].size specifying the size of the chunk of data,

– a variable temporal_surfaces[planeIdx].data specifying the beginning of the chunk.

Where planeIdx indicates the plane to which the chunk belongs.

The output of this process is an entropy decoded temporal signal coefficient group to be stored in TempSigSurface, as described in subsection subclause 9.3.4.

If tile_dimensions_type is not equal to 0, the following information is provided for each chunk:

– a variable temporal_surfaces[planeIdx].tiles pointing to the tiles of the decoded picture;

– a variable temporal_surfaces[planeIdx].rle_only_flag specifying if the Prefix Coding decoder is needed for all tiles.

In this case, a chunk of data is further split to smaller chunks of data, which are termed as tiles. For each tile the following information are provided:

– a variable temporal_surfaces[planeIdx].tiles[tileIdx].size specifying the size of the chunk of tile data;

– a variable temporal_surfaces[planeIdx].tiles[tileIdx].data specifying the beginning of the chunk.

Where planeIdx and tileIdx indicate the plane and tile to which the chunk belongs.

The output of this process is an entropy decoded temporal signal coefficient group to be stored in TempSigSurface as described in subsection 9.3.4.

The entropy decoder consists of two components:

- Prefix Coding decoder.
- Run Length decoder.

9.2 Prefix Coding decoder

9.2.1 Prefix Coding decoder description

If variable rle_only is equal to 1, the Prefix Coding decoder process is skipped, and the process described in subclause 9.3 is invoked. If variable rle_only_flag is equal to 0 the Prefix Coding decoder is applied as specified by the following ordered steps:

1. Initialize the Prefix Coding decoder by reading the code lengths from the stream header size.

   If there are more than 31 non-zero values the stream header is as specified in Figure 16

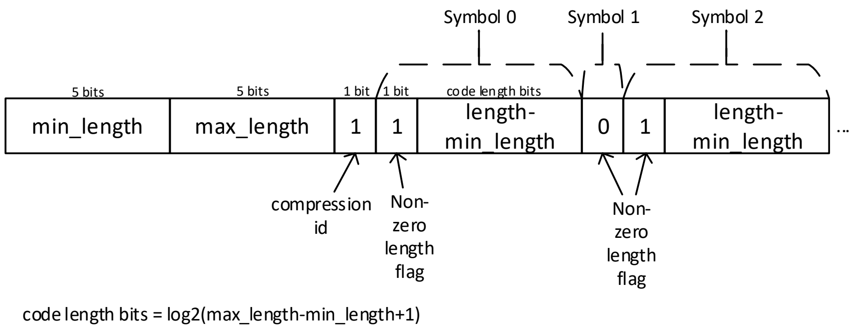

Figure 16 — Prefix Coding decoder stream header for more than 31 non-zeros codes

   Otherwise the stream header is as specified in Figure 17:

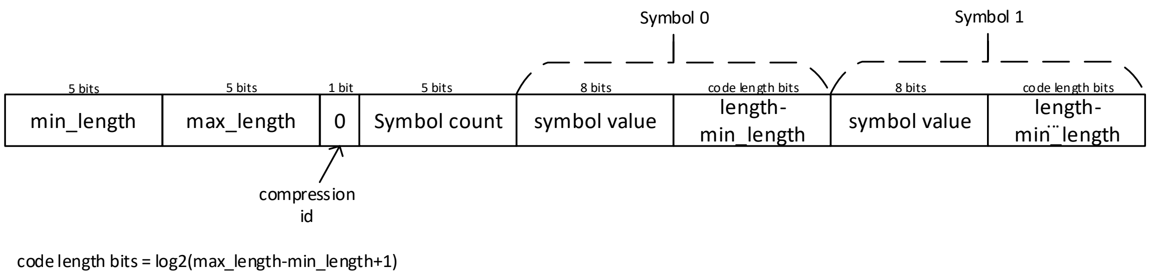

Figure 17 — Prefix Coding decoder normal case

   In the special case for which the frequencies are all zero, the stream header is specified in Figure 18:

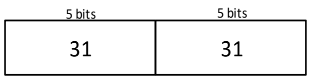

Figure 18 — Special case: Prefix Coding decoder stream header frequencies all zeros

   In the special case where there is only one code in the Prefix Coding tree the stream header is specified in Figure 19:

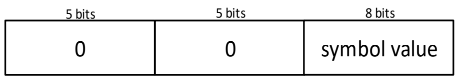

Figure 19 — Special case: only one code in the Prefix Coding tree

2. The following step:

   a. Set code lengths for each symbol

   b. Assign codes to symbols from the code lengths

   c. Generate a table for searching the subsets of codes with identical lengths. Each element of the table records the first index of a given length and the corresponding code (firstIdx, firstCode).

9.2.2 Prefix Coding decoder table generation

Table generation of table at point c) or the step 3 is applied as specified by the following ordered steps:

To find a Prefix Coding Code for a given set of symbols a Prefix Coding tree needs to be created. First the symbols are sorted by frequency, for example, as shown in Table 36:

Table 36 — Symbols sorted by frequency

Symbol	Frequency
A	3
B	8
C	10
D	15
E	20
F	43

The two lowest elements are removed from the list and made into leaves, with a parent node that has a frequency the sum of the two lower element’s frequencies. The partial tree is shown in Figure 20:

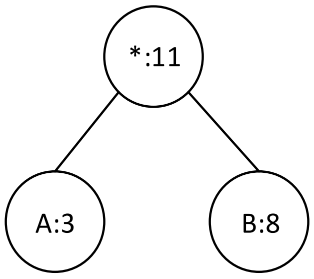

Figure 20 — Partial tree

And the new sorted frequency list is as shown in Table 37:

Table 37 — New sorted frequency

Symbol	Frequency
C	10
*	11
D	15
E	20
F	43

Then the loop is repeated, combining the two lowest elements, as shown in Figure 21:

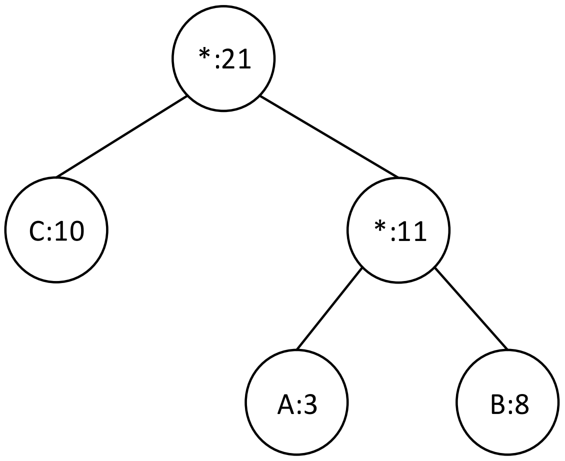

Figure 21 — Repeated loop, showing two lowest elements combined

The new list is as shown in Table 38:

Table 38 — Updated sorted frequency

Symbol	Frequency
D	15
E	20
*	21
F	43

This is repeated until only one element remains in the list, as shown in Figure 22, Table 39, Figure 23, Table 40, Figure 24 and Table 41.

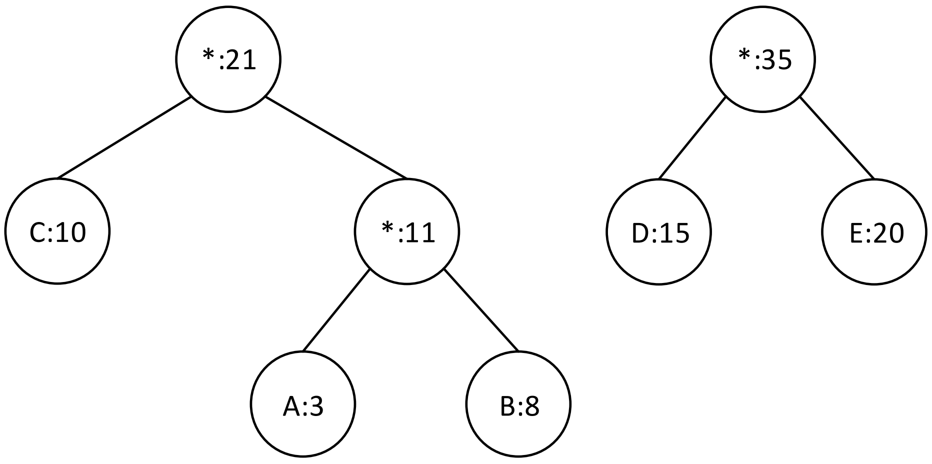

Figure 22 — Loop repeated

Table 39 — Update sorted frequency

Symbol	Frequency
*	21
*	35
F	43

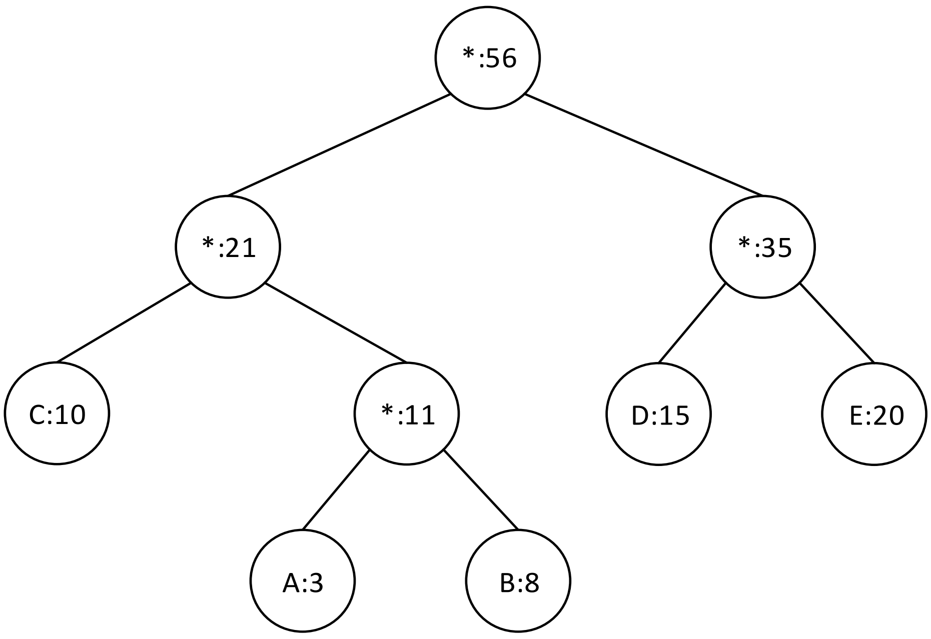

Figure 23 — Loop repeated

Table 40 — Updated sorted frequency

Symbol	**Frequency**
F	43
*	56

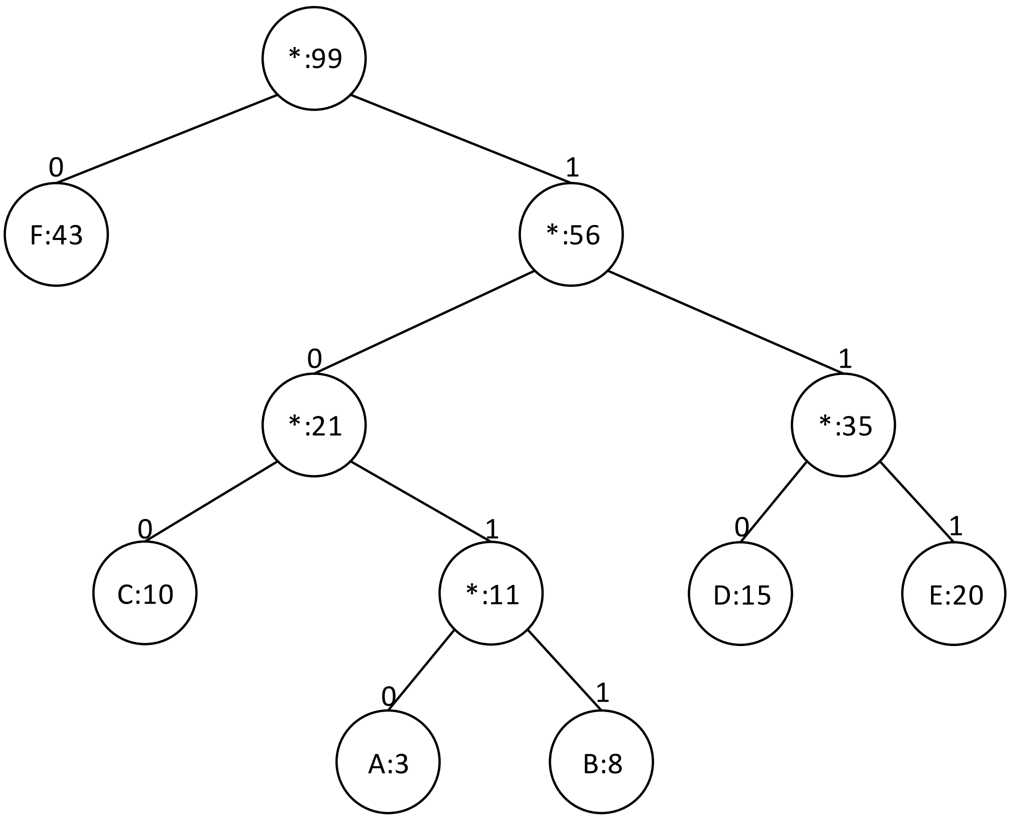

Figure 24 — Loop process completion

Table 41 — One item remaining in list

Symbol	Frequency
*	99

Once the tree is built, to generate the Prefix Coding code for a symbol the tree is traversed from the root to this symbol, appending a 0 each time a left branch is taken and a 1 each time a right branch is taken. In the example above this gives the following code, as specified in Table 42:

Table 42 — Resulting Prefix Coding code

Symbol	Code	Code length
A	1010	4
B	1011	4
C	100	3
D	110	3
E	111	3
F	0	1

The code length of a symbol is the length of its corresponding code. To decode a Prefix Coding code, the tree is traversed beginning at the root, taking a left path if a 0 is read and a right path if a 1 is read. The symbol is found when reaching a leaf.

9.2.3 Prefix Coding decoder for tile data sizes

9.2.3.1 Prefix Coding decoder overview

The decoder reads the Prefix Coding encoded data size of each tile byte by byte. By construction the state of the first byte of data is guaranteed to be LSB Prefix Code state. The decoder uses the state machine shown in Figure 25 to determine the state of the next byte of data. The state tells the decoder how to interpret the current byte of data as described in subclause 9.2.3.2, and shown in Figure 25.

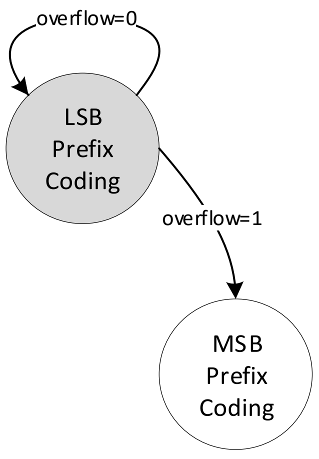

Figure 25 — Prefix Coding decoder state machine

9.2.3.2 Prefix Coding decoder description

The Prefix Coding has 2 states:

LSB Prefix Coding state: this context encodes the 7 less significant bits of a non-zero value. In this state a byte is divided as shown in Figure 26.

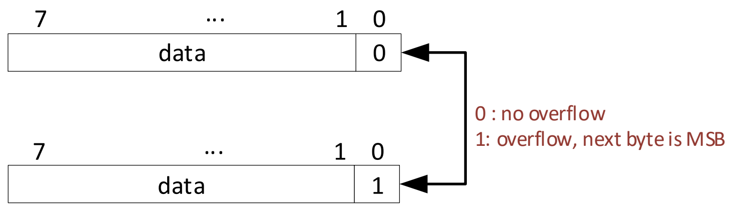

Figure 26 — Prefix Coding encoding of a byte for LSB Prefix Coding state

The overflow bit is set if the value does not fit within 7 bits of data. When the overflow bit is set, the state of the next byte will be MSB Prefix Coding state.

MSB Prefix Coding state: this state encodes bits 8 to 15 of values that do not fit within 7 bits of data. Run Length encoding of a byte for MSB Prefix Coding state is shown in Figure 27.

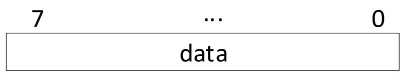

Figure 27 — Run Length encoding of a byte for MSB Prefix Coding state

A frequency table is created for each state for use by the Prefix Coding encoder.

If this process is invoked with surfaces referring to entropy encoded transform coefficients, the decoded values are stored into a temporary buffer tmp_size_per_tile of size nTilesL1 or nTilesL2 (respectively, number of tiles for enhancement sub-layer 1 and sub-layer 2, as derived in subclause 8.3.3), and get mapped to surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].size as follows (planeIdx, levelIdx, layerIdx, and tileIdx have been already specified in subclause 9.1.1):

```
if (levelIdx == 1)
    nTiles = nTilesL1
else
    nTiles = nTilesL2

if (compression_type_size_per_tile == 2) {
    for (tileIdx = 1; tileIdx < nTiles; tileIdx++) {
        tmp_size_per_tile[tileIdx] += tmp_size_per_tile[tileIdx – 1]
    }
}

for (tileIdx = 0; tileIdx < nTiles; tileIdx++) {
    temporal_surfaces[planeIdx].tiles[tileIdx].size = tmp_size_per_tile[tileIdx]
}
```

If this process is invoked with temporal_surfaces referring to an entropy encoded transform signal coefficient group, the decoded values are stored into a temporary buffer tmp_size_per_tile of size nTilesL2 (as derived in subclause 8.3.3) and get mapped to temporal_surfaces[planeIdx].tiles[tileIdx].size as follows (planeIdx and tileIdx have been already specified in subclause 9.1.2):

```
if (compression_type_size_per_tile == 2) {
    for (tileIdx = 1; tileIdx < nTilesL2; tileIdx++) {
        tmp_size_per_tile[tileIdx] += tmp_size_per_tile[tileIdx – 1]
    }
}
```

```
for (tileIdx = 0; tileIdx < nTilesL2; tileIdx++) {
    temporal_surfaces[planeIdx].tiles[tileIdx].size = tmp_size_per_tile[tileIdx]
}
```

9.2.4 Prefix Coding decoder for last bit symbol offset per tile

The same Prefix Coding decoding process as described in subclause 9.2.3 shall be used.

If this process is invoked with surfaces referring to entropy encoded transform coefficients, the decoded values are stored into a temporary buffer tmp_decoded_tile_prefix_last_symbol_bit_offset of size nTilesL1 or nTilesL2 (respectively, number of tiles for enhancement sub-layer 1 and sub-layer 2 as derived in subclause 8.3.3) and get mapped to surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx]. Prefix_last_symbol_bit_offset as follows (planeIdx, levelIdx, layerIdx and tileIdx have been already specified in subclause 9.1.1):

```
if (levelIdx == 1)
    nTiles = nTilesL1
else
    nTiles = nTilesL2
for (tileIdx = 0; tileIdx < nTiles; tileIdx++) {
    surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].prefix_last_symbol_bit_offset =
        tmp_decoded_tile_prefix_last_symbol_bit_offset[tileIdx]
}
```

If this process is invoked with temporal_surfaces referring to an entropy encoded transform signal coefficient group, the decoded values are stored into a temporary buffer tmp_decoded_tile_prefix_last_symbol_bit_offset of size nTilesL2 (as derived in subclause 8.3.3) and get mapped to temporal_surfaces[planeIdx].tiles[tileIdx].prefix_last_symbol_bit_offset as follows (planeIdx and tileIdx have been already specified in subclause 9.1.2):

```
for (tileIdx = 0; tileIdx < nTilesL2; tileIdx++) {
    temporal_surfaces[planeIdx].tiles[tileIdx].prefix_last_symbol_bit_offset =
        tmp_decoded_tile_prefix_last_symbol_bit_offset[tileIdx]
}
```

9.3 RLE decoder

9.3.1 RLE process inputs and outputs

The input of the RLE decoder is a byte stream of Prefix Coding decoded data if rle_only_flag is equal to zero or just a byte stream of raw data if rle_only_flag is equal to 1. The output of this process is a stream of quantized transform coefficients belonging to the chunk pointed by the variables planeIdx, levelIdx and layerIdx as specified in clause 8.3 or the stream of temporal signals belonging to temporal chunk.

9.3.2 RLE decoder for coefficient groups

The run length state machine is used by the Prefix Coding encoding and decoding processed to know which Prefix Coding code to use for the current symbol or code word. The RLE decodes sequences of zeros. It also decodes the frequency tables used to build the Prefix Coding trees.

The run length decoder read the run length encoded data byte by byte. By construction the state of the first byte of data is guaranteed to be LSB RLE state. The decoder uses the state machine shown in Figure 28 to determine the state of the next byte of data. The state tells the decoder how to interpret the current byte of data as described in subclause 9.3.3.

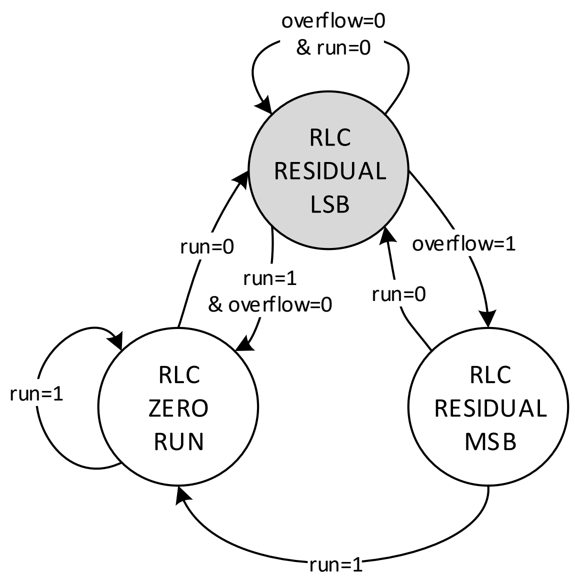

Figure 28 — RLE decoder state machine

9.3.3 RLE decoder description

The RLE has 3 states:

LSB RLE state: this state encodes the 6 less significant bits of a non-zero sample. In this state a byte is divided as shown in Figure 29:

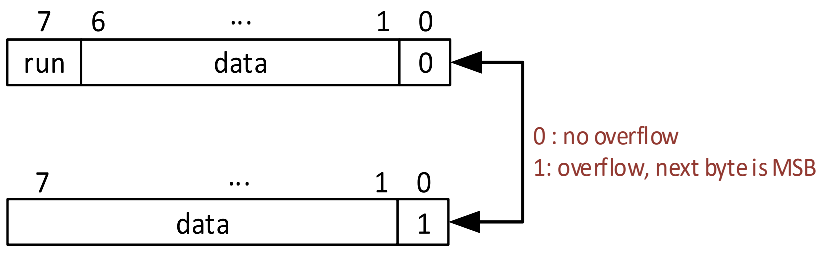

Figure 29 — Run Length encoding of a byte for LSB RLE state

The run bit indicates that the next byte is encoding the count of a run of zeros. The overflow bit is set if the sample value does not fit within 6 bits of data. When the overflow bit is set, the state of the next byte will be MSB RLE state. Therefore, the next state cannot be a run of zeros and bit 7 can be used to encode data instead.

MSB RLE state: this state encodes bits 7 to 13 of sample values that do not fit within 6 bits of data. Bit 7 encodes whether the next byte is a run of zeros. Run Length encoding of a byte for MSB RLE state is as shown in Figure 30:

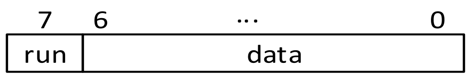

Figure 30 — Run Length encoding of a byte for MSB RLE state

Zero Run state: this state encodes 7 bits of a zero run count. The run bit is high if more bits are needed to encode the count. Run Length encoding of a byte for Zero Run state is as shown in Figure 31:

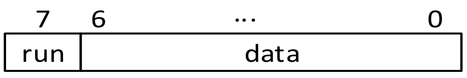

Figure 31 — Run Length encoding of a byte for Zero Run state

A frequency table is created for each state for use by the Prefix Coding encoder.

In order for the decoder to start on a known state, the first symbol in the encoded stream will always be a residual.

9.3.4 RLE decoder for temporal signal coefficient group

9.3.4.1 RLE decoder overview

The Run Length state machine is used by the Prefix Coding encoding and decoding processed to know which Prefix Coding code to use for the current symbol or code word. The RLE decodes sequences of zeros and sequences of ones. It also decodes the frequency tables used to build the Prefix Coding trees.

The Run Length Decoder read the run length encoded data byte by byte. By construction the state of the first byte of data is guaranteed to be true value of the first symbol in the stream. The decoder uses the state machine shown in Figure 32 to determine the state of the next byte of data. The state tells the decoder how to interpret the current byte of data as described in subclause 9.3.4.2.

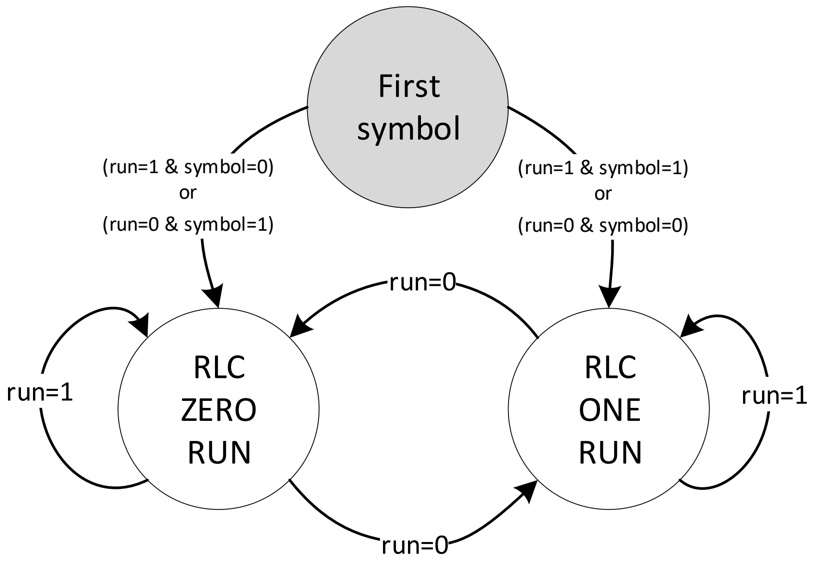

Figure 32 — RLE decoder state machine

9.3.4.2 RLE decoder description

The RLE has 2 states:

Zero Run state: this state encodes 7 bits of a zero run count. The run bit is high if more bits are needed to encode the count. Run Length encoding of a byte for Zero Run state is as shown in Figure 33:

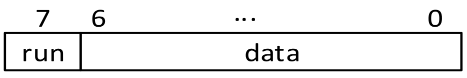

Figure 33 — Run Length encoding of a byte for Zero Run state

One Run state: this state encodes 7 bits of a one run count. The run bit is high if more bits are needed to encode the count. Run Length encoding of a byte for One Run state is as shown in Figure 34:

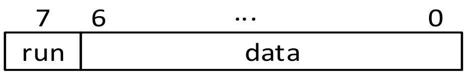

Figure 34 — Run Length encoding of a byte for One Run state

A frequency table is created for each state for use by the Prefix Coding encoder.

In order for the decoder to start on a known state, the first symbol will contain the real value 0 or 1.

RLE decoder writes the 0 and 1 values into temporal signal surface TempSigSurface of the size (PictureWidth / nTbS, PictureHeight / nTbS) where nTbS is transform size.

If temporal_tile_intra_signalling_enabled_flag is equal to 1 and if the value to write at the writing position (x, y) in the TempSigSurface is equal to 1 and x%(32/nTbS) == 0 and y%(32/nTbS) == 0, next writing position is moved to (x + 32/nTbS, y) when (x + 32/nTbS) < (PictureWidth / nTbs), otherwise it is moved to (0, y + 32/nTbS), as shown in Figure 35..

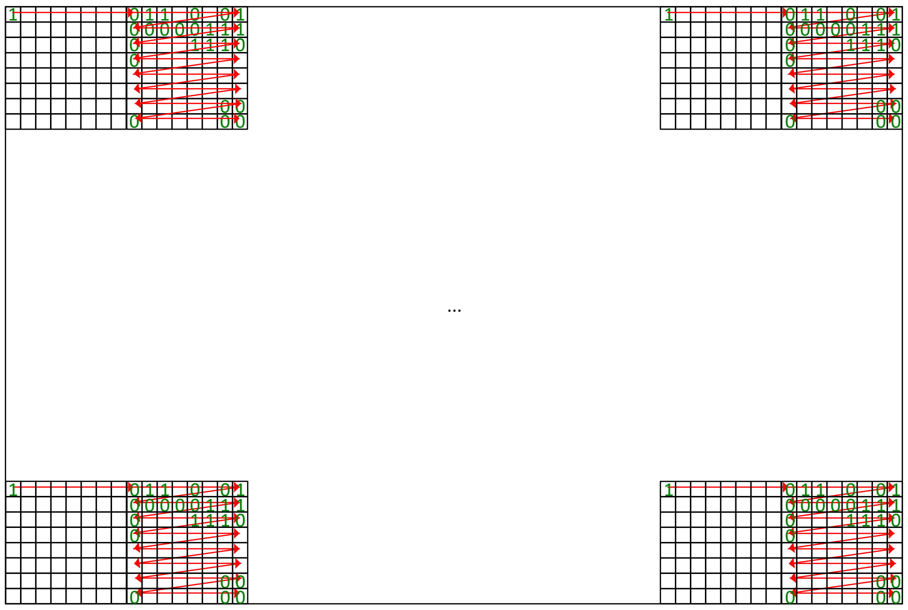

Figure 35 — Run Length Decoder writing values to the temporal signal surface (in the example, nTbs = 4)

9.3.5 RLE decoder for tile entropy_enabled_flag fields

9.3.5.1 RLE decoder overview

The Run Length state machine is used to code the entropy_enabled_flag field of each of the tiles. The RLE decodes sequences of zeros and sequences of ones.

The Run Length Decoder read the run length encoded data byte by byte. By construction the state of the first byte of data is guaranteed to be true value of the first symbol in the stream. The decoder uses the state machine shown in Figure 36 to determine the state of the next byte of data. The state tells the decoder how to interpret the current byte of data as described in subclause 9.3.5.2.

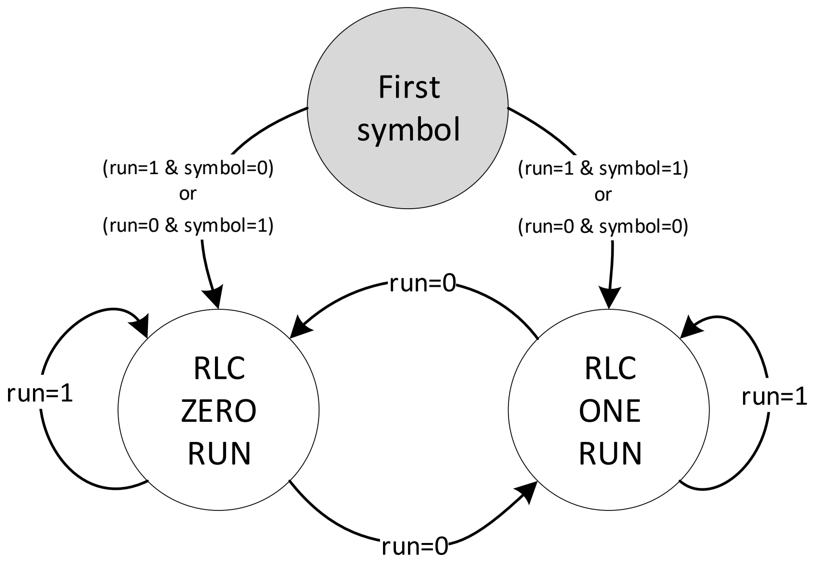

Figure 36 — RLE decoder state machine

9.3.5.2 RLE decoder description

The RLE has 2 states:

Zero Run state: this state encodes 7 bits of a zero run count. The run bit is high if more bits are needed to encode the count. Run Length encoding of a byte for Zero Run stats is as shown in Figure 37:

7	6 … 0
run	data

Figure 37 — Run Length encoding of a byte for Zero Run state

One Run state: this state encodes 7 bits of a one run count. The run bit is equal to 1 if more bits are needed to encode the count. Run Length encoding of a byte for One Run state state is as shown in Figure 38:

7	6 … 0
run	data

Figure 38 — Run Length encoding of a byte for One Run state state

In order for the decoder to start on a known state, the first symbol will contain the real value 0 or 1.

The RLE data is organized in blocks. Each block has an output capacity of 4096 bytes. The RLE switches to a new block in the following cases:

- The current block is full
- The current RLE data is a run and there is less than 5 bytes left in the current block
- The current RLE data is lead to an LSB/MSB pair and there is less than 2 bytes left in the current block

RLE decoder writes the 0 and 1 values into temporary signal surface tmp_decoded_tile_entropy_enabled of size (nPlanes) x (nLevels) x (nLayers) x (nTilesL1 + nTilesL2) x (no_enhancement_bit_flag == 0) + (temporal_signalling_present_flag == 1) x (nPlanes) x (nTilesL2) and get mapped to surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].entropy_enabled_flag and temporal_surfaces[planeIdx].tiles[tileIdx].entropy_enabled_flag as follows:

```
for (planeIdx = 0; planeIdx < nPlanes; ++planeIdx) {
    if (no_enhancement_bit_flag == 0) {
        for (levelIdx = 1; levelIdx <= nLevels; ++levelIdx) {
            if (levelIdx == 1)
                nTiles = nTilesL1
            else
                nTiles = nTilesL2
            for (layerIdx = 0; layer < nLayers; ++layerIdx) {
                for (tileIdx = 0; tileIdx < nTiles; tileIdx++) {
                    surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].entropy_enabled_flag =
                        tmp_decoded_tile_entropy_enabled[tileIdx]
                }
            }
        }
    } else {
        for (levelIdx = 1; levelIdx <= nLevels; ++levelIdx) {
            if (levelIdx == 1)
                nTiles = nTilesL1
            else
                nTiles = nTilesL2
            for (layerIdx = 0; layer < nLayers; ++layerIdx) {
                for (tileIdx = 0; tileIdx < nTiles; tileIdx++)
                    surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].entropy_enabled_flag = 0
            }
        }
    }
    if (temporal_signalling_present_flag == 1) {
        for (tileIdx = 0; tileIdx < nTilesL2; tileIdx++) {
            temporal_surfaces[planeIdx].tiles[tileIdx].entropy_enabled_flag =
                tmp_decoded_tile_entropy_enabled[tileIdx]
        }
    }
}
```

9.4 Parsing process for 0-th order Exp-Golomb codes

This process is invoked when the descriptor of a syntax element in the syntax tables is equal to ue(v).

Inputs to this process are bits from the RBSP.

Outputs of this process are syntax element values.

Syntax elements coded as ue(v) are Exp-Golomb-coded with order 0. The parsing process for these syntax elements begins with reading the bits starting at the current location in the bitstream up to and including the first non-zero bit, and counting the number of leading bits that are equal to 0. This process is specified as follows:

```
leadingZeroBits = −1
for( b = 0; !b; leadingZeroBits++ )                                    (43))
    b = read_bits( 1 )
```

The variable codeNum is then assigned as follows:

$$\text{codeNum} = (2^{\text{leadingZeroBits}} - 1) + \text{read\_bits}(\text{leadingZeroBits}) \qquad (44))$$

where the value returned from read_bits(leadingZeroBits) is interpreted as a binary representation of an unsigned integer with most significant bit written first.

Table 43 illustrates the structure of the 0-th order Exp-Golomb code by separating the bit string into "prefix" and "suffix" bits. The "prefix" bits are those bits that are parsed as specified above for the computation of leadingZeroBits, and are shown as either 0 or 1 in the bit string column of Table 43. The "suffix" bits are those bits that are parsed in the computation of codeNum and are shown as x_i in Table 43, with i in the range of 0 to leadingZeroBits − 1, inclusive. Each x_i is equal to either 0 or 1.

Table 43 – Bit strings with "prefix" and "suffix" bits and assignment to codeNum ranges (informative)

Bit string form	Range of codeNum
1	0
0 1 x_0	1..2
0 0 1 x_1 x_0	3..6
0 0 0 1 x_2 x_1 x_0	7..14
0 0 0 0 1 x_3 x_2 x_1 x_0	15..30
0 0 0 0 0 1 x_4 x_3 x_2 x_1 x_0	31..62
...	...

Table 44 illustrates explicitly the assignment of bit strings to codeNum values.

Table 44 – Exp-Golomb bit strings and codeNum in explicit form and used as ue(v) (informative)

– Bit string	– codeNum
– 1	– 0
– 0 1 0	– 1
– 0 1 1	– 2
– 0 0 1 0 0	– 3
– 0 0 1 0 1	– 4
– 0 0 1 1 0	– 5
– 0 0 1 1 1	– 6
– 0 0 0 1 0 0 0	– 7
– 0 0 0 1 0 0 1	– 8
– 0 0 0 1 0 1 0	– 9
– ...	– ...

The value of the syntax element is equal to codeNum.

Annex A

Profiles, levels and toolsets

(This annex forms an integral part of this International Standard.)

A.1 Overview of profiles, levels and toolsets

Profiles, levels and toolsets specify restrictions on the bitstreams and hence limits on the capabilities needed to decode the bitstreams. Profiles, levels and toolsets may also be used to indicate interoperability points between individual decoder implementations.

NOTE 1 This document does not include individually selectable “options” at the decoder, as this would increase interoperability difficulties.

Each profile specifies a subset of algorithmic features and limits that shall be supported by all decoders conforming to that profile.

NOTE 2 Encoders are not required to make use of any particular subset of features supported in a profile.

Each level specifies a set of limits on the values that may be taken by the syntax elements of this document. The same set of level definitions is used with all profiles, but individual implementations may support a different level for each supported profile. For any given profile, a level generally corresponds to a particular decoder processing load and memory capability.

The profiles that are specified in clause A.3 are also referred to as the profiles specified in Annex A.

A.2 Requirements on video decoder capability

Capabilities of video decoders conforming to this document are specified in terms of the ability to decode video streams conforming to the constraints of profiles and levels specified in this annex and other annexes. When expressing the capabilities of a decoder for a specified profile, the level supported for that profile should also be expressed.

Specific values are specified in this annex and other annexes for the syntax elements profile_idc and level_idc. All other values of profile_idc and level_idc are reserved for future use by ISO/IEC.

NOTE Decoders should not infer that a reserved value of profile_idc between the values specified in this document indicates intermediate capabilities between the specified profiles, as there are no restrictions on the method to be chosen by ISO/IEC for the use of such future reserved values. However, decoders should infer that a reserved value of level_idc between the values specified in this document indicates intermediate capabilities between the specified levels.

A.3 Profiles

A.3.1 Global configuration parameter sets constraints

All constraints for global configuration parameter sets that are specified are constraints for global configuration parameter sets that are activated when the bitstream is decoded.

A.3.2 Main profile

Conformance of a bitstream to the Main profile is indicated by profile_idc equal to 0.

Bitstreams conforming to the Main profile shall have the following constraints:

- Active global configuration data blocks shall have chroma_sampling_type equal to 0 or 1 only.

Decoders conforming to the Main profile at a specific level (identified by a specific value of level_idc) shall be capable of decoding all bitstreams and sublayer representations for which all of the following conditions apply:

- The bitstream is indicated to conform to the Main profile.

- The bitstream or sublayer representation is indicated to conform to a level that is lower than or equal to the specified level.

A.3.3 Main 4:4:4 profile

Conformance of a bitstream to the Main 4:4:4 profile is indicated by profile_idc equal to 1.

Bitstreams conforming to the Main 4:4:4 profile shall have the following constraints:

- Active global configuration data blocks shall have chroma_sampling_type n the range of 0 to 3, inclusive.

Decoders conforming to the Main 4:4:4 profile at a specific level (identified by a specific value of level_idc) shall be capable of decoding all bitstreams and sublayer representations for which all of the following conditions apply:

- The bitstream is indicated to conform to the Main profile.
- The bitstream or sublayer representation is indicated to conform to a level that is lower than or equal to the specified level.

A.4 Levels

Levels in this International Standard are defined based on the following two parameters: count of luma samples of output picture in time (i.e. the Output Sample Rate) and maximum input bit rate for the Coded Picture Buffer LCEVC (CPBL), as defined in Annex C.

Both sample rate and bitrate are considered on observation periods of one second.

The level limits are defined as specified in Table A.1.

Table A.1 — General Level limits

Level	**Sublevel**	**Maximum Output Sample Rate**	**Maximum CPBL bit rate (bit per second per thousand Output Samples)**	**Informative: Example Resolution and Frame Rat**
1	**0**	29,410,000	4	1280x720 (30fps)
1	**1**	29,410,000	40	1280x720 (30fps)
2	**0**	124,560,000	4	1920x1080 (60fps)
2	**1**	124,560,000	40	1920x1080 (60fps)
3	**0**	527,650,000	4	3840x2160 (60 fps)
3	**1**	527,650,000	40	3840x2160 (60 fps)
4	**0**	2,235,160,000	4	7640x4320 (60fps)
4	**1**	2,235,160,000	40	7640x4320 (60fps)

Annex B

Byte stream format

(This annex forms an integral part of this International Standard.)

B.1 Bytestream NAL unit syntax and semantics overview

This annex specifies syntax and semantics of a byte stream format specified for use by applications that deliver some or all of the NAL unit stream as an ordered stream of bytes. For bit-oriented delivery, the bit order for the byte stream format is specified to start with the MSB of the first byte, proceed to the LSB of the first byte, followed by the MSB of the second byte, etc.

The byte stream format consists of a sequence of byte stream NAL unit syntax structures. Each byte stream NAL unit syntax structure contains one 4-byte length indication followed by one nal_unit(NumBytesInNalUnit) syntax structure.

B.2 Byte stream NAL unit syntax and semantics

B.2.1 Byte stream NAL unit syntax

Byte stream NAL unit syntax is specified in Table B.1:

Table B.1 — Byte stream NAL unit syntax

Syntax	**Descriptor**
byte_stream_nal_unit( ) {	
nal_unit_length	u(32)
nal_unit(nal_unit_length)	
}	

B.2.2 Byte stream NAL unit semantics

The order of byte stream NAL units in the byte stream shall follow the decoding order of the NAL units contained in the byte stream NAL units (see clause 7.3.2). The content of each byte stream NAL unit is associated with the same access unit as the NAL unit contained in the byte stream NAL unit (see clause 7.3.2).

nal_unit_length is a 4-byte length field indicating the length of the NAL unit within the nal_unit( ) syntax structure.

Annex C

Hypothetical reference decoder

(This annex forms an integral part of this International Standard.)

C.1 General

This annex specifies the hypothetical reference decoder (HRD) and its use to check bitstream and decoder conformance.

Bitstreams is hereafter intended as the combined bitstream of both base and enhancement encoded data. Although the multiplexing of such combined data is not specified in this document the specification of HRD here must take into account the presence of both provisions.

NOTE 1 In the remainder of this section the terms NAL and NAL units, if not specified explicitly, refer to a generic NAL format (see subclause 7.3.2).

Two types of bitstreams are subject to HRD conformance checking for this International Standard. The first such combined type of bitstream, called Type I bitstream, is a NAL unit stream containing only the VCL NAL units and filler data NAL units for all access units in the bitstream, for both base and enhancement. The second type of bitstream, called a Type II bitstream, contains, in addition to the VCL NAL units and filler data NAL units for all access units in the bitstream, at least one of the following:

additional non-VCL NAL units other than filler data NAL units for either base and/or enhancement

all leading_zero_8bits, zero_byte, start_code_prefix_one_3bytes, and trailing_zero_8bits syntax elements that form a byte stream from the NAL unit stream

NOTE 2 Such a byte stream format, referring to the muxed base and enhancement encoded data, is not specified in this document.

Figure C.1 shows the types of bitstream conformance points checked by the HRD, in the example of an “Annex B” encapsulation as specified in Annex B of ISO/IEC 14496-10.

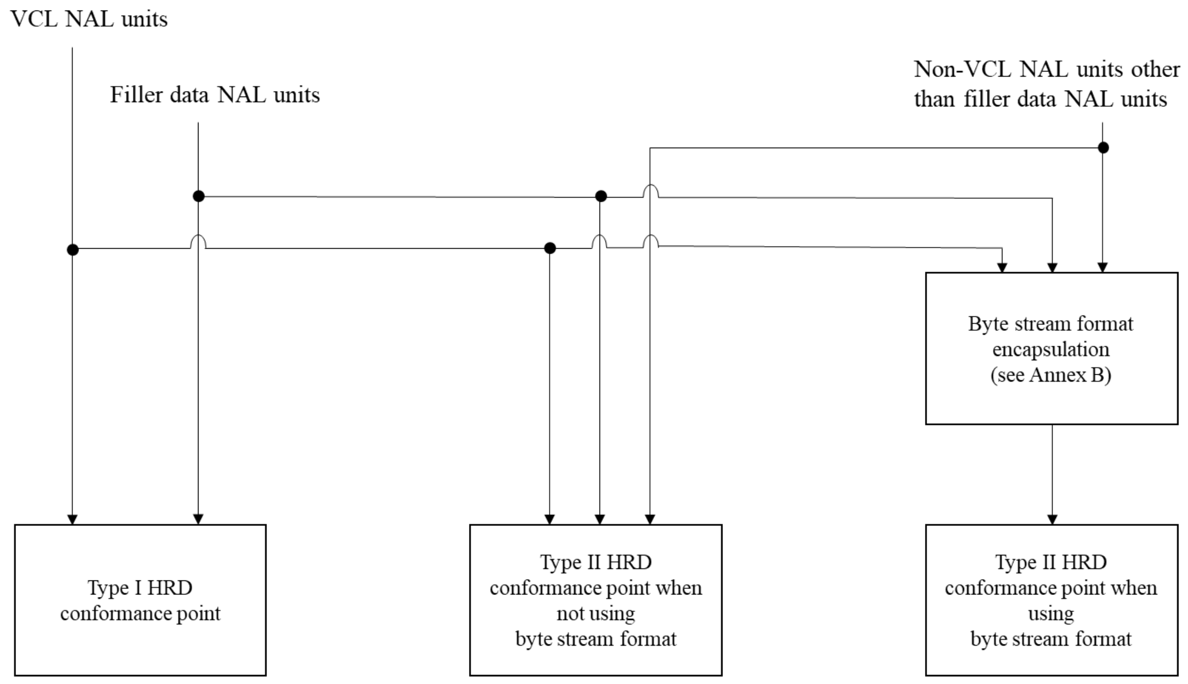
VCL NAL units
Filler data NAL units
Non-VCL NAL units other than filler data NAL units
Byte stream format encapsulation (see Annex B)
Type I HRD conformance point
Type II HRD conformance point when not using byte stream format
Type II HRD conformance point when using byte stream format

Figure C.1 — Structure of byte streams and NAL unit streams for HRD conformance checks

This section defines an HRD as an enhancement of the base HRD, intended as the HRD of the base encoding. Thus, the syntax elements of non-VCL NAL units (or their default values for some of the syntax elements), required for the base HRD, are specified in the semantic definitions relative to the base codec, typically defined in the relative Supplemental Enhancement Information (SEI) and Video Usability Information (VUI), or equivalent clauses" to read as follows "This section defines an HRD as an enhancement of the base HRD, intended as the HRD of the base encoding. Thus, the syntax elements of non-VCL NAL units (or their default values for some of the syntax elements), required for the base HRD, are specified in the semantic definitions relative to the base codec, typically defined in the relative Supplemental Enhancement Information (SEI) and Video Usability Information (VUI), or equivalent clause.

Generally, two types of HRD parameter sets are used in the base encoding. The HRD parameter sets signalled at sequence signalling level, such as global configuration parameter sets/VUI and those at Access Unit level, such as the ones in SEI.

In order to check conformance of a bitstream using this enhanced HRD, all global configuration parameter sets (or equivalent) and picture parameters sets referred to in the VCL NAL units, and corresponding buffering period and picture timing SEI messages (or equivalent) shall be conveyed to this HRD, in a timely manner, either in the bitstream (by non-VCL NAL units), or by other means not specified in this International Standard.

The specification for “presence” of non-VCL NAL units is also satisfied when those NAL units (or just some of them) are conveyed to decoders (or to the HRD) by other means not specified by this International Standard. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

NOTE 3 As an example, synchronization of a non-VCL NAL unit, conveyed by means other than presence in the bitstream, with the NAL units that are present in the bitstream, can be achieved by indicating two points in the bitstream, between which the non-VCL NAL unit would have been present in the bitstream, had the encoder decided to convey it in the bitstream.

When the content of a non-VCL NAL unit is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the non-VCL NAL unit is not required to use the same syntax specified in this annex.

NOTE 4 When HRD information is contained within the bitstream, it is possible to verify the conformance of a bitstream to the requirements of this subclause based solely on information contained in the bitstream. When the HRD information is not present in the bitstream, as is the case for all “stand-alone” Type I bitstreams, conformance can only be verified when the HRD data is supplied by some other means not specified in this International Standard.

The enhanced HRD – or simply HRD hereafter – is the logical combination of elements from both the base HRD and the LCEVC one, consisting of elements such as coded picture buffer (CPB), an instantaneous decoding process, a decoded picture buffer (DPB), and output as shown in Figure C.2.

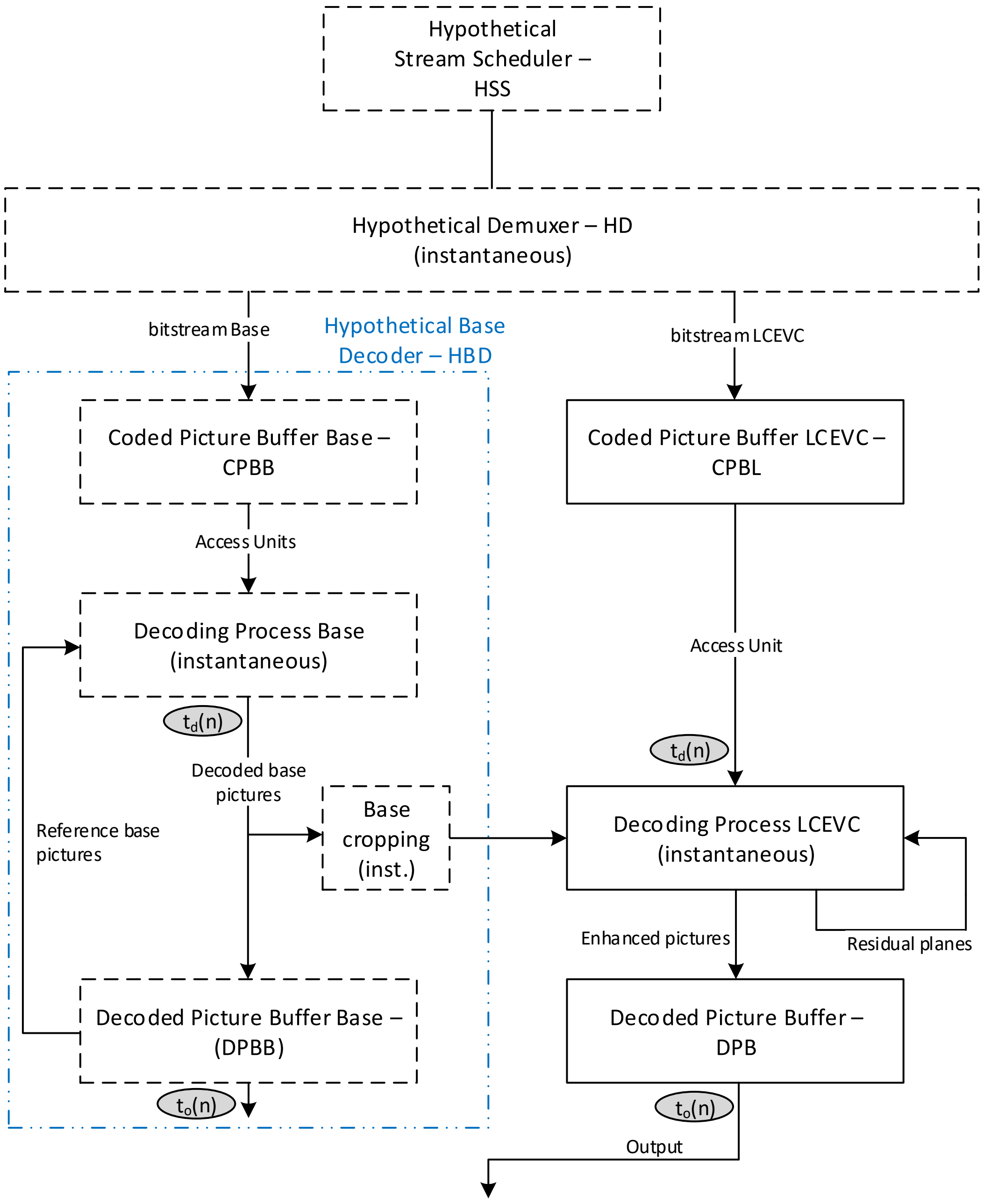
Hypothetical Stream Scheduler – HSS
Hypothetical Demuxer – HD (instantaneous)
bitstream Base
Hypothetical Base Decoder – HBD
bitstream LCEVC
Coded Picture Buffer Base – CPBB
Coded Picture Buffer LCEVC – CPBL
Access Units
Decoding Process Base (instantaneous)
Access Unit
$t_d(n)$
Decoded base pictures
Reference base pictures
Base cropping (inst.)
$t_d(n)$
Decoding Process LCEVC (instantaneous)
Enhanced pictures
Residual planes
Decoded Picture Buffer Base – (DPBB)
Decoded Picture Buffer – DPB
$t_o(n)$
$t_o(n)$
Output

Figure C.2 — HRD buffer model

The bitstream and decoder conformance requirements for the HRD are defined as the ones from the base HRD, to which the buffering and timing of the enhancement are dependent. It is assumed that, regardless of the particular base encoding type, the Hypothetical Base Decoder (HBD) operates logically as follows. Data associated with access units that flow into the base CPB (CPBB) according to a specified arrival schedule are delivered by the Hypothetical Stream Scheduler (HSS) via an Hypothetical Demuxer (HDM), intended as instantaneous. This HDM splits the encoded data between a base bitstream and an LCEVC bitstream. The data associated with each base access unit are removed and decoded instantaneously by the instantaneous decoding process at CPBB removal times. The instantaneous base decoding triggers the removal of enhancement access units from the Coded Picture Buffer LCEVC (CPBL). Both the base decoded pictures, after an instantaneous cropping if required, and the CPBL access units feed the LCEVC decoding process. The LCEVC decoding produces, instantaneously, the enhanced decoded pictures that are placed in the DPB. On the HBD side the Decoded Picture Buffer of the Base (DPBB) operates in the same way it would independently of the enhancement and obeying the conformance requirements of the base HRD. When a picture is placed in the DPB it is removed from the DPB at the base DPB output time, which is strictly the same of the DPBB output time. In such a way the combined decoding is bound to have the very same output timing characteristics of the base decoding alone.

The HRD is initialised as specified by the base HRD parameters, for ex. buffering period SEI message. The removal timing of access units from the CPBB and output timing from the DPBB, which is also the output timing from DPB, are specified in the picture timing or equivalent specification. All timing information relating to a specific access unit shall arrive prior to the CPBB removal time of the access unit.

NOTE 5 While conformance is guaranteed under the assumption that all frame-rates and clocks used to generate the bitstream match exactly the values signalled in the bitstream, in a real system each of these may vary from the signalled or specified value.

All the arithmetic in this annex is done with real values, so that no rounding errors can propagate. For example, the number of bits in a CPBB just prior to or after removal of an access unit is not necessarily an integer.

The variable tc is derived as follows and is called a clock tick.

$$tc = num\_units\_in\_tick \,/\, time\_scale \tag{C.1}$$

The following is specified for expressing the constraints in this Annex.

Let access unit n be the n-th access unit in decoding order with the first access unit being access unit 0.

Let picture n be the coded picture or the decoded picture of access unit n.

C.2 Operation of base coded picture buffer (CPBB)

C.2.1 General

The specifications in this subclause apply independently to each set of CPB parameters that is present and to both the Type I and Type II conformance points shown in Figure C.1.

C.2.2 Timing of bitstream arrival

The timing of bitstream arrival in CPPB is determined by the base HRD timing of bitstream arrival.

The timing of bitstream arrival for CPBL shall be so that:

$t_{baf}(n) < t_{laf}(n)$ for every n

where $t_{baf}(n)$ is the final arrival time of access unit n in the enhancement bitstream and $t_{laf}(n)$ is the final arrival time of access unit n in the base bitstream. This means that the enhancement data cannot arrive later than the base data referring to the same picture.

C.2.3 Timing of coded picture removal

The timing of coded picture removal from CPBB, indicated as $t_{br}(n)$, is determined by the base HRD timing of coded picture removal.

The timing of coded picture removal from CPBL, indicated as $t_{lr}(n)$, is the time the base access unit n has been decoded in the HBD. However, since the decoding is instantaneous in the HBD the two times coincide $t_{lr}(n) = t_{br}(n)$.

In practical terms the decoded picture from the HBD triggers the removal from CPBL and thus the decoding of the enhanced picture.

C.3 Operation of the decoded picture buffer (DPB)

C.3.1 General

The decoded picture buffer contains picture storage buffers waiting to be output at the presentation time. There is only one DPB in the HRD, since there is only one output as enhanced picture.

NOTE The operation of the DPBB, holding reference base pictures, and its conformance requirements are determined by the HBD specification, which is outside the scope of this section.

C.3.2 Removal of pictures from the DPB

The removal of pictures from the DPB is based on the output time, i.e. the presentation time as specified in the HBD.

C.3.3 Picture decoding and output

Picture n is output from DPB at output time $t_o(n)$ determined by the base HRD model. This is typically based on a “dpb output delay”. The details of this dpb output delay or equivalent signalling are not defined in this document.

C.4 Bitstream conformance

A bitstream of coded data conforming to this International Standard fulfils the following requirements.

The bitstream is constructed according to the syntax, semantics, and constraints specified in this International Standard outside of this Annex.

The bitstream is tested by the base HRD as specified in the base HRD bitstream conformance specification.

A CPB overflow is specified as the condition in which the total number of bits in the CPB is larger than the CPB size. Neither the CPBB nor the CPBL shall ever overflow.

A CPB underflow is specified as the condition in which the removal time is lower than the final arrival time for an access unit. Neither the CPBB nor the CPBL shall ever underflow.

Immediately after any decoded picture is added to the DPB, the fullness of the DPB shall be less than or equal to the DPB size as constrained by the HBD conformance specification for the operating point in use, i.e. typically profile and level specified in the bitstream.

All reference pictures shall be available at all times in the internal DPB of the HBD when needed for base decoding prediction. Each picture shall be present in the DPB at its DPB output time unless it is not stored in the DPB at all, or is removed from the DPB according to the HBD specification.

The difference between the output time of a picture and that of the picture immediately following it in output order, shall satisfy the constraint of the HBD for the operating point, such as profile and level, signalled in the bitstream.

C.5 Decoder conformance

C.5.1 General

A decoder conforming to this International Standard fulfils the following requirements.

A decoder claiming conformance to this specification shall be able to decode successfully all conforming bitstreams specified for decoder conformance in subclause C.4**Error! Reference source not found.**, provided that all base HRD related parameters sets referred to in the VCL NAL units, and appropriate buffering period and picture timing metadata are conveyed to the decoder, in a timely manner, either in the bitstream (by non-VCL NAL units), or by external means not specified by this International Standard.

There are two types of conformance that can be claimed by a decoder: output timing conformance and output order conformance.

To check conformance of a decoder, test bitstreams conforming to the claimed operating point, such as profile and level, as specified by subclause C.4, are delivered by a hypothetical stream scheduler (HSS) both to the HRD and to the decoder under test (DUT). All pictures output by the HRD shall also be output by the DUT and, for each picture output by the HRD, the values of all samples that are output by the DUT for the corresponding picture shall be equal to the values of the samples output by the HRD.

For output timing decoder conformance, the HSS operates as described above, with delivery schedules selected only from the subset of values as specified for the base HRD. The same delivery schedule is used for both the HRD and DUT.

For output timing decoder conformance, an HRD as described above is used and the timing (relative to the delivery time of the first bit) of picture output is the same for both HRD and the DUT up to a fixed delay.

For output order decoder conformance, the HSS delivers the bitstream to the DUT “by demand” from the DUT, meaning that the HSS delivers bits (in decoding order) only when the DUT requires more bits to proceed with its processing.

NOTE This means that for this test, the CPBB and CPBL of the DUT could be as small as the size of the largest access unit, respectively for base and enhancement.

A modified HRD as described below is used, and the HSS delivers the bitstream to the HRD, if applicable, by one of the schedules specified in the bitstream such that the bit rate and CPB size are restricted according to the HBD specification. The order of pictures output shall be the same for both HRD and the DUT.

For output order decoder conformance, the HRD CPBB and CPBL sizes are equal to the base HBD ones for the selected operating point while the DPB size is equal to default maximum as per HBD spceification. Removal time from the CPBB for the HRD is equal to final bit arrival time and decoding is immediate. The operation of the DPB of this HRD is described below.

C.5.2 Operation of the output order DPB

C.5.2.1 General

The decoded picture buffer contains picture storage buffers. Prior to initialisation, the DPB is empty (the DPB fullness is set to zero). The following steps all happen instantaneously when an enhanced picure is decoded, and in the order listed.

C.5.2.2 Removal of pictures from the DPB

The removal of pictures from the DPB proceeds as follows.

When a picture is placed in the DPB it is removed from the DPB at the base DPB output time, as specified in the base HRD bitstream.

NOTE This refers to the DPB output from the enhanced HRD, not the internal HBD one.

C.5.2.3 Current picture decoding, storage, and marking

The current picture is decoded and stored in an empty picture storage buffer in the DPB. No marking is applied.

C.5.2.4 “Bumping” process

No “bumping” process is defined for this HRD.

Annex D

Supplemental enhancement information

(This annex forms an integral part of this International Standard.)

D.1 General

This annex specifies syntax and semantics for SEI message payloads.

SEI messages assist in processes related to decoding, display or other purposes. However, SEI messages are not required for constructing the luma or chroma samples by the decoding process. Conforming decoders are not required to process this information for output order conformance to this document (see Annex C for the specification of conformance). Some SEI message information is required to check bitstream conformance and for output timing decoder conformance.

In Annex C including its subclauses, specification for presence of SEI messages are also satisfied when those messages (or some subset of them) are conveyed to decoders (or to the HRD) by other means not specified in this document. When present in the bitstream, SEI messages shall obey the syntax and semantics specified in clause 7.3.5 and this annex. When the content of an SEI message is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the SEI message is not required to use the same syntax specified in this annex. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

D.2 SEI payload syntax

D.2.1 General SEI message syntax

The general SEI message syntax is specified in Table D.1.

Table D.1 — General SEI message syntax

Syntax	**Descriptor**
sei_payload(payloadType, payloadSize) {	
if (payloadType == 1)	
mastering_display_colour_volume(payloadSize)	
else if (payloadType == 2)	
content_light_level_info(payloadSize)	
else	
reserved_sei_message(payloadSize)	
if (more_data_in_payload( )) {	
if (payload_extension_present( ))	
reserved_payload_extension_data	u(v)
payload_bit_equal_to_one /* equal to 1 */	f(1)
while (!byte_aligned( ))	
payload_bit_equal_to_zero /* equal to 0 */	f(1)
}	
}	

D.2.2 Mastering display colour volume SEI message syntax

The mastering display color volume SEI message syntax is specified in Table D.2.

Table D.2 — Mastering display colour volume SEI message syntax

Syntax	**Descriptor**
mastering_display_colour_volume(payloadSize) {	
for(c = 0; c < 3; c++) {	
display_primaries_x[c]	u(16)
display_primaries_y[c]	u(16)
}	
white_point_x	u(16)

Syntax	Descriptor
white_point_y	u(16)
max_display_mastering_luminance	u(32)
min_display_mastering_luminance	u(32)
}	

D.2.3 Content light level information SEI message syntax

The content light level information SEI message syntax is specified inTable D.3.

Table D.3 — Content light level information SEI message syntax

Syntax	Descriptor
content_light_level_info(payloadSize) {	
max_content_light_level	u(16)
max_pic_average_light_level	u(16)
}	

D.2.4 Reserved SEI message syntax

Reserves SEI message syntax is specified in Table D.4.

Table D.4 — Reserved SEI message syntax

Syntax	Descriptor
reserved_sei_message(payloadSize) {	
for(i = 0; i < payloadSize; i++)	b(8)
reserved_sei_message_payload_byte	
}	

D.3 SEI payload semantics

D.3.1 General SEI payload semantics

reserved_payload_extension_data shall not be present in bitstreams conforming to this version of this document. However, decoders conforming to this version of this document shall ignore the presence and value of reserved_payload_extension_data. When present, the length, in bits, of reserved_payload_extension_data is equal to 8 * payloadSize − nEarlierBits − nPayloadZeroBits − 1, where nEarlierBits is the number of bits in the sei_payload( ) syntax structure that precede the reserved_payload_extension_data syntax element and nPayloadZeroBits is the number of payload_bit_equal_to_zero syntax elements at the end of the sei_payload( ) syntax structure.

payload_bit_equal_to_one shall be equal to 1.

payload_bit_equal_to_zero shall be equal to 0.

The semantics and persistence scope for each SEI message are specified in the semantics specification for each particular SEI message.

NOTE Persistence information for SEI messages is informatively summarized in Table D.5.

Table D.5 — Persistence scope of SEI messages (informative)

SEI message	Persistence scope
Mastering display colour volume	The CVS containing the SEI message
Content light level information	The CVS containing the SEI message

D.3.2 Mastering display colour volume SEI message semantics

This SEI message identifies the colour volume (the colour primaries, white point, and luminance range) of a display considered to be the mastering display for the associated video content – e.g., the colour volume of a display that was used for viewing while authoring the video content. The described mastering display is a three-colour additive display system that has been configured to use the indicated mastering colour volume.

This SEI message does not identify the measurement methodologies and procedures used for determining the indicated values or provide any description of the mastering environment. It also does not provide information on colour transformations that would be appropriate to preserve creative intent on displays with colour volumes different from that of the described mastering display.

The information conveyed in this SEI message is intended to be adequate for purposes corresponding to the use of SMPTE ST 2086 (2018).

When a mastering display colour volume SEI message is present for any picture of a CLVS of a particular layer, a mastering display colour volume SEI message shall be present for the first picture of the CLVS. The mastering display colour volume SEI message persists for the current layer in decoding order from the current picture until the end of the CLVS. All mastering display colour volume SEI messages that apply to the same CLVS shall have the same content.

display_primaries_x[c], when in the range of 5 to 37 000, inclusive, specifies the normalized x chromaticity coordinate of the colour primary component c of the mastering display, according to the CIE 1931 definition of x as specified in ISO 11664-1 (see also ISO 11664-3 and CIE 15), in increments of 0.00002. When display_primaries_x[c] is not in the range of 5 to 37 000, inclusive, the normalized x chromaticity coordinate of the colour primary component c of the mastering display is unknown or unspecified or specified by other means not specified in this Specification.

display_primaries_y[c], when in the range of 5 to 42 000, inclusive, specifies the normalized y chromaticity coordinate of the colour primary component c of the mastering display, according to the CIE 1931 definition of y as specified in ISO 11664-1 (see also ISO 11664-3 and CIE 15), in increments of 0.00002. When display_primaries_y[c] is not in the range of 5 to 42 000, inclusive, the normalized y chromaticity coordinate of the colour primary component c of the mastering display is unknown or unspecified or specified by other means not specified in this Specification.

For describing mastering displays that use red, green, and blue colour primaries, it is suggested that index value c equal to 0 should correspond to the green primary, c equal to 1 should correspond to the blue primary and c equal to 2 should correspond to the red colour primary (see also Annex E).

white_point_x, when in the range of 5 to 37 000, inclusive, specifies the normalized x chromaticity coordinate of the white point of the mastering display, according to the CIE 1931 definition of x as specified in ISO 11664-1 (see also ISO 11664-3 and CIE 15), in normalized increments of 0.00002. When white_point_x is not in the range of 5 to 37 000, inclusive, the normalized x chromaticity coordinate of the white point of the mastering display is indicated to be unknown or unspecified or specified by other means not specified in this Specification.

white_point_y, when in the range of 5 to 42 000, inclusive, specifies the normalized y chromaticity coordinate of the white point of the mastering display, according to the CIE 1931 definition of y as specified in ISO 11664-1 (see also ISO 11664-3 and CIE 15), in normalized increments of 0.00002. When white_point_y is not in the range of 5 to 42 000, inclusive, the normalized y chromaticity coordinate of the white point of the mastering display is indicated to be unknown or unspecified or specified by other means not specified in this Specification.

NOTE 1 SMPTE ST 2086 (2018) specifies that the normalized x and y chromaticity coordinate values for the mastering display colour primaries and white point are to be represented with four decimal places. This would correspond with using values of the syntax elements display_primaries_x[c], display_primaries_y[c], white_point_x, and white_point_y, as defined in this Specification, that are multiples of 5.

NOTE 2 An example of the use of values outside the range for which semantics are specified in this Specification is that ANSI/CTA 861-G (2016) uses normalized (x, y) chromaticity coordinate values of (0,0) for the white point to indicate that the white point chromaticity is unknown.

max_display_mastering_luminance, when in the range of 50 000 to 100 000 000, specifies the nominal maximum display luminance of the mastering display in units of 0.0001 candelas per square metre. When max_display_mastering_luminance is not in the range of 50 000 to 100 000 000, the nominal maximum display luminance of the mastering display is indicated to be unknown or unspecified or specified by other means not specified in this Specification.

NOTE 3 SMPTE ST 2086 (2018) specifies that the nominal maximum display luminance of the mastering display is to be specified as a multiple of 1 candela per square meter. This would correspond with using values of the syntax element max_display_mastering_luminance, as defined in this Specification, that are a multiple of 10 000.

NOTE 4 An example of the use of values outside the range for which semantics are specified in this Specification is that ANSI/CTA 861-G (2016) uses the value 0 for the nominal maximum display luminance of the mastering display to indicate that the nominal maximum display luminance of the mastering display is unknown.

min_display_mastering_luminance, when in the range of 1 to 50 000, specifies the nominal minimum display luminance of the mastering display in units of 0.0001 candelas per square metre. When min_display_mastering_luminance is not in the range of 1 to 50 000, the nominal maximum display luminance of the mastering display is unknown or unspecified or specified by other means not specified in this Specification. When max_display_mastering_luminance is equal to 50 000, min_display_mastering_luminance shall not be equal to 50 000.

NOTE 5 SMPTE ST 2086 (2018) specifies that the nominal minimum display luminance of the mastering display is to be specified as a multiple of 0.0001 candelas per square metre, which corresponds to the semantics specified in this Specification.

NOTE 6 An example of the use of values outside the range for which semantics are specified in this Specification is that ANSI/CTA 861-G (2016) uses the value 0 for the nominal minimum display luminance of the mastering display to indicate that the nominal minimum display luminance of the mastering display is unknown.

NOTE 7 Another example of the potential use of values outside the range for which semantics are specified in this Specification is that SMPTE ST 2086 (2018) indicates that values outside the specified range could be used to indicate that the black level and contrast of the mastering display have been adjusted using picture line-up generation equipment (PLUGE).

At the minimum luminance, the mastering display is considered to have the same nominal chromaticity as the white point.

D.3.3 Content light level information SEI message semantics

This SEI message identifies upper bounds for the nominal target brightness light level of the pictures of the CLVS.

The information conveyed in this SEI message is intended to be adequate for purposes corresponding to the use of the Consumer Electronics Association 861.3 specification.

The semantics of the content light level information SEI message are defined in relation to the values of samples in a 4:4:4 representation of red, green, and blue colour primary intensities in the linear light domain for the pictures of the CLVS, in units of candelas per square metre. However, this SEI message does not, by itself, identify a conversion process for converting the sample values of a decoded picture to the samples in a 4:4:4 representation of red, green, and blue colour primary intensities in the linear light domain for the picture.

NOTE 1 Other syntax elements, such as colour_primaries, transfer_characteristics, matrix_coeffs, and the chroma resampling filter hint SEI message, when present, may assist in the identification of such a conversion process.

Given the red, green, and blue colour primary intensities in the linear light domain for the location of a luma sample in a corresponding 4:4:4 representation, denoted as E_R, E_G, and E_B, the maximum component intensity is defined as $E_{Max} = \text{Max}(E_R, \text{Max}(E_G, E_B))$. The light level corresponding to the stimulus is then defined as the CIE 1931 luminance corresponding to equal amplitudes of E_{Max} for all three colour primary intensities for red, green, and blue (with appropriate scaling to reflect the nominal luminance level associated with peak white – e.g., ordinarily scaling to associate peak white with 10 000 candelas per square metre when transfer_characteristics is equal to 16).

NOTE 2 Since the maximum value E_{Max} is used in this definition at each sample location, rather than a direct conversion from E_R, E_G, and E_B to the corresponding CIE 1931 luminance, the CIE 1931 luminance at a location may in some cases be less than the indicated light level. This situation would occur, for example, when E_R and E_G are very small and E_B is large, in which case the indicated light level would be much larger than the true CIE 1931 luminance associated with the (E_R, E_G, E_B) triplet.

All content light level information SEI messages that apply to the same CLVS shall have the same content.

max_content_light_level, when not equal to 0, indicates an upper bound on the maximum light level among all individual samples in a 4:4:4 representation of red, green, and blue colour primary intensities (in the linear light domain) for the pictures of the CLVS, in units of candelas per square metre. When equal to 0, no such upper bound is indicated by max_content_light_level.

max_pic_average_light_level, when not equal to 0, indicates an upper bound on the maximum average light level among the samples in a 4:4:4 representation of red, green, and blue colour primary intensities (in the linear light domain) for any individual picture of the CLVS, in units of candelas per square metre. When equal to 0, no such upper bound is indicated by max_pic_average_light_level.

NOTE 3 When the visually relevant region does not correspond to the entire cropped decoded picture, such as for "letterbox" encoding of video content with a wide picture aspect ratio within a taller cropped decoded picture, the indicated average should be performed only within the visually relevant region.

D.3.4 Reserved SEI message semantics

The reserved SEI message consists of data reserved for future backward-compatible use by ITU-T | ISO/IEC. It is a requirement of bitstream conformance that bitstreams shall not contain reserved SEI messages until and unless the use of such messages has been specified by ITU-T | ISO/IEC. Decoders shall ignore reserved SEI messages.

Annex E

Video usability information

(This annex forms an integral part of this International Standard.)

E.1 General

This Annex specifies syntax and semantics of the VUI parameters.

VUI parameters are not required for constructing the luma or chroma samples by the decoding process. Conforming decoders are not required to process this information for output order conformance to this International Standard (see Annex Cfor the specification of conformance). Some VUI parameters are required to check bitstream conformance and for output timing decoder conformance.

In Annex E, specification for presence of VUI parameters is also satisfied when those parameters (or some subset of them) are conveyed to decoders (or to the HRD) by other means not specified by this International Standard. When present in the bitstream, VUI parameters shall follow the syntax and semantics specified in subclauses E.2 and E.3 and this annex. When the content of VUI parameters is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the VUI parameters is not required to use the same syntax specified in this annex. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

E.2 VUI parameters syntax

The VUI parameters syntax is specified in Table E.1.

Table E.1 — VUI parameters syntax

Syntax	**C**	**Descriptor**
vui_parameters (payload_size) {		
aspect_ratio_info_present_flag	0	u(1)
if (aspect_ratio_info_present_flag) {		
aspect_ratio_idc	0	u(8)
if (aspect_ratio_idc == Extended_SAR) {		
sar_width	0	u(16)
sar_height	0	u(16)
}		
}		
overscan_info_present_flag	0	u(1)
if (overscan_info_present_flag)		
overscan_appropriate_flag	0	u(1)
video_signal_type_present_flag	0	u(1)
if (video_signal_type_present_flag) {		
video_format	0	u(3)
video_full_range_flag	0	u(1)
colour_description_present_flag	0	u(1)
if (colour_description_present_flag) {		
colour_primaries	0	u(8)
transfer_characteristics	0	u(8)
matrix_coefficients	0	u(8)
}		
}		
chroma_loc_info_present_flag	0	u(1)
if (chroma_loc_info_present_flag) {		
chroma_sample_loc_type_top_field	0	ue(v)
chroma_sample_loc_type_bottom_field	0	ue(v)
}		
timing_info_present_flag	0	u(1)
if (timing_info_present_flag) {		

Syntax	C	Descriptor
num_units_in_tick	0	u(32)
time_scale	0	u(32)
fixed_pic_rate_flag	0	u(1)
}		
pic_struct_present_flag	0	u(1)
bitstream_restriction_flag	0	u(1)
if (bitstream_restriction_flag) {		
motion_vectors_over_pic_boundaries_flag	0	u(1)
max_bytes_per_pic_denom	0	ue(v)
max_bits_per_mb_denom	0	ue(v)
log2_max_mv_length_horizontal	0	ue(v)
log2_max_mv_length_vertical	0	ue(v)
num_reorder_pics	0	ue(v)
max_dec_pic_buffering	0	ue(v)
}		
}		

E.3 VUI parameters semantics

aspect_ratio_info_present_flag equal to 1 specifies that aspect_ratio_idc is present. aspect_ratio_info_present_flag equal to 0 specifies that aspect_ratio_idc is not present.

aspect_ratio_idc, sar_width and **sar_height** shall have the meaning of SampleAspectRatio, SarWidth and SarHeight, respectively, as specified in Section 8.6 of ITU-T H.273 | ISO/IEC 23091-2.

overscan_info_present_flag equal to 1 specifies that the overscan_appropriate_flag is present. When overscan_info_present_flag is equal to 0 or is not present, the preferred display method for the video signal is unspecified.

overscan_appropriate_flag equal to 1 indicates that the cropped decoded pictures output are suitable for display using overscan. overscan_appropriate_flag equal to 0 indicates that the cropped decoded pictures output contain visually important information in the entire region out to the edges of the cropping rectangle of the picture, such that the cropped decoded pictures output should not be displayed using overscan. Instead, they should be displayed using either an exact match between the display area and the cropping rectangle, or using underscan.

NOTE 1 For example, overscan_appropriate_flag equal to 1 might be used for entertainment television programming, or for a live view of people in a videoconference, and overscan_appropriate_flag equal to 0 might be used for computer screen capture or security camera content.

video_signal_type_present_flag equal to 1 specifies that video_format, video_full_range_flag and colour_description_present_flag are present. video_signal_type_present_flag equal to 0, specify that video_format, video_full_range_flag and colour_description_present_flag are not present.

video_format indicates the representation of the pictures as specified in Table E.2, before being coded in accordance with this International Standard. When the video_format syntax element is not present, video_format value shall be inferred to be equal to 5.

Table E.2 — Meaning of video_format

video_format	Meaning
0	Component
1	PAL
2	NTSC
3	SECAM
4	MAC
5	Unspecified video format
6	Reserved
7	Reserved

video_full_range_flag indicates the black level and range of the luma and chroma signals as derived from E'$_{Y}$, E'$_{PB}$, and E'$_{PR}$ or E'$_{R}$, E'$_{G}$, and E'$_{B}$ analogue component signals.

When the video_full_range_flag syntax element is not present, the value of video_full_range_flag shall be inferred to be equal to 0.

colour_description_present_flag equal to 1 specifies that colour_primaries, transfer_characteristics and matrix_coefficients are present. colour_description_present_flag equal to 0 specifies that colour_primaries, transfer_characteristics and matrix_coefficients are not present.

colour_primaries shall have the meaning of ColourPrimaries as specified in specified in Section 8.1 of ITU-T H.273 | ISO/IEC 23091-2.

transfer_characteristics shall have the meaning of TransferCharacteristics as specified in Section 8.2 of ITU-T H.273 | ISO/IEC 23091-2.

matrix_coefficients shall have the meaning of MatrixCoefficients as specified in as specified in Section 8.3 of ITU-T H.273 | ISO/IEC 23091-2.

chroma_loc_info_present_flag equal to 1 specifies that chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field are present. chroma_loc_info_present_flag equal to 0 specifies that chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field are not present.

chroma_sample_loc_type_top_field and **chroma_sample_loc_type_bottom_field** specify the location of chroma samples for the top field and the bottom field as shown in Figure E.1. The value of chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field shall be in the range of 0 to 5, inclusive. When the chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field are not present, the values of chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field shall be inferred to be equal to 0.

NOTE 2 When coding progressive source material, chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field should have the same value.

Location of chroma samples for top and bottom fields as a function of chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field are shown in Figure E.1.

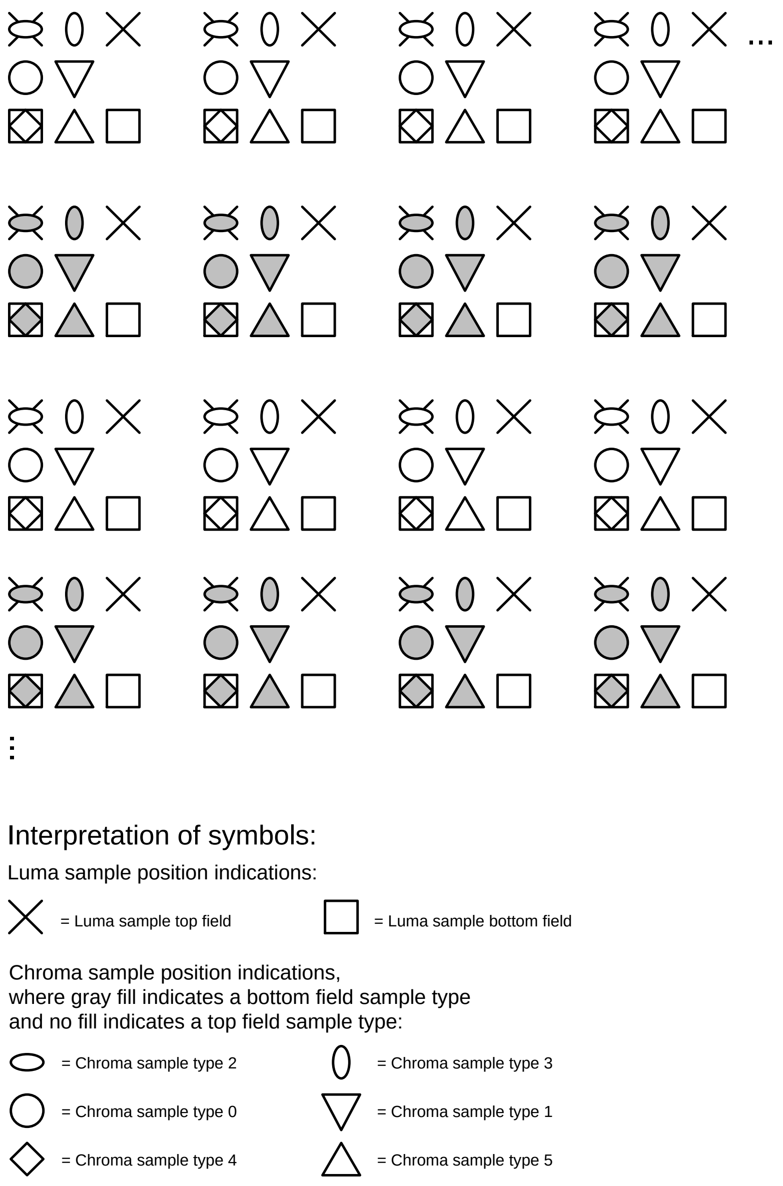
...
⋮
Interpretation of symbols:
Luma sample position indications:
= Luma sample top field
= Luma sample bottom field
Chroma sample position indications, where gray fill indicates a bottom field sample type and no fill indicates a top field sample type:
= Chroma sample type 2
= Chroma sample type 3
= Chroma sample type 0
= Chroma sample type 1
= Chroma sample type 4
= Chroma sample type 5

Figure E.1 — Chroma location samples

timing_info_present_flag equal to 1 specifies that num_units_in_tick, time_scale and fixed_pic_rate_flag are present in the bitstream. timing_info_present_flag equal to 0 specifies that num_units_in_tick, time_scale and fixed_pic_rate_flag are not present in the bitstream.

num_units_in_tick is the number of time units of a clock operating at the frequency time_scale Hz that corresponds to one increment (called a clock tick) of a clock tick counter. num_units_in_tick shall be greater than 0. A clock tick is the minimum interval of time that can be represented in the coded data. For example, when the clock frequency of a video signal is 60 000 ÷ 1001 Hz, time_scale may be equal to 60 000 and num_units_in_tick may be equal to 1001. See Equation (C.1).

time_scale is the number of time units that pass in one second. For example, a time coordinate system that measures time using a 27 MHz clock has a time_scale of 27 000 000. time_scale shall be greater than 0.

fixed_pic_rate_flag equal to 1 indicates that the temporal distance between the HRD output times of any two consecutive pictures in output order is constrained as follows. fixed_pic_rate_flag equal to 0 indicates that no such constraints apply to the temporal distance between the HRD output times of any two consecutive pictures in output order.

For each picture n where n indicates the n-th picture (in output order) that is output and picture n is not the last picture in the bitstream (in output order) that is output, the value of $\Delta t_{fi,dpb}(n)$ is specified by

$$\Delta t_{fi,dpb}(n) = \Delta t_{o,dpb}(n) \div \text{DeltaTfiDivisor} \quad \text{(E.1)}$$

When fixed_pic_rate_flag is equal to 1 for a coded video sequence containing picture n, the value computed for $\Delta t_{fi,dpb}(n)$ shall be equal to t_c as specified in Equation (C.1) (using the value of t_c for the coded video sequence containing picture n) when either or both of the following conditions are true for the following picture n_n that is specified for use in Equation C-13.

- picture n_n is in the same coded video sequence as picture n.
- picture n_n is in a different coded video sequence and fixed_pic_rate_flag is equal to 1 in the coded video sequence containing picture n_n and the value of num_units_in_tick ÷ time_scale is the same for both coded video sequences.

bitstream_restriction_flag equal to 1, specifies that the following coded video sequence bitstream restriction parameters are present. bitstream_restriction_flag equal to 0, specifies that the following coded video sequence bitstream restriction parameters are not present.

max_bytes_per_pic_denom indicates a number of bytes not exceeded by the sum of the sizes of the VCL NAL units associated with any coded picture in the coded video sequence.

The number of bytes that represent a picture in the NAL unit stream is specified for this purpose as the total number of bytes of VCL NAL unit data (i.e., the total of the NumBytesInNALunit variables for the VCL NAL units) for the picture. The value of max_bytes_per_pic_denom shall be in the range of 0 to 16, inclusive.

Depending on max_bytes_per_pic_denom the following applies.

If max_bytes_per_pic_denom is equal to 0, no limits are indicated.

Otherwise (max_bytes_per_pic_denom is not equal to 0), no coded picture shall be represented in the coded video sequence by more than the following number of bytes.

$$(\text{PicSizeInMbs} * \text{RawMbBits}) \div (8 * \text{max\_bytes\_per\_pic\_denom}) \quad \text{(E.2)}$$

When the max_bytes_per_pic_denom syntax element is not present, the value of max_bytes_per_pic_denom shall be inferred to be equal to 2.

max_bits_per_mb_denom indicates the maximum number of coded bits of macroblock_layer( ) data for any macroblock in any picture of the coded video sequence. The value of max_bits_per_mb_denom shall be in the range of 0 to 16, inclusive.

Depending on max_bits_per_mb_denom the following applies.

If max_bits_per_mb_denom is equal to 0, no limit is specified.

Otherwise (max_bits_per_mb_denom is not equal to 0), no coded macroblock_layer( ) shall be represented in the bitstream by more than the following number of bits.

$$(128 + \text{RawMbBits}) \div \text{max\_bits\_per\_mb\_denom} \quad \text{(E.3)}$$

Depending on entropy_coding_mode_flag, the bits of macroblock_layer( ) data are counted as follows.

If entropy_coding_mode_flag is equal to 0, the number of bits of macroblock_layer( ) data is given by the number of bits in the macroblock_layer( ) syntax structure for a macroblock.

Otherwise (entropy_coding_mode_flag is equal to 1), the number of bits of macroblock_layer( ) data for a macroblock is given by the number of times read_bits(1) is called in subclauses 9.3.3.2.2 and 9.3.3.2.3 when parsing the macroblock_layer( ) associated with the macroblock.

When the max_bits_per_mb_denom is not present, the value of max_bits_per_mb_denom shall be inferred to be equal to 1.

log2_max_mv_length_horizontal and **log2_max_mv_length_vertical** indicate the maximum absolute value of a decoded horizontal and vertical motion vector component, respectively, in ¼ luma sample units, for all pictures in the coded video sequence. A value of n asserts that no value of a motion vector component shall exceed the range from -2^n to $2^n - 1$, inclusive, in units of ¼ luma sample displacement. The value of log2_max_mv_length_horizontal shall be in the range of 0 to 16, inclusive. The value of log2_max_mv_length_vertical shall be in the range of 0 to 16, inclusive. When log2_max_mv_length_horizontal is not present, the values of log2_max_mv_length_horizontal and log2_max_mv_length_vertical shall be inferred to be equal to 16.

NOTE 9 The maximum absolute value of a decoded vertical or horizontal motion vector component is also constrained by profile and level limits as specified in Annex A.

num_reorder_pics indicates the maximum number of frames, complementary field pairs, or non-paired fields that precede any frame, complementary field pair, or non-paired field in the coded video sequence in decoding order and follow it in output order. The value of num_reorder_pics shall be in the range of 0 to max_dec_pic_buffering, inclusive. When the num_reorder_pics syntax element is not present, the value of num_reorder_pics value shall be inferred to be equal to max_dec_pic_buffering.

max_dec_pic_buffering specifies the required size of the HRD decoded picture buffer (DPB) in units of picture storage buffers. The coded video sequence shall not require a decoded picture buffer with size of more than Max(1, max_dec_pic_buffering) picture storage buffers to enable the output of decoded pictures at the output times specified by dpb_output_delay of the picture timing SEI messages. The value of max_dec_pic_buffering shall be in the range of num_ref_pics to MaxDpbSize (as specified in subclause A.3.1 or A.3.2), inclusive. When the max_dec_pic_buffering syntax element is not present, the value of max_dec_pic_buffering shall be inferred to be equal to MaxDpbSize.