INTERNATIONAL STANDARD

ISO/IEC 14496-29

First edition
2015-04-01

Information technology — Coding of audio-visual objects —

Part 29: Web video coding

Technologies de l'information — Codage des objets audiovisuels —

Partie 29: Codage vidéo Web

Reference number
ISO/IEC 14496-29:2015(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: Foreword - Supplementary information

The committee responsible for this document is ISO/IEC JTC 1, *Information technology*, SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

ISO/IEC 14496 consists of the following parts, under the general title *Information technology — Coding of audio-visual objects*:

— *Part 1: Systems*

— *Part 2: Visual*

— *Part 3: Audio*

— *Part 4: Conformance testing*

— *Part 5: Reference software*

— *Part 6: Delivery Multimedia Integration Framework (DMIF)*

— *Part 7: Optimized reference software for coding of audio-visual objects*

— *Part 8: Carriage of ISO/IEC 14496 contents over IP networks*

— *Part 9: Reference hardware description*

— *Part 10: Advanced Video Coding*
— *Part 11: Scene description and application engine*
— *Part 12: ISO base media file format*
— *Part 13: Intellectual Property Management and Protection (IPMP) extensions*
— *Part 14: MP4 file format*
— *Part 15: Advanced Video Coding (AVC) file format*
— *Part 16: Animation Framework eXtension (AFX)*
— *Part 17: Streaming text format*
— *Part 18: Font compression and streaming*
— *Part 19: Synthesized texture stream*
— *Part 20: Lightweight Application Scene Representation (LASeR) and Simple Aggregation Format (SAF)*
— *Part 21: MPEG-J Graphics Framework eXtensions (GFX)*
— *Part 22: Open Font Format*
— *Part 23: Symbolic Music Representation*
— *Part 24: Audio and systems interaction*
— *Part 25: 3D Graphics Compression Model*
— *Part 26: Audio conformance*
— *Part 27: 3D Graphics conformance*
— *Part 28: Composite font representation*
— *Part 29: Web video coding*

Introduction

This International Standard specifies Web Video Coding, a technology that is compatible with the Constrained Baseline Profile of ISO/IEC 14996-10. Only the subset that is specified in Annex A for the Constrained Baseline Profile is a normative specification, while all remaining aspects are informative. This text is derived from ISO/IEC 14996-10, with which the section numbers in this specification are aligned, and that specification may additionally be consulted if desired, as an aid to understanding this Specification.

Information technology — Coding of audio-visual objects — Part 29: Web video coding

1 Scope

This Part of ISO/IEC 14496 specifies Web Video Coding for coding of audio-visual objects.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

- ISO 11664-1, *Colorimetry — Part 1: CIE standard colorimetric observers*.
- ISO/IEC 14496-10: *Information technology – Coding of audio-visual objects – Part 10: Advanced Video Coding*

3 Definitions

For the purposes of this document, the following definitions apply:

3.1 **access unit**: A set of *NAL units* that are consecutive in *decoding order* and contain exactly one *primary coded picture*. In addition to the *primary coded picture*, an access unit may also contain one *auxiliary coded picture*, or other *NAL units* not containing *slices* of a *coded picture*. The decoding of an access unit always results in a *decoded picture*.

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

3.3 **bitstream**: A sequence of bits that forms the representation of *coded pictures* and associated data forming one or more *coded video sequences*. Bitstream is a collective term used to refer either to a *NAL unit stream* or a *byte stream*.

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

3.5 **[void]**

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

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

3.8 **byte-aligned**: A position in a *bitstream* is byte-aligned when the position is an integer multiple of 8 bits from the position of the first bit in the *bitstream*. 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.

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

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

3.11 **[void]**

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

NOTE – 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.

3.13 **coded frame**: A *coded representation* of a *frame*.

3.14 **coded picture**: A *coded representation* of a *picture*.

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

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

3.17 **[void]**

3.18 **coded slice NAL unit**: A *NAL unit* containing a *slice* that is not a *slice* of an *auxiliary coded picture*.

3.19 **coded video sequence**: A sequence of *access units* that consists, in decoding order, of an *IDR access unit* followed by zero or more non-IDR *accessunits* including all subsequent *access units* up to but not including any subsequent *IDR access unit*.

3.20 **component**: An array or single sample from one of the three arrays (*luma* and two *chroma*) that make up a *frame* in 4:2:0 colour format.

3.21 **DC transform coefficient**: A *transform coefficient* for which the *frequency index* is zero in all dimensions.

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

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

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

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

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

3.27 **decoding process**: The process specified in this International Standard that reads a *bitstream* and derives *decoded pictures* from it.

3.28 **[void]**

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

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

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

3.32 **encoding process**: A process, not specified in this International Standard, that produces a *bitstream* conforming to this International Standard.

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

3.34 **frame**: A *frame* contains an array of *luma* samples and two corresponding arrays of *chroma* samples in 4:2:0 format.

3.35 **frame macroblock**: A *macroblock* representing samples of a *coded frame*. All *macroblocks* of a *coded frame* are **frame macroblock**s.

3.36 **[void]**

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

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

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

3.40 **I slice**: A *slice* that is decoded using *intra prediction* only.

3.41 **informative**: A term used to refer to content provided in this International Standard that is not an integral part of this International Standard. Informative content does not establish any mandatory requirements for conformance to this International Standard.

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

3.43 **instantaneous decoding refresh (IDR) picture**: A *coded picture* for which the variable IdrPicFlag is equal to 1. An IDR picture causes the *decoding process* to mark all *reference pictures* as "unused for reference" immediately after the decoding of the IDR picture. All *coded pictures* that follow an IDR picture in *decoding order* can be decoded without *inter prediction* from any *picture* that precedes the IDR picture in *decoding order*. The first *picture* of each *coded video sequence* in *decoding order* is an IDR picture.

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

3.45 **inter prediction**: A *prediction* derived from decoded samples of *reference pictures* other than the current *decoded picture*.

3.46 **interpretation sample value**: A possibly-altered value corresponding to a decoded sample value of an *auxiliary coded picture* that may be generated for use in the *display process*. Interpretation sample values are not used in the *decoding process* and have no normative effect on the *decoding process*.

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

3.48 **intra prediction**: A *prediction* derived from the decoded samples of the same decoded *slice*.

3.49 **intra slice**: See *I slice*.

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

3.51 **layer**: One of a set of syntactical structures in a non-branching hierarchical relationship. Higher layers contain lower layers. The coding layers are the *coded video sequence*, *picture*, *slice*, and *macroblock* layers.

3.52 **level**: A defined set of constraints on the values that may be taken by the *syntax elements* and variables of this International Standard. The same set of levels is defined for all *profiles*, with most aspects of the definition of each level being in common across different *profiles*. Individual implementations may, within specified constraints, support a different level for each supported *profile*. In a different context, a level is the value of a *transform coefficient* prior to *scaling* (see the definition of *transform coefficient level*).

3.53 **list**: A one-dimensional array of *syntax elements* or variables.

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

NOTE – 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.

3.55 **macroblock**: A 16x16 *block* of *luma* samples and two corresponding *blocks* of *chroma* samples of a *picture* that has three sample arrays, or a 16x16 *block* of samples of a monochrome *picture* or a *picture* that is coded using three separate colour planes. The division of a *slice* into macroblocks is a *partitioning*.

3.56 **macroblock address**: a macroblock address is the index of a *macroblock* in a *macroblock raster scan* of the *picture* starting with zero for the top-left *macroblock* in a *picture*.

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

3.58 **macroblock partition**: A *block* of *luma* samples and two corresponding *blocks* of *chroma* samples resulting from a *partitioning* of a *macroblock* for *inter prediction* for a *picture* that has three sample arrays or a *block* of *luma* samples resulting from a *partitioning* of a *macroblock* for *inter prediction* for a monochrome *picture* or a *picture* that is coded using three separate colour planes.

3.59 **matrix**: A two-dimensional array of *syntax elements* or variables.

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

3.61 **memory management control operation**: Seven operations that control *reference picture marking*.

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

3.63 **must**: A term used in expressing an observation about a requirement or an implication of a requirement that is specified elsewhere in this International Standard. This term is used exclusively in an *informative* context.

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

3.65 **NAL unit stream**: A sequence of *NAL units*.

3.66 **non-reference frame**: A *frame* coded with nal_ref_idc equal to 0.

3.67 **non-reference picture**: A *picture* coded with nal_ref_idc equal to 0. A *non-reference picture* is not used for *inter prediction* of any other *pictures*.

3.68 **note**: A term used to prefix *informative* remarks. This term is used exclusively in an *informative* context.

3.69 **output order**: The order in which the *decoded pictures* are output from the *decoded picture buffer*.

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

3.71 **parameter**: A *syntax element* of a *sequence parameter set* or a *picture parameter set*. Parameter is also used as part of the defined term *quantisation parameter*.

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

3.73 **picture**: A collective term for a *frame*.

3.74 **picture parameter set**: A *syntax structure* containing *syntax elements* that apply to zero or more entire *coded pictures* as determined by the pic_parameter_set_id *syntax element* found in each *slice header*.

3.75 **picture order count**: A variable that is associated with each *coded picture* and has a value that is non-decreasing with increasing *picture* position in *output order* relative to the first output *picture* of the previous *IDR picture* in *decoding order* or relative to the previous *picture*, in *decoding order*, that contains a *memory management control operation* that marks all *reference pictures* as "unused for reference".

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

3.77 **prediction process**: The use of a *predictor* to provide an estimate of the sample value or data element currently being decoded.

3.78 **predictive slice**: See *P slice*.

3.79 **predictor**: A combination of specified values or previously decoded sample values or data elements used in the *decoding process* of subsequent sample values or data elements.

3.80 **primary coded picture**: The coded representation of a *picture* to be used by the *decoding process* for a bitstream conforming to this International Standard. The primary coded picture contains all *macroblocks* of the *picture*. The only *pictures* that have a normative effect on the *decoding process* are primary coded pictures. *e.*

3.81 **profile**: A specified subset of the syntax of this International Standard.

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

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

3.84 **raster scan**: A mapping of a rectangular two-dimensional pattern to a one-dimensional pattern such that the first entries in the one-dimensional pattern are from the first top row of the two-dimensional pattern scanned from left to right, followed similarly by the second, third, etc., rows of the pattern (going down) each scanned from left to right.

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

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

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

3.88 **reference frame**: A *reference frame* may be used for *inter prediction* when *P slices* of a *coded frame* are decoded. See also *reference picture*.

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

3.90 **reference picture**: A *picture* with nal_ref_idc not equal to 0. A *reference picture* contains samples that may be used for *inter prediction* in the *decoding process* of subsequent *pictures* in *decoding order*.

3.91 **reference picture list**: A list of *reference pictures* that is used for *inter prediction* of a *P slice*. For the *decoding process* of a *P slice,* there is one reference picture list.

3.92 **reference picture list 0**: A *reference picture list* used for *inter prediction* of a *Pslice*. All *inter prediction* used for *P slices* uses reference picture list 0.

3.93 **reference picture marking**: Specifies, in the bitstream, how the *decoded pictures* are marked for *inter prediction*.

3.94 **reserved**: The term reserved, when used in the clauses specifying some values of a particular *syntax element*, are for future use by ITU-T | ISO/IEC. These values shall not be used in *bitstreams* conforming to this International Standard, but may be used in future extensions of this International Standard by ITU-T | ISO/IEC.

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

3.96 **run**: A number of consecutive data elements represented in the decoding process. In one context, the number of zero-valued *transform coefficient levels* preceding a non-zero *transform coefficient level* in the list of *transform coefficient levels* generated by a *zig-zag scan*. In other contexts, run refers to a number of *macroblocks*.

3.97 **sample aspect ratio**: Specifies, for assisting the *display process*, which is not specified in this International Standard, the ratio between the intended horizontal distance between the columns and the intended vertical distance between the rows of the *luma* sample array in a *frame*. Sample aspect ratio is expressed as *h*:*v*, where *h* is horizontal width and *v* is vertical height (in arbitrary units of spatial distance).

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

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

3.100 **shall**: A term used to express mandatory requirements for conformance to this International Standard. 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 results to the *decoding process* described herein conforms to the *decoding process* requirements of this International Standard.

3.101 **should**: A term used to refer to behaviour of an implementation that is encouraged to be followed under anticipated ordinary circumstances, but is not a mandatory requirement for conformance to this International Standard.

3.102 **skipped macroblock**: A *macroblock* for which no data is coded other than an indication that the *macroblock* is to be decoded as "skipped". This indication may be common to several *macroblocks*.

3.103 **slice**: An integer number of *macroblocks* ordered consecutively in the *raster scan* within the *primary coded picture*. The *macroblock addresses* are derived from the first *macroblock address* in a slice (as represented in the *slice header*) and, when a *picture* is coded using three separate colour planes, a colour plane identifier.

3.104 **[void]**

3.105 **[void]**

3.106 **slice header**: A part of a coded *slice* containing the data elements pertaining to the first or all *macroblocks* represented in the *slice*.

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

3.108 **start code prefix**: A unique sequence of three *bytes* equal to 0x000001 embedded in the *byte stream* as a prefix to each *NAL unit*. The location of a start code prefix can be used by a *decoder* to identify the beginning of a new *NAL unit* and the end of a previous *NAL unit*. Emulation of start code prefixes is prevented within *NAL units* by the inclusion of *emulation prevention bytes*.

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

3.110 **sub-macroblock**: One quarter of the samples of a *macroblock*, i.e., an 8x8 *luma block* and two corresponding *chroma blocks* of which one corner is located at a corner of the *macroblock* for a *picture* that has three sample arrays or an 8x8 *luma block* of which one corner is located at a corner of the *macroblock* for a monochrome *picture* or a *picture* that is coded using three separate colour planes.

3.111 **sub-macroblock partition**: A *block* of *luma* samples and two corresponding *blocks* of *chroma* samples resulting from a *partitioning* of a *sub-macroblock* for *inter prediction* for a *picture* that has three sample arrays or a *block* of *luma* samples resulting from a *partitioning* of a *sub-macroblock* for *inter prediction* for a monochrome *picture* or a *picture* that is coded using three separate colour planes.

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

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

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

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

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

3.117 **unspecified:** The term unspecified, when used in the clauses specifying some values of a particular *syntax element*, indicates that the values have no specified meaning in this International Standard and will not have a specified meaning in the future as an integral part of this International Standard.

3.118 **variable length coding (VLC)**: A reversible procedure for entropy coding that assigns shorter bit strings to *symbols* expected to be more frequent and longer bit strings to *symbols* expected to be less frequent.

3.119 **VCL NAL unit**: A collective term for *coded slice NAL units*.

3.120 **zig-zag scan**: A specific sequential ordering of *transform coefficient levels* from (approximately) the lowest spatial frequency to the highest. Zig-zag scan is used for *transform coefficient levels* in *frame macroblocks*.

4 Abbreviations

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

CAVLC Context-based Adaptive Variable Length Coding

CBR Constant Bit Rate

CPB Coded Picture Buffer

DPB Decoded Picture Buffer

DUT Decoder under test

FIFO First-In, First-Out

HRD Hypothetical Reference Decoder

HSS Hypothetical Stream Scheduler

IDR Instantaneous Decoding Refresh

LSB Least Significant Bit

MB Macroblock

MSB Most Significant Bit

NAL Network Abstraction Layer

RBSP Raw Byte Sequence Payload

SEI Supplemental Enhancement Information

SODB String Of Data Bits

UUID Universal Unique Identifier

VBR Variable Bit Rate

VCL Video Coding Layer

VLC Variable Length Coding

VUI Video Usability Information

5 Conventions

NOTE – The mathematical operators used in this Specification are similar to those used in the C programming language. However, integer division and arithmetic shift operations are specifically defined. Numbering and counting conventions generally begin from 0.

5.1 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.

5.2 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.

5.3 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.

5.4 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 positive integer values of y. Bits shifted into the MSBs as a result of the right shift have a value equal to the MSB of x prior to the shift operation.

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

5.5 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).

5.6 Range notation

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

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

5.7 Mathematical functions

The following mathematical functions are defined as follows:

$BitDepth_Y$ and $BitDepth_C$ are both specified to be equal to 8 in this standard

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

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

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

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

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

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

$$\text{InverseRasterScan}(\,a, b, c, d, e\,) = \begin{cases} (a \% (d / b)) * b & ; \quad e == 0 \\ (a / (d / b)) * c & ; \quad e == 1 \end{cases} \tag{5-7}$$

Log2(x) returns the base-2 logarithm of x. (5-8)

Log10(x) returns the base-10 logarithm of x. (5-9)

$$\text{Median}(\,x, y, z\,) = x + y + z - \text{Min}(\,x, \text{Min}(\,y, z\,)\,) - \text{Max}(\,x, \text{Max}(\,y, z\,)\,) \tag{5-10}$$

$$\text{Min}(\,x, y\,) = \begin{cases} x & ; \quad x <= y \\ y & ; \quad x > y \end{cases} \tag{5-11}$$

$$\text{Max}(\,x, y\,) = \begin{cases} x & ; \quad x >= y \\ y & ; \quad x < y \end{cases} \tag{5-12}$$

$$\text{Round}(\,x\,) = \text{Sign}(\,x\,) * \text{Floor}(\,\text{Abs}(\,x\,) + 0.5\,) \tag{5-13}$$

$$\text{Sign}(\,x\,) = \begin{cases} 1 & ; \quad x >= 0 \\ -1 & ; \quad x < 0 \end{cases} \tag{5-14}$$

$$\text{Sqrt}(\,x\,) = \sqrt{x} \tag{5-15}$$

5.8 Order of operation precedence

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

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

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

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

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

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

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

5.9 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), its one or two syntax categories, and one or two descriptors for its method of coded representation. The decoding process behaves according to the value of the syntax element and to the values of previously decoded syntax elements. When a value of a syntax element is used in the syntax tables or the text, it appears in regular (i.e., not bold) type.

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

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

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

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

Functions that are not syntax functions (including mathematical functions specified in subclause 5.7) 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).

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}.

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.

5.10 Text description of logical operations

In the text, a statement of logical operations as would be described in pseudo-code as

```
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 in pseudo-code as

```
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 any 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 in pseudo-code as:

```
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

5.11 Processes

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

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

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

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

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

6.1 Bitstream formats

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

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

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

6.2 Source, decoded, and output picture formats

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

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

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

- Luma (Y) only (monochrome), with or without an auxiliary array.
- Luma and two Chroma (YCbCr or YCgCo), with or without an auxiliary array.
- Green, Blue and Red (GBR, also known as RGB), with or without an auxiliary array.
- Arrays representing other unspecified monochrome or tri-stimulus colour samplings (for example, YZX, also known as XYZ), with or without an auxiliary array.

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

Table 6-1 – Chroma Format

chroma_format_idc	Chroma Format
1	4:2:0

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.

For the purposes of this version of this Specification, the value of chroma_format_idc shall be inferred to be equal to 1 and the chroma format shall be inferred to be 4:2:0 as shown in Table 6-1.

The width and height of the luma sample arrays are each an integer multiple of 16. In coded video sequences using 4:2:0 chroma sampling, the width and height of chroma sample arrays are each an integer multiple of 8. The width or height of pictures output from the decoding process need not be an integer multiple of 16 and can be specified using a cropping rectangle.

The syntax for the luma and (when present) chroma arrays are ordered such when data for all three colour components is present, the data for the luma array is first, followed by any data for the Cb array, followed by any data for the Cr array, unless otherwise specified.

The number of bits necessary for the representation of each of the samples in the luma and chroma arrays in a coded video sequence is equal to 8, regardless of whether the sample is a sample of the luma array or a sample of the chroma arrays.

The nominal vertical and horizontal relative locations of luma and chroma samples in frames are shown in Figure 6-1. Alternative chroma sample relative locations may be indicated in video usability information (see Annex E).

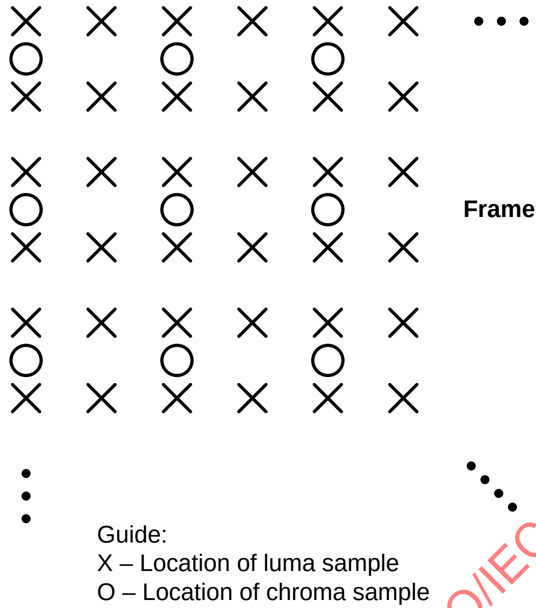

Figure 6-1 – Nominal vertical and horizontal locations of 4:2:0 luma and chroma samples in a frame

The samples are processed in units of macroblocks. The luma array for each macroblock is 16 samples in both width and height. The variables MbWidthC and MbHeightC, which specify the width and height, respectively, of the chroma arrays for each macroblock, are derived as follows:

MbWidthC and MbHeightC are derived as

$$\text{MbWidthC} = 16 / 2 \quad (6\text{-}1)$$
$$\text{MbHeightC} = 16 / 2 \quad (6\text{-}2)$$

6.3 Spatial subdivision of pictures and slices

This subclause specifies how a picture is partitioned into slices and macroblocks. Pictures are divided into slices. A slice is a sequence of macroblocks. Each macroblock is comprised of one 16x16 luma array and two corresponding chroma sample arrays. Each macroblock represents a spatial rectangular region of the picture. For example, a picture may be divided into two slices as shown in Figure 6-2.

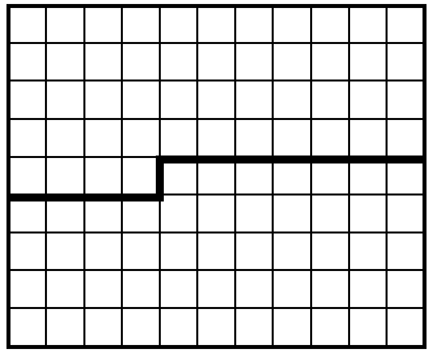

Figure 6-2 – A picture with 11 by 9 macroblocks that is partitioned into two slices

6.4 Inverse scanning processes and derivation processes for neighbours

This subclause specifies inverse scanning processes; i.e., the mapping of indices to locations, and derivation processes for neighbours.

6.4.1 Inverse macroblock scanning process

Input to this process is a macroblock address mbAddr.

Output of this process is the location (x, y) of the upper-left luma sample for the macroblock with address mbAddr relative to the upper-left sample of the picture.

The inverse macroblock scanning process is specified as follows:

$$x = \text{InverseRasterScan}(\ \text{mbAddr}, 16, 16, \text{PicWidthInSamples}_L, 0\ ) \quad (6\text{-}3)$$

$$y = \text{InverseRasterScan}(\ \text{mbAddr}, 16, 16, \text{PicWidthInSamples}_L, 1\ ) \quad (6\text{-}4)$$

6.4.2 Inverse macroblock partition and sub-macroblock partition scanning process

Macroblocks or sub-macroblocks may be partitioned, and the partitions are scanned for inter prediction as shown in Figure 6-3. The outer rectangles refer to the samples in a macroblock or sub-macroblock, respectively. The rectangles refer to the partitions. The number in each rectangle specifies the index of the inverse macroblock partition scan or inverse sub-macroblock partition scan.

The functions MbPartWidth( ), MbPartHeight( ), SubMbPartWidth( ), and SubMbPartHeight( ) describing the width and height of macroblock partitions and sub-macroblock partitions are specified in Tables 7-9 and 7-12. MbPartWidth( ) and MbPartHeight( ) are set to appropriate values for each macroblock, depending on the macroblock type. SubMbPartWidth( ) and SubMbPartHeight( ) are set to appropriate values for each sub-macroblock of a macroblock with mb_type equal to P_8x8 or P_8x8ref0, depending on the sub-macroblock type.

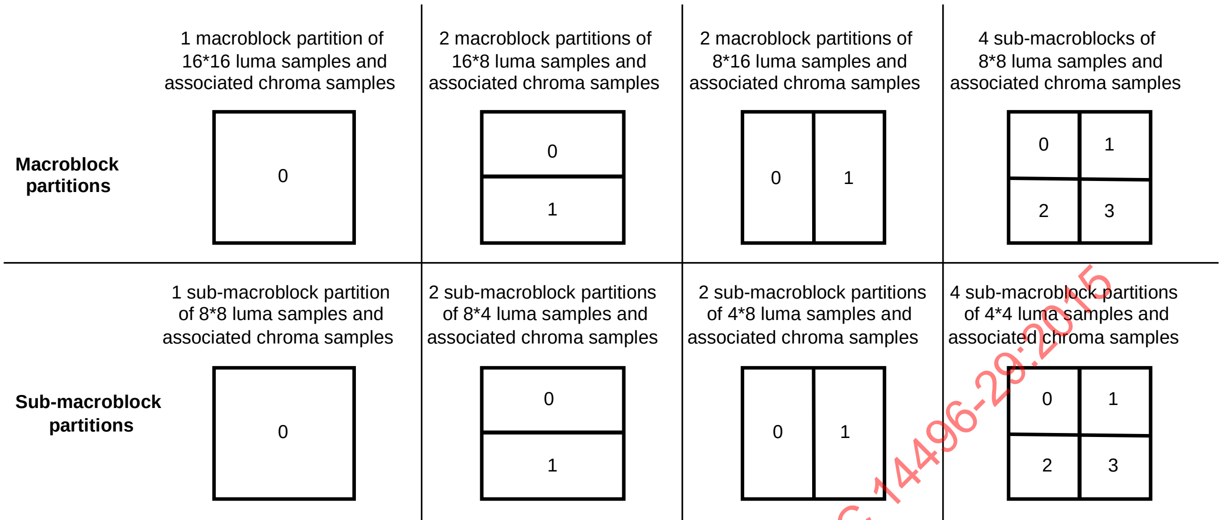

Figure 6-3 – Macroblock partitions, sub-macroblock partitions, macroblock partition scans, and sub-macroblock partition scans

6.4.2.1 Inverse macroblock partition scanning process

Input to this process is the index of a macroblock partition mbPartIdx.

Output of this process is the location (x, y) of the upper-left luma sample for the macroblock partition mbPartIdx relative to the upper-left sample of the macroblock.

The inverse macroblock partition scanning process is specified by

x = InverseRasterScan(mbPartIdx, MbPartWidth(mb_type), MbPartHeight(mb_type), 16, 0) (6-5)

y = InverseRasterScan(mbPartIdx, MbPartWidth(mb_type), MbPartHeight(mb_type), 16, 1) (6-6)

6.4.2.2 Inverse sub-macroblock partition scanning process

Inputs to this process are the index of a macroblock partition mbPartIdx and the index of a sub-macroblock partition subMbPartIdx.

Output of this process is the location (x, y) of the upper-left luma sample for the sub-macroblock partition subMbPartIdx relative to the upper-left sample of the sub-macroblock.

The inverse sub-macroblock partition scanning process is specified as follows:

– If mb_type is equal to P_8x8 or P_8x8ref0

x = InverseRasterScan(subMbPartIdx, SubMbPartWidth(sub_mb_type[mbPartIdx]),
SubMbPartHeight(sub_mb_type[mbPartIdx]), 8, 0) (6-7)

y = InverseRasterScan(subMbPartIdx, SubMbPartWidth(sub_mb_type[mbPartIdx]),
SubMbPartHeight(sub_mb_type[mbPartIdx]), 8, 1) (6-8)

– Otherwise (mb_type is not equal to P_8x8 or P_8x8ref0), x = InverseRasterScan(subMbPartIdx, 4, 4, 8, 0) (6-9)

y = InverseRasterScan(subMbPartIdx, 4, 4, 8, 1) (6-10)

6.4.3 Inverse 4x4 luma block scanning process

Input to this process is the index of a 4x4 luma block luma4x4BlkIdx.

Output of this process is the location (x, y) of the upper-left luma sample for the 4x4 luma block with index luma4x4BlkIdx relative to the upper-left luma sample of the macroblock.

Figure 6-4 shows the scan for the 4x4 luma blocks.

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

Figure 6-4 – Scan for 4x4 luma blocks

The inverse 4x4 luma block scanning process is specified by

x = InverseRasterScan(luma4x4BlkIdx / 4, 8, 8, 16, 0) +
InverseRasterScan(luma4x4BlkIdx % 4, 4, 4, 8, 0) (6-11)

y = InverseRasterScan(luma4x4BlkIdx / 4, 8, 8, 16, 1) +
InverseRasterScan(luma4x4BlkIdx % 4, 4, 4, 8, 1) (6-12)

6.4.4 (void)

6.4.5 Inverse 8x8 luma block scanning process

Input to this process is the index of an 8x8 luma block luma8x8BlkIdx.

Output of this process is the location (x, y) of the upper-left luma sample for the 8x8 luma block with index luma8x8BlkIdx relative to the upper-left luma sample of the macroblock.

Figure 6-5 shows the scan for the 8x8 luma blocks.

0	1
2	3

Figure 6-5 – Scan for 8x8 luma blocks

The inverse 8x8 luma block scanning process is specified by:

x = InverseRasterScan(luma8x8BlkIdx, 8, 8, 16, 0) (6-13)

y = InverseRasterScan(luma8x8BlkIdx, 8, 8, 16, 1) (6-14)

6.4.6 (void)

6.4.7 Inverse 4x4 chroma block scanning process

Input to this process is the index of a 4x4 chroma block chroma4x4BlkIdx.

Output of this process is the location (x, y) of the upper-left chroma sample for a 4x4 chroma block with index chroma4x4BlkIdx relative to the upper-left chroma sample of the macroblock.

The inverse 4x4chroma block scanning process is specified by

x = InverseRasterScan(chroma4x4BlkIdx, 4, 4, 8, 0) (6-15)

y = InverseRasterScan(chroma4x4BlkIdx, 4, 4, 8, 1) (6-16)

6.4.8 Derivation process of the availability for macroblock addresses

Input to this process is a macroblock address mbAddr.

Output of this process is the availability of the macroblock mbAddr.

NOTE – The meaning of availability is determined when this process is invoked.

The macroblock is marked as available, unless any of the following conditions are true, in which case the macroblock is marked as not available:

– mbAddr < 0,

– mbAddr > CurrMbAddr,

– the macroblock with address mbAddr belongs to a different slice than the macroblock with address CurrMbAddr.

6.4.9 Derivation process for neighbouring macroblock addresses and their availability

The outputs of this process are:

– mbAddrA: the address and availability status of the macroblock to the left of the current macroblock,

– mbAddrB: the address and availability status of the macroblock above the current macroblock,

– mbAddrC: the address and availability status of the macroblock above-right of the current macroblock,

– mbAddrD: the address and availability status of the macroblock above-left of the current macroblock.

Figure 6-6 shows the relative spatial locations of the macroblocks with mbAddrA, mbAddrB, mbAddrC, and mbAddrD relative to the current macroblock with CurrMbAddr.

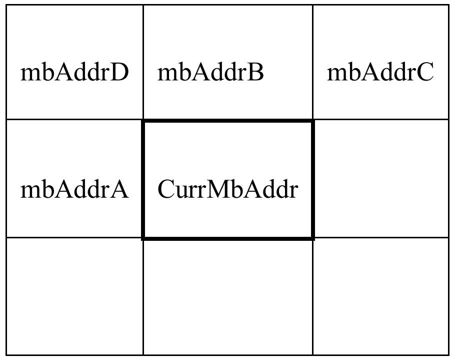

Figure 6-6 – Neighbouring macroblocks for a given macroblock

Input to the process in subclause 6.4.8 is mbAddrA = CurrMbAddr − 1 and the output is whether the macroblock mbAddrA is available. In addition, mbAddrA is marked as not available when CurrMbAddr % PicWidthInMbs is equal to 0.

Input to the process in subclause 6.4.8 is mbAddrB = CurrMbAddr − PicWidthInMbs and the output is whether the macroblock mbAddrB is available.

Input to the process in subclause 6.4.8 is mbAddrC = CurrMbAddr − PicWidthInMbs + 1 and the output is whether the macroblock mbAddrC is available. In addition, mbAddrC is marked as not available when (CurrMbAddr + 1) % PicWidthInMbs is equal to 0.

Input to the process in subclause 6.4.8 is mbAddrD = CurrMbAddr − PicWidthInMbs − 1 and the output is whether the macroblock mbAddrD is available. In addition, mbAddrD is marked as not available when CurrMbAddr % PicWidthInMbs is equal to 0.

6.4.10 (void)

6.4.11 Derivation processes for neighbouring macroblocks, blocks, and partitions

Subclause 6.4.11.1 specifies the derivation process for neighbouring macroblocks.

Subclause 6.4.11.4 specifies the derivation process for neighbouring 4x4 luma blocks.

Subclause 6.4.11.5 specifies the derivation process for neighbouring 4x4 chroma blocks.

Subclause 6.4.11.7 specifies the derivation process for neighbouring partitions.

Table 6-2 specifies the values for the difference of luma location (xD, yD) for the input and the replacement for N in mbAddrN, mbPartIdxN, subMbPartIdxN, luma8x8BlkIdxN, luma4x4BlkIdxN, cb4x4BlkIdxN, cr4x4BlkIdxN, and chroma4x4BlkIdxN for the output. These input and output assignments are used in subclauses 6.4.11.1 to 6.4.11.7. The variable predPartWidth is specified when Table 6-2 is referred to.

Table 6-2 – Specification of input and output assignments for subclauses 6.4.11.1 to 6.4.11.7

N	xD	yD
A	−1	0
B	0	−1
C	predPartWidth	−1
D	−1	−1

Figure 6-7 illustrates the relative location of the neighbouring macroblocks, blocks, or partitions A, B, C, and D to the current macroblock, partition, or block, when the current macroblock, partition, or block is in frame coding mode.

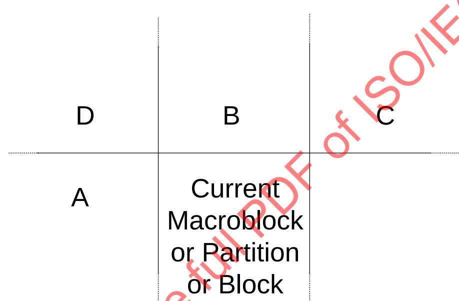

Figure 6-7 – Determination of the neighbouring macroblock, blocks, and partitions (informative)

6.4.11.1 Derivation process for neighbouring macroblocks

Outputs of this process are:

– mbAddrA: the address of the macroblock to the left of the current macroblock and its availability status,

– mbAddrB: the address of the macroblock above the current macroblock and its availability status.

mbAddrN (with N being A or B) is derived as specified by the following ordered steps:

1. The difference of luma location (xD, yD) is set according to Table 6-2.
2. The derivation process for neighbouring locations as specified in subclause 6.4.12 is invoked for luma locations with (xN, yN) equal to (xD, yD), and the output is assigned to mbAddrN.

6.4.11.2 (void)

6.4.11.3 (void)

6.4.11.4 Derivation process for neighbouring 4x4 luma blocks

Input to this process is a 4x4 luma block index luma4x4BlkIdx.

Outputs of this process are:

– mbAddrA: either equal to CurrMbAddr or the address of the macroblock to the left of the current macroblock and its availability status,
– luma4x4BlkIdxA: the index of the 4x4 luma block to the left of the 4x4 block with index luma4x4BlkIdx and its availability status,
– mbAddrB: either equal to CurrMbAddr or the address of the macroblock above the current macroblock and its availability status,
– luma4x4BlkIdxB: the index of the 4x4 luma block above the 4x4 block with index luma4x4BlkIdx and its availability status.

mbAddrN and luma4x4BlkIdxN (with N being A or B) are derived as specified by the following ordered steps:

1. The difference of luma location (xD, yD) is set according to Table 6-2.
2. The inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with luma4x4BlkIdx as the input and (x, y) as the output.
3. The luma location (xN, yN) is specified by

$$xN = x + xD \quad (6\text{-}17)$$

$$yN = y + yD \quad (6\text{-}18)$$

4. The derivation process for neighbouring locations as specified in subclause 6.4.12 is invoked for luma locations with (xN, yN) as the input and the output is assigned to mbAddrN and (xW, yW).
5. The variable luma4x4BlkIdxN is derived as follows:
   – If mbAddrN is not available, luma4x4BlkIdxN is marked as not available.
   – Otherwise (mbAddrN is available), the derivation process for 4x4 luma block indices as specified in subclause 6.4.13.1 is invoked with the luma location (xW, yW) as the input and the output is assigned to luma4x4BlkIdxN.

6.4.11.5 Derivation process for neighbouring 4x4 chroma blocks

Input to this process is a 4x4 chroma block index chroma4x4BlkIdx.

Outputs of this process are:

– mbAddrA (either equal to CurrMbAddr or the address of the macroblock to the left of the current macroblock) and its availability status,
– chroma4x4BlkIdxA (the index of the 4x4 chroma block to the left of the 4x4 chroma block with index chroma4x4BlkIdx) and its availability status,
– mbAddrB (either equal to CurrMbAddr or the address of the macroblock above the current macroblock) and its availability status,
– chroma4x4BlkIdxB (the index of the 4x4 chroma block above the 4x4 chroma block with index chroma4x4BlkIdx) and its availability status.

mbAddrN and chroma4x4BlkIdxN (with N being A or B) are derived as specified by the following ordered steps:

1. The difference of chroma location (xD, yD) is set according to Table 6-2.
2. The inverse 4x4 chroma block scanning process as specified in subclause 6.4.7 is invoked with chroma4x4BlkIdx as the input and(x, y) as the output.
3. The chroma location (xN, yN) is specified by

$$xN = x + xD \quad (6\text{-}19)$$

yN = y + yD (6-20)

4. The derivation process for neighbouring locations as specified in subclause 6.4.12 is invoked for chroma locations with (xN, yN) as the input and the output is assigned to mbAddrN and (xW, yW).

5. The variable chroma4x4BlkIdxN is derived as follows:

   – If mbAddrN is not available, chroma4x4BlkIdxN is marked as not available.

   – Otherwise (mbAddrN is available), the derivation process for 4x4 chroma block indices as specified in subclause 6.4.13.2 is invoked with the chroma location (xW, yW) as the input and the output is assigned to chroma4x4BlkIdxN.

6.4.11.6 (void)

6.4.11.7 Derivation process for neighbouring partitions

Inputs to this process are:

– a macroblock partition index mbPartIdx

– a current sub-macroblock type currSubMbType

– a sub-macroblock partition index subMbPartIdx

Outputs of this process are:

– mbAddrA\mbPartIdxA\subMbPartIdxA: specifying the macroblock or sub-macroblock partition to the left of the current macroblock and its availability status, or the sub-macroblock partition CurrMbAddr\mbPartIdx\subMbPartIdx and its availability status,

– mbAddrB\mbPartIdxB\subMbPartIdxB: specifying the macroblock or sub-macroblock partition above the current macroblock and its availability status, or the sub-macroblock partition CurrMbAddr\mbPartIdx\subMbPartIdx and its availability status,

– mbAddrC\mbPartIdxC\subMbPartIdxC: specifying the macroblock or sub-macroblock partition to the right-above of the current macroblock and its availability status, or the sub-macroblock partition CurrMbAddr\mbPartIdx\subMbPartIdx and its availability status,

– mbAddrD\mbPartIdxD\subMbPartIdxD: specifying the macroblock or sub-macroblock partition to the left-above of the current macroblock and its availability status, or the sub-macroblock partition CurrMbAddr\mbPartIdx\subMbPartIdx and its availability status.

mbAddrN, mbPartIdxN, and subMbPartIdxN (with N being A, B, C, or D) are derived as specified by the following ordered steps:

1. The inverse macroblock partition scanning process as described in subclause 6.4.2.1 is invoked with mbPartIdx as the input and (x, y) as the output.

2. The location of the upper-left luma sample inside a macroblock partition (xS, yS) is derived as follows:

   – If mb_type is equal to P_8x8 or P_8x8ref0, the inverse sub-macroblock partition scanning process as described in subclause 6.4.2.2 is invoked with subMbPartIdx as the input and (xS, yS) as the output.

   – Otherwise, (xS, yS) are set to (0, 0).

3. The variable predPartWidth in Table 6-2 is specified as follows:

   – If mb_type is equal to P_Skip, predPartWidth = 16.

   – Otherwise, if mb_type is equal to P_8x8 or P_8x8ref0, predPartWidth = SubMbPartWidth(sub_mb_type[mbPartIdx]).

   – Otherwise, predPartWidth = MbPartWidth(mb_type).

4. The difference of luma location (xD, yD) is set according to Table 6-2.
5. The neighbouring luma location (xN, yN) is specified by

$$xN = x + xS + xD \quad (6\text{-}21)$$

$$yN = y + yS + yD \quad (6\text{-}22)$$

6. The derivation process for neighbouring locations as specified in subclause 6.4.12 is invoked for luma locations with (xN, yN) as the input and the output is assigned to mbAddrN and (xW, yW).
7. Depending on mbAddrN, the following applies:
   – If mbAddrN is not available, the macroblock or sub-macroblock partition mbAddrN\mbPartIdxN\subMbPartIdxN is marked as not available.
   – Otherwise (mbAddrN is available), the following ordered steps are specified:
      a. Let mbTypeN be the syntax element mb_type of the macroblock with macroblock address mbAddrN and, when mbTypeN is equal to P_8x8 or P_8x8ref0, let subMbTypeN be the syntax element list sub_mb_type of the macroblock with macroblock address mbAddrN.
      b. The derivation process for macroblock and sub-macroblock partition indices as specified in subclause 6.4.13.4 is invoked with the luma location (xW, yW), the macroblock type mbTypeN, and, when mbTypeN is equal to P_8x8 or P_8x8ref0, the list of sub-macroblock types subMbTypeN as the inputs and the outputs are the macroblock partition index mbPartIdxN and the sub-macroblock partition index subMbPartIdxN.
      c. When the partition given by mbPartIdxN and subMbPartIdxN is not yet decoded, the macroblock partition mbPartIdxN and the sub-macroblock partition subMbPartIdxN are marked as not available.

      NOTE – The latter condition is, for example, the case when mbPartIdx = 2, subMbPartIdx = 3, xD = 4, yD = −1, i.e., when neighbour C of the last 4x4 luma block of the third sub-macroblock is requested.

6.4.12 Derivation process for neighbouring locations

Input to this process is a luma or chroma location (xN, yN) expressed relative to the upper left corner of the current macroblock.

Outputs of this process are:

– mbAddrN: either equal to CurrMbAddr or to the address of neighbouring macroblock that contains (xN, yN) and its availability status,
– (xW, yW): the location (xN, yN) expressed relative to the upper-left corner of the macroblock mbAddrN (rather than relative to the upper-left corner of the current macroblock).

Let maxW and maxH be variables specifying maximum values of the location components xN, xW, and yN, yW, respectively. maxW and maxH are derived as follows:

– If this process is invoked for neighbouring luma locations,

$$maxW = maxH = 16 \quad (6\text{-}23)$$

– Otherwise (this process is invoked for neighbouring chroma locations),

$$maxW = MbWidthC \quad (6\text{-}24)$$

$$maxH = MbHeightC \quad (6\text{-}25)$$

6.4.12.1 Specification for neighbouring locations in frames

The derivation process for neighbouring macroblock addresses and their availability in subclause 6.4.9 is invoked with mbAddrA, mbAddrB, mbAddrC, and mbAddrD as well as their availability status as the output.

Table 6-3 specifies mbAddrN depending on (xN, yN).

Table 6-3 – Specification of mbAddrN

xN	**yN**	**mbAddrN**
< 0	< 0	mbAddrD
< 0	0..maxH − 1	mbAddrA
0..maxW − 1	< 0	mbAddrB
0..maxW − 1	0..maxH − 1	CurrMbAddr
> maxW − 1	< 0	mbAddrC
> maxW − 1	0..maxH − 1	not available
	> maxH − 1	not available

The neighbouring location (xW, yW) relative to the upper-left corner of the macroblock mbAddrN is derived as

$$xW = (xN + maxW) \% maxW \quad (6\text{-}26)$$

$$yW = (yN + maxH) \% maxH \quad (6\text{-}27)$$

6.4.13 Derivation processes for block and partition indices

Subclause 6.4.13.1 specifies the derivation process for 4x4 luma block indices.

Subclause 6.4.13.2 specifies the derivation process for 4x4 chroma block indices.

Subclause 6.4.13.3 specifies the derivation process for 8x8 luma block indices.

Subclause 6.4.13.4 specifies the derivation process for macroblock and sub-macroblock partition indices.

6.4.13.1 Derivation process for 4x4 luma block indices

Input to this process is a luma location (xP, yP) relative to the upper-left luma sample of a macroblock.

Output of this process is a 4x4 luma block index luma4x4BlkIdx.

The 4x4 luma block index luma4x4BlkIdx is derived by

$$luma4x4BlkIdx = 8 * (yP / 8) + 4 * (xP / 8) + 2 * ((yP \% 8) / 4) + ((xP \% 8) / 4) \quad (6\text{-}28)$$

6.4.13.2 Derivation process for 4x4 chroma block indices

Input to this process is a chroma location (xP, yP) relative to the upper-left chroma sample of a macroblock.

Output of this process is a 4x4 chroma block index chroma4x4BlkIdx.

The 4x4 chroma block index chroma4x4BlkIdx is derived by

$$chroma4x4BlkIdx = 2 * (yP / 4) + (xP / 4) \quad (6\text{-}29)$$

6.4.13.3 Derivation process for 8x8 luma block indices

Input to this process is a luma location (xP, yP) relative to the upper-left luma sample of a macroblock.

Outputs of this process is an 8x8 luma block index luma8x8BlkIdx.

The 8x8 luma block index luma8x8BlkIdx is derived by

$$\text{luma8x8BlkIdx} = 2 * (yP / 8) + (xP / 8) \quad (6\text{-}30)$$

6.4.13.4 Derivation process for macroblock and sub-macroblock partition indices

Inputs to this process are:

– a luma location (xP, yP) relative to the upper-left luma sample of a macroblock,

– a macroblock type mbType,

– when mbType is equal to P_8x8 or P_8x8ref0, a list of sub-macroblock types subMbType with 4 elements.

Outputs of this process are:

– a macroblock partition index mbPartIdx,

– a sub-macroblock partition index subMbPartIdx.

The macroblock partition index mbPartIdx is derived as follows:

– If mbType specifies an I macroblock type, mbPartIdx is set equal to 0.

– Otherwise (mbType does not specify an I macroblock type), mbPartIdx is derived by

$$\text{mbPartIdx} = (16 / \text{MbPartWidth}(\text{mbType})) * (yP / \text{MbPartHeight}(\text{mbType})) + (xP / \text{MbPartWidth}(\text{mbType})) \quad (6\text{-}31)$$

The sub-macroblock partition index subMbPartIdx is derived as follows:

– If mbType is not equal to P_8x8 or P_8x8ref0, subMbPartIdx is set equal to 0.

– Otherwise (mbType is equal to P_8x8 or P_8x8ref0), subMbPartIdx is derived by

$$\text{subMbPartIdx} = (8 / \text{SubMbPartWidth}(\text{subMbType}[\text{mbPartIdx}])) * ((yP \% 8) / \text{SubMbPartHeight}(\text{subMbType}[\text{mbPartIdx}])) + ((xP \% 8) / \text{SubMbPartWidth}(\text{subMbType}[\text{mbPartIdx}])) \quad (6\text{-}32)$$

7 Syntax and semantics

7.1 Normative Syntax and Semantics

7.1.1 Normative and Informative Technologies

The normative requirements of this specification extend only to the technologies required to implement the profile specified in A.2.1. All other aspects of this specification are informative only, and not normative. Specifically, a conforming decoder is not required to handle a number of technologies, including but not limited to the following:

a) field coding (i.e. frame_mbs_only_flag equal to 0);

b) color sampling formats other than 4:2:0;

c) picture size scaling;

d) BitDepth_Y and BitDepth_C values other than 8;

e) bipredictive and switching slice types (i.e. slice types other than I and P slices);

f) weighted prediction modes other than the default (i.e. weighted_pred_flag or weighted_bipred_idc not equal to 0);

g) entropy coding modes other than CAVLC (i.e. entropy_coding_mode_flag not equal to 0);

h) 8x8 inverse transform block size;

i) arbitrary slice order;

j) more than one slice group per picture;

7.1.2 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 means for identifying entry points into the bitstream and means to identify and handle non-conforming bitstreams. The methods for identifying and handling errors and other such situations are not specified here.

The following table lists examples of pseudo code used to describe the syntax. When **syntax_element** appears, it specifies that a syntax element is parsed from the bitstream and the bitstream pointer is advanced to the next position beyond the syntax element in the bitstream parsing process.

	Descriptor
/* A statement can be a syntax element with an associated descriptor or can be an expression used to specify conditions for the existence, type, and quantity of syntax elements, as in the following two examples */	
syntax_element	ue(v)
conditioning statement	
/* A group of statements enclosed in curly brackets is a compound statement and is treated functionally as a single statement. */	
{	
statement	
statement	
…	
}	
/* A "while" structure specifies a test of whether a condition is true, and if true, specifies evaluation of a statement (or compound statement) repeatedly until the condition is no longer true */	
while(condition)	
statement	
/* A "do … while" structure specifies evaluation of a statement once, followed by a test of whether a condition is true, and if true, specifies repeated evaluation of the statement until the condition is no longer true */	
do	
statement	
while(condition)	
/* An "if … else" structure specifies a test of whether a condition is true, and if the condition is true, specifies evaluation of a primary statement, otherwise, specifies evaluation of an alternative statement. The "else" part of the structure and the associated alternative statement is omitted if no alternative statement evaluation is needed */	

if(condition)	
primary statement	
else	
alternative statement	
/* A "for" structure specifies evaluation of an initial statement, followed by a test of a condition, and if the condition is true, specifies repeated evaluation of a primary statement followed by a subsequent statement until the condition is no longer true. */	
for(initial statement; condition; subsequent statement)	
primary statement	

7.2 Specification of syntax functions, categories, and descriptors

The functions presented here are used in the syntactical description. These functions 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.

byte_aligned( ) is specified as follows:

– If the current position in the bitstream is on a byte boundary, i.e., the next bit in the bitstream is the first bit in a byte, the return value of byte_aligned( ) is equal to TRUE.

– Otherwise, the return value of byte_aligned( ) is equal to FALSE.

more_data_in_byte_stream( ), which is used only in the byte stream NAL unit syntax structure specified in Annex B, is specified as follows:

– If more data follow in the byte stream, the return value of more_data_in_byte_stream( ) is equal to TRUE.

– Otherwise, the return value of more_data_in_byte_stream( ) is equal to FALSE.

more_rbsp_data( ) is specified as follows:

– If there is no more data in the RBSP, the return value of more_rbsp_data( ) is equal to FALSE.

– Otherwise, the RBSP data is searched for the last (least significant, right-most) bit equal to 1 that is present in the RBSP. Given the position of this bit, which is the first bit (rbsp_stop_one_bit) of the rbsp_trailing_bits( ) syntax structure, the following applies:

  – If there is more data in an RBSP before the rbsp_trailing_bits( ) syntax structure, the return value of more_rbsp_data( ) is equal to TRUE.

  – Otherwise, the return value of more_rbsp_data( ) is equal to FALSE.

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

more_rbsp_trailing_data( ) is specified as follows:

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

– Otherwise, the return value of more_rbsp_trailing_data( ) is equal to FALSE.

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

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

The following descriptors specify the parsing process of each syntax element

– 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).

– ce(v): context-adaptive variable-length entropy-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.2.

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

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

– me(v): mapped Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.1.

– se(v): signed integer Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.1.

– te(v): truncated Exp-Golomb-coded syntax element with left bit first. The parsing process for this descriptor is specified in subclause 9.1.

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

– ue(v): unsigned integer Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.1.

7.3 Syntax in tabular form

7.3.1 NAL unit syntax

nal_unit(NumBytesInNALunit) {	**Descriptor**
forbidden_zero_bit	f(1)
nal_ref_idc	u(2)
nal_unit_type	u(5)
NumBytesInRBSP = 0	
nalUnitHeaderBytes = 1	
for(i = nalUnitHeaderBytes; i < NumBytesInNALunit; i++)	
if(i + 2 < NumBytesInNALunit && next_bits(24) = = 0x000003) {	
rbsp_byte[NumBytesInRBSP++ **]**	b(8)
rbsp_byte[NumBytesInRBSP++ **]**	b(8)
i += 2	
emulation_prevention_three_byte /* equal to 0x03 */	f(8)
} else	
rbsp_byte[NumBytesInRBSP++ **]**	b(8)
}	

7.3.2 Raw byte sequence payloads and RBSP trailing bits syntax

7.3.2.1 Sequence parameter set RBSP syntax

seq_parameter_set_rbsp( ) {	**Descriptor**
seq_parameter_set_data( )	
rbsp_trailing_bits( )	
}	

7.3.2.1.1 Sequence parameter set data syntax

seq_parameter_set_data( ) {	**Descriptor**
profile_idc	u(8)
constraint_set0_flag /* normally equal to 1 */	u(1)
constraint_set1_flag /* normally equal to 1 */	u(1)
constraint_set2_flag /* normally equal to 1 */	u(1)
constraint_set3_flag	u(1)
constraint_set4_flag /* equal to 0; ignored by decoders */	u(1)
constraint_set5_flag /* equal to 0; ignored by decoders */	u(1)
reserved_zero_2bits /* equal to 0 */	u(2)
level_idc	u(8)
seq_parameter_set_id	ue(v)
log2_max_frame_num_minus4	ue(v)
pic_order_cnt_type	ue(v)
if(pic_order_cnt_type = = 0)	
log2_max_pic_order_cnt_lsb_minus4	ue(v)
else if(pic_order_cnt_type = = 1) {	
delta_pic_order_always_zero_flag	u(1)
offset_for_non_ref_pic	se(v)
offset_for_top_to_bottom_field	se(v)
num_ref_frames_in_pic_order_cnt_cycle	ue(v)
for(i = 0; i < num_ref_frames_in_pic_order_cnt_cycle; i++)	
offset_for_ref_frame[i]	se(v)
}	
max_num_ref_frames	ue(v)
gaps_in_frame_num_value_allowed_flag	u(1)
pic_width_in_mbs_minus1	ue(v)
pic_height_in_map_units_minus1	ue(v)
frame_mbs_only_flag /*equal to 1*/	u(1)
direct_8x8_inference_flag	u(1)
frame_cropping_flag	u(1)
if(frame_cropping_flag) {	
frame_crop_left_offset	ue(v)
frame_crop_right_offset	ue(v)
frame_crop_top_offset	ue(v)
frame_crop_bottom_offset	ue(v)
}	
vui_parameters_present_flag	u(1)
if(vui_parameters_present_flag)	
vui_parameters( )	
}	

7.3.2.2 Picture parameter set RBSP syntax

pic_parameter_set_rbsp( ) {	**Descriptor**
pic_parameter_set_id	ue(v)
seq_parameter_set_id	ue(v)
entropy_coding_mode_flag /*equal to zero*/	u(1)
bottom_field_pic_order_in_frame_present_flag	u(1)
num_slice_groups_minus1 /*equal to zero*/	ue(v)
num_ref_idx_l0_default_active_minus1	ue(v)
num_ref_idx_l1_default_active_minus1	ue(v)
weighted_pred_flag /* = 0 */	u(1)
weighted_bipred_idc /* = 0 */	u(2)
pic_init_qp_minus26 /* relative to 26 */	se(v)
pic_init_qs_minus26 /* relative to 26 */	se(v)
chroma_qp_index_offset	se(v)
deblocking_filter_control_present_flag	u(1)
constrained_intra_pred_flag	u(1)
redundant_pic_cnt_present_flag /* equal to zero*/	u(1)
rbsp_trailing_bits( )	
}	

7.3.2.3 Supplemental enhancement information RBSP syntax

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

7.3.2.3.1 Supplemental enhancement information message syntax

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

7.3.2.4 Access unit delimiter RBSP syntax

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

7.3.2.5 End of sequence RBSP syntax

end_of_seq_rbsp( ) {	**Descriptor**
}	

7.3.2.6 End of stream RBSP syntax

end_of_stream_rbsp( ) {	**Descriptor**
}	

7.3.2.7 Filler data RBSP syntax

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

7.3.2.8 Slice layer RBSP syntax

slice_layer_rbsp( ) {	**Descriptor**
slice_header( )	
slice_data( ) /* all categories of slice_data( ) syntax */	
rbsp_slice_trailing_bits( )	
}	

7.3.2.9 (void)

7.3.2.10 RBSP slice trailing bits syntax

rbsp_slice_trailing_bits( ) {	**Descriptor**
rbsp_trailing_bits( )	
}	

7.3.2.11 RBSP trailing bits syntax

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

7.3.3 Slice header syntax

slice_header( ) {	**Descriptor**
first_mb_in_slice	ue(v)
slice_type	ue(v)
pic_parameter_set_id	ue(v)
frame_num	u(v)
if(IdrPicFlag)	
idr_pic_id	ue(v)
if(pic_order_cnt_type = = 0) {	
pic_order_cnt_lsb	u(v)
if(bottom_field_pic_order_in_frame_present_flag)	
delta_pic_order_cnt_bottom	se(v)
}	
if(pic_order_cnt_type = = 1 && !delta_pic_order_always_zero_flag) {	
delta_pic_order_cnt[0]	se(v)
if(bottom_field_pic_order_in_frame_present_flag)	
delta_pic_order_cnt[1]	se(v)
}	
if(slice_type = = P) {	
num_ref_idx_active_override_flag	u(1)
if(num_ref_idx_active_override_flag)	
num_ref_idx_l0_active_minus1	ue(v)
}	
ref_pic_list_modification( )	
if(nal_ref_idc != 0)	
dec_ref_pic_marking( )	
slice_qp_delta	se(v)
if(deblocking_filter_control_present_flag) {	
disable_deblocking_filter_idc	ue(v)
if(disable_deblocking_filter_idc != 1) {	
slice_alpha_c0_offset_div2	se(v)
slice_beta_offset_div2	se(v)
}	
}	
}	

7.3.3.1 Reference picture list modification syntax

ref_pic_list_modification( ) {	**Descriptor**
if(slice_type % 5 != 2 && slice_type % 5 != 4) {	
ref_pic_list_modification_flag_l0	u(1)
if(ref_pic_list_modification_flag_l0)	
do {	
modification_of_pic_nums_idc	ue(v)
if(modification_of_pic_nums_idc = = 0 || modification_of_pic_nums_idc = = 1)	
abs_diff_pic_num_minus1	ue(v)
else if(modification_of_pic_nums_idc = = 2)	
long_term_pic_num	ue(v)
} while(modification_of_pic_nums_idc != 3)	
}	
}	

7.3.3.2 (void)

7.3.3.3 Decoded reference picture marking syntax

dec_ref_pic_marking( ) {	**Descriptor**
if(IdrPicFlag) {	
no_output_of_prior_pics_flag	u(1)
long_term_reference_flag	u(1)
} else {	
adaptive_ref_pic_marking_mode_flag	u(1)
if(adaptive_ref_pic_marking_mode_flag)	
do {	
memory_management_control_operation	ue(v)
if(memory_management_control_operation = = 1 || memory_management_control_operation = = 3)	
difference_of_pic_nums_minus1	ue(v)
if(memory_management_control_operation = = 2)	
long_term_pic_num	ue(v)
if(memory_management_control_operation = = 3 || memory_management_control_operation = = 6)	
long_term_frame_idx	ue(v)
if(memory_management_control_operation = = 4)	
max_long_term_frame_idx_plus1	ue(v)
} while(memory_management_control_operation != 0)	
}	
}	

7.3.4 Slice data syntax

slice_data( ) {	**Descriptor**
CurrMbAddr = first_mb_in_slice	
moreDataFlag = 1	
prevMbSkipped = 0	
do {	
if(slice_type != I) {	
mb_skip_run	ue(v)
prevMbSkipped = (mb_skip_run > 0)	
for(i=0; i<mb_skip_run; i++)	
CurrMbAddr = NextMbAddress(CurrMbAddr)	
if(mb_skip_run > 0)	
moreDataFlag = more_rbsp_data( )	
}	
if(moreDataFlag)	
macroblock_layer( )	
moreDataFlag = more_rbsp_data( )	
CurrMbAddr = NextMbAddress(CurrMbAddr)	
} while(moreDataFlag)	
}	

7.3.5 Macroblock layer syntax

macroblock_layer( ) {	**Descriptor**
mb_type	ue(v)
if(mb_type = = I_PCM) {	
while(!byte_aligned( ))	
pcm_alignment_zero_bit	f(1)
for(i = 0; i < 256; i++)	
pcm_sample_luma[i]	u(v)
for(i = 0; i < 128)	
pcm_sample_chroma[i]	u(v)
} else {	
if(mb_type != I_4x4 && MbPartPredMode(mb_type, 0) != Intra_16x16 && NumMbPart(mb_type) = = 4)	
sub_mb_pred(mb_type)	
else	
mb_pred(mb_type)	
if(MbPartPredMode(mb_type, 0) != Intra_16x16)	
coded_block_pattern	
if(CodedBlockPatternLuma > 0 || CodedBlockPatternChroma > 0 || MbPartPredMode(mb_type, 0) = = Intra_16x16) {	
mb_qp_delta	se(v)
residual( )	
}	
}	
}	

7.3.5.1 Macroblock prediction syntax

mb_pred(mb_type) {	**Descriptor**
if(MbPartPredMode(mb_type, 0) = = Intra_4x4 || MbPartPredMode(mb_type, 0) = = Intra_16x16) {	
if(MbPartPredMode(mb_type, 0) = = Intra_4x4)	
for(luma4x4BlkIdx=0; luma4x4BlkIdx<16; luma4x4BlkIdx++) {	
prev_intra4x4_pred_mode_flag[luma4x4BlkIdx]	u(1)
if(!prev_intra4x4_pred_mode_flag[luma4x4BlkIdx])	
rem_intra4x4_pred_mode[luma4x4BlkIdx]	u(3)
}	
intra_chroma_pred_mode	ue(v)
} else {	
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)	
if(num_ref_idx_l0_active_minus1 > 0)	
ref_idx_l0[mbPartIdx]	te(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)	
for(compIdx = 0; compIdx < 2; compIdx++)	
mvd_l0[mbPartIdx][0][compIdx]	se(v)
}	
}	

7.3.5.2 Sub-macroblock prediction syntax

sub_mb_pred(mb_type) {	**Descriptor**
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)	
sub_mb_type[mbPartIdx **]**	ue(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)	
if(num_ref_idx_l0_active_minus1 > 0 && mb_type != P_8x8ref0	
ref_idx_l0[mbPartIdx **]**	te(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)	
for(subMbPartIdx = 0; subMbPartIdx < NumSubMbPart(sub_mb_type[mbPartIdx]); subMbPartIdx++)	
for(compIdx = 0; compIdx < 2; compIdx++)	
mvd_l0[mbPartIdx **][** subMbPartIdx **][** compIdx **]**	se(v)
}	

7.3.5.3 Residual data syntax

residual( ) {	**Descriptor**
residual_luma(i16x16DClevel, i16x16AClevel, level4x4)	
Intra16x16DCLevel = i16x16DClevel	
Intra16x16ACLevel = i16x16AClevel	
LumaLevel4x4 = level4x4	
for(iCbCr = 0; iCbCr < 2; iCbCr++)	
if(CodedBlockPatternChroma & 3) /* chroma DC residual present */	
residual_block(ChromaDCLevel[iCbCr], 0, 3, 4)	
else	
for(i = 0; i < 4; i++)	
ChromaDCLevel[iCbCr][i] = 0	
for(iCbCr = 0; iCbCr < 2; iCbCr++)	
for(i4x4 = 0; i4x4 < 4; i4x4++)	
if(CodedBlockPatternChroma & 2) /* chroma AC residual present */	
residual_block(ChromaACLevel[iCbCr][i4x4], 0,14,15)	
else	
for(i = 0; i < 15; i++)	
ChromaACLevel[iCbCr][i4x4][i] = 0	
}	

7.3.5.3.1 Residual luma syntax

residual_luma(i16x16DClevel, i16x16AClevel, level4x4) {	**Descriptor**
if(MbPartPredMode(mb_type, 0) = = Intra_16x16)	
residual_block(i16x16DClevel, 0, 15, 16)	
for(i8x8 = 0; i8x8 < 4; i8x8++)	
for(i4x4 = 0; i4x4 < 4; i4x4++)	
if(CodedBlockPatternLuma & (1 << i8x8))	
if(MbPartPredMode(mb_type, 0) = = Intra_16x16)	
residual_block(i16x16AClevel[i8x8*4+ i4x4], 0, 14, 15)	
else	
residual_block(level4x4[i8x8* 4 + i4x4], 0, 15, 16)	
else if(MbPartPredMode(mb_type, 0) = = Intra_16x16)	
for(i = 0; i < 15; i++)	
i16x16AClevel[i8x8 * 4 + i4x4][i] = 0	
else	
for(i = 0; i < 16; i++)	
level4x4[i8x8 * 4 + i4x4][i] = 0	
}	

7.3.5.3.2 Residual block CAVLC syntax

residual_block(coeffLevel, startIdx, endIdx, maxNumCoeff) {	**Descriptor**
for(i = 0; i < maxNumCoeff; i++)	
coeffLevel[i] = 0	
coeff_token	ce(v)
if(TotalCoeff(coeff_token) > 0) {	
if(TotalCoeff(coeff_token) > 10 && TrailingOnes(coeff_token) < 3)	
suffixLength = 1	
else	
suffixLength = 0	
for(i = 0; i < TotalCoeff(coeff_token); i++)	
if(i < TrailingOnes(coeff_token)) {	
trailing_ones_sign_flag	u(1)
levelVal[i] = 1 − 2 * trailing_ones_sign_flag	
} else {	
level_prefix	ce(v)
levelCode = (level_prefix << suffixLength)	
if(suffixLength > 0 || level_prefix >= 14) {	
level_suffix	u(v)
levelCode += level_suffix	
}	
if(i = = TrailingOnes(coeff_token) && TrailingOnes(coeff_token) < 3)	
levelCode += 2	
if(levelCode % 2 = = 0)	
levelVal[i] = (levelCode + 2) >> 1	
else	
levelVal[i] = (−levelCode − 1) >> 1	
if(suffixLength = = 0)	
suffixLength = 1	
if(Abs(levelVal[i]) > (3 << (suffixLength − 1)) && suffixLength < 6)	
suffixLength++	
}	
if(TotalCoeff(coeff_token) <endIdx − startIdx + 1) {	
total_zeros	ce(v)
zerosLeft = total_zeros	
} else	
zerosLeft = 0	
for(i = 0; i < TotalCoeff(coeff_token) − 1; i++) {	
if(zerosLeft > 0) {	
run_before	ce(v)
runVal[i] = run_before	
} else	
runVal[i] = 0	
zerosLeft = zerosLeft − runVal[i]	
}	
runVal[TotalCoeff(coeff_token) − 1] = zerosLeft	
coeffNum = −1	
for(i = TotalCoeff(coeff_token) − 1; i >= 0; i− −) {	
coeffNum += runVal[i] + 1	

coeffLevel[startIdx + coeffNum] = levelVal[i]	
}	
}	
}	

7.4 Semantics

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

7.4.1 NAL unit semantics

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

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 International Standard.

forbidden_zero_bit shall be equal to 0.

nal_ref_idc not equal to 0 specifies that the content of the NAL unit contains a sequence parameter set, a picture parameter set, or a slice of a reference picture.

For coded video sequences conforming to one or more of the profiles specified in Annex A that are decoded using the decoding process specified in clauses 2-9, nal_ref_idc equal to 0 for a NAL unit containing a slice indicates that the slice is part of a non-reference picture.

nal_ref_idc shall not be equal to 0 for sequence parameter set or picture parameter set NAL units. When nal_ref_idc is equal to 0 for one NAL unit with nal_unit_type in the range of 1 to 4, inclusive, of a particularpicture, it shall be equal to 0 for all NAL units with nal_unit_type in the range of 1 to 4, inclusive, of the picture.

nal_ref_idc shall not be equal to 0 for NAL units with nal_unit_type equal to 5.

nal_ref_idc shall be equal to 0 for all NAL units having nal_unit_type equal to 6, 9, 10, 11, or 12.

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

For coded video sequences conforming to one or more of the profiles specified in Annex A that are decoded using the decoding process specified in clauses 2-9, VCL and non-VCL NAL units are specified in Table 7-1 in the column labelled "Annex A NAL unit type class".

Table 7-1 – NAL unit type codes, syntax element categories, and NAL unit type classes

nal_unit_type	Content of NAL unit and RBSP syntax structure	Annex A NAL unit type class
0	Unspecified	non-VCL
1	Coded slice of a non-IDR picture slice_layer_ rbsp( )	VCL
2	Reserved	VCL
3	Reserved	VCL
4	Reserved	VCL
5	Coded slice of an IDR picture slice_layer_rbsp( )	VCL
6	Supplemental enhancement information (SEI) sei_rbsp( )	non-VCL
7	Sequence parameter set seq_parameter_set_rbsp( )	non-VCL
8	Picture parameter set pic_parameter_set_rbsp( )	non-VCL
9	Access unit delimiter access_unit_delimiter_rbsp( )	non-VCL
10	End of sequence end_of_seq_rbsp( )	non-VCL
11	End of stream end_of_stream_rbsp( )	non-VCL
12	Filler data filler_data_rbsp( )	non-VCL
13	Reserved	non-VCL
14	Reserved	non-VCL
15	Reserved	non-VCL
16..18	Reserved	non-VCL
19	Reserved	non-VCL
20	Reserved	non-VCL
21..23	Reserved	non-VCL
24..31	Unspecified	non-VCL

NAL units that use nal_unit_type equal to 0 or in the range of 24..31, inclusive, shall not affect the decoding process specified in this International Standard.

NOTE – NAL unit types 0 and 24..31 may be used as determined by the application. No decoding process for these values of nal_unit_type is specified in this International Standard. Since different applications might use NAL unit types 0 and 24..31 for different purposes, particular care must be exercised in the design of encoders that generate NAL units with nal_unit_type equal to 0 or in the range of 24 to 31, inclusive, and in the design of decoders that interpret the content of NAL units with nal_unit_type equal to 0 or in the range of 24 to 31, inclusive.

Decoders shall ignore (remove from the bitstream and discard) the contents of all NAL units that use reserved values of nal_unit_type.

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

In the text, coded slice NAL unit collectively refers to a coded slice of a non-IDR picture NAL unit or to a coded slice of an IDR picture NAL unit. The variable IdrPicFlag is specified as

IdrPicFlag = ((nal_unit_type == 5) ? 1 : 0) (7-1)

When the value of nal_unit_type is equal to 5 for a NAL unit containing a slice of a particular picture, the picture shall not contain NAL units with nal_unit_type in the range of 1 to 4, inclusive. For coded video sequences conforming to one or more of the profiles specified in Annex A that are decoded using the decoding process specified in clauses 2-9,such a picture is referred to as an IDR picture.

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

The RBSP contains an SODB as follows:

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

– Otherwise, the RBSP contains the SODB as follows:

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

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 shall be 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 shall be as specified in Table 7-1.

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

NOTE – When nal_unit_type is equal to 0, particular care must be exercised in the design of encoders to avoid the presence of the above-listed three-byte and four-byte patterns at the beginning of the NAL unit syntax structure, as the syntax element emulation_prevention_three_byte cannot be the third byte of a NAL unit.

7.4.1.1 Encapsulation of an SODB within an RBSP (informative)

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

The form of encapsulation of an SODB within an RBSP and the use of the emulation_prevention_three_byte for encapsulation of an RBSP within a NAL unit is specified for the following purposes:

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

– to enable identification of the end of the SODB within the NAL unit by searching the RBSP for the rbsp_stop_one_bit starting at the end of the RBSP,

The encoder can produce a NAL unit from an RBSP by the following procedure:

1. The RBSP data is searched for byte-aligned bits of the following binary patterns:

   '00000000 00000000 000000xx' (where xx represents any 2 bit pattern: 00, 01, 10, or 11),

   and a byte equal to 0x03 is inserted to replace these bit patterns with the patterns:

   '00000000 00000000 00000011 000000xx',

2. The resulting sequence of bytes is prefixed with the first byte of the NAL unit containing the syntax elements forbidden_zero_bit, nal_ref_idc, and nal_unit_type, where nal_unit_type indicates the type of RBSP data structure the NAL unit contains.

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

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

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

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

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

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

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

7.4.1.2.1 Order of sequence and picture parameter set RBSPs and their activation

This subclause specifies the activation process of picture and sequence parameter sets for coded video sequences that conform to one or more of the profiles specified in Annex A and are decoded using the decoding process specified in clauses 2-9.

NOTE – The sequence and picture parameter set mechanism decouples the transmission of infrequently changing information from the transmission of coded macroblock data. Sequence and picture parameter sets may, in some applications, be conveyed "out-of-band" using a reliable transport mechanism.

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

When a picture parameter set RBSP (with a particular value of pic_parameter_set_id) is not active and it is referred to by a coded slice NAL unit (using that value of pic_parameter_set_id), it is activated. This picture parameter set RBSP is called the active picture parameter set RBSP until it is deactivated by the activation of another picture parameter set RBSP. A picture parameter set RBSP, with that particular value of pic_parameter_set_id, shall be available to the decoding process prior to its activation.

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

When a picture parameter set NAL unit with a particular value of pic_parameter_set_id is received, its content replaces the content of the previous picture parameter set NAL unit, in decoding order, with the same value of pic_parameter_set_id (when a previous picture parameter set NAL unit with the same value of pic_parameter_set_id was present in the bitstream).

NOTE – A decoder must be capable of simultaneously storing the contents of the picture parameter sets for all values of pic_parameter_set_id. The content of the picture parameter set with a particular value of pic_parameter_set_id is overwritten when a new picture parameter set NAL unit with the same value of pic_parameter_set_id is received.

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

When a sequence parameter set RBSP (with a particular value of seq_parameter_set_id) is not already active and it is referred to by activation of a picture parameter set RBSP (using that value of seq_parameter_set_id) or is referred to by an SEI NAL unit containing a buffering period SEI message (using that value of seq_parameter_set_id), it is activated. This sequence parameter set RBSP is called the active sequence parameter set RBSP until it is deactivated by the activation of another sequence parameter set RBSP. A sequence parameter set RBSP, with that particular value of seq_parameter_set_id, shall be available to the decoding process prior to its activation. An activated sequence parameter set RBSP shall remain active for the entire coded video sequence.

NOTE – Because an IDR access unit begins a new coded video sequence and an activated sequence parameter set RBSP must remain active for the entire coded video sequence, a sequence parameter set RBSP can only be activated by a buffering period SEI message when the buffering period SEI message is part of an IDR access unit.

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

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

When a sequence parameter set NAL unit with a particular value of seq_parameter_set_id is received, its content replaces the content of the previous sequence parameter set NAL unit, in decoding order, with the same value of seq_parameter_set_id (when a previous sequence parameter set NAL unit with the same value of seq_parameter_set_id was present in the bitstream).

NOTE – A decoder must be capable of simultaneously storing the contents of the sequence parameter sets for all values of seq_parameter_set_id. The content of the sequence parameter set with a particular value of seq_parameter_set_id is overwritten when a new sequence parameter set NAL unit with the same value of seq_parameter_set_id is received.

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

During operation of the decoding process (see clause 8), the values of parameters of the active picture parameter set and the active sequence parameter set shall be considered in effect.

7.4.1.2.2 Order of access units and association to coded video sequences

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

A coded video sequence consists of one or more access units. For coded video sequences that conform to one or more of the profiles specified in Annex A and are decoded using the decoding process specified in clauses 2-9, the order of NAL units and coded pictures and their association to access units is described in subclause 7.4.1.2.3.

The first access unit of each coded video sequence is an IDR access unit. All subsequent access units in the coded video sequence are non-IDR access units.

It is a requirement of bitstream conformance that, when two consecutive access units in decoding order within a coded video sequence both contain non-reference pictures, the value of picture order count for each coded frame in the first such access unit shall be less than or equal to the value of picture order count for each coded frame in the second such access unit.

It is a requirement of bitstream conformance that, when present, an access unit following an access unit that contains an end of sequence NAL unit shall be an IDR access unit.

It is a requirement of bitstream conformance that, when an SEI NAL unit contains data that pertain to more than one access unit (for example, when the SEI NAL unit has a coded video sequence as its scope), it shall be contained in the first access unit to which it applies.

It is a requirement of bitstream conformance that, when an end of stream NAL unit is present in an access unit, this access unit shall be the last access unit in the bitstream and the end of stream NAL unit shall be the last NAL unit in that access unit.

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

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

An access unit consists of one primary coded picture, , and zero or more non-VCL NAL units. The association of VCL NAL units to primary is described in subclause 7.4.1.2.5.

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

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

– access unit delimiter NAL unit (when present),

– sequence parameter set NAL unit (when present),

– picture parameter set NAL unit (when present),

– SEI NAL unit (when present),

– NAL units with nal_unit_type in the range of 14 to 18, inclusive (when present),

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

The constraints for the detection of the first VCL NAL unit of a primary coded picture are specified in subclause 7.4.1.2.4.

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

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

– When any SEI NAL units are present, they shall precede the primary coded picture.

– When an SEI NAL unit containing a buffering period SEI message (see Annex D) is present, the buffering period SEI message shall be the first SEI message payload of the first SEI NAL unit in the access unit.

– When an end of sequence NAL unit is present, it shall follow the primary coded picture

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

– NAL units having nal_unit_type equal to 0, 12, or in the range of 20 to 31, inclusive, shall not precede the first VCL NAL unit of the primary coded picture.

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

NOTE – When a NAL unit having nal_unit_type equal to 7 or 8 is present in an access unit, it may or may not be referred to in the coded pictures of the access unit in which it is present, and may be referred to in coded pictures of subsequent access units.

The structure of access units not containing any NAL units with nal_unit_type equal to 0, 7, 8, or in the range of 12 to 18, inclusive, or in the range of 20 to 31, inclusive, is shown in Figure 7-1.

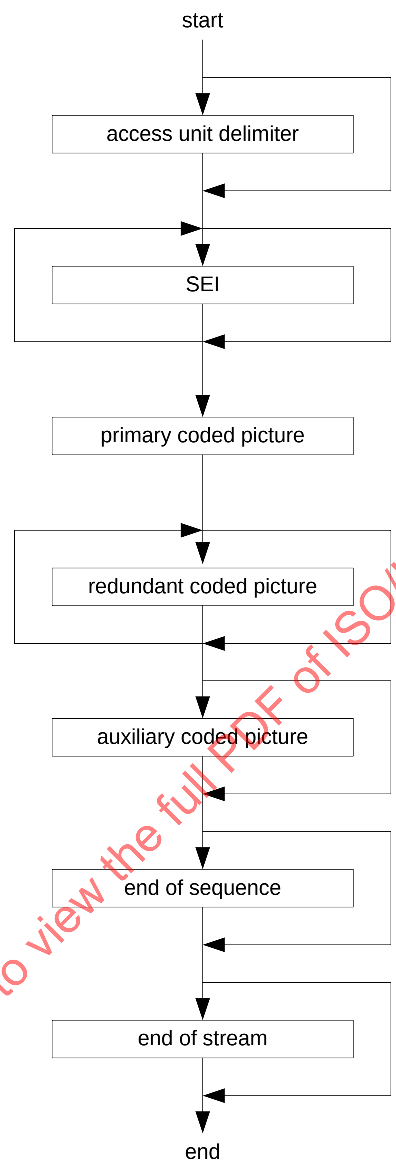

Figure 7-1 – Structure of an access unit not containing any NAL units with nal_unit_type equal to 0, 7, 8, or in the range of 12 to 18, inclusive, or in the range of 20 to 31, inclusive

7.4.1.2.4 Detection of the first VCL NAL unit of a primary coded picture

This subclause specifies constraints on VCL NAL unit syntax that are sufficient to enable the detection of the first VCL NAL unit of each primary coded picture for coded video sequences that conform to one or more of the profiles specified in Annex A and are decoded using the decoding process specified in clauses 2-9.

Any coded slice NAL unit of the primary coded picture of the current access unit shall be different from any coded slice NAL unit of the primary coded picture of the previous access unit in one or more of the following ways:

– frame_num differs in value. The value of frame_num used to test this condition is the value of frame_num that appears in the syntax of the slice header, regardless of whether that value is inferred to have been equal to 0 for subsequent use in the decoding process due to the presence of memory_management_control_operation equal to 5.

NOTE – A consequence of the above statement is that a primary coded picture having frame_num equal to 1 cannot contain a memory_management_control_operation equal to 5 unless some other condition listed below is fulfilled for the next primary coded picture that follows after it (if any).

– pic_parameter_set_id differs in value.

– nal_ref_idc differs in value with one of the nal_ref_idc values being equal to 0.

– pic_order_cnt_type is equal to 0 for both and pic_order_cnt_lsb differs in value.– pic_order_cnt_type is equal to 1 for both and either delta_pic_order_cnt[0] differs in value, or delta_pic_order_cnt[1] differs in value.

– IdrPicFlag differs in value.

– IdrPicFlag is equal to 1 for both and idr_pic_id differs in value.

NOTE – Some of the VCL NAL units in non-VCL NAL units (e.g., an access unit delimiter NAL unit) may also be used for the detection of the boundary between access units, and may therefore aid in the detection of the start of a new primary coded picture.

7.4.1.2.5 Order of VCL NAL units and association to coded pictures

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

Each VCL NAL unit is part of a coded picture.

The order of the VCL NAL units within a coded IDR picture is constrained as follows:

– the order of coded slice of an IDR picture NAL units shall be in the order of increasing macroblock address.

The order of the VCL NAL units within a coded non-IDR picture is constrained as follows:

– the order of coded slice of a non-IDR picture NAL units shall be in the order of increasing macroblock address.

NAL units having nal_unit_type equal to 12 may be present in the access unit but shall not precede the first VCL NAL unit of the primary coded picture within the access unit.

NAL units having nal_unit_type equal to 0 or in the range of 24 to 31, inclusive, which are unspecified, may be present in the access unit but shall not precede the first VCL NAL unit of the primary coded picture within the access unit.

NAL units having nal_unit_type in the range of 20 to 23, inclusive, shall not precede the first VCL NAL unit of the primary coded picture within the access unit.

7.4.2 Raw byte sequence payloads and RBSP trailing bits semantics

7.4.2.1 Sequence parameter set RBSP semantics

7.4.2.1.1 Sequence parameter set data semantics

profile_idc and **level_idc** indicate the profile and level to which the coded video sequence conforms.

constraint_set0_flag equal to 1, when profile_idc is equal to 66, has no meaning and should be equal to 1 and its value shall be ignored by decoders. When profile_idc is equal to 77, constraint_set0_flag equal to 1 indicates that the coded video sequence obeys all constraints specified in subclause A.2.1 and constraint_set0_flag equal to 0 indicates that the coded video sequence may or may not obey all constraints specified in subclause A.2.1. When profile_idc is not equal to 66 or 77, constraint_set0_flag is interpreted together with constraint_set1_flag as specified below.

constraint_set1_flag equal to 1, when profile_idc is equal to 66 or constraint_set0_flag is equal to 1, indicates that the coded video sequence obeys all constraints specified in subclause A.2.1. When profile_idc is not equal to 66 and constraint_set0_flag is not equal to 1, constraint_set1_flag equal to 0 indicates that the coded video sequence may or may not obey all constraints specified in subclause A.2.1.

constraint_set2_flag is not used in this specification; the value shall be ignored by decoders, and should be set to 1 by encoders.

constraint_set3_flag is specified as follows:

– If profile_idc is equal to 66, 77, or 88 and level_idc is equal to 11, constraint_set3_flag equal to 1 indicates that the coded video sequence obeys all constraints specified in Annex A for level 1b and constraint_set3_flag equal to 0 indicates that the coded video sequence obeys all constraints specified in Annex A for level 1.1.
– Otherwise the value of 1 for constraint_set3_flag is reserved for future use by ITU-T | ISO/IEC. In this case, decoders shall ignore the value of constraint_set3_flag.

constraint_set4_flag is reserved for future use by ITU-T | ISO/IEC; the value shall be ignored by decoders, and shall be set to 0 by encoders.

constraint_set5_flag is reserved for future use by ITU-T | ISO/IEC; the value shall be ignored by decoders, and shall be set to 0 by encoders.

reserved_zero_2bits shall be equal to 0. Other values of reserved_zero_2bits may be specified in the future by ITU-T | ISO/IEC. Decoders shall ignore the value of reserved_zero_2bits.

seq_parameter_set_id identifies the sequence parameter set that is referred to by the picture parameter set. The value of seq_parameter_set_id shall be in the range of 0 to 31, inclusive.

NOTE – When feasible, encoders should use distinct values of seq_parameter_set_id when the values of other sequence parameter set syntax elements differ rather than changing the values of the syntax elements associated with a specific value of seq_parameter_set_id.

log2_max_frame_num_minus4 specifies the value of the variable MaxFrameNum that is used in frame_num related derivations as follows:

$$\text{MaxFrameNum} = 2^{(\text{log2\_max\_frame\_num\_minus4} + 4)} \tag{7-2}$$

The value of log2_max_frame_num_minus4 shall be in the range of 0 to 12, inclusive.

pic_order_cnt_type specifies the method to decode picture order count (as specified in subclause 8.2.1). The value of pic_order_cnt_type shall be in the range of 0 to 2, inclusive.

pic_order_cnt_type shall not be equal to 2 in a coded video sequence that contains an access unit containing a non-reference frame followed immediately by an access unit containing a non-reference picture,

log2_max_pic_order_cnt_lsb_minus4 specifies the value of the variable MaxPicOrderCntLsb that is used in the decoding process for picture order count as specified in subclause 8.2.1 as follows:

$$\text{MaxPicOrderCntLsb} = 2^{(\text{log2\_max\_pic\_order\_cnt\_lsb\_minus4} + 4)} \tag{7-3}$$

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

delta_pic_order_always_zero_flag equal to 1 specifies that delta_pic_order_cnt[0] and delta_pic_order_cnt[1] are not present in the slice headers of the sequence and shall be inferred to be equal to 0. delta_pic_order_always_zero_flag equal to 0 specifies that delta_pic_order_cnt[0] is present in the slice headers of the sequence and delta_pic_order_cnt[1] may be present in the slice headers of the sequence.

offset_for_non_ref_pic is used to calculate the picture order count of a non-reference picture as specified in subclause 8.2.1. The value of offset_for_non_ref_pic shall be in the range of $-2^{31} + 1$ to $2^{31} - 1$, inclusive.

num_ref_frames_in_pic_order_cnt_cycle is used in the decoding process for picture order count as specified in subclause 8.2.1. The value of num_ref_frames_in_pic_order_cnt_cycle shall be in the range of 0 to 255, inclusive.

offset_for_ref_frame[i **]** is an element of a list of num_ref_frames_in_pic_order_cnt_cycle values used in the decoding process for picture order count as specified in subclause 8.2.1. The value of offset_for_ref_frame[i] shall be in the range of $-2^{31} + 1$ to $2^{31} - 1$, inclusive.

When pic_order_cnt_type is equal to 1, the variable ExpectedDeltaPerPicOrderCntCycle is derived by

```
ExpectedDeltaPerPicOrderCntCycle = 0
for( i = 0;  i < num_ref_frames_in_pic_order_cnt_cycle;  i++ )
    ExpectedDeltaPerPicOrderCntCycle += offset_for_ref_frame[ i ]          (7-4)
```

max_num_ref_frames specifies the maximum number of short-term and long-term reference frames that may be used by the decoding process for inter prediction of any picture in the coded video sequence. max_num_ref_frames also determines the size of the sliding window operation as specified in subclause 8.2.5.3. The value of max_num_ref_frames shall be in the range of 0 toMaxDpbFrames (as specified in subclause A.3.1), inclusive.

gaps_in_frame_num_value_allowed_flag specifies the allowed values of frame_num as specified in subclause 7.4.3 and the decoding process in case of an inferred gap between values of frame_num as specified in subclause 8.2.5.2.

pic_width_in_mbs_minus1 plus 1 specifies the width of each decoded picture in units of macroblocks.

The variable for the picture width in units of macroblocks is derived as

$$PicWidthInMbs = pic\_width\_in\_mbs\_minus1 + 1 \quad (7\text{-}5)$$

The variable for picture width for the luma component is derived as

$$PicWidthInSamples_L = PicWidthInMbs * 16 \quad (7\text{-}6)$$

The variable for picture width for the chroma components is derived as

$$PicWidthInSamples_C = PicWidthInMbs * MbWidthC \quad (7\text{-}7)$$

frame_mbs_only_flag shall be equal to 1 and specifies that every coded picture of the coded video sequence is a coded frame containing only frame macroblocks.

The allowed range of values for pic_width_in_mbs_minus1, pic_height_in_map_units_minus1, is specified by constraints in Annex A.

pic_height_in_map_units_minus1 plus 1 is the height of a frame in units of macroblocks.

The variable FrameHeightInMbs is derived as

$$FrameHeightInMbs = (2 - frame\_mbs\_only\_flag) * PicHeightInMapUnits \quad (7\text{-}8)$$

direct_8x8_inference_flag is not used and may have any value.

frame_cropping_flag equal to 1 specifies that the frame cropping offset parameters follow next in the sequence parameter set. frame_cropping_flag equal to 0 specifies that the frame cropping offset parameters are not present.

frame_crop_left_offset, frame_crop_right_offset, frame_crop_top_offset, frame_crop_bottom_offset specify the samples of the pictures in the coded video sequence that are output from the decoding process, in terms of a rectangular region specified in frame coordinates for output.

The variables CropUnitX and CropUnitY are derived as follows:

$$CropUnitX = 2 \quad (7\text{-}9)$$
$$CropUnitY = 2 * (2 - frame\_mbs\_only\_flag) \quad (7\text{-}10)$$

The frame cropping rectangle contains luma samples with horizontal frame coordinates from CropUnitX * frame_crop_left_offset to $PicWidthInSamples_L$ − (CropUnitX * frame_crop_right_offset + 1) and vertical frame coordinates from CropUnitY * frame_crop_top_offset to (16 * FrameHeightInMbs) − (CropUnitY * frame_crop_bottom_offset + 1), inclusive. The value of frame_crop_left_offset shall be in the range of 0 to ($PicWidthInSamples_L$ / CropUnitX) − (frame_crop_right_offset + 1), inclusive; and the value of frame_crop_top_offset shall be in the range of 0 to (16 * FrameHeightInMbs / CropUnitY) − (frame_crop_bottom_offset + 1), inclusive.

When frame_cropping_flag is equal to 0, the values of frame_crop_left_offset, frame_crop_right_offset, frame_crop_top_offset, and frame_crop_bottom_offset shall be inferred to be equal to 0.

The corresponding specified samples of the two chroma arrays are the samples having frame coordinates (x / 2, y / 2), where (x, y) are the frame coordinates of the specified luma samples.

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

7.4.2.2 Picture parameter set RBSP semantics

pic_parameter_set_id identifies the picture parameter set that is referred to in the slice header. The value of pic_parameter_set_id shall be in the range of 0 to 255, inclusive.

seq_parameter_set_id refers to the active sequence parameter set. The value of seq_parameter_set_id shall be in the range of 0 to 31, inclusive.

entropy_coding_mode_flag selects the entropy decoding method to be applied for the syntax elements. It shall be equal to 0, and the method specified by the left descriptor in the syntax table is applied (Exp-Golomb coded, see subclause 9.1 or CAVLC, see subclause 9.2).

bottom_field_pic_order_in_frame_present_flag equal to 1 specifies that the syntax elements delta_pic_order_cnt_bottom (when pic_order_cnt_type is equal to 0) or delta_pic_order_cnt[1] (when pic_order_cnt_type is equal to 1), which are related to picture order counts for the bottom field of a coded frame, are present in the slice headers for coded frames as specified in subclause 7.3.3. bottom_field_pic_order_in_frame_present_flag equal to 0 specifies that the syntax elements delta_pic_order_cnt_bottom and delta_pic_order_cnt[1] are not present in the slice headers.

num_slice_groups_minus1 shall be equal to 0.

num_ref_idx_l0_default_active_minus1 specifies how num_ref_idx_l0_active_minus1 is inferred for P slices with num_ref_idx_active_override_flag equal to 0. The value of num_ref_idx_l0_default_active_minus1 shall be in the range of 0 to 31, inclusive.

num_ref_idx_l1_default_active_minus1 is not used and shall be in the range of 0 to 31, inclusive.

weighted_pred_flag shall be equal to 0.

weighted_bipred_idc shall be equal to 0.

pic_init_qp_minus26 specifies the initial value minus 26 of $SliceQP_Y$ for each slice. The initial value is modified at the slice layer when a non-zero value of slice_qp_delta is decoded, and is modified further when a non-zero value of mb_qp_delta is decoded at the macroblock layer. The value of pic_init_qp_minus26 shall be in the range of −(26 + 0) to +25, inclusive.

pic_init_qs_minus26 is not used and shall be in the range of −26 to +25, inclusive.

chroma_qp_index_offset specifies the offset that shall be added to QP_Y and QS_Y for addressing the table of QP_C values for the Cb chroma component. The value of chroma_qp_index_offset shall be in the range of −12 to +12, inclusive.

deblocking_filter_control_present_flag equal to 1 specifies that a set of syntax elements controlling the characteristics of the deblocking filter is present in the slice header. deblocking_filter_control_present_flag equal to 0 specifies that the set of syntax elements controlling the characteristics of the deblocking filter is not present in the slice headers and their inferred values are in effect.

constrained_intra_pred_flag equal to 0 specifies that intra prediction allows usage of residual data and decoded samples of neighbouring macroblocks coded using Inter macroblock prediction modes for the prediction of macroblocks coded using Intra macroblock prediction modes. constrained_intra_pred_flag equal to 1 specifies constrained intra prediction, in which case prediction of macroblocks coded using Intra macroblock prediction modes only uses residual data and decoded samples from I macroblock types.

redundant_pic_cnt_present_flag shall be equal to 0 and specifies that the redundant_pic_cnt syntax element is not present in slice headers that refer to the picture parameter set.

7.4.2.3 Supplemental enhancement information RBSP semantics

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

7.4.2.3.1 Supplemental enhancement information message semantics

An SEI RBSP contains one or more SEI messages. Each SEI message consists of the variables specifying the type payloadType and size payloadSize of the SEI payload. SEI payloads, identified herein as the sei_payload( ) syntax structure, are specified by Annex D. The derived SEI payload size payloadSize is specified in bytes and shall be equal to the number of RBSP bytes in the SEI payload.

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

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

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

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

7.4.2.4 Access unit delimiter RBSP semantics

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

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

NOTE – The value of primary_pic_type applies to the slice_type values in all slice headers of the primary coded picture, including the slice_type syntax elements in all NAL units with nal_unit_type equal to 1 or 5.

Table 7-2 – Meaning of primary_pic_type

primary_pic_type	slice_type values that may be present in the primary coded picture
0	2, 7
1	0, 2, 5, 7
2	0, 2, 5, 7
3	4, 9
4	3, 4, 8, 9
5	2, 4, 7, 9
6	0, 2, 3, 4, 5, 7, 8, 9
7	0, 2, 3, 4, 5, 7, 8, 9

7.4.2.5 End of sequence RBSP semantics

The end of sequence RBSP specifies that the next subsequent access unit in the bitstream in decoding order (if any) shall be an IDR access unit. The syntax content of the SODB and RBSP for the end of sequence RBSP are empty. No normative decoding process is specified for an end of sequence RBSP.

7.4.2.6 End of stream RBSP semantics

The end of stream RBSP indicates that no additional NAL units shall be present in the bitstream that are subsequent to the end of stream RBSP in decoding order. The syntax content of the SODB and RBSP for the end of stream RBSP are empty. No normative decoding process is specified for an end of stream RBSP.

NOTE – When an end of stream NAL unit is present, the bitstream is considered to end (for purposes of the scope of this International Standard). In some system environments, another bitstream may follow after the bitstream that has ended, either immediately or at some time thereafter, possibly within the same communication channel. Under such circumstances, the scope of this International Standard applies only to the processing of each of these individual bitstreams. No requirements are specified herein regarding the transition between such bitstreams (e.g., in regard to timing, buffering operation, etc.).

7.4.2.7 Filler data RBSP semantics

The filler data RBSP contains zero or more bytes. No normative decoding process is specified for a filler data RBSP.

ff_byte is a byte. It is a requirement of bitstream conformance that the value of ff_byte shall be equal to 0xFF.

7.4.2.8 Slice layer without partitioning RBSP semantics

The slice layer without partitioning RBSP consists of a slice header and slice data.

7.4.2.9 (void)

7.4.2.10 RBSP slice trailing bits semantics

In this Specification, the RBSP trailing bits syntax and semantics are the same as the RBSP trailing bits syntax and semantics.

7.4.2.11 RBSP trailing bits semantics

rbsp_stop_one_bit shall be equal to 1.

rbsp_alignment_zero_bit shall be equal to 0.

7.4.3 Slice header semantics

When present, the value of the slice header syntax elements pic_parameter_set_id, frame_num, , idr_pic_id, pic_order_cnt_lsb, delta_pic_order_cnt[0], and delta_pic_order_cnt[1] shall be the same in all slice headers of a coded picture.

first_mb_in_slice specifies the address of the first macroblock in the slice. the value of first_mb_in_slice is constrained as follows:

– the value of first_mb_in_slice shall not be less than the value of first_mb_in_slice for any other slice of the current picture that precedes the current slice in decoding order.

The first macroblock address of the slice is derived as follows:

– first_mb_in_slice is the macroblock address of the first macroblock in the slice, and first_mb_in_slice shall be in the range of 0 to PicSizeInMbs − 1, inclusive.

slice_type specifies the coding type of the slice according to Table 7-3. Reserved slice_type values shall not be present in the slice header.

Table 7-3 – Name association to slice_type

slice_type	Name of slice_type
0	P (P slice)
1	Reserved
2	I (I slice)
3	Reserved
4	Reserved
5	P (P slice)
6	Reserved
7	I (I slice)
8	Reserved
9	Reserved

When slice_type has a value in the range 5..9, it is a requirement of bitstream conformance that all other slices of the current coded picture shall have a value of slice_type equal to the current value of slice_type or equal to the current value of slice_type minus 5.

NOTE – Values of slice_type in the range 5..9 can be used by an encoder to indicate that all slices of a picture have the same value of (slice_type % 5). Values of slice_type in the range 5..9 are otherwise equivalent to corresponding values in the range 0..4.

When nal_unit_type is equal to 5 (IDR picture), slice_type shall be equal to 2 or 7.

When max_num_ref_frames is equal to 0, slice_type shall be equal to 2 or 7.

pic_parameter_set_id specifies the picture parameter set in use. The value of pic_parameter_set_id shall be in the range of 0 to 255, inclusive.

frame_num is used as an identifier for pictures and shall be represented by log2_max_frame_num_minus4 + 4 bits in the bitstream. frame_num is constrained as follows:

The variable PrevRefFrameNum is derived as follows:

– If the current picture is an IDR picture, PrevRefFrameNum is set equal to 0.

– Otherwise (the current picture is not an IDR picture), PrevRefFrameNum is set as follows:

  – If the decoding process for gaps in frame_num specified in subclause 8.2.5.2 was invoked by the decoding process for an access unit that contained a non-reference picture that followed the previous access unit in decoding order that contained a reference picture, PrevRefFrameNum is set equal to the value of frame_num for the last of the "non-existing" reference frames inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2.

  – Otherwise, PrevRefFrameNum is set equal to the value of frame_num for the previous access unit in decoding order that contained a reference picture.

The value of frame_num is constrained as follows:

– If the current picture is an IDR picture, frame_num shall be equal to 0.

– Otherwise (the current picture is not an IDR picture), referring to the primary coded picture in the previous access unit in decoding order that contains a reference picture as the preceding reference picture, the value of frame_num for the current picture shall not be equal to PrevRefFrameNum unless all of the following three conditions are true:

  a) The current picture and the preceding reference picture belong to consecutive access units in decoding order.

  b) (void).

c) One or more of the following conditions is true:

– The preceding reference picture is an IDR picture,

– The preceding reference picture includes a memory_management_control_operation syntax element equal to 5,

NOTE – When the preceding reference picture includes a memory_management_control_operation syntax element equal to 5, PrevRefFrameNum is equal to 0.

– There is a primary coded picture that precedes the preceding reference picture and the primary coded picture that precedes the preceding reference picture does not have frame_num equal to PrevRefFrameNum,

– There is a primary coded picture that precedes the preceding reference picture and the primary coded picture that precedes the preceding reference picture is not a reference picture.

When the value of frame_num is not equal to PrevRefFrameNum, it is a requirement of bitstream conformance that the following constraints shall be obeyed:

a) There shall not be any previous frame in decoding order that is currently marked as "used for short-term reference" that has a value of frame_num equal to any value taken on by the variable UnusedShortTermFrameNum in the following:

```
UnusedShortTermFrameNum = ( PrevRefFrameNum + 1 ) % MaxFrameNum
while( UnusedShortTermFrameNum  !=  frame_num )                              (7-11)
    UnusedShortTermFrameNum = ( UnusedShortTermFrameNum + 1 ) % MaxFrameNum
```

b) The value of frame_num is constrained as follows:

– If gaps_in_frame_num_value_allowed_flag is equal to 0, the value of frame_num for the current picture shall be equal to (PrevRefFrameNum + 1) % MaxFrameNum.

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

  – If frame_num is greater than PrevRefFrameNum, there shall not be any non-reference pictures in the bitstream that follow the previous reference picture and precede the current picture in decoding order in which either of the following conditions is true:

    – The value of frame_num for the non-reference picture is less than PrevRefFrameNum,

    – The value of frame_num for the non-reference picture is greater than the value of frame_num for the current picture.

  – Otherwise (frame_num is less than PrevRefFrameNum), there shall not be any non-reference pictures in the bitstream that follow the previous reference picture and precede the current picture in decoding order in which both of the following conditions are true:

    – The value of frame_num for the non-reference picture is less than PrevRefFrameNum,

    – The value of frame_num for the non-reference picture is greater than the value of frame_num for the current picture.

A picture including a memory_management_control_operation equal to 5 shall have frame_num constraints as described above and, after the decoding of the current picture and the processing of the memory management control operations, the picture shall be inferred to have had frame_num equal to 0 for all subsequent use in the decoding process, except as specified in subclause 7.4.1.2.4.

idr_pic_id identifies an IDR picture. The values of idr_pic_id in all the slices of an IDR picture shall remain unchanged. When two consecutive access units in decoding order are both IDR access units, the value of idr_pic_id in the slices of the first such IDR access unit shall differ from the idr_pic_id in the second such IDR access unit. The value of idr_pic_id shall be in the range of 0 to 65535, inclusive.

NOTE – It is not prohibited for multiple IDR pictures in a bitstream to have the same value of idr_pic_id unless such pictures occur in two consecutive access units in decoding order.

pic_order_cnt_lsb specifies the picture order count modulo MaxPicOrderCntLsb for a coded frame. The length of the pic_order_cnt_lsb syntax element is log2_max_pic_order_cnt_lsb_minus4 + 4 bits. The value of the pic_order_cnt_lsb shall be in the range of 0 to MaxPicOrderCntLsb − 1, inclusive.

delta_pic_order_cnt[0 **]** specifies the picture order count difference from the expected picture order count for the top field of a coded frame or for a coded field as specified in subclause 8.2.1. The value of delta_pic_order_cnt[0] shall be in the range of $-2^{31} + 1$ to $2^{31} - 1$, inclusive. When this syntax element is not present in the bitstream for the current slice, it shall be inferred to be equal to 0.

redundant_pic_cnt shall be equal to 0 and specifies that slices belong to the primary coded picture.

When the value of nal_ref_idc in one VCL NAL unit of an access unit is equal to 0, the value of nal_ref_idc in all other VCL NAL units of the same access unit shall be equal to 0.

The marking status of reference pictures and the value of frame_num after the decoded reference picture marking process as specified in subclause 8.2.5 is invoked for the primary coded picture of the same access unit shall be identical regardless whether the primary coded picture of the access unit would be decoded.

num_ref_idx_active_override_flag equal to 1 specifies that the syntax element num_ref_idx_l0_active_minus1 is present for P slices. num_ref_idx_active_override_flag equal to 0 specifies that the syntax element num_ref_idx_l0_active_minus1 is not present.

When the current slice is a P slice and the value of num_ref_idx_l0_default_active_minus1 in the picture parameter set exceeds 15, num_ref_idx_active_override_flag shall be equal to 1.

num_ref_idx_l0_active_minus1specifies the maximum reference index for reference picture list 0 that shall be used to decode the slice.

When the current slice is a P slice and num_ref_idx_l0_active_minus1 is not present, num_ref_idx_l0_active_minus1 shall be inferred to be equal to num_ref_idx_l0_default_active_minus1.

The range of num_ref_idx_l0_active_minus1 is specified as follows:

num_ref_idx_l0_active_minus1 shall be in the range of 0 to 15, inclusive.

slice_qp_delta specifies the initial value of QP_Y to be used for all the macroblocks in the slice until modified by the value of mb_qp_delta in the macroblock layer. The initial QP_Y quantisation parameter for the slice is computed as

$$SliceQP_Y = 26 + pic\_init\_qp\_minus26 + slice\_qp\_delta \quad (7\text{-}12)$$

The value of slice_qp_delta shall be limited such that $SliceQP_Y$ is in the range of −0 to +51, inclusive.

disable_deblocking_filter_idc specifies whether the operation of the deblocking filter shall be disabled across some block edges of the slice and specifies for which edges the filtering is disabled. When disable_deblocking_filter_idc is not present in the slice header, the value of disable_deblocking_filter_idc shall be inferred to be equal to 0.

The value of disable_deblocking_filter_idc shall be in the range of 0 to 2, inclusive.

slice_alpha_c0_offset_div2 specifies the offset used in accessing the α and t_{C0} deblocking filter tables for filtering operations controlled by the macroblocks within the slice. From this value, the offset that shall be applied when addressing these tables shall be computed as

$$FilterOffsetA = slice\_alpha\_c0\_offset\_div2 \ll 1 \quad (7\text{-}13)$$

The value of slice_alpha_c0_offset_div2 shall be in the range of −6 to +6, inclusive. When slice_alpha_c0_offset_div2 is not present in the slice header, the value of slice_alpha_c0_offset_div2 shall be inferred to be equal to 0.

slice_beta_offset_div2 specifies the offset used in accessing the β deblocking filter table for filtering operations controlled by the macroblocks within the slice. From this value, the offset that is applied when addressing the β table of the deblocking filter shall be computed as

$$FilterOffsetB = slice\_beta\_offset\_div2 \ll 1 \quad (7\text{-}14)$$

The value of slice_beta_offset_div2 shall be in the range of −6 to +6, inclusive. When slice_beta_offset_div2 is not present in the slice header the value of slice_beta_offset_div2 shall be inferred to be equal to 0.

7.4.3.1 Reference picture list modification semantics

The syntax elements modification_of_pic_nums_idc, abs_diff_pic_num_minus1, and long_term_pic_num specify the change from the initial reference picture lists to the reference picture lists to be used for decoding the slice.

ref_pic_list_modification_flag_l0 equal to 1 specifies that the syntax element modification_of_pic_nums_idc is present for specifying reference picture list 0. ref_pic_list_modification_flag_l0 equal to 0 specifies that this syntax element is not present.

When ref_pic_list_modification_flag_l0 is equal to 1, the number of times that modification_of_pic_nums_idc is not equal to 3 following ref_pic_list_modification_flag_l0 shall not exceed num_ref_idx_l0_active_minus1 + 1.

When RefPicList0[num_ref_idx_l0_active_minus1] in the initial reference picture list produced as specified in subclause 8.2.4.2 is equal to "no reference picture", ref_pic_list_modification_flag_l0 shall be equal to 1 and modification_of_pic_nums_idc shall not be equal to 3 until RefPicList0[num_ref_idx_l0_active_minus1] in the modified list produced as specified in subclause 8.2.4.3 is not equal to "no reference picture".

modification_of_pic_nums_idc together with abs_diff_pic_num_minus1 or long_term_pic_num specifies which of the reference pictures are re-mapped. The values of modification_of_pic_nums_idc are specified in Table 7-4. The value of the first modification_of_pic_nums_idc that follows immediately after ref_pic_list_modification_flag_l0 shall not be equal to 3.

Table 7-4 – modification_of_pic_nums_idc operations for modification of reference picture lists

modification_of_pic_nums_idc	modification specified
0	abs_diff_pic_num_minus1 is present and corresponds to a difference to subtract from a picture number prediction value
1	abs_diff_pic_num_minus1 is present and corresponds to a difference to add to a picture number prediction value
2	long_term_pic_num is present and specifies the long-term picture number for a reference picture
3	End loop for modification of the initial reference picture list

abs_diff_pic_num_minus1 plus 1 specifies the absolute difference between the picture number of the picture being moved to the current index in the list and the picture number prediction value. abs_diff_pic_num_minus1 shall be in the range of 0 to MaxPicNum − 1. The allowed values of abs_diff_pic_num_minus1 are further restricted as specified in subclause 8.2.4.3.1.

long_term_pic_num specifies the long-term picture number of the picture being moved to the current index in the list. When decoding a coded frame, long_term_pic_num shall be equal to a LongTermPicNum assigned to one of the reference frames marked as "used for long-term reference".

7.4.3.2 (void)

7.4.3.3 Decoded reference picture marking semantics

The syntax elements no_output_of_prior_pics_flag, long_term_reference_flag, adaptive_ref_pic_marking_mode_flag, memory_management_control_operation, difference_of_pic_nums_minus1, long_term_frame_idx, long_term_pic_num, and max_long_term_frame_idx_plus1 specify marking of the reference pictures.

The marking of a reference picture can be "unused for reference", "used for short-term reference", or "used for long-term reference", but only one among these three. When a reference picture is referred to as being marked as "used for reference", this collectively refers to the picture being marked as "used for short-term reference" or "used for long-term reference" (but not both). A reference picture that is marked as "used for short-term reference" is referred to as a short-term reference picture. A reference picture that is marked as "used for long-term reference" is referred to as a long-term reference picture.

The content of the decoded reference picture marking syntax structure shall be the same in all slice headers of the primary coded picture.

no_output_of_prior_pics_flag specifies how the previously-decoded pictures in the decoded picture buffer are treated after decoding of an IDR picture. See Annex C. When the IDR picture is the first IDR picture in the bitstream, the value of no_output_of_prior_pics_flag has no effect on the decoding process. When the IDR picture is not the first IDR picture in the bitstream and the value of PicWidthInMbs, FrameHeightInMbs, or max_dec_frame_buffering derived from the active sequence parameter set is different from the value of PicWidthInMbs, FrameHeightInMbs, or max_dec_frame_buffering derived from the sequence parameter set active for the preceding picture, no_output_of_prior_pics_flag equal to 1 may (but should not) be inferred by the decoder, regardless of the actual value of no_output_of_prior_pics_flag.

long_term_reference_flag equal to 0 specifies that the MaxLongTermFrameIdx variable is set equal to "no long-term frame indices" and that the IDR picture is marked as "used for short-term reference". long_term_reference_flag equal to 1 specifies that the MaxLongTermFrameIdx variable is set equal to 0 and that the current IDR picture is marked "used for long-term reference" and is assigned LongTermFrameIdx equal to 0. When max_num_ref_frames is equal to 0, long_term_reference_flag shall be equal to 0.

adaptive_ref_pic_marking_mode_flag selects the reference picture marking mode of the currently decoded picture as specified in Table 7-5. adaptive_ref_pic_marking_mode_flag shall be equal to 1 when the number of frames, that are currently marked as "used for long-term reference" is equal to Max(max_num_ref_frames, 1).

Table 7-5 – Interpretation of adaptive_ref_pic_marking_mode_flag

adaptive_ref_pic_marking_mode_flag	Reference picture marking mode specified
0	Sliding window reference picture marking mode: A marking mode providing a first-in first-out mechanism for short-term reference pictures.
1	Adaptive reference picture marking mode: A reference picture marking mode providing syntax elements to specify marking of reference pictures as "unused for reference" and to assign long-term frame indices.

memory_management_control_operation specifies a control operation to be applied to affect the reference picture marking. The memory_management_control_operation syntax element is followed by data necessary for the operation specified by the value of memory_management_control_operation. The values and control operations associated with memory_management_control_operation are specified in Table 7-6. The memory_management_control_operation syntax elements are processed by the decoding process in the order in which they appear in the slice header, and the semantics constraints expressed for each memory_management_control_operation apply at the specific position in that order at which that individual memory_management_control_operation is processed.

For interpretation of memory_management_control_operation, the term "reference picture" refers to a reference frame.

memory_management_control_operation shall not be equal to 1 in a slice header unless the specified reference picture is marked as "used for short-term reference" when the memory_management_control_operation is processed by the decoding process.

memory_management_control_operation shall not be equal to 2 in a slice header unless the specified long-term picture number refers to a reference picture that is marked as "used for long-term reference" when the memory_management_control_operation is processed by the decoding process.

memory_management_control_operation shall not be equal to 3 in a slice header unless the specified reference picture is marked as "used for short-term reference" when the memory_management_control_operation is processed by the decoding process.

memory_management_control_operation shall not be equal to 3 or 6 if the value of the variable MaxLongTermFrameIdx is equal to "no long-term frame indices" when the memory_management_control_operation is processed by the decoding process.

Not more than one memory_management_control_operation equal to 4 shall be present in a slice header.

Not more than one memory_management_control_operation equal to 5 shall be present in a slice header.

Not more than one memory_management_control_operation equal to 6 shall be present in a slice header.

memory_management_control_operation shall not be equal to 5 in a slice header unless no memory_management_control_operation in the range of 1 to 3 is present in the same decoded reference picture marking syntax structure.

A memory_management_control_operation equal to 5 shall not follow a memory_management_control_operation equal to 6 in the same slice header.

When a memory_management_control_operation equal to 6 is present, any memory_management_control_operation equal to 2, 3, or 4 that follows the memory_management_control_operation equal to 6 within the same slice header shall not specify the current picture to be marked as "unused for reference".

NOTE – These constraints prohibit any combination of multiple memory_management_control_operation syntax elements that would specify the current picture to be marked as "unused for reference". However, some other combinations of memory_management_control_operation syntax elements are permitted that may affect the marking status of other reference pictures more than once in the same slice header. In particular, it is permitted for a memory_management_control_operation equal to 3 that specifies a long-term frame index to be assigned to a particular short-term reference picture to be followed in the same slice header by a memory_management_control_operation equal to 2, 3, 4 or 6 that specifies the same reference picture to subsequently be marked as "unused for reference".

Table 7-6 – Memory management control operation (memory_management_control_operation) values

memory_management_control_operation	Memory Management Control Operation
0	End memory_management_control_operation syntax element loop
1	Mark a short-term reference picture as "unused for reference"
2	Mark a long-term reference picture as "unused for reference"
3	Mark a short-term reference picture as "used for long-term reference" and assign a long-term frame index to it
4	Specify the maximum long-term frame index and mark all long-term reference pictures having long-term frame indices greater than the maximum value as "unused for reference"
5	Mark all reference pictures as "unused for reference" and set the MaxLongTermFrameIdx variable to "no long-term frame indices"
6	Mark the current picture as "used for long-term reference" and assign a long-term frame index to it

difference_of_pic_nums_minus1 is used (with memory_management_control_operation equal to 3 or 1) to assign a long-term frame index to a short-term reference picture or to mark a short-term reference picture as "unused for reference". When the associated memory_management_control_operation is processed by the decoding process, the resulting picture number derived from difference_of_pic_nums_minus1 shall be a picture number assigned to one of the reference pictures marked as "used for reference" and not previously assigned to a long-term frame index.

The resulting picture number is constrained such that the resulting picture number shall be one of the set of picture numbers assigned to reference frames.

long_term_pic_num is used (with memory_management_control_operation equal to 2) to mark a long-term reference picture as "unused for reference". When the associated memory_management_control_operation is processed by the decoding process, long_term_pic_num shall be equal to a long-term picture number assigned to one of the reference pictures that is currently marked as "used for long-term reference".

The resulting long-term picture number is constrained such that the resulting long-term picture number shall be one of the set of long-term picture numbers assigned to reference frames.

long_term_frame_idx is used (with memory_management_control_operation equal to 3 or 6) to assign a long-term frame index to a picture. When the associated memory_management_control_operation is processed by the decoding process, the value of long_term_frame_idx shall be in the range of 0 to MaxLongTermFrameIdx, inclusive.

max_long_term_frame_idx_plus1 minus 1 specifies the maximum value of long-term frame index allowed for long-term reference pictures (until receipt of another value of max_long_term_frame_idx_plus1). The value of max_long_term_frame_idx_plus1 shall be in the range of 0 to max_num_ref_frames, inclusive.

7.4.4 Slice data semantics

mb_skip_run specifies the number of consecutive skipped macroblocks for which, when decoding a P slice, mb_type shall be inferred to be P_Skip and the macroblock type is collectively referred to as a P macroblock type. The value of mb_skip_run shall be in the range of 0 to PicSizeInMbs − CurrMbAddr, inclusive.

end_of_slice_flag equal to 0 specifies that another macroblock is following in the slice. end_of_slice_flag equal to 1 specifies the end of the slice and that no further macroblock follows.

The function NextMbAddress(n) used in the slice data syntax table is specified as:

NextMbAddress(n) = n + 1

7.4.5 Macroblock layer semantics

mb_type specifies the macroblock type. The semantics of mb_type depend on the slice type.

Tables and semantics are specified for the various macroblock types for I, P slices. Each table presents the value of mb_type, the name of mb_type, the number of macroblock partitions used (given by the NumMbPart(mb_type) function), the prediction mode of the macroblock (when it is not partitioned) or the first partition (given by the MbPartPredMode(mb_type, 0) function) and the prediction mode of the second partition (given by the MbPartPredMode(mb_type, 1) function). When a value is not applicable it is designated by "na". In the text, the value of mb_type may be referred to as the macroblock type, the value of MbPartPredMode( ) may be referred to in the text by "macroblock (partition) prediction mode", and a value X of MbPartPredMode( ) may be referred to in the text by "X macroblock (partition) prediction mode" or as "X prediction macroblocks".

Table 7-7 shows the allowed collective macroblock types for each slice_type.

Table 7-7 – Allowed collective macroblock types for slice_type

slice_type	allowed collective macroblock types
I (slice)	I (see Table 7-8) (macroblock types)
P (slice)	P (see Table 7-9) and I (see Table 7-8) (macroblock types)

Macroblock types that may be collectively referred to as I macroblock types are specified in Table 7-8.

The macroblock types for I slices are all I macroblock types.

Table 7-8 – Macroblock types for I slices

mb_type	Name of mb_type	MbPartPredMode (mb_type, 0)	Intra16x16PredMode	CodedBlockPatternChroma	CodedBlockPatternLuma
0	I_4x4	Intra_4x4	na	Equation 7-15	Equation 7-15
1	I_16x16_0_0_0	Intra_16x16	0	0	0
2	I_16x16_1_0_0	Intra_16x16	1	0	0
3	I_16x16_2_0_0	Intra_16x16	2	0	0
4	I_16x16_3_0_0	Intra_16x16	3	0	0
5	I_16x16_0_1_0	Intra_16x16	0	1	0
6	I_16x16_1_1_0	Intra_16x16	1	1	0
7	I_16x16_2_1_0	Intra_16x16	2	1	0
8	I_16x16_3_1_0	Intra_16x16	3	1	0
9	I_16x16_0_2_0	Intra_16x16	0	2	0
10	I_16x16_1_2_0	Intra_16x16	1	2	0
11	I_16x16_2_2_0	Intra_16x16	2	2	0
12	I_16x16_3_2_0	Intra_16x16	3	2	0
13	I_16x16_0_0_1	Intra_16x16	0	0	15
14	I_16x16_1_0_1	Intra_16x16	1	0	15
15	I_16x16_2_0_1	Intra_16x16	2	0	15
16	I_16x16_3_0_1	Intra_16x16	3	0	15
17	I_16x16_0_1_1	Intra_16x16	0	1	15
18	I_16x16_1_1_1	Intra_16x16	1	1	15
19	I_16x16_2_1_1	Intra_16x16	2	1	15
20	I_16x16_3_1_1	Intra_16x16	3	1	15
21	I_16x16_0_2_1	Intra_16x16	0	2	15
22	I_16x16_1_2_1	Intra_16x16	1	2	15

23	I_16x16_2_2_1	Intra_16x16	2	2	15
24	I_16x16_3_2_1	Intra_16x16	3	2	15
25	I_PCM	na	na	na	na

The following semantics are assigned to the macroblock types in Table 7-8:

– I_NxN: A mnemonic name for mb_type equal to 0 with MbPartPredMode(mb_type, 0) equal to Intra_4x4.

– I_16x16_0_0_0, I_16x16_1_0_0, I_16x16_2_0_0, I_16x16_3_0_0, I_16x16_0_1_0, I_16x16_1_1_0, I_16x16_2_1_0, I_16x16_3_1_0, I_16x16_0_2_0, I_16x16_1_2_0, I_16x16_2_2_0, I_16x16_3_2_0, I_16x16_0_0_1, I_16x16_1_0_1, I_16x16_2_0_1, I_16x16_3_0_1, I_16x16_0_1_1, I_16x16_1_1_1, I_16x16_2_1_1, I_16x16_3_1_1, I_16x16_0_2_1, I_16x16_1_2_1, I_16x16_2_2_1, I_16x16_3_2_1: the macroblock is coded as an Intra_16x16 prediction macroblock.

To each Intra_16x16 prediction macroblock, an Intra16x16PredMode is assigned, which specifies the Intra_16x16 prediction mode, and values of CodedBlockPatternLuma and CodedBlockPatternChroma are assigned as specified in Table 7-8.

Intra_4x4 specifies the macroblock prediction mode and specifies that the Intra_4x4 prediction process is invoked as specified in subclause 8.3.1. Intra_4x4 is an Intra macroblock prediction mode.

Intra_16x16 specifies the macroblock prediction mode and specifies that the Intra_16x16 prediction process is invoked as specified in subclause 8.3.3. Intra_16x16 is an Intra macroblock prediction mode.

For a macroblock coded with mb_type equal to I_PCM, the Intra macroblock prediction mode shall be inferred.

Macroblock types that may be collectively referred to as P macroblock types are specified in Table 7-9.

The macroblock types for P and SP slices are specified in Tables 7-9 and 7-8. mb_type values 0 to 4 are specified in Table 7-9 and mb_type values 5 to 30 are specified in Table 7-8, indexed by subtracting 5 from the value of mb_type.

Table 7-9 – Macroblock type values 0 to 4 for P and SP slices

mb_type	Name of mb_type	NumMbPart (mb_type)	MbPartPredMode (mb_type, 0)	MbPartPredMode (mb_type, 1)	MbPartWidth (mb_type)	MbPartHeight (mb_type)
0	P_L0_16x16	1	Pred_L0	na	16	16
1	P_L0_L0_16x8	2	Pred_L0	Pred_L0	16	8
2	P_L0_L0_8x16	2	Pred_L0	Pred_L0	8	16
3	P_8x8	4	na	na	8	8
4	P_8x8ref0	4	na	na	8	8
inferred	P_Skip	1	Pred_L0	na	16	16

The following semantics are assigned to the macroblock types in Table 7-9:

– P_L0_16x16: the samples of the macroblock are predicted with one luma macroblock partition of size 16x16 luma samples and associated chroma samples.

– P_L0_L0_MxN, with MxN being replaced by 16x8 or 8x16: the samples of the macroblock are predicted using two luma partitions of size MxN equal to 16x8, or two luma partitions of size MxN equal to 8x16, and associated chroma samples, respectively.

– P_8x8: for each sub-macroblock an additional syntax element (sub_mb_type[mbPartIdx] with mbPartIdx being the macroblock partition index for the corresponding sub-macroblock) is present in the bitstream that specifies the type of the corresponding sub-macroblock (see subclause 7.4.5.2).

– P_8x8ref0: has the same semantics as P_8x8 but no syntax element for the reference index (ref_idx_l0[mbPartIdx] with mbPartIdx = 0..3) is present in the bitstream and ref_idx_l0[mbPartIdx] shall be inferred to be equal to 0 for all sub-macroblocks of the macroblock (with indices mbPartIdx = 0..3).

– P_Skip: no further data is present for the macroblock in the bitstream.

The following semantics are assigned to the macroblock prediction modes (for macroblocks that are not partitioned) and macroblock partition prediction modes (for macroblocks that are partitioned) specified by MbPartPredMode( ) in Table 7-9:

– Pred_L0: specifies that the Inter prediction process is invoked using list 0 prediction. Pred_L0 is an Inter macroblock prediction mode (for macroblocks that are not partitioned) and an Inter macroblock partition prediction mode (for macroblocks that are partitioned).

When mb_type is equal to any of the values specified in Table 7-9, the macroblock is coded in an Inter macroblock prediction mode.

pcm_alignment_zero_bit is a bit equal to 0.

pcm_sample_luma[i] is a sample value. The pcm_sample_luma[i] values represent luma sample values in the raster scan within the macroblock. The number of bits used to represent each of these samples is BitDepth_Y. (BitDepth_Y is equal to 8 in this standard.) pcm_sample_luma[i] shall not be equal to zero.

pcm_sample_chroma[i] is a sample value. The first MbWidthC * MbHeightC pcm_sample_chroma[i] values represent Cb sample values in the raster scan within the macroblock and the remaining MbWidthC * MbHeightC pcm_sample_chroma[i] values represent Cr sample values in the raster scan within the macroblock. The number of bits used to represent each of these samples is BitDepth_C. (BitDepth_C is equal to 8 in this standard.) pcm_sample_chroma[i] shall not be equal to zero.

coded_block_pattern specifies which of the four 8x8 luma blocks and associated chroma blocks of a macroblock may contain non-zero transform coefficient levels. When coded_block_pattern is present in the bitstream, the variables CodedBlockPatternLuma and CodedBlockPatternChroma are derived as

CodedBlockPatternLuma = coded_block_pattern % 16
CodedBlockPatternChroma = coded_block_pattern / 16 (7-15)

When the macroblock type is not equal to P_Skip or I_PCM, the following applies:

– If the macroblock prediction mode is equal Intra_16x16, the following applies:

  – the value of CodedBlockPatternLuma specifies the following.

    – If CodedBlockPatternLuma is equal to 0, all AC transform coefficient levels of the luma component of the macroblock are equal to 0 for all 16 of the 4x4 blocks in the 16x16 luma block.

    – Otherwise (CodedBlockPatternLuma is not equal to 0), CodedBlockPatternLuma is equal to 15, at least one of the AC transform coefficient levels of the luma component of the macroblock shall be non-zero, and the AC transform coefficient levels are scanned for all 16 of the 4x4 blocks in the 16x16 block.

– Otherwise (the macroblock prediction mode is not equal to Intra_16x16),coded_block_pattern is present in the bitstream, and the following applies:

– each of the four LSBs of CodedBlockPatternLuma specifies, for one of the four 8x8 luma blocks of the macroblock, the following.
  – If the corresponding bit of CodedBlockPatternLuma is equal to 0, all transform coefficient levels of the luma transform blocks in the 8x8 luma block are equal to zero.
  – Otherwise (the corresponding bit of CodedBlockPatternLuma is equal to 1), one or more transform coefficient levels of one or more of the luma transform blocks in the 8x8 luma block shall be non-zero valued and the transform coefficient levels of the corresponding transform blocks are scanned.

When the macroblock type is not equal to P_Skip or I_PCM, CodedBlockPatternChroma is interpreted as specified in Table 7-10.

Table 7-10 – Specification of CodedBlockPatternChroma values

CodedBlockPatternChroma	Description
0	All chroma transform coefficient levels are equal to 0.
1	One or more chroma DC transform coefficient levels shall be non-zero valued. All chroma AC transform coefficient levels are equal to 0.
2	Zero or more chroma DC transform coefficient levels are non-zero valued. One or more chroma AC transform coefficient levels shall be non-zero valued.

mb_qp_delta can change the value of QP_Y in the macroblock layer. The decoded value of mb_qp_delta shall be in the range of $-(26 + 0 / 2)$ to $+(25 + 0 / 2)$, inclusive. mb_qp_delta shall be inferred to be equal to 0 when it is not present for any macroblock (including P_Skip macroblock types).

The value of QP_Y is derived as

$$QP_Y = ((QP_{Y,PREV} + mb\_qp\_delta + 52 + 2 * 0) \% (52 + 0)) - 0 \quad (7\text{-}16)$$

where $QP_{Y,PREV}$ is the luma quantisation parameter, QP_Y, of the previous macroblock in decoding order in the current slice. For the first macroblock in the slice $QP_{Y,PREV}$ is initially set equal to $SliceQP_Y$ derived in Equation 7-12 at the start of each slice.

The value of QP'_Y is derived as

$$QP'_Y = QP_Y + 0 \quad (7\text{-}17)$$

7.4.5.1 Macroblock prediction semantics

All samples of the macroblock are predicted. The prediction modes are derived using the following syntax elements.

prev_intra4x4_pred_mode_flag[luma4x4BlkIdx **]**and **rem_intra4x4_pred_mode[** luma4x4BlkIdx **]** specify the Intra_4x4 prediction of the 4x4 luma block with index luma4x4BlkIdx = 0..15.

intra_chroma_pred_mode specifies the type of spatial prediction used for chroma in macroblocks using Intra_4x4 or Intra_16x16 prediction, as shown in Table 7-11. The value of intra_chroma_pred_mode shall be in the range of 0 to 3, inclusive.

Table 7-11 – Relationship between intra_chroma_pred_mode and spatial prediction modes

intra_chroma_pred_mode	Intra Chroma Prediction Mode
0	DC
1	Horizontal
2	Vertical
3	Plane

ref_idx_l0[mbPartIdx **]** when present, specifies the index in reference picture list 0 of the reference picture to be used for prediction.

The range of ref_idx_l0[mbPartIdx], the index in list 0 of the reference picture, is specified such that value of ref_idx_l0[mbPartIdx] shall be in the range of 0 to num_ref_idx_l0_active_minus1, inclusive.

When only one reference picture is used for inter prediction, the values of ref_idx_l0[mbPartIdx] shall be inferred to be equal to 0.

mvd_l0[mbPartIdx **][** 0 **][** compIdx **]** specifies the difference between a list 0 motion vector component to be used and its prediction. The index mbPartIdx specifies to which macroblock partition mvd_l0 is assigned. The partitioning of the macroblock is specified by mb_type. The horizontal motion vector component difference is decoded first in decoding order and is assigned compIdx = 0. The vertical motion vector component is decoded second in decoding order and is assigned compIdx = 1. The range of the components of mvd_l0[mbPartIdx][0][compIdx] is specified by constraints on the motion vector variable values derived from it as specified in Annex A.

7.4.5.2 Sub-macroblock prediction semantics

sub_mb_type[mbPartIdx **]** specifies the sub-macroblock types.

Tables and semantics are specified for the various sub-macroblock types for P macroblock types. Each table presents the value of sub_mb_type[mbPartIdx], the name of sub_mb_type[mbPartIdx], the number of sub-macroblock partitions used (given by the NumSubMbPart(sub_mb_type[mbPartIdx]) function), and the prediction mode of the sub-macroblock (given by the SubMbPredMode(sub_mb_type[mbPartIdx]) function). In the text, the value of sub_mb_type[mbPartIdx] may be referred to by "sub-macroblock type". In the text, the value of SubMbPredMode( ) may be referred to by "sub-macroblock prediction mode" or "macroblock partition prediction mode".

The interpretation of sub_mb_type[mbPartIdx] for P macroblock types is specified in Table 7-12, where the row for "inferred" specifies values inferred when sub_mb_type[mbPartIdx] is not present.

Table 7-12 – Sub-macroblock types in P macroblocks

sub_mb_type[mbPartIdx]	Name of sub_mb_type[mbPartIdx]	NumSubMbPart (sub_mb_type[mbPartIdx])	SubMbPredMode (sub_mb_type[mbPartIdx])	SubMbPartWidth (sub_mb_type[mbPartIdx])	SubMbPartHeight (sub_mb_type[mbPartIdx])
inferred	na	na	na	na	na
0	P_L0_8x8	1	Pred_L0	8	8
1	P_L0_8x4	2	Pred_L0	8	4
2	P_L0_4x8	2	Pred_L0	4	8
3	P_L0_4x4	4	Pred_L0	4	4

The following semantics are assigned to the sub-macroblock types in Table 7-12:

P_L0_MxN, with MxN being replaced by 8x8, 8x4, 4x8, or 4x4: the samples of the sub-macroblock are predicted using one luma partition of size MxN equal to 8x8, two luma partitions of size MxN equal to 8x4, or two luma partitions of size MxN equal to 4x8, or four luma partitions of size MxN equal to 4x4, and associated chroma samples, respectively.

The following semantics are assigned to the sub-macroblock prediction modes (or macroblock partition prediction modes)specified by SubMbPredMode( ) in Table 7-12:

Pred_L0: see semantics for Table 7-9.

ref_idx_l0[mbPartIdx **]**has the same semantics as ref_idx_l0 in subclause 7.4.5.1.

mvd_l0[mbPartIdx **][** subMbPartIdx **][** compIdx **]**has the same semantics as mvd_l0 in subclause 7.4.5.1, except that it is applied to the sub-macroblock partition index with subMbPartIdx. The indices mbPartIdx and subMbPartIdx specify to which macroblock partition and sub-macroblock partition mvd_l0 is assigned.

7.4.5.3 Residual data semantics

The syntax structure residual_block( ), which is used for parsing the transform coefficient levels is set equal to residual_block_cavlc, which is used for parsing the syntax elements for transform coefficient levels.

The syntax structure residual_luma(i16x16DClevel, i16x16AClevel, level4x4, level8x8, startIdx, endIdx) is used with the first four variables in brackets being its output and being assigned as follows.

Intra16x16DCLevel is set equal to i16x16DClevel, Intra16x16ACLevel is set equal to i16x16AClevel, LumaLevel4x4 is set equal to level4x4, and LumaLevel8x8 is set equal to level8x8.

The following applies:

– For each chroma component, indexed by iCbCr = 0..1, the DC transform coefficient levels of the 4 * NumC8x8 4x4 chroma blocks are parsed into the iCbCr-th list ChromaDCLevel[iCbCr].

– For each of the 4x4 chroma blocks, indexed by i4x4 = 0..3 and i8x8 = 0..NumC8x8 − 1, of each chroma component, indexed by iCbCr = 0..1, the 15 AC transform coefficient levels are parsed into the (i8x8*4 + i4x4)-th list of the iCbCr-th chroma component ChromaACLevel[iCbCr][i8x8*4 + i4x4].

7.4.5.3.1 Residual luma data semantics

Output of this syntax structure are the variables i16x16DClevel, i16x16AClevel, level4x4, and level8x8.

Depending on mb_type, the syntax structure residual_block(coeffLevel, startIdx, endIdx, maxNumCoeff) is used with the arguments coeffLevel, which is a list containing the maxNumCoeff transform coefficient levels that are parsed in residual_block( ), startIdx, endIdx, and maxNumCoeff as follows.

Depending on MbPartPredMode(mb_type, 0), the following applies:

– If MbPartPredMode(mb_type, 0) is equal to Intra_16x16, the transform coefficient levels are parsed into the list i16x16DClevel and into the 16 lists i16x16AClevel[i]. i16x16DClevel contains the 16 transform coefficient levels of the DC transform coefficient levels for each 4x4 luma block. For each of the 16 4x4 luma blocks indexed by i = 0..15, the 15 AC transform coefficients levels of the i-th block are parsed into the i-th list i16x16AClevel[i].

– Otherwise (MbPartPredMode(mb_type, 0) is not equal to Intra_16x16), the following applies:

  for each of the 16 4x4 luma blocks indexed by i = 0..15, the 16 transform coefficient levels of the i-th block are parsed into the i-th list level4x4[i].

7.4.5.3.2 Residual block CAVLC semantics

The function TotalCoeff(coeff_token) that is used in subclause 7.3.5.3.2 returns the number of non-zero transform coefficient levels derived from coeff_token.

The function TrailingOnes(coeff_token) that is used in subclause 7.3.5.3.2 returns the trailing ones derived from coeff_token.

coeff_token specifies the total number of non-zero transform coefficient levels and the number of trailing one transform coefficient levels in a transform coefficient level scan. A trailing one transform coefficient level is one of up to three consecutive non-zero transform coefficient levels having an absolute value equal to 1 at the end of a scan of non-zero transform coefficient levels. The range of coeff_token is specified in subclause 9.2.1.

trailing_ones_sign_flag specifies the sign of a trailing one transform coefficient level as follows:

– If trailing_ones_sign_flag is equal to 0, the corresponding transform coefficient level is decoded as +1.

– Otherwise (trailing_ones_sign_flag equal to 1), the corresponding transform coefficient level is decoded as −1.

level_prefix and **level_suffix** specify the value of a non-zero transform coefficient level. The range of level_prefix and level_suffix is specified in subclause 9.2.2.

total_zeros specifies the total number of zero-valued transform coefficient levels that are located before the position of the last non-zero transform coefficient level in a scan of transform coefficient levels. The range of total_zeros is specified in subclause 9.2.3.

run_before specifies the number of consecutive transform coefficient levels in the scan with zero value before a non-zero valued transform coefficient level. The range of run_before is specified in subclause 9.2.3.

coeffLevel contains maxNumCoeff transform coefficient levels for the current list of transform coefficient levels.

8 Decoding process

Outputs of this process are decoded samples of the current picture (sometimes referred to by the variable CurrPic).

the number of sample arrays of the current picture is as follows:

– the current picture consists of 3 sample arrays S_L, S_{Cb}, S_{Cr}.

This clause describes the decoding process, given syntax elements and upper-case variables from clause 7.

The decoding process is specified such that all decoders shall produce numerically identical results. Any decoding process that produces identical results to the process described here conforms to the decoding process requirements of this International Standard.

Each picture referred to in this clause is a complete primary coded picture. Each slice referred to in this clause is a slice of a primary coded picture.

The decoding process is structured as follows:

the decoding process is invoked a single time with the current picture being the output.

An overview of the decoding process is given as follows:

- The decoding of NAL units is specified in subclause 8.1.
- The processes in subclause 8.2 specify decoding processes using syntax elements in the slice layer and above:
  - Variables and functions relating to picture order count are derived in subclause 8.2.1. (only needed to be invoked for one slice of a picture)
  - When the frame_num of the current picture is not equal to PrevRefFrameNum and is not equal to (PrevRefFrameNum + 1) % MaxFrameNum, the decoding process for gaps in frame_num is performed according to subclause 8.2.5.2 prior to the decoding of any slices of the current picture.
  - At the beginning of the decoding process for each P slice, the decoding process for reference picture lists construction specified in subclause 8.2.4 is invoked for derivation of reference picture list 0 (RefPicList0).
  - When the current picture is a reference picture and after all slices of the current picture have been decoded, the decoded reference picture marking process in subclause 8.2.5 specifies how the current picture is used in the decoding process of inter prediction in later decoded pictures.
- The processes in subclauses 8.3, 8.4, 8.5, 8.6 and 8.7 specify decoding processes using syntax elements in the macroblock layer and above.
  - The intra prediction process for I macroblocks, except for I_PCM macroblocks as specified in subclause 8.3, has intra prediction samples as its output. For I_PCM macroblocks subclause 8.3 directly specifies a picture construction process. The output are constructed samples prior to the deblocking filter process.
  - The inter prediction process for P macroblocks is specified in subclause 8.4 with inter prediction samples being the output.
  - The transform coefficient decoding process and picture construction process prior to deblocking filter process are specified in subclause 8.5. That process derives samples for I macroblocks and for P macroblocks in P slices. The output are constructed samples prior to the deblocking filter process.
  - The constructed samples prior to the deblocking filter process that are next to the edges of blocks and macroblocks are processed by a deblocking filter as specified in subclause 8.6 with the output being the decoded samples.

8.1 NAL unit decoding process

Inputs to this process are NAL units.

Outputs of this process are the RBSP syntax structures encapsulated within the NAL units.

The decoding process for each NAL unit extracts the RBSP syntax structure from the NAL unit and then operates the decoding processes specified for the RBSP syntax structure in the NAL unit as follows.

Subclause 8.2 describes the decoding process for NAL units with nal_unit_type equal to 1 through 5.

Subclause 8.3 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1, and 5.

Subclause 8.4 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1.

Subclause 8.5 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1 and 5.

NAL units with nal_unit_type equal to 7 and 8 contain sequence parameter sets and picture parameter sets, respectively. Picture parameter sets are used in the decoding processes of other NAL units as determined by reference to a picture parameter set within the slice headers of each picture. Sequence parameter sets are used in the decoding processes of other NAL units as determined by reference to a sequence parameter set within the picture parameter sets of each sequence.

No normative decoding process is specified for NAL units with nal_unit_type equal to 6, 9, 10, 11, and 12.

8.2 Slice decoding process

8.2.1 Decoding process for picture order count

Outputs of this process are the functions PicOrderCnt(picX) and DiffPicOrderCnt(picA, picB) and the variable PicOrderCnt.

Each coded frame is associated with a picture order count, called PicOrderCnt.

PicOrderCnt indicates the picture order of the corresponding frame relative to the previous IDR picture or the previous reference picture including a memory_management_control_operation equal to 5 in decoding order.

PicOrderCnt is derived by invoking one of the decoding processes for picture order count type 0, 1, and 2 in subclauses 8.2.1.1, 8.2.1.2, and 8.2.1.3, respectively. When the current picture includes a memory_management_control_operation equal to 5, after the decoding of the current picture, tempPicOrderCnt is set equal to PicOrderCnt(CurrPic), and PicOrderCnt is set equal to 0.

NOTE –When the decoding process for a picture currPic that includes a memory_management_control_operation equal to 5 refers to the values of PicOrderCnt for the picture currPic (including references to the function PicOrderCnt( ) with the picture currPic as the argument and references to the function DiffPicOrderCnt( ) with one of the arguments being currPic), the values of PicOrderCnt that is derived as specified in subclauses 8.2.1.1, 8.2.1.2, and 8.2.1.3 for the picture currPic are used. When the decoding process for a picture refers to the values PicOrderCnt of the previous picture prevMmco5Pic in decoding order that includes a memory_management_control_operation equal to 5 (including references via the functions PicOrderCnt( ) or DiffPicOrderCnt( )), the values of PicOrderCnt that is used for the picture prevMmco5Pic are the values after the modification specified in the paragraph above (resulting in PicOrderCnt equal to 0).

The bitstream shall not contain data that result in PicOrderCnt not equal to 0 for a coded IDR frame.

When the current picture is not an IDR picture, the following applies:

1) Consider the list variable listD containing as elements the PicOrderCnt values associated with the list of pictures including all of the following:

   a. The first picture in the list is the previous picture of any of the following types:

      – an IDR picture,

      – a picture containing a memory_management_control_operation equal to 5.

   b. The following additional pictures:

      – If pic_order_cnt_type is equal to 0, all other pictures that follow in decoding order after the first picture in the list are not "non-existing" frames inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2 and either precede the current picture in decoding order or are the current picture. When pic_order_cnt_type is equal to 0 and the current picture is not a "non-existing" frame inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2, the current picture is included in listD prior to the invoking of the decoded reference picture marking process.

– Otherwise (pic_order_cnt_type is not equal to 0), all other pictures that follow in decoding order after the first picture in the list and either precede the current picture in decoding order or are the current picture. When pic_order_cnt_type is not equal to 0, the current picture is included in listD prior to the invoking of the decoded reference picture marking process.

2) Consider the list variable listO which contains the elements of listD sorted in ascending order. listO shall not contain a PicOrderCnt that has a value equal to another PicOrderCnt.

The bitstream shall not contain data that result in values of PicOrderCnt, PicOrderCntMsb, or FrameNumOffset used in the decoding process as specified in subclauses 8.2.1.1 to 8.2.1.3 that exceed the range of values from -2^{31} to $2^{31} - 1$, inclusive.

The function PicOrderCnt(picX) is specified as follows:

PicOrderCnt(picX) = PicOrderCnt of the frame picX (8-1)

Then DiffPicOrderCnt(picA, picB) is specified as follows:

DiffPicOrderCnt(picA, picB) = PicOrderCnt(picA) − PicOrderCnt(picB) (8-2)

The bitstream shall not contain data that result in values of DiffPicOrderCnt(picA, picB) used in the decoding process that exceed the range of -2^{15} to $2^{15} - 1$, inclusive.

NOTE – Let X be the current picture and Y and Z be two other pictures in the same sequence, Y and Z are considered to be in the same output order direction from X when both DiffPicOrderCnt(X, Y) and DiffPicOrderCnt(X, Z) are positive or both are negative.

When the current picture includes a memory_management_control_operation equal to 5, PicOrderCnt(CurrPic) shall be greater than PicOrderCnt(any other picture in listD).

8.2.1.1 Decoding process for picture order count type 0

This process is invoked when pic_order_cnt_type is equal to 0.

Input to this process is PicOrderCntMsb of the previous reference picture in decoding order as specified in this subclause.

Outputs of this process is PicOrderCnt.

The variables prevPicOrderCntMsb and prevPicOrderCntLsb are derived as follows:

– If the current picture is an IDR picture, prevPicOrderCntMsb is set equal to 0 and prevPicOrderCntLsb is set equal to 0.

– Otherwise (the current picture is not an IDR picture), the following applies:

  – If the previous reference picture in decoding order included a memory_management_control_operation equal to 5, the following applies:

    – prevPicOrderCntMsb is set equal to 0 and prevPicOrderCntLsb is set equal to the value of PicOrderCnt for the previous reference picture in decoding order.

  – Otherwise (the previous reference picture in decoding order did not include a memory_management_control_operation equal to 5), prevPicOrderCntMsb is set equal to PicOrderCntMsb of the previous reference picture in decoding order and prevPicOrderCntLsb is set equal to the value of pic_order_cnt_lsb of the previous reference picture in decoding order.

PicOrderCntMsb of the current picture is derived as specified by the following pseudo-code:

```
if( ( pic_order_cnt_lsb < prevPicOrderCntLsb ) &&
   ( ( prevPicOrderCntLsb − pic_order_cnt_lsb ) >= ( MaxPicOrderCntLsb / 2 ) ) )
    PicOrderCntMsb = prevPicOrderCntMsb + MaxPicOrderCntLsb                    (8-3)
else if( ( pic_order_cnt_lsb > prevPicOrderCntLsb ) &&
      ( ( pic_order_cnt_lsb − prevPicOrderCntLsb ) > ( MaxPicOrderCntLsb / 2 ) ) )
    PicOrderCntMsb = prevPicOrderCntMsb − MaxPicOrderCntLsb
else
    PicOrderCntMsb = prevPicOrderCntMsb
```

PicOrderCnt is derived as

```
PicOrderCnt = PicOrderCntMsb + pic_order_cnt_lsb                              (8-4)
```

8.2.1.2 Decoding process for picture order count type 1

This process is invoked when pic_order_cnt_type is equal to 1.

Input to this process is FrameNumOffset of the previous picture in decoding order as specified in this subclause.

Outputs of this process is PicOrderCnt

The value of PicOrderCnt is derived as specified in this subclause. Let prevFrameNum be equal to the frame_num of the previous picture in decoding order.

When the current picture is not an IDR picture, the variable prevFrameNumOffset is derived as follows:

– If the previous picture in decoding order included a memory_management_control_operation equal to 5, prevFrameNumOffset is set equal to 0.

– Otherwise (the previous picture in decoding order did not include a memory_management_control_operation equal to 5), prevFrameNumOffset is set equal to the value of FrameNumOffset of the previous picture in decoding order.

NOTE – When gaps_in_frame_num_value_allowed_flag is equal to 1, the previous picture in decoding order may be a "non-existing" frame inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2.

The variable FrameNumOffset is derived as specified by the following pseudo-code:

```
if( IdrPicFlag == 1 )
    FrameNumOffset = 0
else if( prevFrameNum > frame_num )                                           (8-5)
    FrameNumOffset = prevFrameNumOffset + MaxFrameNum
else
    FrameNumOffset = prevFrameNumOffset
```

The variable absFrameNum is derived as specified by the following pseudo-code:

```
if( num_ref_frames_in_pic_order_cnt_cycle != 0 )
    absFrameNum = FrameNumOffset + frame_num
else                                                                          (8-6)
    absFrameNum = 0
if( nal_ref_idc == 0 && absFrameNum > 0 )
    absFrameNum = absFrameNum − 1
```

When absFrameNum > 0, picOrderCntCycleCnt and frameNumInPicOrderCntCycle are derived as

```
picOrderCntCycleCnt = ( absFrameNum − 1 ) / num_ref_frames_in_pic_order_cnt_cycle
frameNumInPicOrderCntCycle = ( absFrameNum − 1 ) % num_ref_frames_in_pic_order_cnt_cycle    (8-7)
```

The variable expectedPicOrderCnt is derived as specified by the following pseudo-code:

```
if( absFrameNum > 0 ){
    expectedPicOrderCnt = picOrderCntCycleCnt * ExpectedDeltaPerPicOrderCntCycle
    for( i = 0; i <= frameNumInPicOrderCntCycle; i++ )
        expectedPicOrderCnt = expectedPicOrderCnt + offset_for_ref_frame[ i ]
} else
    expectedPicOrderCnt = 0
if( nal_ref_idc == 0 )                                                        (8-8)
    expectedPicOrderCnt = expectedPicOrderCnt + offset_for_non_ref_pic
```

The variables PicOrderCnt are derived as specified by the following pseudo-code:

PicOrderCnt = expectedPicOrderCnt + delta_pic_order_cnt[0] (8-9)

8.2.1.3 Decoding process for picture order count type 2

This process is invoked when pic_order_cnt_type is equal to 2.

Outputs of this process is PicOrderCnt .

Let prevFrameNum be equal to the frame_num of the previous picture in decoding order.

When the current picture is not an IDR picture, the variable prevFrameNumOffset is derived as follows:

– If the previous picture in decoding order included a memory_management_control_operation equal to 5, prevFrameNumOffset is set equal to 0.

– Otherwise (the previous picture in decoding order did not include a memory_management_control_operation equal to 5), prevFrameNumOffset is set equal to the value of FrameNumOffset of the previous picture in decoding order.

NOTE – When gaps_in_frame_num_value_allowed_flag is equal to 1, the previous picture in decoding order may be a "non-existing" frame inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2.

The variable FrameNumOffset is derived as specified by the following pseudo-code:

```
if( IdrPicFlag  = =  1)
    FrameNumOffset = 0
else if( prevFrameNum  >  frame_num )                                    (8-10)
    FrameNumOffset = prevFrameNumOffset + MaxFrameNum
else
    FrameNumOffset = prevFrameNumOffset
```

The variable tempPicOrderCnt is derived as specified by the following pseudo-code:

```
if( IdrPicFlag  = =  1 )
    tempPicOrderCnt = 0
else if( nal_ref_idc  = =  0 )                                           (8-11)
    tempPicOrderCnt = 2 * ( FrameNumOffset + frame_num ) − 1
else
    tempPicOrderCnt = 2 * ( FrameNumOffset + frame_num )
```

The variables PicOrderCnt is derived as specified by the following pseudo-code:

PicOrderCnt = tempPicOrderCnt (8-12)

NOTE – Picture order count type 2 cannot be used in a coded video sequence that contains consecutive non-reference pictures that would result in more than one of these pictures having the same value of PicOrderCnt

NOTE –Picture order count type 2 results in an output order that is the same as the decoding order.

8.2.2 (void)

8.2.3 (void)

8.2.4 Decoding process for reference picture lists construction

This process is invoked at the beginning of the decoding process for each P slice.

Decoded reference pictures are marked as "used for short-term reference" or "used for long-term reference" as specified by the bitstream and specified in subclause 8.2.5. Short-term reference pictures are identified by the value of frame_num. Long-term reference pictures are assigned a long-term frame index as specified by the bitstream and specified in subclause 8.2.5.

Subclause 8.2.4.1 is invoked to specify the assignment of variables FrameNum, FrameNumWrap, and PicNum to each of the short-term reference pictures, and the assignment of variable LongTermPicNum to each of the long-term reference pictures.

Reference pictures are addressed through reference indices as specified in subclause 8.4.2.1. A reference index is an index into a reference picture list. When decoding a P slice, there is a single reference picture list RefPicList0. At the beginning of the decoding process for each slice, reference picture list RefPicList0, are derived as specified by the following ordered steps:

1. An initial reference picture list RefPicList0 are derived as specified in subclause 8.2.4.2.
2. When ref_pic_list_modification_flag_l0 is equal to 1 or, the initial reference picture list RefPicList0 are modified as specified in subclause 8.2.4.3.

   NOTE – The modification process for reference picture lists specified in subclause 8.2.4.3allows the contents of RefPicList0 to be modified in a flexible fashion. In particular, it is possible for a picture that is currently marked "used for reference" to be inserted into RefPicList0 even when the picture is not in the initial reference picture list derived as specified in subclause 8.2.4.2.

The number of entries in the modified reference picture list RefPicList0 is num_ref_idx_l0_active_minus1 + 1,. A reference picture may appear at more than one index in the modified reference picture lists RefPicList0.

8.2.4.1 Decoding process for picture numbers

This process is invoked when the decoding process for reference picture lists construction specified in subclause 8.2.4, the decoded reference picture marking process specified in subclause 8.2.5, or the decoding process for gaps in frame_num specified in subclause 8.2.5.2 is invoked.

The variables FrameNum, FrameNumWrap, PicNum, LongTermFrameIdx, and LongTermPicNum are used for the initialisation process for reference picture lists in subclause 8.2.4.2, the modification process for reference picture lists in subclause 8.2.4.3, the decoded reference picture marking process in subclause 8.2.5, and the decoding process for gaps in frame_num in subclause 8.2.5.2.

To each short-term reference picture the variables FrameNum and FrameNumWrap are assigned as follows. First, FrameNum is set equal to the syntax element frame_num that has been decoded in the slice header(s) of the corresponding short-term reference picture. Then the variable FrameNumWrap is derived as

```
if( FrameNum > frame_num )
    FrameNumWrap = FrameNum − MaxFrameNum                (8-13)
else
    FrameNumWrap = FrameNum
```

where the value of frame_num used in Equation 8-13 is the frame_num in the slice header(s) for the current picture.

Each long-term reference picture has an associated value of LongTermFrameIdx (that was assigned to it as specified in subclause 8.2.5).

To each short-term reference picture a variable PicNum is assigned, and to each long-term reference picture a variable LongTermPicNum is assigned. T

8.2.4.2 Initialisation process for reference picture lists

This initialisation process is invoked when decoding a P, slice header.

RefPicList0 have initial entries as specified in subclause 8.2.4.2.1.

When the number of entries in the initial RefPicList0 produced as specified in subclause 8.2.4.2.1 is greater than num_ref_idx_l0_active_minus1 + 1 the extra entries past position num_ref_idx_l0_active_minus1 are discarded from the initial reference picture list.

When the number of entries in the initial RefPicList0 produced as specified in subclause 8.2.4.2.1 is less than num_ref_idx_l0_active_minus1 + 1, the remaining entries in the initial reference picture list are set equal to "no reference picture".

8.2.4.2.1 Initialisation process for the reference picture list for P slices in frames

This initialisation process is invoked when decoding a P slice in a coded frame.

When this process is invoked, there shall be at least one reference frame that is currently marked as "used for reference" (i.e., as "used for short-term reference" or "used for long-term reference") and is not marked as "non-existing".

The reference picture list RefPicList0 is ordered so that short-term reference frames s have lower indices than long-term reference frames The short-term reference frames are ordered starting with the frame with the highest PicNum value and proceeding through in descending order to the frame with the lowest PicNum value.

The long-term reference frames are ordered starting with the frame with the lowest LongTermPicNum value and proceeding through in ascending order to the frame with the highest LongTermPicNum value.

For example, when three reference frames are marked as "used for short-term reference" with PicNum equal to 300, 302, and 303 and two reference frames are marked as "used for long-term reference" with LongTermPicNum equal to 0 and 3, the initial index order is:

– RefPicList0[0] is set equal to the short-term reference picture with PicNum = 303,

– RefPicList0[1] is set equal to the short-term reference picture with PicNum = 302,

– RefPicList0[2] is set equal to the short-term reference picture with PicNum = 300,

– RefPicList0[3] is set equal to the long-term reference picture with LongTermPicNum = 0,

– RefPicList0[4] is set equal to the long-term reference picture with LongTermPicNum = 3.

8.2.4.3 Modification process for reference picture lists

When ref_pic_list_modification_flag_l0 is equal to 1, the following applies:

1. Let refIdxL0 be an index into the reference picture list RefPicList0. It is initially set equal to 0.
2. The corresponding syntax elements modification_of_pic_nums_idc are processed in the order they occur in the bitstream. For each of these syntax elements, the following applies:
   - If modification_of_pic_nums_idc is equal to 0 or equal to 1, the process specified in subclause 8.2.4.3.1 is invoked with refIdxL0 as input, and the output is assigned to refIdxL0.
   - Otherwise, if modification_of_pic_nums_idc is equal to 2, the process specified in subclause 8.2.4.3.2 is invoked with refIdxL0 as input, and the output is assigned to refIdxL0.
   - Otherwise (modification_of_pic_nums_idc is equal to 3), the modification process for reference picture list RefPicList0 is finished.

8.2.4.3.1 Modification process of reference picture lists for short-term reference pictures

Input to this process is an index refIdxL0

Output of this process is an incremented index refIdxL0.

The variable picNumL0NoWrap is derived as follows:

– If modification_of_pic_nums_idc is equal to 0,

```
if( picNumL0Pred − ( abs_diff_pic_num_minus1 + 1 ) < 0 )
    picNumL0NoWrap = picNumL0Pred − ( abs_diff_pic_num_minus1 + 1 ) + MaxPicNum        (8-14)
else
    picNumL0NoWrap = picNumL0Pred − ( abs_diff_pic_num_minus1 + 1 )
```

– Otherwise (modification_of_pic_nums_idc is equal to 1),

```
if( picNumL0Pred + ( abs_diff_pic_num_minus1 + 1 ) >= MaxPicNum )
    picNumL0NoWrap = picNumL0Pred + ( abs_diff_pic_num_minus1 + 1 ) − MaxPicNum        (8-15)
else
    picNumL0NoWrap = picNumL0Pred + ( abs_diff_pic_num_minus1 + 1 )
```


picNumL0Pred is the prediction value for the variable picNumL0NoWrap. When the process specified in this subclause is invoked the first time for a slice (that is, for the first occurrence of modification_of_pic_nums_idc equal to 0 or 1 in the ref_pic_list_modification( ) syntax), picNumL0Pred is initially set equal to CurrPicNum. After each assignment of picNumL0NoWrap, the value of picNumL0NoWrap is assigned to picNumL0Pred.

The variable picNumL0 is derived as specified by the following pseudo-code:

```
if( picNumL0NoWrap > CurrPicNum )
    picNumL0 = picNumL0NoWrap − MaxPicNum                                   (8-16)
else
    picNumL0 = picNumL0NoWrap
```

picNumL0 shall be equal to the PicNum of a reference picture that is marked as "used for short-term reference" and shall not be equal to the PicNum of a short-term reference picture that is marked as "non-existing".

The following procedure is conducted to place the picture with short-term picture number picNumL0 into the index position refIdxL0, shift the position of any other remaining pictures to later in the list, and increment the value of refIdxL0.

```
for( cIdx = num_ref_idx_l0_active_minus1 + 1; cIdx > refIdxL0; cIdx− − )
   RefPicList0[ cIdx ] = RefPicList0[ cIdx − 1]
RefPicList0[ refIdxL0++ ] = short-term reference picture with PicNum equal to picNumL0
nIdx = refIdxL0
for( cIdx = refIdxL0; cIdx <= num_ref_idx_l0_active_minus1 + 1; cIdx++ )    (8-17)
   if( PicNumF( RefPicList0[ cIdx ] ) != picNumL0 )
      RefPicList0[ nIdx++ ] = RefPicList0[ cIdx ]
```

where the function PicNumF(RefPicList0[cIdx]) is derived as follows:

– If the picture RefPicList0[cIdx] is marked as "used for short-term reference", PicNumF(RefPicList0[cIdx]) is the PicNum of the picture RefPicList0[cIdx].
– Otherwise (the picture RefPicList0[cIdx] is not marked as "used for short-term reference"), PicNumF(RefPicList0[cIdx]) is equal to MaxPicNum.

NOTE – A value of MaxPicNum can never be equal to picNumL0.

NOTE – Within this pseudo-code procedure, the length of the list RefPicList0 is temporarily made one element longer than the length needed for the final list. After the execution of this procedure, only elements 0 through num_ref_idx_l0_active_minus1 of the list need to be retained.

8.2.4.3.2 Modification process of reference picture lists for long-term reference pictures

Input to this process is an index refIdxL0.

Output of this process is an incremented index refIdxL0.

The following procedure is conducted to place the picture with long-term picture number long_term_pic_num into the index position refIdxL0, shift the position of any other remaining pictures to later in the list, and increment the value of refIdxL0.

```
for( cIdx = num_ref_idx_l0_active_minus1 + 1; cIdx > refIdxL0; cIdx− − )
   RefPicList0[ cIdx ] = RefPicList0[ cIdx − 1]
RefPicList0[ refIdxL0++ ] = long-term reference picture with LongTermPicNum equal to long_term_pic_num
nIdx = refIdxL0
for( cIdx = refIdxL0; cIdx <= num_ref_idx_l0_active_minus1 + 1; cIdx++ )    (8-18)
   if( LongTermPicNumF( RefPicList0[ cIdx ] ) != long_term_pic_num )
      RefPicList0[ nIdx++ ] = RefPicList0[ cIdx ]
```

where the function LongTermPicNumF(RefPicList0[cIdx]) is derived as follows:

– If the picture RefPicList0[cIdx] is marked as "used for long-term reference", LongTermPicNumF(RefPicList0[cIdx]) is the LongTermPicNum of the picture RefPicList0[cIdx].

– Otherwise (the picture RefPicList0[cIdx] is not marked as "used for long-term reference"), LongTermPicNumF(RefPicList0[cIdx]) is equal to 2 * (MaxLongTermFrameIdx + 1).

NOTE – A value of 2 * (MaxLongTermFrameIdx + 1) can never be equal to long_term_pic_num.

NOTE – Within this pseudo-code procedure, the length of the list RefPicList0 is temporarily made one element longer than the length needed for the final list. After the execution of this procedure, only elements 0 through num_ref_idx_l0_active_minus1 of the list need to be retained.

8.2.5 Decoded reference picture marking process

This process is invoked for decoded pictures when nal_ref_idc is not equal to 0.

NOTE – The decoding process for gaps in frame_num that is specified in subclause 8.2.5.2 may also be invoked when nal_ref_idc is equal to 0, as specified in clause 8.

A decoded picture with nal_ref_idc not equal to 0, referred to as a reference picture, is marked as "used for short-term reference" or "used for long-term reference". A picture that is marked as "used for short-term reference" is identified by its FrameNum. A picture that is marked as "used for long-term reference" is identified by its LongTermFrameIdx.

Frames marked as "used for short-term reference" or as "used for long-term reference" can be used as a reference for inter prediction when decoding a frame until the frame, is marked as "unused for reference".

A picture can be marked as "unused for reference" by the sliding window reference picture marking process, a first-in, first-out mechanism specified in subclause 8.2.5.3 or by the adaptive memory control reference picture marking process, a customised adaptive marking operation specified in subclause 8.2.5.4.

A short-term reference picture is identified for use in the decoding process by its variables FrameNum and FrameNumWrap and its picture number PicNum, and a long-term reference picture is identified for use in the decoding process by its long-term picture number LongTermPicNum. When the current picture is not an IDR picture, subclause 8.2.4.1 is invoked to specify the assignment of the variables FrameNum, FrameNumWrap, PicNum and LongTermPicNum.

8.2.5.1 Sequence of operations for decoded reference picture marking process

Decoded reference picture marking proceeds in the following ordered steps:

1. All slices of the current picture are decoded.
2. Depending on whether the current picture is an IDR picture, the following applies:
   - If the current picture is an IDR picture, the following ordered steps are specified:
     a. All reference pictures are marked as "unused for reference"
     b. Depending on long_term_reference_flag, the following applies:
        - If long_term_reference_flag is equal to 0, the IDR picture is marked as "used for short-term reference" and MaxLongTermFrameIdx is set equal to "no long-term frame indices".
        - Otherwise (long_term_reference_flag is equal to 1), the IDR picture is marked as "used for long-term reference", the LongTermFrameIdx for the IDR picture is set equal to 0, and MaxLongTermFrameIdx is set equal to 0.
   - Otherwise (the current picture is not an IDR picture), the following applies:
     - If adaptive_ref_pic_marking_mode_flag is equal to 0, the process specified in subclause 8.2.5.3 is invoked.
     - Otherwise (adaptive_ref_pic_marking_mode_flag is equal to 1), the process specified in subclause 8.2.5.4 is invoked.
3. When the current picture is not an IDR picture and it was not marked as "used for long-term reference" by memory_management_control_operation equal to 6, it is marked as "used for short-term reference".

It is a requirement of bitstream conformance that, after marking the current decoded reference picture, the total number of frames shall not be greater than Max(max_num_ref_frames, 1).

8.2.5.2 Decoding process for gaps in frame_num

This process is invoked when frame_num is not equal to PrevRefFrameNum and is not equal to (PrevRefFrameNum + 1) % MaxFrameNum.

NOTE – Although this process is specified as a subclause within subclause 8.2.5 (which defines a process that is invoked only when nal_ref_idc is not equal to 0), this process may also be invoked when nal_ref_idc is equal to 0 (as specified in clause 8). The reasons for the location of this subclause within the structure of this International Standard are historical.

NOTE – This process can only be invoked for a conforming bitstream when gaps_in_frame_num_value_allowed_flag is equal to 1. When gaps_in_frame_num_value_allowed_flag is equal to 0 and frame_num is not equal to PrevRefFrameNum and is not equal to (PrevRefFrameNum + 1) % MaxFrameNum, the decoding process should infer an unintentional loss of pictures.

When this process is invoked, a set of values of frame_num pertaining to "non-existing" pictures is derived as all values taken on by UnusedShortTermFrameNum in Equation 7-11 except the value of frame_num for the current picture.

For each of the values of frame_num pertaining to "non-existing" pictures, in the order in which the values of UnusedShortTermFrameNum are generated by Equation 7-11, the following ordered steps are specified:

1. The decoding process for picture numbers as specified in subclause 8.2.4.1 is invoked.
2. The sliding window decoded reference picture marking process as specified in subclause 8.2.5.3 is invoked.
3. The decoding process generates a frame and the generated frame is marked as "non-existing" and "used for short-term reference". The sample values of the generated frame may be set to any value.

The following constraints shall be obeyed:

a) (void)

b) The bitstream shall not contain data that result in the derivation of a reference picture that is marked as "non-existing" in any invocation of the reference picture selection process specified in subclause 8.4.2.1.

c) The bitstream shall not contain data that result in a variable picNumL0 that is equal to the PicNum of a picture marked as "non-existing" in any invocation of the modification process for reference picture lists for short-term reference pictures specified in subclause 8.2.4.3.1.

d) The bitstream shall not contain data that result in a variable picNumL0 that is equal to the PicNum of a picture marked as "non-existing" in any invocation of the assignment process of a LongTermFrameIdx to a short-term reference picture specified in subclause 8.2.5.4.3.

NOTE – The above constraints specify that frames that are marked as "non-existing" by the process specified in this subclause must not be referenced in the inter prediction process (subclause 8.4), the modification commands for reference picture lists for short-term reference pictures (subclause 8.2.4.3.1), or the assignment process of a LongTermFrameIdx to a short-term reference picture (subclause 8.2.5.4.3).

8.2.5.3 Sliding window decoded reference picture marking process

This process is invoked when adaptive_ref_pic_marking_mode_flag is equal to 0.

Depending on the properties of the current picture as specified below, the following applies:

1. Let numShortTerm be the total number of reference frames marked as "used for short-term reference". Let numLongTerm be the total number of reference frames marked as "used for long-term reference".
2. When numShortTerm + numLongTerm is equal to Max(max_num_ref_frames, 1), the condition that numShortTerm is greater than 0 shall be fulfilled, and the short-term reference frame that has the smallest value of FrameNumWrap is marked as "unused for reference".

8.2.5.4 Adaptive memory control decoded reference picture marking process

This process is invoked when adaptive_ref_pic_marking_mode_flag is equal to 1.

The memory_management_control_operation commands with values of 1 to 6 are processed in the order they occur in the bitstream after the current picture has been decoded. For each of these memory_management_control_operation commands, one of the processes specified in subclauses 8.2.5.4.1 to 8.2.5.4.6 is invoked depending on the value of memory_management_control_operation. The memory_management_control_operation command with value of 0 specifies the end of memory_management_control_operation commands.

Memory management control operations are applied to pictures as follows:

– memory_management_control_operation commands are applied to the frames specified.

8.2.5.4.1 Marking process of a short-term reference picture as "unused for reference"

This process is invoked when memory_management_control_operation is equal to 1.

Let picNumX be specified by

$$picNumX = CurrPicNum - (difference\_of\_pic\_nums\_minus1 + 1). \quad (8\text{-}19)$$

The value of picNumX is used to mark a short-term reference picture as "unused for reference" as follows:

– the short-term reference frame specified by picNumX is marked as "unused for reference".

8.2.5.4.2 Marking process of a long-term reference picture as "unused for reference"

This process is invoked when memory_management_control_operation is equal to 2.

The value of LongTermPicNum is used to mark a long-term reference picture as "unused for reference" as follows:

– the long-term reference frame having LongTermPicNum equal to long_term_pic_num is marked as "unused for reference".

8.2.5.4.3 Assignment process of a LongTermFrameIdx to a short-term reference picture

This process is invoked when memory_management_control_operation is equal to 3.

Given the syntax element difference_of_pic_nums_minus1, the variable picNumX is obtained as specified in subclause 8.2.5.4.1. picNumX shall refer to a frame marked as "used for short-term reference" and not marked as "non-existing".

When LongTermFrameIdx equal to long_term_frame_idx is already assigned to a long-term reference frame, that frame is marked as "unused for reference".

The value of LongTermFrameIdx is used to mark a picture from "used for short-term reference" to "used for long-term reference" as follows:

– The marking of the short-term reference frame specified by picNumX is changed from "used for short-term reference" to "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.

8.2.5.4.4 Decoding process for MaxLongTermFrameIdx

This process is invoked when memory_management_control_operation is equal to 4.

All pictures for which LongTermFrameIdx is greater than max_long_term_frame_idx_plus1 − 1 and that are marked as "used for long-term reference" are marked as "unused for reference".

The variable MaxLongTermFrameIdx is derived as follows:

– If max_long_term_frame_idx_plus1 is equal to 0, MaxLongTermFrameIdx is set equal to "no long-term frame indices".

– Otherwise (max_long_term_frame_idx_plus1 is greater than 0), MaxLongTermFrameIdx is set equal to max_long_term_frame_idx_plus1 − 1.

NOTE – The memory_management_control_operation command equal to 4 can be used to mark long-term reference pictures as "unused for reference". The frequency of transmitting max_long_term_frame_idx_plus1 is not specified by this International Standard. However, the encoder should send a memory_management_control_operation command equal to 4 upon receiving an error message, such as an intra refresh request message.

8.2.5.4.5 Marking process of all reference pictures as "unused for reference" and setting MaxLongTermFrameIdx to "no long-term frame indices"

This process is invoked when memory_management_control_operation is equal to 5.

All reference pictures are marked as "unused for reference" and the variable MaxLongTermFrameIdx is set equal to "no long-term frame indices".

8.2.5.4.6 Process for assigning a long-term frame index to the current picture

This process is invoked when memory_management_control_operation is equal to 6.

When a variable LongTermFrameIdx equal to long_term_frame_idx is already assigned to a long-term reference frame, that frame is marked as "unused for reference".

The current picture is marked as "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.

8.3 Intra prediction process

This process is invoked for I macroblock types.

Inputs to this process are constructed samples prior to the deblocking filter process and, for Intra_4x4 prediction modes, the values of Intra4x4PredMode from neighbouring macroblocks.

Outputs of this process are specified as follows:

– If the macroblock prediction mode is Intra_4x4, the outputs are constructed luma samples prior to the deblocking filter process and chroma prediction samples of the macroblock $pred_C$, where C is equal to Cb and Cr.

– Otherwise, if mb_type is not equal to I_PCM, the outputs are luma prediction samples of the macroblock $pred_L$ and chroma prediction samples of the macroblock $pred_C$, where C is equal to Cb and Cr.

– Otherwise (mb_type is equal to I_PCM), the outputs are constructed luma and chroma samples prior to the deblocking filter process.

The variable MvCnt is set equal to 0.

Depending on the value of mb_type the following applies:

– If mb_type is equal to I_PCM, the sample construction process for I_PCM macroblocks as specified in subclause 8.3.5 is invoked.

– Otherwise (mb_type is not equal to I_PCM), the following applies:

1. The decoding processes for Intra prediction modes are described for the luma component as follows:

   – If the macroblock prediction mode is equal to Intra_4x4, the Intra_4x4 prediction process for luma samples as specified in subclause 8.3.1 is invoked.

   – Otherwise (the macroblock prediction mode is equal to Intra_16x16), the Intra_16x16 prediction process as specified in subclause 8.3.3 is invoked with S'_L as the input and the outputs are luma prediction samples of the macroblock $pred_L$.

2. the Intra prediction process for chroma samples as specified in subclause 8.3.4 is invoked with S'_{Cb}, and S'_{Cr} as the inputs and the outputs are chroma prediction samples of the macroblock $pred_{Cb}$ and $pred_{Cr}$.

Samples used in the Intra prediction process are the sample values prior to alteration by any deblocking filter operation.

8.3.1 Intra_4x4 prediction process for luma samples

This process is invoked when the macroblock prediction mode is equal to Intra_4x4.

Inputs to this process are the values of Intra4x4PredMode (if available) from neighbouring macroblocks or macroblock pairs.

The luma component of a macroblock consists of 16 blocks of 4x4 luma samples. These blocks are inverse scanned using the 4x4 luma block inverse scanning process as specified in subclause 6.4.3.

For all 4x4 luma blocks of the luma component of a macroblock with luma4x4BlkIdx = 0..15, the derivation process for the Intra4x4PredMode as specified in subclause 8.3.1.1 is invoked with luma4x4BlkIdx as well as Intra4x4PredMode that are previously (in decoding order) derived for adjacent macroblocks as the input and the variable Intra4x4PredMode[luma4x4BlkIdx] as the output.

For each luma block of 4x4 samples indexed using luma4x4BlkIdx = 0..15, the following ordered steps are specified:

1. The Intra_4x4 sample prediction process in subclause 8.3.1.2 is invoked with luma4x4BlkIdx and the array S'_L containing constructed luma samples prior to the deblocking filter process from adjacent luma blocks as the inputs and the outputs are the Intra_4x4 luma prediction samples $pred4x4_L[x, y]$ with x, y = 0..3.
2. The position of the upper-left sample of a 4x4 luma block with index luma4x4BlkIdx inside the current macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (xO, yO).
3. The values of the prediction samples $pred_L[xO + x, yO + y]$ with x, y = 0..3 are derived by

$$pred_L[xO + x, yO + y] = pred4x4_L[x, y] \quad (8\text{-}20)$$

4. The transform coefficient decoding process and picture construction process prior to deblocking filter process in subclause 8.5 is invoked with $pred_L$ and luma4x4BlkIdx as the input and the constructed samples for the current 4x4 luma block S'_L as the output.

8.3.1.1 Derivation process for Intra4x4PredMode

Inputs to this process are the index of the 4x4 luma block luma4x4BlkIdx and variable arrays Intra4x4PredMode (if available) that are previously (in decoding order) derived for adjacent macroblocks.

Output of this process is the variable Intra4x4PredMode[luma4x4BlkIdx].

Table 8-1 specifies the values for Intra4x4PredMode[luma4x4BlkIdx] and the associated names.

Table 8-1 – Specification of Intra4x4PredMode[luma4x4BlkIdx] and associated names

Intra4x4PredMode[luma4x4BlkIdx]	Name of Intra4x4PredMode[luma4x4BlkIdx]
0	Intra_4x4_Vertical (prediction mode)
1	Intra_4x4_Horizontal (prediction mode)
2	Intra_4x4_DC (prediction mode)
3	Intra_4x4_Diagonal_Down_Left (prediction mode)
4	Intra_4x4_Diagonal_Down_Right (prediction mode)
5	Intra_4x4_Vertical_Right (prediction mode)
6	Intra_4x4_Horizontal_Down (prediction mode)
7	Intra_4x4_Vertical_Left (prediction mode)
8	Intra_4x4_Horizontal_Up (prediction mode)

Intra4x4PredMode[luma4x4BlkIdx] labelled 0, 1, 3, 4, 5, 6, 7, and 8 represent directions of predictions as illustrated in Figure 8-1.

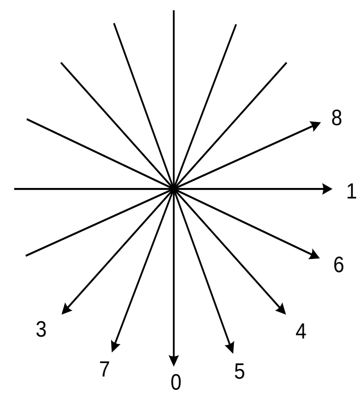

Figure 8-1 – Intra_4x4 prediction mode directions (informative)

Intra4x4PredMode[luma4x4BlkIdx] is derived as specified by the following ordered steps:

1. The process specified in subclause 6.4.11.4 is invoked with luma4x4BlkIdx given as input and the output is assigned to mbAddrA, luma4x4BlkIdxA, mbAddrB, and luma4x4BlkIdxB.
2. The variable dcPredModePredictedFlag is derived as follows:
   – If any of the following conditions are true, dcPredModePredictedFlag is set equal to 1
     – the macroblock with address mbAddrA is not available
     – the macroblock with address mbAddrB is not available
     – the macroblock with address mbAddrA is available and coded in an Inter macroblock prediction mode and constrained_intra_pred_flag is equal to 1
     – the macroblock with address mbAddrB is available and coded in an Inter macroblock prediction mode and constrained_intra_pred_flag is equal to 1
   – Otherwise, dcPredModePredictedFlag is set equal to 0.
3. For N being either replaced by A or B, the variables intraMxMPredModeN are derived as follows:
   – If dcPredModePredictedFlag is equal to 1 or the macroblock with address mbAddrN is not coded in Intra_4x4 macroblock prediction mode, intraMxMPredModeN is set equal to 2 (Intra_4x4_DC prediction mode).
   – Otherwise (dcPredModePredictedFlag is equal to 0 and the macroblock with address mbAddrN is coded in Intra_4x4 macroblock prediction mode), the following applies:
     – If the macroblock with address mbAddrN is coded in Intra_4x4 macroblock prediction mode, intraMxMPredModeN is set equal to Intra4x4PredMode[luma4x4BlkIdxN], where Intra4x4PredMode is the variable array assigned to the macroblock mbAddrN.
4. Intra4x4PredMode[luma4x4BlkIdx] is derived by applying the following procedure:

predIntra4x4PredMode = Min(intraMxMPredModeA, intraMxMPredModeB)
if(prev_intra4x4_pred_mode_flag[luma4x4BlkIdx])
  Intra4x4PredMode[luma4x4BlkIdx] = predIntra4x4PredMode
else (8-21)
  if(rem_intra4x4_pred_mode[luma4x4BlkIdx] < predIntra4x4PredMode)
    Intra4x4PredMode[luma4x4BlkIdx] = rem_intra4x4_pred_mode[luma4x4BlkIdx]
  else
    Intra4x4PredMode[luma4x4BlkIdx] = rem_intra4x4_pred_mode[luma4x4BlkIdx] + 1

8.3.1.2 Intra_4x4 sample prediction

This process is invoked for each 4x4 luma block of a macroblock with macroblock prediction mode equal to Intra_4x4 followed by the transform decoding process and picture construction process prior to deblocking for each 4x4 luma block.

Inputs to this process are:

– the index of a 4x4 luma block luma4x4BlkIdx,

– an ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array cS_L containing constructed luma samples prior to the deblocking filter process of neighbouring macroblocks.

Output of this process are the prediction samples $pred4x4_L$[x, y], with x, y = 0..3, for the 4x4 luma block with index luma4x4BlkIdx.

The position of the upper-left sample of a 4x4 luma block with index luma4x4BlkIdx inside the current macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (xO, yO).

The 13 neighbouring samples p[x, y] that are constructed luma samples prior to the deblocking filter process, with x = −1, y = −1..3 and x = 0..7, y = −1, are derived as specified by the following ordered steps:

1. The luma location (xN, yN) is specified by

$$xN = xO + x \quad (8\text{-}22)$$

$$yN = yO + y \quad (8\text{-}23)$$

2. The derivation process for neighbouring locations in subclause 6.4.12 is invoked for luma locations with (xN, yN) as input and mbAddrN and (xW, yW) as output.

3. Each sample p[x, y] with x = −1, y = −1..3 and x = 0..7, y = −1 is derived as follows:

   – If any of the following conditions are true, the sample p[x, y] is marked as "not available for Intra_4x4 prediction"

     – mbAddrN is not available,

     – the macroblock mbAddrN is coded in an Inter macroblock prediction mode and constrained_intra_pred_flag is equal to 1,

     – the macroblock mbAddrN has mb_type equal to SI and constrained_intra_pred_flag is equal to 1 and the current macroblock does not have mb_type equal to SI,

     – x is greater than 3 and luma4x4BlkIdx is equal to 3 or 11.

   – Otherwise, the sample p[x, y] is marked as "available for Intra_4x4 prediction" and the value of the sample p[x, y] is derived as specified by the following ordered steps:

     a. The location of the upper-left luma sample of the macroblock mbAddrN is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddrN as the input and the output is assigned to (xM, yM).

b. the sample value p[x, y] is derived as follows:

$$p[x, y] = cS_L[xM + xW, yM + yW] \quad (8\text{-}24)$$

When samples p[x, −1], with x = 4..7, are marked as "not available for Intra_4x4 prediction," and the sample p[3, −1] is marked as "available for Intra_4x4 prediction," the sample value of p[3, −1] is substituted for sample values p[x, −1], with x = 4..7, and samples p[x, −1], with x = 4..7, are marked as "available for Intra_4x4 prediction".

NOTE – Each block is assumed to be constructed into a picture array prior to decoding of the next block.

Depending on Intra4x4PredMode[luma4x4BlkIdx], one of the Intra_4x4 prediction modes specified in subclauses 8.3.1.2.1 to 8.3.1.2.9 is invoked.

8.3.1.2.1 Specification of Intra_4x4_Vertical prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 0.

This mode shall be used only when the samples p[x, −1] with x = 0..3 are marked as "available for Intra_4x4 prediction".

The values of the prediction samples $pred4x4_L$[x, y], with x, y = 0..3, are derived by

$$pred4x4_L[x, y] = p[x, -1], \text{ with } x, y = 0..3 \quad (8\text{-}25)$$

8.3.1.2.2 Specification of Intra_4x4_Horizontal prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 1.

This mode shall be used only when the samples p[−1, y], with y = 0..3, are marked as "available for Intra_4x4 prediction".

The values of the prediction samples $pred4x4_L$[x, y], with x, y = 0..3, are derived by

$$pred4x4_L[x, y] = p[-1, y], \text{ with } x,y = 0..3 \quad (8\text{-}26)$$

8.3.1.2.3 Specification of Intra_4x4_DC prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 2.

The values of the prediction samples $pred4x4_L$[x, y], with x, y = 0..3, are derived as follows:

– If all samples p[x, −1], with x = 0..3, and p[−1, y], with y = 0..3, are marked as "available for Intra_4x4 prediction", the values of the prediction samples $pred4x4_L$[x, y], with x, y = 0..3, are derived by

$$pred4x4_L[x, y] = (p[0, -1] + p[1, -1] + p[2, -1] + p[3, -1] + p[-1, 0] + p[-1, 1] + p[-1, 2] + p[-1, 3] + 4) >> 3 \quad (8\text{-}27)$$

– Otherwise, if any samples p[x, −1], with x = 0..3, are marked as "not available for Intra_4x4 prediction" and all samples p[−1, y], with y = 0..3, are marked as "available for Intra_4x4 prediction", the values of the prediction samples $pred4x4_L$[x, y], with x, y = 0..3, are derived by

$$pred4x4_L[x, y] = (p[-1, 0] + p[-1, 1] + p[-1, 2] + p[-1, 3] + 2) >> 2 \quad (8\text{-}28)$$

– Otherwise, if any samples p[−1, y], with y = 0..3, are marked as "not available for Intra_4x4 prediction" and all samples p[x, −1], with x = 0 .. 3, are marked as "available for Intra_4x4 prediction", the values of the prediction samples $pred4x4_L$[x, y], with x, y = 0 .. 3, are derived by

$$pred4x4_L[x, y] = (p[0, -1] + p[1, -1] + p[2, -1] + p[3, -1] + 2) >> 2 \quad (8\text{-}29)$$

– Otherwise (some samples p[x, −1], with x = 0..3, and some samples p[−1, y], with y = 0..3, are marked as "not available for Intra_4x4 prediction"), the values of the prediction samples $pred4x4_L$[x, y], with x, y = 0..3, are derived by (wherein $BitDepth_Y$ is equal to 8 in this standard):

$$pred4x4_L[x, y] = (1 << (BitDepth_Y - 1)) \quad (8\text{-}30)$$

NOTE – A 4x4 luma block can always be predicted using this mode.

8.3.1.2.4 Specification of Intra_4x4_Diagonal_Down_Left prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 3.

This mode shall be used only when the samples p[x, −1] with x = 0..7 are marked as "available for Intra_4x4 prediction".

The values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3, are derived as follows:

– If x is equal to 3 and y is equal to 3,

$$pred4x4_L[x, y] = (p[6, -1] + 3 * p[7, -1] + 2) >> 2 \quad (8\text{-}31)$$

– Otherwise (x is not equal to 3 or y is not equal to 3),

$$pred4x4_L[x, y] = (p[x + y, -1] + 2 * p[x + y + 1, -1] + p[x + y + 2, -1] + 2) >> 2 \quad (8\text{-}32)$$

8.3.1.2.5 Specification of Intra_4x4_Diagonal_Down_Right prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 4.

This mode shall be used only when the samples p[x, −1] with x = 0..3 and p[−1, y] with y = −1..3 are marked as "available for Intra_4x4 prediction".

The values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3, are derived as follows:

– If x is greater than y,

$$pred4x4_L[x, y] = (p[x - y - 2, -1] + 2 * p[x - y - 1, -1] + p[x - y, -1] + 2) >> 2 \quad (8\text{-}33)$$

– Otherwise if x is less than y,

$$pred4x4_L[x, y] = (p[-1, y - x - 2] + 2 * p[-1, y - x - 1] + p[-1, y - x] + 2) >> 2 \quad (8\text{-}34)$$

– Otherwise (x is equal to y),

$$pred4x4_L[x, y] = (p[0, -1] + 2 * p[-1, -1] + p[-1, 0] + 2) >> 2 \quad (8\text{-}35)$$

8.3.1.2.6 Specification of Intra_4x4_Vertical_Right prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 5.

This mode shall be used only when the samples p[x, −1] with x = 0..3 and p[−1, y] with y = −1..3 are marked as "available for Intra_4x4 prediction".

Let the variable zVR be set equal to 2 * x − y.

The values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3, are derived as follows:

– If zVR is equal to 0, 2, 4, or 6,

$$pred4x4_L[x, y] = (p[x - (y >> 1) - 1, -1] + p[x - (y >> 1), -1] + 1) >> 1 \quad (8\text{-}36)$$

– Otherwise, if zVR is equal to 1, 3, or 5,

$$pred4x4_L[x, y] = (p[x - (y >> 1) - 2, -1] + 2 * p[x - (y >> 1) - 1, -1] + p[x - (y >> 1), -1] + 2) >> 2 \quad (8\text{-}37)$$

– Otherwise, if zVR is equal to −1,

$$pred4x4_L[x, y] = (p[-1, 0] + 2 * p[-1, -1] + p[0, -1] + 2) >> 2 \quad (8\text{-}38)$$

– Otherwise (zVR is equal to −2 or −3),

$$pred4x4_L[x, y] = (p[-1, y - 1] + 2 * p[-1, y - 2] + p[-1, y - 3] + 2) >> 2 \quad (8\text{-}39)$$

8.3.1.2.7 Specification of Intra_4x4_Horizontal_Down prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 6.

This mode shall be used only when the samples p[x, −1] with x = 0..3 and p[−1, y] with y = −1..3 are marked as "available for Intra_4x4 prediction".

Let the variable zHD be set equal to 2 * y − x.

The values of the prediction samples $pred4x4_L$[x, y], with x, y = 0..3, are derived as follows:

– If zHD is equal to 0, 2, 4, or 6,

$$pred4x4_L[x, y] = (p[-1, y - (x >> 1) - 1] + p[-1, y - (x >> 1)] + 1) >> 1 \quad (8\text{-}40)$$

– Otherwise, if zHD is equal to 1, 3, or 5,

$$pred4x4_L[x, y] = (p[-1, y - (x >> 1) - 2] + 2 * p[-1, y - (x >> 1) - 1] + p[-1, y - (x >> 1)] + 2) >> 2 \quad (8\text{-}41)$$

– Otherwise, if zHD is equal to −1,

$$pred4x4_L[x, y] = (p[-1, 0] + 2 * p[-1, -1] + p[0, -1] + 2) >> 2 \quad (8\text{-}42)$$

– Otherwise (zHD is equal to −2 or −3),

$$pred4x4_L[x, y] = (p[x - 1, -1] + 2 * p[x - 2, -1] + p[x - 3, -1] + 2) >> 2 \quad (8\text{-}43)$$

8.3.1.2.8 Specification of Intra_4x4_Vertical_Left prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 7.

This mode shall be used only when the samples p[x, −1] with x = 0..7 are marked as "available for Intra_4x4 prediction".

The values of the prediction samples $pred4x4_L$[x, y], with x, y = 0..3, are derived as follows:

– If y is equal to 0 or 2,

$$pred4x4_L[x, y] = (p[x + (y >> 1), -1] + p[x + (y >> 1) + 1, -1] + 1) >> 1 \quad (8\text{-}44)$$

– Otherwise (y is equal to 1 or 3),

$$pred4x4_L[x, y] = (p[x + (y >> 1), -1] + 2 * p[x + (y >> 1) + 1, -1] + p[x + (y >> 1) + 2, -1] + 2) >> 2 \quad (8\text{-}45)$$

8.3.1.2.9 Specification of Intra_4x4_Horizontal_Up prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 8.

This mode shall be used only when the samples p[−1, y] with y = 0..3 are marked as "available for Intra_4x4 prediction".

Let the variable zHU be set equal to x + 2 * y.

The values of the prediction samples $pred4x4_L$[x, y], with x, y = 0..3, are derived as follows:

– If zHU is equal to 0, 2, or 4

$$pred4x4_L[x, y] = (p[-1, y + (x >> 1)] + p[-1, y + (x >> 1) + 1] + 1) >> 1 \quad (8\text{-}46)$$

– Otherwise, if zHU is equal to 1 or 3

$$pred4x4_L[x, y] = (p[-1, y + (x >> 1)] + 2 * p[-1, y + (x >> 1) + 1] + p[-1, y + (x >> 1) + 2] + 2) >> 2 \quad (8\text{-}47)$$

– Otherwise, if zHU is equal to 5,

$$pred4x4_L[x, y] = (p[-1, 2] + 3 * p[-1, 3] + 2) >> 2 \quad (8-48)$$

– Otherwise (zHU is greater than 5),

$$pred4x4_L[x, y] = p[-1, 3] \quad (8-49)$$

8.3.2 (void)

8.3.3 Intra_16x16 prediction process for luma samples

This process is invoked when the macroblock prediction mode is equal to Intra_16x16. It specifies how the Intra prediction luma samples for the current macroblock are derived.

Input to this process is a ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array cS_L containing constructed luma samples prior to the deblocking filter process of neighbouring macroblocks.

Outputs of this process are Intra prediction luma samples for the current macroblock $pred_L[x, y]$.

The 33 neighbouring samples p[x, y] that are constructed luma samples prior to the deblocking filter process, with x = −1, y = −1..15 and with x = 0..15, y = −1, are derived as specified by the following ordered steps:

1. The derivation process for neighbouring locations in subclause 6.4.12 is invoked for luma locations with (x, y) assigned to (xN, yN) as input and mbAddrN and (xW, yW) as output.
2. Each sample p[x, y] with x = −1, y = −1..15 and with x = 0..15, y = −1 is derived as follows:
   - If any of the following conditions are true, the sample p[x, y] is marked as "not available for Intra_16x16 prediction":
     - mbAddrN is not available,
     - the macroblock mbAddrN is coded in an Inter macroblock prediction mode and constrained_intra_pred_flag is equal to 1,
     - the macroblock mbAddrN has mb_type equal to SI and constrained_intra_pred_flag is equal to 1.
   - Otherwise, the sample p[x, y] is marked as "available for Intra_16x16 prediction" and the value of the sample p[x, y] is derived as specified by the following ordered steps:
     a. The location of the upper-left luma sample of the macroblock mbAddrN is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddrN as the input and the output is assigned to (xM, yM).
     b. the sample value p[x, y] is derived as follows:

$$p[x, y] = cS_L[xM + xW, yM + yW] \quad (8-50)$$

Let $pred_L[x, y]$ with x, y = 0..15 denote the prediction samples for the 16x16 luma block samples.

Intra_16x16 prediction modes are specified in Table 8-2.

Table 8-2 – Specification of Intra16x16PredMode and associated names

Intra16x16PredMode	Name of Intra16x16PredMode
0	Intra_16x16_Vertical (prediction mode)
1	Intra_16x16_Horizontal (prediction mode)
2	Intra_16x16_DC (prediction mode)
3	Intra_16x16_Plane (prediction mode)

Depending on Intra16x16PredMode, one of the Intra_16x16 prediction modes specified in subclauses 8.3.3.1 to 8.3.3.4 is invoked.

8.3.3.1 Specification of Intra_16x16_Vertical prediction mode

This Intra_16x16 prediction mode shall be used only when the samples p[x, −1] with x = 0..15 are marked as "available for Intra_16x16 prediction".

The values of the prediction samples $pred_L$[x, y], with x, y = 0..15, are derived by

$$pred_L[x, y] = p[x, -1], \text{ with } x, y = 0..15 \quad (8\text{-}51)$$

8.3.3.2 Specification of Intra_16x16_Horizontal prediction mode

This Intra_16x16 prediction mode shall be used only when the samples p[−1, y] with y = 0..15 are marked as "available for Intra_16x16 prediction".

The values of the prediction samples $pred_L$[x, y], with x, y = 0..15, are derived by

$$pred_L[x, y] = p[-1, y], \text{ with } x, y = 0..15 \quad (8\text{-}52)$$

8.3.3.3 Specification of Intra_16x16_DC prediction mode

This Intra_16x16 prediction mode operates, depending on whether the neighbouring samples are marked as "available for Intra_16x16 prediction", as follows:

– If all neighbouring samples p[x, −1], with x = 0..15, and p[−1, y], with y = 0..15, are marked as "available for Intra_16x16 prediction", the prediction for all luma samples in the macroblock is given by:

$$pred_L[x, y] = \left(\sum_{x'=0}^{15} p[x',-1] + \sum_{y'=0}^{15} p[-1,y'] + 16\right) >> 5, \text{ with } x, y = 0..15 \quad (8\text{-}53)$$

– Otherwise, if any of the neighbouring samples p[x, −1], with x = 0..15, are marked as "not available for Intra_16x16 prediction" and all of the neighbouring samples p[−1, y], with y = 0..15, are marked as "available for Intra_16x16 prediction", the prediction for all luma samples in the macroblock is given by:

$$pred_L[x, y] = \left(\sum_{y'=0}^{15} p[-1,y'] + 8\right) >> 4, \text{ with } x, y = 0..15 \quad (8\text{-}54)$$

– Otherwise, if any of the neighbouring samples p[−1, y], with y = 0..15, are marked as "not available for Intra_16x16 prediction" and all of the neighbouring samples p[x, −1], with x = 0..15, are marked as "available for Intra_16x16 prediction", the prediction for all luma samples in the macroblock is given by:

$$pred_L[x, y] = \left(\sum_{x'=0}^{15} p[x',-1] + 8\right) >> 4, \text{ with } x, y = 0..15 \quad (8\text{-}55)$$

– Otherwise (some of the neighbouring samples p[x, −1], with x = 0..15, and some of the neighbouring samples p[−1, y], with y = 0..15, are marked as "not available for Intra_16x16 prediction"), the prediction for all luma samples in the macroblock is given by:

$$pred_L[x, y] = (1 << (BitDepth_Y - 1)), \text{ with } x, y = 0..15 \qquad (8\text{-}56)$$

8.3.3.4 Specification of Intra_16x16_Plane prediction mode

This Intra_16x16 prediction mode shall be used only when the samples p[x, −1] with x =−1..15 and p[−1, y] with y = 0..15 are marked as "available for Intra_16x16 prediction".

The values of the prediction samples $pred_L[x, y]$, with x, y = 0..15, are derived by

$$pred_L[x, y] = Clip1_Y((a + b * (x - 7) + c * (y - 7) + 16) >> 5), \text{ with } x, y = 0..15, \qquad (8\text{-}57)$$

where

$$a = 16 * (p[-1, 15] + p[15, -1]) \qquad (8\text{-}58)$$

$$b = (5 * H + 32) >> 6 \qquad (8\text{-}59)$$

$$c = (5 * V + 32) >> 6 \qquad (8\text{-}60)$$

and H and V are specified as

$$H = \sum_{x'=0}^{7} (x'+1) * (p[8 + x', -1] - p[6 - x', -1]) \qquad (8\text{-}61)$$

$$V = \sum_{y'=0}^{7} (y'+1) * (p[-1, 8 + y'] - p[-1, 6 - y']) \qquad (8\text{-}62)$$

8.3.4 Intra prediction process for chroma samples

This process is invoked for I macroblock types. It specifies how the Intra prediction chroma samples for the current macroblock are derived. (ChromaArrayType = 1 in this standard).

Inputs to this process are two $(PicWidthInSamples_C)x(PicHeightInSamples_C)$ arrays cS_{Cb} and cS_{Cr} containing constructed chroma samples prior to the deblocking filter process of neighbouring macroblocks.

Outputs of this process are Intra prediction chroma samples for the current macroblock $pred_{Cb}[x, y]$ and $pred_{Cr}[x, y]$.

The following applies:

the following text specifies the Intra prediction chroma samples for the current macroblock $pred_{Cb}[x, y]$ and $pred_{Cr}[x, y]$.

Both chroma blocks (Cb and Cr) of the macroblock use the same prediction mode. The prediction mode is applied to each of the chroma blocks separately. The process specified in this subclause is invoked for each chroma block. In the remainder of this subclause, chroma block refers to one of the two chroma blocks and the subscript C is used as a replacement of the subscript Cb or Cr.

The neighbouring samples p[x, y] that are constructed chroma samples prior to the deblocking filter process, with x = −1, y = −1..MbHeightC − 1 and with x = 0..MbWidthC − 1, y = −1, are derived as specified by the following ordered steps:

1. The derivation process for neighbouring locations in subclause 6.4.12 is invoked for chroma locations with (x, y) assigned to (xN, yN) as input and mbAddrN and (xW, yW) as output.

2. Each sample p[x, y] is derived as follows:

   – If any of the following conditions are true, the sample p[x, y] is marked as "not available for Intra chroma prediction":

– mbAddrN is not available,

– the macroblock mbAddrN is coded in an Inter macroblock prediction mode and constrained_intra_pred_flag is equal to 1,

– the macroblock mbAddrN has mb_type equal to SI and constrained_intra_pred_flag is equal to 1 and the current macroblock does not have mb_type equal to SI.

– Otherwise, the sample p[x, y] is marked as "available for Intra chroma prediction" and the value of the sample p[x, y] is derived as specified by the following ordered steps:

a. The location of the upper-left luma sample of the macroblock mbAddrN is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddrN as the input and the output is assigned to (xL, yL).

b. The location (xM, yM) of the upper-left chroma sample of the macroblock mbAddr is derived by:

$$xM = (xL >> 4) * MbWidthC \quad (8\text{-}63)$$
$$yM = ((yL >> 4)* MbHeightC) + (yL \% 2) \quad (8\text{-}64)$$

c. the sample value p[x, y] is derived as follows:

$$p[x, y] = cS_C[xM + xW, yM + yW] \quad (8\text{-}65)$$

Let $pred_C[x, y]$ with x = 0..MbWidthC − 1, y = 0..MbHeightC − 1 denote the prediction samples for the chroma block samples.

Intra chroma prediction modes are specified in Table 8-3.

Table 8-3 – Specification of Intra chroma prediction modes and associated names

intra_chroma_pred_mode	Name of intra_chroma_pred_mode
0	Intra_Chroma_DC (prediction mode)
1	Intra_Chroma_Horizontal (prediction mode)
2	Intra_Chroma_Vertical (prediction mode)
3	Intra_Chroma_Plane (prediction mode)

Depending on intra_chroma_pred_mode, one of the Intra chroma prediction modes specified in subclauses 8.3.4.1 to 8.3.4.4is invoked.

8.3.4.1 Specification of Intra_Chroma_DC prediction mode

This Intra chroma prediction mode is invoked when intra_chroma_pred_mode is equal to 0. (ChromaArrayType = 1 in this standard).

For each chroma block of 4x4 samples indexed by chroma4x4BlkIdx = 0..(1 << (ChromaArrayType + 1)) − 1, the following applies:

– The position of the upper-left sample of a 4x4 chroma block with index chroma4x4BlkIdx inside the current macroblock is derived by invoking the inverse 4x4 chroma block scanning process in subclause 6.4.7 with chroma4x4BlkIdx as the input and the output being assigned to (xO, yO).

– Depending on the values of xO and yO, the following applies:

– If (xO, yO) is equal to (0, 0) or xO and yO are greater than 0, the values of the prediction samples $pred_C[x + xO, y + yO]$ with x, y = 0..3 are derived as follows:

– If all samples p[x + xO, −1], with x = 0..3, and p[−1, y +yO], with y = 0..3, are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C$[x + xO, y + yO], with x, y = 0..3, are derived as:

$$pred_C[\, x + xO, y + yO \,] = \left(\sum_{x'=0}^{3} p[x'+xO, -1] + \sum_{y'=0}^{3} p[-1, y'+yO] + 4 \right) >> 3 \text{, with x, y = 0..3.} \qquad (8\text{-}66)$$

– Otherwise, if any samples p[x + xO, −1], with x = 0..3, are marked as "not available for Intra chroma prediction" and all samples p[−1, y +yO], with y = 0..3, are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C$[x + xO, y + yO], with x, y = 0..3, are derived as:

$$pred_C[\, x + xO, y + yO \,] = \left(\sum_{y'=0}^{3} p[-1, y'+yO] + 2 \right) >> 2 \text{, with x, y = 0..3.} \qquad (8\text{-}67)$$

– Otherwise, if any samples p[−1, y +yO], with y = 0..3, are marked as "not available for Intra chroma prediction" and all samples p[x + xO, −1], with x = 0..3, are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C$[x + xO, y + yO], with x, y = 0..3, are derived as:

$$pred_C[\, x + xO, y + yO \,] = \left(\sum_{x'=0}^{3} p[x'+xO, -1] + 2 \right) >> 2 \text{, with x, y = 0..3.} \qquad (8\text{-}68)$$

– Otherwise (some samples p[x + xO, −1], with x = 0..3, and some samples p[−1, y +yO], with y = 0..3, are marked as "not available for Intra chroma prediction"), the values of the prediction samples $pred_C$[x + xO, y + yO], with x, y = 0..3, are derived as (wherein $BitDepth_C$ is equal to 8 in this standard):

$$pred_C[\, x + xO, y + yO \,] = (1 << (BitDepth_C - 1)) \text{, with x, y = 0..3.} \qquad (8\text{-}69)$$

– Otherwise, if xO is greater than 0 and yO is equal to 0, the values of the prediction samples $pred_C$[x + xO, y + yO] with x, y = 0..3 are derived as follows:

– If all samples p[x + xO, −1], with x = 0..3, are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C$[x + xO, y + yO], with x, y = 0..3, are derived as:

$$pred_C[\, x + xO, y + yO \,] = \left(\sum_{x'=0}^{3} p[x'+xO, -1] + 2 \right) >> 2 \text{, with x, y = 0..3.} \qquad (8\text{-}70)$$

– Otherwise, if all samples p[−1, y +yO], with y = 0..3, are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C$[x + xO, y + yO], with x, y = 0..3, are derived as:

$$pred_C[\, x + xO, y + yO \,] = \left(\sum_{y'=0}^{3} p[-1, y'+yO] + 2 \right) >> 2 \text{, with x, y = 0..3.} \qquad (8\text{-}71)$$

– Otherwise (some samples p[x + xO, −1], with x = 0..3, and some samples p[−1, y +yO], with y = 0..3, are marked as "not available for Intra chroma prediction"), the values of the prediction samples $pred_C$[x + xO, y + yO], with x, y = 0..3, are derived as:

$$pred_C[\, x + xO, y + yO \,] = (1 << (BitDepth_C - 1)) \text{, with x, y = 0..3.} \qquad (8\text{-}72)$$

– Otherwise (xO is equal to 0 and yO is greater than 0), the values of the prediction samples $pred_C$[x + xO, y + yO] with x, y = 0..3 are derived as follows:

– If all samples p[−1, y +yO], with y = 0..3, are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C$[x + xO, y + yO], with x, y = 0..3, are derived as:

$$pred_C[\ x + xO, y + yO\ ] = \left(\sum_{y'=0}^{3} p[-1, y'+yO] + 2 \right) >> 2\text{, with x, y = 0..3.} \quad (8\text{-}73)$$

– Otherwise, if all samples p[x + xO, −1], with x = 0..3, are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C$[x + xO, y + yO], with x, y = 0..3, are derived as:

$$pred_C[\ x + xO, y + yO\ ] = (\ 1 << (\ BitDepth_C - 1\ )\ )\text{, with x, y = 0..3.} \quad (8\text{-}74)$$

– Otherwise (some samples p[x + xO, −1], with x = 0..3, and some samples p[−1, y +yO], with y = 0..3, are marked as "not available for Intra chroma prediction"), the values of the prediction samples $pred_C$[x + xO, y + yO], with x, y = 0..3, are derived as:

$$pred_C[\ x + xO, y + yO\ ] = (\ 1 << (\ BitDepth_C - 1\ )\ )\text{, with x, y = 0..3.} \quad (8\text{-}75)$$

8.3.4.2 Specification of Intra_Chroma_Horizontal prediction mode

This Intra chroma prediction mode is invoked when intra_chroma_pred_mode is equal to 1.

This mode shall be used only when the samples p[−1, y] with y = 0..MbHeightC − 1 are marked as "available for Intra chroma prediction".

The values of the prediction samples $pred_C$[x, y] are derived as:

$$pred_C[\ x, y\ ] = p[\ -1, y\ ]\text{, with x = 0..MbWidthC − 1 and y = 0..MbHeightC − 1} \quad (8\text{-}76)$$

8.3.4.3 Specification of Intra_Chroma_Vertical prediction mode

This Intra chroma prediction mode is invoked when intra_chroma_pred_mode is equal to 2.

This mode shall be used only when the samples p[x, −1] with x = 0..MbWidthC − 1 are marked as "available for Intra chroma prediction".

The values of the prediction samples $pred_C$[x, y] are derived as:

$$pred_C[\ x, y\ ] = p[\ x, -1\ ]\text{, with x = 0..MbWidthC − 1 and y = 0..MbHeightC − 1} \quad (8\text{-}77)$$

8.3.4.4 Specification of Intra_Chroma_Plane prediction mode

This Intra chroma prediction mode is invoked when intra_chroma_pred_mode is equal to 3. (ChromaArrayType = 1 in this standard)

This mode shall be used only when the samples p[x, −1], with x = 0..MbWidthC − 1 and p[−1, y], with y = −1..MbHeightC − 1 are marked as "available for Intra chroma prediction".

The values of the prediction samples $pred_C$[x, y] are derived by:

$$pred_C[\ x, y\ ] = Clip1_C(\ (\ a + b * (\ x - 3\ ) + c * (\ y - 3\ ) + 16\ ) >> 5\ ),$$
$$\text{with x = 0..MbWidthC − 1 and y = 0..MbHeightC − 1} \quad (8\text{-}78)$$

where

$$a = 16 * (\ p[\ -1, MbHeightC - 1\ ] + p[\ MbWidthC - 1, -1\ ]\ ) \quad (8\text{-}79)$$

$$b = (\ 34\ * H + 32\ ) >> 6 \quad (8\text{-}80)$$

$$c = (\ 34\ * V + 32\ ) >> 6 \quad (8\text{-}81)$$

and H and V are specified as:

$$H = \sum_{x'=0}^{3+xCF} (x'+1) * (p[4 + xCF + x', -1] - p[2 + xCF - x', -1]) \quad (8-82)$$

$$V = \sum_{y'=0}^{3+yCF} (y'+1) * (p[-1, 4 + yCF + y'] - p[-1, 2 + yCF - y']) \quad (8-83)$$

8.3.5 Sample construction process for I_PCM macroblocks

This process is invoked when mb_type is equal to I_PCM.

The position of the upper-left luma sample of the current macroblock is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with CurrMbAddr as input and the output being assigned to (xP, yP).

The constructed luma samples prior to the deblocking process are generated as specified by:

```
for( i = 0; i < 256; i++ )
   S'L[ xP + ( i % 16 ), yP + ( i / 16 ) ) ] = pcm_sample_luma[ i ]                    (8-84)
```

The constructed chroma samples prior to the deblocking process are generated as specified by:

```
for( i = 0; i < MbWidthC * MbHeightC; i++ ) {
   S'Cb[ ( xP / 2 ) + ( i % MbWidthC ),
        ( ( yP + 2− 1 ) / 2) + ( i / MbWidthC ) ] =
         pcm_sample_chroma[ i ]                                                        (8-85)
   S'Cr[ ( xP / 2 ) + ( i % MbWidthC ),
        ( ( yP + 2− 1 ) / 2) + ( i / MbWidthC ) ] =
         pcm_sample_chroma[ i + MbWidthC * MbHeightC ]
}
```

8.4 Inter prediction process

This process is invoked when decoding P macroblock types.

Outputs of this process are Inter prediction samples for the current macroblock that are a 16x16 array $pred_L$ of luma samples and two (MbWidthC)x(MbHeightC) arrays $pred_{Cb}$ and $pred_{Cr}$ of chroma samples, one for each of the chroma components Cb and Cr.

The partitioning of a macroblock is specified by mb_type. Each macroblock partition is referred to by mbPartIdx. When the macroblock partitioning consists of partitions that are equal to sub-macroblocks, each sub-macroblock can be further partitioned into sub-macroblock partitions as specified by sub_mb_type[mbPartIdx]. Each sub-macroblock partition is referred to by subMbPartIdx. When the macroblock partitioning does not consist of sub-macroblocks, subMbPartIdx is set equal to 0.

The following steps are specified for each macroblock partition or for each sub-macroblock partition.

The functions MbPartWidth( ), MbPartHeight( ), SubMbPartWidth( ), and SubMbPartHeight( ) describing the width and height of macroblock partitions and sub-macroblock partitions are specified in Tables 7-9 and 7-12.

The range of the macroblock partition index mbPartIdx is derived as follows:

– mbPartIdx proceeds over values 0..NumMbPart(mb_type) − 1.

For each value of mbPartIdx, the variables partWidth and partHeight for each macroblock partition or sub-macroblock partition in the macroblock are derived as follows:

– If mb_type is not equal to P_8x8 or P_8x8ref0, subMbPartIdx is set equal to 0, and partWidth and partHeight are derived as:

partWidth = MbPartWidth(mb_type) (8-86)

partHeight = MbPartHeight(mb_type) (8-87)

– Otherwise, if mb_type is equal to P_8x8 or P_8x8ref0, subMbPartIdx proceeds over values 0..NumSubMbPart(sub_mb_type[mbPartIdx]) − 1, and partWidth and partHeight are derived as:

partWidth = SubMbPartWidth(sub_mb_type[mbPartIdx]) (8-88)

partHeight = SubMbPartHeight(sub_mb_type[mbPartIdx]). (8-89)

The variables partWidthC and partHeightC are derived as:

partWidthC = partWidth / 2 (8-90)
partHeightC = partHeight / 2 (8-91)

Let the variable MvCnt be initially set equal to 0 before any invocation of subclause 8.4.1 for the macroblock.

The Inter prediction process for a macroblock partition mbPartIdx and a sub-macroblock partition subMbPartIdx consists of the following ordered steps:

1. The derivation process for motion vector components and reference indices as specified in subclause 8.4.1 is invoked.

   Inputs to this process are:

   – a macroblock partition mbPartIdx,

   – a sub-macroblock partition subMbPartIdx.

   Outputs of this process are:

   – luma motion vector mvL0 and the chroma motion vector mvCL0

   – reference indices refIdxL0

   – prediction list utilization flag predFlagL0

   – the sub-macroblock partition motion vector count subMvCnt.

2. The variable MvCnt is incremented by subMvCnt.

3. (void)

4. The decoding process for Inter prediction samples as specified in subclause 8.4.2 is invoked.

   Inputs to this process are:

   – a macroblock partition mbPartIdx,

   – a sub-macroblock partition subMbPartIdx,

   – variables specifying partition width and height for luma and chroma (if available), partWidth, partHeight, partWidthC (if available), and partHeightC (if available),

   – luma motion vector mvL0 and the chroma motion vector mvCL0,

   – reference index refIdxL0,

   – prediction list utilization flag predFlagL0,

   Outputs of this process are inter prediction samples (pred); which are a (partWidth)x(partHeight) array $predPart_L$ of prediction luma samples and two (partWidthC)x(partHeightC) arrays $predPart_{Cr}$, and $predPart_{Cb}$ of prediction chroma samples, one for each of the chroma components Cb and Cr.

For use in derivation processes of variables invoked later in the decoding process, the following assignments are made:

MvL0[mbPartIdx][subMbPartIdx] = mvL0 (8-92)

$RefIdxL0[mbPartIdx] = refIdxL0$ (8-93)

$PredFlagL0[mbPartIdx] = predFlagL0$ (8-94)

The location of the upper-left sample of the macroblock partition relative to the upper-left sample of the macroblock is derived by invoking the inverse macroblock partition scanning process as described in subclause 6.4.2.1 with mbPartIdx as the input and (xP, yP) as the output.

The location of the upper-left sample of the sub-macroblock partition relative to the upper-left sample of the macroblock partition is derived by invoking the inverse sub-macroblock partition scanning process as described in subclause 6.4.2.2 with subMbPartIdx as the input and (xS, yS) as the output.

The macroblock prediction is formed by placing the macroblock or sub-macroblock partition prediction samples in their correct relative positions in the macroblock, as follows.

The variable $pred_L[xP + xS + x, yP + yS + y]$ with $x = 0..partWidth - 1$, $y = 0..partHeight - 1$ is derived by:

$$pred_L[xP + xS + x, yP + yS + y] = predPart_L[x, y] \quad (8\text{-}95)$$

The variable $pred_C$ with $x = 0..partWidthC - 1$, $y = 0..partHeightC - 1$, and C in $pred_C$ and $predPart_C$ being replaced by Cb or Cr is derived by:

$$pred_C[xP / 2 + xS / 2 + x, yP / 2+ yS / 2+ y] = predPart_C[x, y] \quad (8\text{-}96)$$

8.4.1 Derivation process for motion vector components and reference indices

Inputs to this process are:

– a macroblock partition mbPartIdx,

– a sub-macroblock partition subMbPartIdx.

Outputs of this process are:

– luma motion vector mvL0 and the chroma motion vector mvCL0,

– reference index refIdxL0,

– prediction list utilization flag predFlagL0,

– a motion vector count variable subMvCnt.

For the derivation of the variables mvL0 as well as refIdxL0, the following applies:

1. The variables refIdxL0 and predFlagL0 are derived as follows:
   - If MbPartPredMode(mb_type, mbPartIdx) or SubMbPredMode(sub_mb_type[mbPartIdx]) is equal to Pred_L0,

     $refIdxL0 = ref\_idx\_l0[mbPartIdx]$ (8-97)

     $predFlagL0 = 1$ (8-98)

   - Otherwise, the variables refIdxL0 and predFlagL0 are specified by

     $refIdxL0 = -1$ (8-99)

     $predFlagL0 = 0$ (8-100)

2. The motion vector count variable subMvCnt is set equal to predFlagL0.

3. The variable currSubMbType is derived as follows:

   currSubMbType is set equal to "na".

4. When predFlagL0 is equal to 1, the derivation process for luma motion vector prediction in subclause 8.4.1.3 is invoked with mbPartIdx subMbPartIdx, refIdxL0, and currSubMbType as the inputs and the output being mvpL0. The luma motion vectors are derived by

$$mvL0[\ 0\ ] = mvpL0[\ 0\ ] + mvd\_l0[\ mbPartIdx\ ][\ subMbPartIdx\ ][\ 0\ ] \quad (8\text{-}101)$$

$$mvL0[\ 1\ ] = mvpL0[\ 1\ ] + mvd\_l0[\ mbPartIdx\ ][\ subMbPartIdx\ ][\ 1\ ] \quad (8\text{-}102)$$

When predFlagL0 is equal to 1, the derivation process for chroma motion vectors in subclause 8.4.1.4 is invoked with mvL0 and refIdxL0 as input and the output being mvCL0.

8.4.1.1 Derivation process for luma motion vectors for skipped macroblocks in P slices

This process is invoked when mb_type is equal to P_Skip.

Outputs of this process are the motion vector mvL0 and the reference index refIdxL0.

The reference index refIdxL0 for a skipped macroblock is derived as:

$$refIdxL0 = 0. \quad (8\text{-}103)$$

For the derivation of the motion vector mvL0 of a P_Skip macroblock type, the following ordered steps are specified:

1. The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx set equal to 0, subMbPartIdx set equal to 0, currSubMbType set equal to "na", and listSuffixFlag set equal to 0 as input and the output is assigned to mbAddrA, mbAddrB, mvL0A, mvL0B, refIdxL0A, and refIdxL0B.
2. The variable mvL0 is specified as follows:
   - If any of the following conditions are true, both components of the motion vector mvL0 are set equal to 0:
     - mbAddrA is not available,
     - mbAddrB is not available,
     - refIdxL0A is equal to 0 and both components of mvL0A are equal to 0,
     - refIdxL0B is equal to 0 and both components of mvL0B are equal to 0.
   - Otherwise, the derivation process for luma motion vector prediction as specified in subclause 8.4.1.3 is invoked with mbPartIdx = 0, subMbPartIdx = 0, refIdxL0, and currSubMbType = "na" as inputs and the output is assigned to mvL0.

NOTE – The output is directly assigned to mvL0, since the predictor is equal to the actual motion vector.

8.4.1.2 (void)

8.4.1.3 Derivation process for luma motion vector prediction

Inputs to this process are:

- the macroblock partition index mbPartIdx,
- the sub-macroblock partition index subMbPartIdx,
- the reference index of the current partition refIdxL0
- the variable currSubMbType.

Output of this process is the prediction mvpL0 of the motion vector mvL0 .

The derivation process for the neighbouring blocks for motion data in subclause 8.4.1.3.2 is invoked with mbPartIdx, subMbPartIdx, currSubMbType, and listSuffixFlag = 0 as the input and with mbAddrN\mbPartIdxN\subMbPartIdxN, reference indices refIdxL0N and the motion vectors mvL0N with N being replaced by A, B, or C as the output.

The motion vector predictor mvpL0 is derived as follows:

– If MbPartWidth(mb_type) is equal to 16, MbPartHeight(mb_type) is equal to 8, mbPartIdx is equal to 0, and refIdxL0B is equal to refIdxL0, the motion vector predictor mvpL0 is derived by:

$$mvpL0 = mvL0B \quad (8\text{-}104)$$

– Otherwise, if MbPartWidth(mb_type) is equal to 16, MbPartHeight(mb_type) is equal to 8, mbPartIdx is equal to 1, and refIdxL0A is equal to refIdxL0, the motion vector predictor mvpL0 is derived by:

$$mvpL0 = mvL0A \quad (8\text{-}105)$$

– Otherwise, if MbPartWidth(mb_type) is equal to 8, MbPartHeight(mb_type) is equal to 16, mbPartIdx is equal to 0, and refIdxL0A is equal to refIdxL0, the motion vector predictor mvpL0 is derived by:

$$mvpL0 = mvL0A \quad (8\text{-}106)$$

– Otherwise, if MbPartWidth(mb_type) is equal to 8, MbPartHeight(mb_type) is equal to 16, mbPartIdx is equal to 1, and refIdxL0C is equal to refIdxL0, the motion vector predictor mvpL0 is derived by:

$$mvpL0 = mvL0C \quad (8\text{-}107)$$

– Otherwise, the derivation process for median luma motion vector prediction in subclause 8.4.1.3.1 is invoked with mbAddrN\mbPartIdxN\subMbPartIdxN, mvL0N, refIdxL0N with N being replaced by A, B, or C, and refIdxL0 as the inputs and the output is assigned to the motion vector predictor mvpL0.

Figure 8-2 illustrates the non-median prediction as specified in Equations 8-104 to 8-107.

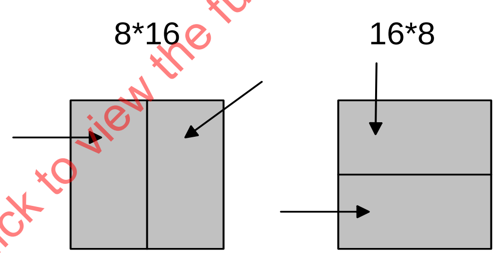

Figure 8-2 – Directional segmentation prediction (informative)

8.4.1.3.1 Derivation process for median luma motion vector prediction

Inputs to this process are:

– the neighbouring partitions mbAddrN\mbPartIdxN\subMbPartIdxN (with N being replaced by A, B, or C),

– the motion vectors mvL0N (with N being replaced by A, B, or C) of the neighbouring partitions,

– the reference indices refIdxL0N (with N being replaced by A, B, or C) of the neighbouring partitions,

– the reference index refIdxL0 of the current partition.

Output of this process is the motion vector prediction mvpL0.

The variable mvpL0 is derived as specified by the following ordered steps:

1. When both partitions mbAddrB\mbPartIdxB\subMbPartIdxB and mbAddrC\mbPartIdxC\subMbPartIdxC are not available and mbAddrA\mbPartIdxA\subMbPartIdxA is available,

$$\mathrm{mvL0B} = \mathrm{mvL0A} \tag{8-108}$$

$$\mathrm{mvL0C} = \mathrm{mvL0A} \tag{8-109}$$

$$\mathrm{refIdxL0B} = \mathrm{refIdxL0A} \tag{8-110}$$

$$\mathrm{refIdxL0C} = \mathrm{refIdxL0A} \tag{8-111}$$

2. Depending on reference indices refIdxL0A, refIdxL0B, or refIdxL0C, the following applies:

   – If one and only one of the reference indices refIdxL0A, refIdxL0B, or refIdxL0C is equal to the reference index refIdxL0 of the current partition, the following applies. Let refIdxL0N be the reference index that is equal to refIdxL0, the motion vector mvL0N is assigned to the motion vector prediction mvpL0:

$$\mathrm{mvpL0} = \mathrm{mvL0N} \tag{8-112}$$

   – Otherwise, each component of the motion vector prediction mvpL0 is given by the median of the corresponding vector components of the motion vector mvL0A, mvL0B, and mvL0C:

$$\mathrm{mvpL0}[\,0\,] = \mathrm{Median}(\,\mathrm{mvL0A}[\,0\,], \mathrm{mvL0B}[\,0\,], \mathrm{mvL0C}[\,0\,]\,) \tag{8-113}$$

$$\mathrm{mvpL0}[\,1\,] = \mathrm{Median}(\,\mathrm{mvL0A}[\,1\,], \mathrm{mvL0B}[\,1\,], \mathrm{mvL0C}[\,1\,]\,) \tag{8-114}$$

8.4.1.3.2 Derivation process for motion data of neighbouring partitions

Inputs to this process are:

– the macroblock partition index mbPartIdx,

– the sub-macroblock partition index subMbPartIdx,

– the current sub-macroblock type currSubMbType,

– the list suffix flag listSuffixFlag.

Outputs of this process are (with N being replaced by A, B, or C)

– mbAddrN\mbPartIdxN\subMbPartIdxN specifying neighbouring partitions,

– the motion vectors mvL0N of the neighbouring partitions,

– the reference indices refIdxL0N of the neighbouring partitions.

Variable names that include the string "L0" are interpreted with the 0 being equal to listSuffixFlag.

The partitions mbAddrN\mbPartIdxN\subMbPartIdxN with N being either A, B, or C are derived in the following ordered steps:

1. Let mbAddrD\mbPartIdxD\subMbPartIdxD be variables specifying an additional neighbouring partition.

2. The process in subclause 6.4.11.7 is invoked with mbPartIdx, currSubMbType, and subMbPartIdx as input and the output is assigned to mbAddrN\mbPartIdxN\subMbPartIdxN with N being replaced by A, B, C, or D.

3. When the partition mbAddrC\mbPartIdxC\subMbPartIdxC is not available, the following applies:

$$\mathrm{mbAddrC} = \mathrm{mbAddrD} \tag{8-115}$$

$$\mathrm{mbPartIdxC} = \mathrm{mbPartIdxD} \tag{8-116}$$

subMbPartIdxC = subMbPartIdxD (8-117)

The motion vectors mvL0N and reference indices refIdxL0N (with N being A, B, or C) are derived as follows:

– If the macroblock partition or sub-macroblock partition mbAddrN\mbPartIdxN\subMbPartIdxN is not available or mbAddrN is coded in an Intra macroblock prediction mode or predFlagL0 of mbAddrN\mbPartIdxN\subMbPartIdxN is equal to 0, both components of mvL0N are set equal to 0 and refIdxL0N is set equal to −1.

– Otherwise, the following ordered steps are specified:

 1. The motion vector mvL0N and reference index refIdxL0N are set equal to MvL0[mbPartIdxN][subMbPartIdxN] and RefIdxL0[mbPartIdxN], respectively, which are the motion vector mvL0 and reference index refIdxL0 that have been assigned to the (sub-)macroblock partition mbAddrN\mbPartIdxN\subMbPartIdxN.

 2. The variables mvL0N[1] and refIdxL0N are further processed as follows:

  – Otherwise, the vertical motion vector component mvL0N[1] and the reference index refIdxL0N remain unchanged.

8.4.1.4 Derivation process for chroma motion vectors

Inputs to this process are a luma motion vector mvL0 and a reference index refIdxL0.

Output of this process is a chroma motion vector mvCL0.

A chroma motion vector is derived from the corresponding luma motion vector.

The precision of the chroma motion vector components is 1 ÷ (4 * 2) horizontally and 1 ÷ (4 * 2) vertically.

NOTE – For example, when using the 4:2:0 chroma format, since the units of luma motion vectors are one-quarter luma sample units and chroma has half horizontal and vertical resolution compared to luma, the units of chroma motion vectors are one-eighth chroma sample units, i.e., a value of 1 for the chroma motion vector refers to a one-eighth chroma sample displacement. For example, when the luma vector applies to 8x16 luma samples, the corresponding chroma vector in 4:2:0 chroma format applies to 4x8 chroma samples and when the luma vector applies to 4x4 luma samples, the corresponding chroma vector in 4:2:0 chroma format applies to 2x2 chroma samples.

For the derivation of the motion vector mvCL0, the following applies:

– the horizontal and vertical components of the chroma motion vector mvCL0 are derived as:

mvCL0[0] = mvL0[0] (8-118)
mvCL0[1] = mvL0[1] (8-119)

8.4.2 Decoding process for Inter prediction samples

Inputs to this process are:

– a macroblock partition mbPartIdx,

– a sub-macroblock partition subMbPartIdx,

– variables specifying partition width and height for luma and chroma (if available), partWidth, partHeight, partWidthC (if available) and partHeightC (if available),

– luma motion vectors mvL0 and chroma motion vectors mvCL0,

– reference indices refIdxL0,

– prediction list utilization flags, predFlagL0,

Outputs of this process are the Inter prediction samples predPart, which are a (partWidth)x(partHeight) array $predPart_L$ of prediction luma samples two (partWidthC)x(partHeightC) arrays $predPart_{Cb}$, $predPart_{Cr}$ of prediction chroma samples, one for each of the chroma components Cb and Cr.

Let $predPartL0_L$ be (partWidth)x(partHeight) arrays of predicted luma sample values. Let $predPartL0_{Cb}$, and $predPartL0_{Cr}$, be (partWidthC)x(partHeightC) arrays of predicted chroma sample values.

When predFlagL0 is equal to 1, the following applies:

– The reference picture consisting of an ordered two-dimensional array $refPicL0_L$ of luma samples and two ordered two-dimensional arrays $refPicL0_{Cb}$ and $refPicL0_{Cr}$ of chroma samples is derived by invoking the process specified in subclause 8.4.2.1 with refIdxL0 and RefPicList0 given as input.

– The array $predPartL0_L$ and the arrays $predPartL0_{Cb}$ and $predPartL0_{Cr}$ are derived by invoking the process specified in subclause 8.4.2.2 with the current partition specified by mbPartIdx\subMbPartIdx, the motion vectors mvL0, mvCL0 (if available), and the reference arrays with $refPicL0_L$, $refPicL0_{Cb}$ (if available), and $refPicL0_{Cr}$ (if available) given as input.

For C being replaced by L, Cb (if available), or Cr (if available), the array $predPart_C$ of the prediction samples of component C is derived by

$$predPart_C[x, y] = predPartL0_C[x, y] \quad (8\text{-}120)$$

8.4.2.1 Reference picture selection process

Input to this process is a reference index refIdxL0

Output of this process is a reference picture consisting of a two-dimensional array of luma samples $refPicL0_L$ and two two-dimensional arrays of chroma samples $refPicL0_{Cb}$ and $refPicL0_{Cr}$.

The reference picture list RefPicList0 (which has been derived as specified in subclause 8.2.4) consists of the following.

– each entry of RefPicList0 is a reference frame

For the derivation of the reference picture, the following applies:

– the reference frame RefPicList0[refIdxL0] is the output. The output reference frame consists of a $(PicWidthInSamples_L)$x$(PicHeightInSamples_L)$ array of luma samples $refPicL0_L$ and two $(PicWidthInSamples_C)$x$(PicHeightInSamples_C)$ arrays of chroma samples $refPicL0_{Cb}$ and $refPicL0_{Cr}$.

The following applies:

– the reference picture sample arrays $refPicL0_L$, $refPicL0_{Cb}$, and $refPicL0_{Cr}$ correspond to decoded sample arrays S_L, S_{Cb}, S_{Cr} derived in subclause 8.7 for a previously-decoded reference frame.

8.4.2.2 Fractional sample interpolation process

Inputs to this process are:

– the current partition given by its partition index mbPartIdx and its sub-macroblock partition index subMbPartIdx,

– the width and height partWidth, partHeight of this partition in luma-sample units,

– a luma motion vector mvL0 given in quarter-luma-sample units,

– a chroma motion vector mvCL0 with a precision of one-(4*2)-th chroma-sample units horizontally and one-(4*2)-th chroma-sample units vertically,

– the selected reference picture sample arrays $refPicL0_L$, $refPicL0_{Cb}$, and $refPicL0_{Cr}$.

Outputs of this process are:

– a (partWidth)x(partHeight) array $predPartL0_L$ of prediction luma sample values,

– two (partWidthC)x(partHeightC) arrays $predPartL0_{Cb}$, and $predPartL0_{Cr}$ of prediction chroma sample values.

Let (xA_L, yA_L) be the location given in full-sample units of the upper-left luma sample of the current partition given by mbPartIdx\subMbPartIdx relative to the upper-left luma sample location of the given two-dimensional array of luma samples.

Let ($xInt_L$, $yInt_L$) be a luma location given in full-sample units and ($xFrac_L$, $yFrac_L$) be an offset given in quarter-sample units. These variables are used only inside this subclause for specifying general fractional-sample locations inside the reference sample arrays $refPicL0_L$, $refPicL0_{Cb}$(if available), and $refPicL0_{Cr}$(if available).

For each luma sample location ($0 <= x_L < partWidth$, $0 <= y_L < partHeight$) inside the prediction luma sample array $predPartL0_L$, the corresponding prediction luma sample value $predPartL0_L[x_L, y_L]$ is derived as specified by the following ordered steps:

1. The variables $xInt_L$, $yInt_L$, $xFrac_L$, and $yFrac_L$ are derived by:

$$xInt_L = xA_L + (mvL0[0] >> 2) + x_L \quad (8\text{-}121)$$
$$yInt_L = yA_L + (mvL0[1] >> 2) + y_L \quad (8\text{-}122)$$

$$xFrac_L = mvL0[0] \& 3 \quad (8\text{-}123)$$
$$yFrac_L = mvL0[1] \& 3 \quad (8\text{-}124)$$

2. The prediction luma sample value $predPartL0_L[x_L, y_L]$ is derived by invoking the process specified in subclause 8.4.2.2.1 with ($xInt_L$, $yInt_L$), ($xFrac_L$, $yFrac_L$) and $refPicL0_L$ given as input.

Let ($xInt_C$, $yInt_C$) be a chroma location given in full-sample units and ($xFrac_C$, $yFrac_C$) be an offset given in one-(4*2)-th chroma-sample units horizontally and one-(4*2)-th chroma-sample units vertically. These variables are used only inside this subclause for specifying general fractional-sample locations inside the reference sample arrays $refPicL0_{Cb}$, and $refPicL0_{Cr}$.

For each chroma sample location ($0 <= x_C < partWidthC$, $0 <= y_C < partHeightC$) inside the prediction chroma sample arrays $predPartL0_{Cb}$ and $predPartL0_{Cr}$, the corresponding prediction chroma sample values $predPartL0_{Cb}[x_C, y_C]$ and $predPartL0_{Cr}[x_C, y_C]$ are derived as specified by the following ordered steps:

1. the variables $xInt_C$, $yInt_C$, $xFrac_C$, and $yFrac_C$ are derived as follows:

$$xInt_C = (xA_L / 2) + (mvCL0[0] >> 3) + x_C \quad (8\text{-}125)$$
$$yInt_C = (yA_L / 2) + (mvCL0[1] >> 3) + y_C \quad (8\text{-}126)$$

$$xFrac_C = mvCL0[0] \& 7 \quad (8\text{-}127)$$
$$yFrac_C = mvCL0[1] \& 7 \quad (8\text{-}128)$$

2. the following applies:

   – The prediction sample value $predPartL0_{Cb}[x_C, y_C]$ is derived by invoking the process specified in subclause 8.4.2.2.2 with ($xInt_C$, $yInt_C$), ($xFrac_C$, $yFrac_C$) and $refPicL0_{Cb}$ given as input.

   – The prediction sample value $predPartL0_{Cr}[x_C, y_C]$ is derived by invoking the process specified in subclause 8.4.2.2.2 with ($xInt_C$, $yInt_C$), ($xFrac_C$, $yFrac_C$) and $refPicL0_{Cr}$ given as input.

8.4.2.2.1 Luma sample interpolation process

Inputs to this process are:

– a luma location in full-sample units ($xInt_L$, $yInt_L$),

– a luma location offset in fractional-sample units ($xFrac_L$, $yFrac_L$),

– the luma sample array of the selected reference picture $refPicL0_L$.

Output of this process is a predicted luma sample value $predPartL0_L[x_L, y_L]$.

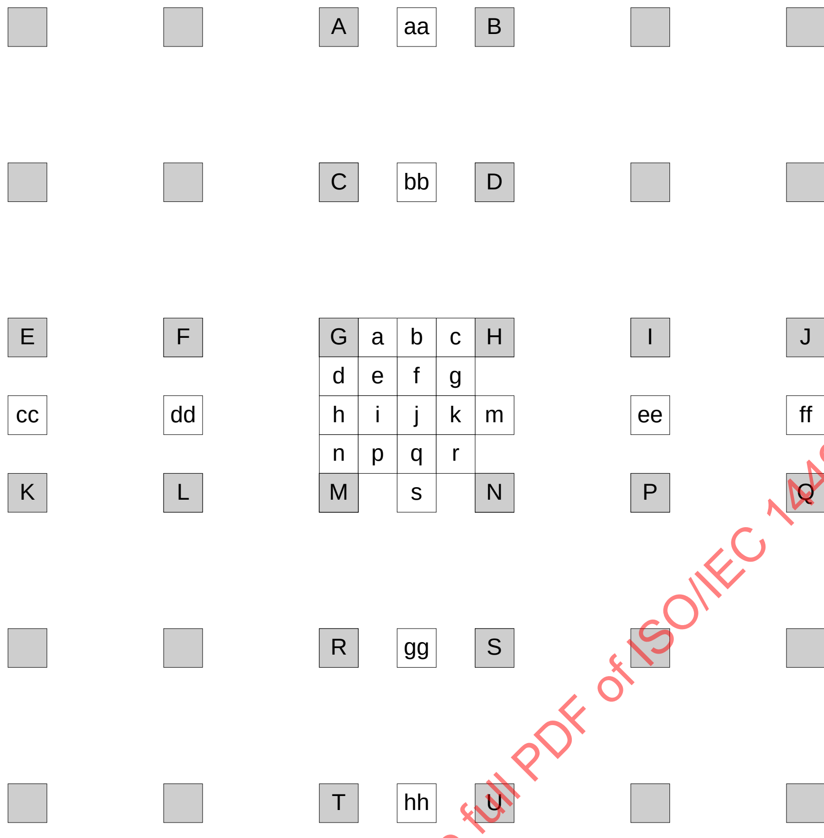

Figure 8-3 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for quarter sample luma interpolation

The variable $refPicHeightEffective_L$, which is the height of the effective reference picture luma array, is derived as follows:

– $refPicHeightEffective_L$ is set equal to $PicHeightInSamples_L$.

In Figure 8-3, the positions labelled with upper-case letters within shaded blocks represent luma samples at full-sample locations inside the given two-dimensional array $refPicL0_L$ of luma samples. These samples may be used for generating the predicted luma sample value $predPartL0_L[\, x_L, y_L \,]$. The locations (xZ_L, yZ_L) for each of the corresponding luma samples Z, where Z may be A, B, C, D, E, F, G, H, I, J, K, L, M, N, P, Q, R, S, T, or U, inside the given array $refPicL0_L$ of luma samples are derived as:

$$xZ_L = \text{Clip3}(\, 0,\ PicWidthInSamples_L - 1,\ xInt_L + xDZ_L \,) \quad (8\text{-}129)$$
$$yZ_L = \text{Clip3}(\, 0,\ refPicHeightEffective_L - 1,\ yInt_L + yDZ_L \,) \quad (8\text{-}130)$$

Table 8-4 specifies (xDZ_L, yDZ_L) for different replacements of Z.

Table 8-4 – Differential full-sample luma locations

Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	P	Q	R	S	T	U
xDZ_L	0	1	0	1	−2	−1	0	1	2	3	−2	−1	0	1	2	3	0	1	0	1
yDZ_L	−2	−2	−1	−1	0	0	0	0	0	0	1	1	1	1	1	1	2	2	3	3

Given the luma samples 'A' to 'U' at full-sample locations (xA_L, yA_L) to (xU_L, yU_L), the luma samples 'a' to 's' at fractional sample positions are derived by the following rules. The luma prediction values at half sample positions are derived by applying a 6-tap filter with tap values (1, −5, 20, 20, −5, 1). The luma prediction values at quarter sample positions are derived by averaging samples at full and half sample positions. The process for each fractional position is described below.

– The samples at half sample positions labelled b are derived by first calculating intermediate values denoted as $b_1$by applying the 6-tap filter to the nearest integer position samples in the horizontal direction. The samples at half sample positions labelled h are derived by first calculating intermediate values denoted as $h_1$by applying the 6-tap filter to the nearest integer position samples in the vertical direction:

$$b_1 = (E - 5 * F + 20 * G + 20 * H - 5 * I + J) \quad (8\text{-}131)$$
$$h_1 = (A - 5 * C + 20 * G + 20 * M - 5 * R + T) \quad (8\text{-}132)$$

The final prediction values b and h are derived using

$$b = \text{Clip1}_Y((b_1 + 16) >> 5) \quad (8\text{-}133)$$
$$h = \text{Clip1}_Y((h_1 + 16) >> 5) \quad (8\text{-}134)$$

– The samples at half sample position labelled as j are derived by first calculating intermediate value denoted as j_1 by applying the 6-tap filter to the intermediate values of the closest half sample positions in either the horizontal or vertical direction because these yield an equal result:

$$j_1 = cc - 5 * dd + 20 * h_1 + 20 * m_1 - 5 * ee + ff\text{, or} \quad (8\text{-}135)$$
$$j_1 = aa - 5 * bb + 20 * b_1 + 20 * s_1 - 5 * gg + hh \quad (8\text{-}136)$$

where intermediate values denoted as aa, bb, gg, s_1 and hh are derived by applying the 6-tap filter horizontally in the same manner as the derivation of b_1 and intermediate values denoted as cc, dd, ee, m_1 and ff are derived by applying the 6-tap filter vertically in the same manner as the derivation of h_1. The final prediction value j is derived using

$$j = \text{Clip1}_Y((j_1 + 512) >> 10) \quad (8\text{-}137)$$

– The final prediction values s and m are derived from s_1 and m_1 in the same manner as the derivation of b and h, as given by

$$s = \text{Clip1}_Y((s_1 + 16) >> 5) \quad (8\text{-}138)$$
$$m = \text{Clip1}_Y((m_1 + 16) >> 5) \quad (8\text{-}139)$$

– The samples at quarter sample positions labelled as a, c, d, n, f, i, k, and q are derived by averaging with upward rounding of the two nearest samples at integer and half sample positions using

$$a = (G + b + 1) >> 1 \quad (8\text{-}140)$$
$$c = (H + b + 1) >> 1 \quad (8\text{-}141)$$
$$d = (G + h + 1) >> 1 \quad (8\text{-}142)$$
$$n = (M + h + 1) >> 1 \quad (8\text{-}143)$$
$$f = (b + j + 1) >> 1 \quad (8\text{-}144)$$
$$i = (h + j + 1) >> 1 \quad (8\text{-}145)$$
$$k = (j + m + 1) >> 1 \quad (8\text{-}146)$$
$$q = (j + s + 1) >> 1 \quad (8\text{-}147)$$

– The samples at quarter sample positions labelled as e, g, p, and r are derived by averaging with upward rounding of the two nearest samples at half sample positions in the diagonal direction using

$e = (b + h + 1) >> 1$ (8-148)
$g = (b + m + 1) >> 1$ (8-149)
$p = (h + s + 1) >> 1$ (8-150)
$r = (m + s + 1) >> 1.$ (8-151)

The luma location offset in fractional-sample units ($xFrac_L$, $yFrac_L$) specifies which of the generated luma samples at full-sample and fractional-sample locations is assigned to the predicted luma sample value $predPartL0_L[x_L, y_L]$. This assignment is done according to Table 8-5. The value of $predPartL0_L[x_L, y_L]$ is the output.

Table 8-5 – Assignment of the luma prediction sample $predPartL0_L[x_L, y_L]$

$xFrac_L$	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
$yFrac_L$	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
$predPartL0_L[x_L, y_L]$	G	d	h	n	a	e	i	p	b	f	j	q	c	g	k	r

8.4.2.2.2 Chroma sample interpolation process

Inputs to this process are:

– a chroma location in full-sample units ($xInt_C$, $yInt_C$),

– a chroma location offset in fractional-sample units ($xFrac_C$, $yFrac_C$),

– chroma component samples from the selected reference picture $refPicL0_C$.

Output of this process is a predicted chroma sample value $predPartL0_C[x_C, y_C]$.

In Figure 8-4, the positions labelled with A, B, C, and D represent chroma samples at full-sample locations inside the given two-dimensional array $refPicL0_C$ of chroma samples.

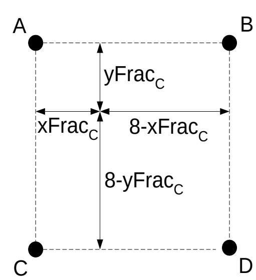

Figure 8-4 – Fractional sample position dependent variables in chroma interpolation and surrounding integer position samples A, B, C, and D

The variable $refPicHeightEffective_C$,which is the height of the effective reference picture chroma array, is derived as follows:

– refPicHeightEffective$_C$ is set equal to PicHeightInSamples$_C$.

The sample coordinates specified in Equations 8-152 through 8-159 are used for generating the predicted chroma sample value predPartL0$_C$[x$_C$, y$_C$].

$$xA_C = \text{Clip3}(\ 0,\ \text{PicWidthInSamples}_C - 1,\ \text{xInt}_C\ ) \quad (8\text{-}152)$$
$$xB_C = \text{Clip3}(\ 0,\ \text{PicWidthInSamples}_C - 1,\ \text{xInt}_C + 1\ ) \quad (8\text{-}153)$$
$$xC_C = \text{Clip3}(\ 0,\ \text{PicWidthInSamples}_C - 1,\ \text{xInt}_C\ ) \quad (8\text{-}154)$$
$$xD_C = \text{Clip3}(\ 0,\ \text{PicWidthInSamples}_C - 1,\ \text{xInt}_C + 1\ ) \quad (8\text{-}155)$$

$$yA_C = \text{Clip3}(\ 0,\ \text{refPicHeightEffective}_C - 1,\ \text{yInt}_C\ ) \quad (8\text{-}156)$$
$$yB_C = \text{Clip3}(\ 0,\ \text{refPicHeightEffective}_C - 1,\ \text{yInt}_C\ ) \quad (8\text{-}157)$$
$$yC_C = \text{Clip3}(\ 0,\ \text{refPicHeightEffective}_C - 1,\ \text{yInt}_C + 1\ ) \quad (8\text{-}158)$$
$$yD_C = \text{Clip3}(\ 0,\ \text{refPicHeightEffective}_C - 1,\ \text{yInt}_C + 1\ ) \quad (8\text{-}159)$$

Given the chroma samples A, B, C, and D at full-sample locations specified in Equations 8-152 through 8-159, the predicted chroma sample value predPartL0$_C$[x$_C$, y$_C$] is derived as:

$$\text{predPartL0}_C[\ x_C,\ y_C\ ] = (\ (\ 8 - \text{xFrac}_C\ ) * (\ 8 - \text{yFrac}_C\ ) * A + \text{xFrac}_C * (\ 8 - \text{yFrac}_C\ ) * B + (\ 8 - \text{xFrac}_C\ ) * \text{yFrac}_C * C + \text{xFrac}_C * \text{yFrac}_C * D + 32\ ) >> 6 \quad (8\text{-}160)$$

8.5 Transform coefficient decoding process and picture construction process prior to deblocking filter process

Inputs to this process are Intra16x16DCLevel (if available), Intra16x16ACLevel (if available), LumaLevel4x4 (if available), LumaLevel8x8 (if available), ChromaDCLevel (if available), ChromaACLevel (if available), CbLevel4x4 (if available), CrLevel4x4 (if available), CbLevel8x8 (if available), CrLevel8x8 (if available), and available Inter or Intra prediction sample arrays for the current macroblock for the applicable components pred$_L$, pred$_{Cb}$, or pred$_{Cr}$.

NOTE – When decoding a macroblock in Intra_4x4 macroblock prediction mode, the luma component of the macroblock prediction array may not be complete, since for each 4x4 luma block, the Intra_4x4 prediction process for luma samples as specified in subclause 8.3.1 and the process specified in this subclause are iterated.

Outputs of this process are the constructed sample arrays prior to the deblocking filter process for the applicable components S'_L, S'_{Cb}, or S'_{Cr}.

NOTE – When decoding a macroblock in Intra_4x4 macroblock prediction mode, the luma component of the macroblock constructed sample arrays prior to the deblocking filter process may not be complete, since for each 4x4 luma block, the Intra_4x4 prediction process for luma samples as specified in subclause 8.3.1 and the process specified in this subclause are iterated.

This subclause specifies transform coefficient decoding and picture construction prior to the deblocking filter process.

When the current macroblock is coded as P_Skip, all values of LumaLevel4x4, LumaLevel8x8, CbLevel4x4, CbLevel8x8, CrLevel4x4, CrLevel8x8, ChromaDCLevel, ChromaACLevel are set equal to 0 for the current macroblock.

8.5.1 Specification of transform decoding process for 4x4 luma residual blocks

When the current macroblock prediction mode is not equal to Intra_16x16, the variable LumaLevel4x4 contains the levels for the luma transform coefficients. For a 4x4 luma block indexed by luma4x4BlkIdx = 0..15, the following ordered steps are specified:

1. The inverse scanning process for 4x4 transform coefficients and scaling lists as specified in subclause 8.5.6 is invoked with LumaLevel4x4[luma4x4BlkIdx] as the input and the two-dimensional array c as the output.

2. The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.12 is invoked with c as the input and r as the output.

3. (void)

4. The position of the upper-left sample of a 4x4 luma block with index luma4x4BlkIdx inside the macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (xO, yO).

5. The 4x4 array u with elements u_{ij} for i, j = 0..3 is derived as:

$$u_{ij} = \text{Clip1}_Y(\text{pred}_L[xO + j, yO + i] + r_{ij}) \quad (8\text{-}161)$$

6. The picture construction process prior to deblocking filter process in subclause 8.5.14 is invoked with u and luma4x4BlkIdx as the inputs.

8.5.2 Specification of transform decoding process for luma samples of Intra_16x16 macroblock prediction mode

When the current macroblock prediction mode is equal to Intra_16x16, the variables Intra16x16DCLevel and Intra16x16ACLevel contain the levels for the luma transform coefficients. The transform coefficient decoding proceeds in the following ordered steps:

1. The 4x4 luma DC transform coefficients of all 4x4 luma blocks of the macroblock are decoded.
   a. The inverse scanning process for 4x4 transform coefficients and scaling lists as specified in subclause 8.5.6 is invoked with Intra16x16DCLevel as the input and the two-dimensional array c as the output.
   b. The scaling and transformation process for luma DC transform coefficients for Intra_16x16 macroblock type as specified in subclause 8.5.10 is invoked with BitDepth_Y, QP'_Y, and c as the input and dcY as the output.

2. The 16x16 array rMb is derived by processing the 4x4 luma blocks indexed by luma4x4BlkIdx = 0..15, and for each 4x4 luma block, the following ordered steps are specified:
   a. The variable lumaList, which is a list of 16 entries, is derived. The first entry of lumaList is the corresponding value from the array dcY. Figure 8-5 shows the assignment of the indices of the array dcY to the luma4x4BlkIdx. The two numbers in the small squares refer to indices i and j in dcY_{ij}, and the numbers in large squares refer to luma4x4BlkIdx.

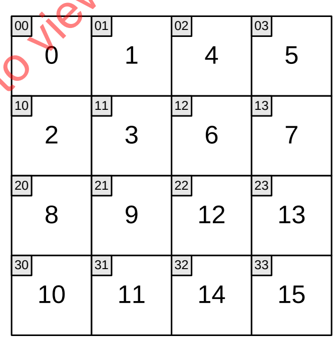

Figure 8-5 – Assignment of the indices of dcY to luma4x4BlkIdx

The elements in lumaList with index k = 1..15 are specified as:

$$\text{lumaList}[k] = \text{Intra16x16ACLevel}[\text{luma4x4BlkIdx}][k - 1] \quad (8\text{-}162)$$

   b. The inverse scanning process for 4x4 transform coefficients and scaling lists as specified in subclause 8.5.6 is invoked with lumaList as the input and the two-dimensional array c as the output.
   c. The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.12 is invoked with c as the input and r as the output.

d. The position of the upper-left sample of a 4x4 luma block with index luma4x4BlkIdx inside the macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (xO, yO).

e. The elements rMb[x, y] of the 16x16 array rMb with x = xO..xO + 3 and y = yO..yO + 3are derived by

$$rMb[\ xO + j,\ yO + i\ ] = r_{ij} \qquad (8\text{-}163)$$

3. (void)

4. The 16x16 array u with elements u_{ij} for i, j = 0..15 is derived as

$$u_{ij} = \mathrm{Clip1}_Y(\ pred_L[j,\ i\ ] + rMb[\ j,\ i\ ]\ ) \qquad (8\text{-}164)$$

5. The picture construction process prior to deblocking filter process in subclause 8.5.14 is invoked with u as the input.

8.5.3 (void)

8.5.4 Specification of transform decoding process for chroma samples

This process is invoked for each chroma component Cb and Cr separately.

For each chroma component, the variables ChromaDCLevel[iCbCr] and ChromaACLevel[iCbCr], with iCbCr set equal to 0 for Cb and iCbCr set equal to 1 for Cr, contain the levels for both components of the chroma transform coefficients.

Let the variable numChroma4x4Blks be set equal to (MbWidthC / 4) * (MbHeightC / 4).

For each chroma component, the transform decoding proceeds separately in the following ordered steps:

1. The numChroma4x4Blks chroma DC transform coefficients of the 4x4 chroma blocks of the component indexed by iCbCr of the macroblock are decoded as specified in the following ordered steps:

   a. The 2x2 array c is derived using the inverse raster scanning process applied to ChromaDCLevel as follows:

$$c = \begin{bmatrix} \mathrm{ChromaDCLevel}[\,iCbCr\,][\,0\,] & \mathrm{ChromaDCLevel}[\,iCbCr\,][\,1\,] \\ \mathrm{ChromaDCLevel}[\,iCbCr\,][\,2\,] & \mathrm{ChromaDCLevel}[\,iCbCr\,][\,3\,] \end{bmatrix} \qquad (8\text{-}165)$$

   b. The scaling and transformation process for chroma DC transform coefficients as specified in subclause 8.5.11 is invoked with c as the input and dcC as the output.

2. The (MbWidthC)x(MbHeightC) array rMb is derived by processing the 4x4 chroma blocks indexed by chroma4x4BlkIdx = 0..numChroma4x4Blks − 1 of the component indexed by iCbCr, and for each 4x4 chroma block, the following ordered steps are specified:

   a. The variable chromaList, which is a list of 16 entries, is derived. The first entry of chromaList is the corresponding value from the array dcC. Figure 8-6 shows the assignment of the indices of the array dcC to the chroma4x4BlkIdx. The two numbers in the small squares refer to indices i and j in dcC_{ij}, and the numbers in large squares refer to chroma4x4BlkIdx.

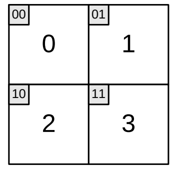

Figure 8-6 – Assignment of the indices of dcC to chroma4x4BlkIdx:

The elements in chromaList with index k = 1..15 are specified as:

$$\text{chromaList}[\,k\,] = \text{ChromaACLevel}[\,\text{chroma4x4BlkIdx}\,][\,k - 1\,] \quad (8\text{-}166)$$

b. The inverse scanning process for 4x4 transform coefficients and scaling lists as specified in subclause 8.5.6 is invoked with chromaList as the input and the two-dimensional array c as the output.

c. The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.12 is invoked with c as the input and r as the output.

d. The position of the upper-left sample of a 4x4 chroma block with index chroma4x4BlkIdx inside the current macroblock is derived by invoking the inverse 4x4 chroma block scanning process as specified in subclause 6.4.7 with chroma4x4BlkIdx as the input and the output being assigned to (xO, yO).

e. The elements rMb[x, y] of the (MbWidthC)x(MbHeightC) array rMb with x = xO..xO + 3 and y = yO..yO + 3 are derived by:

$$\text{rMb}[\,xO + j,\ yO + i\,] = r_{ij} \quad (8\text{-}167)$$

3. (void)

4. The (MbWidthC)x(MbHeightC) array u with elements u_{ij} for i = 0..MbHeightC − 1 and j = 0..MbWidthC − 1 is derived as:

$$u_{ij} = \text{Clip1}_C(\ \text{pred}_C[\,j, i\,] + \text{rMb}[\,j, i\,]\ ) \quad (8\text{-}168)$$

5. The picture construction process prior to deblocking filter process in subclause 8.5.14 is invoked with u as the input.

8.5.5 (void)

8.5.6 Inverse scanning process for 4x4 transform coefficients and scaling lists

Input to this process is a list of 16 values.

Output of this process is a variable c containing a two-dimensional array of 4x4 values. In the case of transform coefficients, these 4x4 values represent levels assigned to locations in the transform block. In the case of applying the inverse scanning process to a scaling list, the output variable c contains a two-dimensional array representing a 4x4 scaling matrix.

When this subclause is invoked with a list of transform coefficient levels as the input, the sequence of transform coefficient levels is mapped to the transform coefficient level positions. Table 8-6 specifies the mapping: inverse zig-zag scan. The inverse zig-zag scan is used for transform coefficients in frame macroblocks.

When this subclause is invoked with a scaling list as the input, the sequence of scaling list entries is mapped to the positions in the corresponding scaling matrix. For this mapping, the inverse zig-zag scan is used.

Figure 8-7 illustrates the scan.

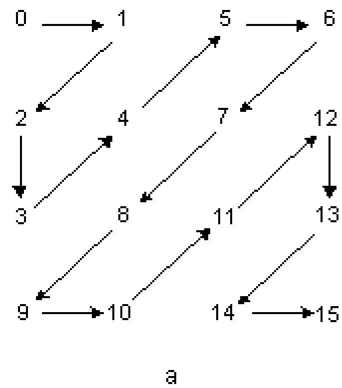

Figure 8-7 – 4x4 block scan: the zig-zag scan.

Table 8-6 provides the mapping from the index idx of input list of 16 elements to indices i and j of the two-dimensional array c.

Table 8-6 – Specification of mapping of idx to c_{ij} for zig-zag

idx	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
zig-zag	c_{00}	c_{01}	c_{10}	c_{20}	c_{11}	c_{02}	c_{03}	c_{12}	c_{21}	c_{30}	c_{31}	c_{22}	c_{13}	c_{23}	c_{32}	c_{33}

8.5.7 (void)

8.5.8 Derivation process for chroma quantisation parameters

Outputs of this process are:

– QP_C: the chroma quantisation parameter for each chroma component Cb and Cr,

NOTE – QP quantisation parameter values QP_Y and QS_Y are always in the range of −0 to 51, inclusive. QP quantisation parameter values QP_C and QS_C are always in the range of −0 to 39, inclusive.

The value of QP_C for a chroma component is determined from the current value of QP_Y and the value of chroma_qp_index_offset.

NOTE – The scaling equations are specified such that the equivalent transform coefficient level scaling factor doubles for every increment of 6 in QP_Y. Thus, there is an increase in the factor used for scaling of approximately 12 % for each increase of 1 in the value of QP_Y.

The value of QP_C for each chroma component is determined as specified in Table 8-7 based on the index denoted as qP_I.

The variable qP_{Offset} for each chroma component is derived as follows:

– If the chroma component is the Cb component, qP_{Offset} is specified as:

$$qP_{Offset} = \text{chroma\_qp\_index\_offset} \quad (8\text{-}169)$$

– Otherwise (the chroma component is the Cr component), qP_{Offset} is specified as:

$$qP_{Offset} = \text{chroma\_qp\_index\_offset} \quad (8\text{-}170)$$

The value of qP_I for each chroma component is derived as:

$$qP_I = \text{Clip3}(\ -0,\ 51,\ QP_Y + qP_{Offset}\ ) \quad (8\text{-}171)$$

The value of QP'_C for the chroma components is derived as:

$$QP'_C = QP_C + 0 \quad (8\text{-}172)$$

Table 8-7 – Specification of QP_C as a function of qP_I

qP_I	<30	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
QP_C	$= qP_I$	29	30	31	32	32	33	34	34	35	35	36	36	37	37	37	38	38	38	39	39	39	39

8.5.9 Derivation process for scaling functions

Outputs of this process are:

– LevelScale4x4: the scaling factor for 4x4 block transform luma or chroma coefficient levels.

LevelScale4x4(m, i, j) is specified by:

$$\text{LevelScale4x4}(\ m,\ i,\ j) = 16 * \text{normAdjust4x4}(\ m,\ i,\ j\ ) \quad (8\text{-}173)$$

where

$$\text{normAdjust4x4}(m,i,j) = \begin{cases} v_{m0} & \text{for } (\ i \,\%\, 2,\ j \,\%\, 2\ ) \text{ equal to } (0,0), \\ v_{m1} & \text{for } (\ i \,\%\, 2,\ j \,\%\, 2\ ) \text{ equal to } (1,1), \\ v_{m2} & \text{otherwise;} \end{cases} \quad (8\text{-}174)$$

where the first and second subscripts of v are row and column indices, respectively, of the matrix specified as:

$$v = \begin{bmatrix} 10 & 16 & 13 \\ 11 & 18 & 14 \\ 13 & 20 & 16 \\ 14 & 23 & 18 \\ 16 & 25 & 20 \\ 18 & 29 & 23 \end{bmatrix}. \quad (8\text{-}175)$$

8.5.10 Scaling and transformation process for DC transform coefficients for Intra_16x16 macroblock type

Inputs to this process are:

– the variables bitDepth and qP,

– transform coefficient level values for DC transform coefficients of Intra_16x16 macroblocks as a 4x4 array c with elements c_{ij}, where i and j form a two-dimensional frequency index.

Outputs of this process are 16 scaled DC values for 4x4 blocks of Intra_16x16 macroblocks as a 4x4 array dcY with elements dcY_{ij}.

The following applies:

– the following text of this process specifies the output.

The inverse transform for the 4x4 luma DC transform coefficients is specified by:

$$f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} * \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}. \tag{8-176}$$

The bitstream shall not contain data that result in any element f_{ij} of f with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

After the inverse transform, the scaling is performed as follows:

– If qP is greater than or equal to 36, the scaled result is derived as:

$$dcY_{ij} = (f_{ij} * \text{LevelScale4x4}(qP \% 6, 0, 0)) << (qP / 6 - 6), \quad \text{with } i, j = 0…3 \tag{8-177}$$

– Otherwise (qP is less than 36), the scaled result is derived as:

$$dcY_{ij} = (f_{ij} * \text{LevelScale4x4}(qP \% 6, 0, 0) + (1 << (5 - qP / 6))) >> (6 - qP / 6), \quad \text{with } i, j = 0…3 \tag{8-178}$$

The bitstream shall not contain data that result in any element dcY_{ij} of dcY with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

NOTE – When entropy_coding_mode_flag is equal to 0 and qP is less than 10 and profile_idc is equal to 66, 77, or 88, the range of values that can be represented for the elements c_{ij} of c is not sufficient to represent the full range of values of the elements dcY_{ij} of dcY that could be necessary to form a close approximation of the content of any possible source picture by use of the Intra_16x16 macroblock type.

NOTE – Since the range limit imposed on the elements dcY_{ij} of dcY is imposed after the right shift in Equation 8-178, a larger range of values must be supported in the decoder prior to the right shift.

8.5.11 Scaling and transformation process for chroma DC transform coefficients

Inputs to this process are transform coefficient level values for chroma DC transform coefficients of one chroma component of the macroblock as an (MbWidthC / 4)x(MbHeightC / 4) array c with elements c_{ij}, where i and j form a two-dimensional frequency index.

Outputs of this process are the scaled DC values as an (MbWidthC / 4)x(MbHeightC / 4) array dcC with elements dcC_{ij}.

The variables bitDepth and qP are set equal to BitDepth_C and QP'_C, respectively.

The following ordered steps are specified:

1. The transformation process for chroma DC transform coefficients as specified in subclause 8.5.11.1 is invoked with bitDepth and c as the inputs and the output is assigned to the (MbWidthC / 4)x(MbHeightC / 4) array f of chroma DC values with elements f_{ij}.

2. The scaling process for chroma DC transform coefficients as specified in subclause 8.5.11.2 is invoked with bitDepth, qP, and f as the inputs and the output is assigned to the (MbWidthC / 4)x(MbHeightC / 4) array dcC of scaled chroma DC values with elements dcC_{ij}.

8.5.11.1 Transformation process for chroma DC transform coefficients

Inputs of this process are transform coefficient level values for chroma DC transform coefficients of one chroma component of the macroblock as an (MbWidthC / 4)x(MbHeightC / 4) array c with elements c_{ij}, where i and j form a two-dimensional frequency index.

Outputs of this process are the DC values as an (MbWidthC / 4)x(MbHeightC / 4) array f with elements f_{ij}.

The inverse transform is specified as follows:

– the inverse transform for the 2x2 chroma DC transform coefficients is specified as:

$$f = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} * \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (8\text{-}179)$$

8.5.11.2 Scaling process for chroma DC transform coefficients

Inputs of this process are:

– the variables bitDepth and qP,

– DC values as an (MbWidthC / 4)x(MbHeightC / 4) array f with elements f_{ij}.

Outputs of this process are scaled DC values as an (MbWidthC / 4)x(MbHeightC / 4) array dcC with elements dcC_{ij}.

The bitstream shall not contain data that result in any element f_{ij} of f with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

Scaling is performed as follows:

– the scaled result is derived as:

$$dcC_{ij} = ((f_{ij} * \text{LevelScale4x4}(qP \% 6, 0, 0)) << (qP / 6)) >> 5, \quad \textit{with} \quad i, j = 0, 1 \quad (8\text{-}180)$$

The bitstream shall not contain data that result in any element dcC_{ij} of dcC with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

NOTE – When qP is less than 4 and profile_idc is equal to 66, 77, or 88, the range of values that can be represented for the elements c_{ij} of c in subclause 8.5.11.1 may not be sufficient to represent the full range of values of the elements dcC_{ij} of dcC that could be necessary to form a close approximation of the content of any possible source picture.

NOTE – Since the range limit imposed on the elements dcC_{ij} of dcC is imposed after the right shift in Equation 8-180, a larger range of values must be supported in the decoder prior to the right shift.

8.5.12 Scaling and transformation process for residual 4x4 blocks

Input to this process is a 4x4 array c with elements c_{ij} which is either an array relating to a residual block of the luma component or an array relating to a residual block of a chroma component.

Outputs of this process are residual sample values as 4x4 array r with elements r_{ij}.

The variable bitDepth is derived as follows:

– If the input array c relates to a luma residual block, bitDepth is set equal to BitDepth_Y.

– Otherwise (the input array c relates to a chroma residual block), bitDepth is set equal to BitDepth_C.

The variable sMbFlag is derived as follows:

– sMbFlag is set equal to 0.

The variable qP is derived as follows:

– If the input array c relates to a luma residual block and sMbFlag is equal to 0,

$$qP = QP'_Y \quad (8\text{-}181)$$

– Otherwise, if the input array c relates to a luma residual block and sMbFlag is equal to 1,

$$qP = QS_Y \quad (8\text{-}182)$$

– Otherwise, if the input array c relates to a chroma residual block and sMbFlag is equal to 0,

$$qP = QP'_C \quad (8\text{-}183)$$

– Otherwise (the input array c relates to a chroma residual block and sMbFlag is equal to 1),

$$qP = QS_C \quad (8\text{-}184)$$

The following applies:

– the following ordered steps are specified:

1. The scaling process for residual 4x4 blocks as specified in subclause 8.5.12.1 is invoked with bitDepth, qP, and c as the inputs and the output is assigned to the 4x4 array d of scaled transform coefficients with elements d_{ij}.
2. The transformation process for residual 4x4 blocks as specified in subclause 8.5.12.2 is invoked with bitDepth and d as the inputs and the output is assigned to the 4x4 array r of residual sample values with elements r_{ij}.

8.5.12.1 Scaling process for residual 4x4 blocks

Inputs of this process are:

– the variables bitDepth and qP,

– a 4x4 array c with elements c_{ij} which is either an array relating to a residual block of luma component or an array relating to a residual block of a chroma component.

Output of this process is a 4x4 array of scaled transform coefficients d with elements d_{ij}.

The bitstream shall not contain data that result in any element c_{ij} of c with i, j = 0..3 that exceeds the range of integer values from $-2^{(7\,+\,\mathrm{bitDepth})}$ to $2^{(7\,+\,\mathrm{bitDepth})} - 1$, inclusive.

Scaling of 4x4 block transform coefficient levels c_{ij} proceeds as follows:

– If all of the following conditions are true:

  – i is equal to 0,

  – j is equal to 0,

  – c relates to a luma residual block coded using Intra_16x16 macroblock prediction mode or c relates to a chroma residualblock.

  the variable d_{00} is derived by

$$d_{00} = c_{00} \qquad (8\text{-}185)$$

– Otherwise, the following applies:

  – If qP is greater than or equal to 24, the scaled result is derived as

$$d_{ij} = (c_{ij} * \mathrm{LevelScale4x4}(qP\%6,i,j)) << (qP / 6 - 4), \text{ with } i,j = 0..3 \text{ except as noted above} \qquad (8\text{-}186)$$

  – Otherwise (qP is less than 24), the scaled result is derived as

$$d_{ij} = (c_{ij} * \mathrm{LevelScale4x4}(qP \% 6, i, j) + 2^{3-qP/6}) >> (4 - qP / 6), \text{ with } i, j = 0..3 \text{ except as noted above} \qquad (8\text{-}187)$$

The bitstream shall not contain data that result in any element d_{ij} of d with i, j = 0..3 that exceeds the range of integer values from $-2^{(7\,+\,\mathrm{bitDepth})}$ to $2^{(7\,+\,\mathrm{bitDepth})} - 1$, inclusive.

8.5.12.2 Transformation process for residual 4x4 blocks

Inputs of this process are:

– the variable bitDepth,

– a 4x4 array of scaled transform coefficients d with elements d_{ij}.

Outputs of this process are residual sample values as 4x4 array r with elements r_{ij}.

The bitstream shall not contain data that result in any element d_{ij} of d with i, j = 0..3 that exceeds the range of integer values from $-2^{(7\,+\,\mathrm{bitDepth})}$ to $2^{(7\,+\,\mathrm{bitDepth})} - 1$, inclusive.

The transform process shall convert the block of scaled transform coefficients to a block of output samples in a manner mathematically equivalent to the following.

First, each (horizontal) row of scaled transform coefficients is transformed using a one-dimensional inverse transform as follows.

A set of intermediate values is computed as follows:

$$e_{i0} = d_{i0} + d_{i2}, \quad \text{with} \quad i = 0..3 \tag{8-188}$$

$$e_{i1} = d_{i0} - d_{i2}, \quad \text{with} \quad i = 0..3 \tag{8-189}$$

$$e_{i2} = (d_{i1} >> 1) - d_{i3}, \quad \text{with} \quad i = 0..3 \tag{8-190}$$

$$e_{i3} = d_{i1} + (d_{i3} >> 1), \quad \text{with} \quad i = 0..3 \tag{8-191}$$

The bitstream shall not contain data that result in any element e_{ij} of e with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

Then, the transformed result is computed from these intermediate values as follows:

$$f_{i0} = e_{i0} + e_{i3}, \quad \text{with} \quad i = 0..3 \tag{8-192}$$

$$f_{i1} = e_{i1} + e_{i2}, \quad \text{with} \quad i = 0..3 \tag{8-193}$$

$$f_{i2} = e_{i1} - e_{i2}, \quad \text{with} \quad i = 0..3 \tag{8-194}$$

$$f_{i3} = e_{i0} - e_{i3}, \quad \text{with} \quad i = 0..3 \tag{8-195}$$

The bitstream shall not contain data that result in any element f_{ij} of f with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

Then, each (vertical) column of the resulting matrix is transformed using the same one-dimensional inverse transform as follows.

A set of intermediate values is computed as follows:

$$g_{0j} = f_{0j} + f_{2j}, \quad \text{with} \quad j = 0..3 \tag{8-196}$$

$$g_{1j} = f_{0j} - f_{2j}, \quad \text{with} \quad j = 0..3 \tag{8-197}$$

$$g_{2j} = (f_{1j} >> 1) - f_{3j}, \quad \text{with} \quad j = 0..3 \tag{8-198}$$

$$g_{3j} = f_{1j} + (f_{3j} >> 1), \quad \text{with} \quad j = 0..3 \tag{8-199}$$

The bitstream shall not contain data that result in any element g_{ij} of g with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

Then, the transformed result is computed from these intermediate values as follows:

$$h_{0j} = g_{0j} + g_{3j}, \quad \text{with} \quad j = 0..3 \tag{8-200}$$

$$h_{1j} = g_{1j} + g_{2j}, \quad \text{with} \quad j = 0..3 \tag{8-201}$$

$$h_{2j} = g_{1j} - g_{2j}, \quad \text{with} \quad j = 0..3 \tag{8-202}$$

$$h_{3j} = g_{0j} - g_{3j}, \quad \text{with} \quad j = 0..3 \tag{8-203}$$

The bitstream shall not contain data that result in any element h_{ij} of h with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 33$, inclusive.

After performing both the one-dimensional horizontal and the one-dimensional vertical inverse transforms to produce an array of transformed samples, the final constructed residual sample values is derived as:

$$r_{ij} = (h_{ij} + 2^5) >> 6 \quad \text{with} \quad i, j = 0..3 \tag{8-204}$$

8.5.13 (void)

8.5.14 Picture construction process prior to deblocking filter process

Inputs to this process are:

– a sample array u with elements u_{ij} which is a 16x16 luma block or an (MbWidthC)x(MbHeightC) chroma block or a 4x4 luma block or a 4x4 chroma block,

– when u is not a 16x16 luma block or an (MbWidthC)x(MbHeightC) chroma block, a block index luma4x4BlkIdx or chroma4x4BlkIdx.

The position of the upper-left luma sample of the current macroblock is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with CurrMbAddr as input and the output being assigned to (xP, yP).

When u is a luma block, for each sample u_{ij} of the luma block, the following ordered steps are specified:

1. Depending on the size of the block u, the following applies:

   – If u is a 16x16 luma block, the position (xO, yO) of the upper-left sample of the 16x16 luma block inside the macroblock is set equal to (0, 0) and the variable nE is set equal to 16.

   – Otherwise, if u is an 4x4 luma block, the position of the upper-left sample of the 4x4 luma block with index luma4x4BlkIdx inside the macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (xO, yO), and the variable nE is set equal to 4.

2. The following applies:

$$S'_L[xP + xO + j, yP + yO + i] = u_{ij} \quad \text{with } i, j = 0..nE - 1 \tag{8-205}$$

When u is a chroma block, for each sample u_{ij} of the chroma block, the following ordered steps are specified:

1. The subscript C in the variable S'_C is replaced with Cb for the Cb chroma component and with Cr for the Cr chroma component.

2. Depending on the size of the block u, the following applies:

   – If u is an (MbWidthC)x(MbHeightC) Cb or Cr block, the variable nW is set equal to MbWidthC, the variable nH is set equal to MbHeightC, and the position (xO, yO) of the upper-left sample of the (nW)x(nH) Cb or Cr block inside the macroblock is set equal to (0, 0).

   – Otherwise, if u is a 4x4 Cb or Cr block, the variables nW and nH are set equal to 4, and the position of the upper-left sample of a 4x4 Cb or Cr block with index chroma4x4BlkIdx inside the macroblock is derived as follows:

     – the position of the upper-left sample of the 4x4 chroma block with index chroma4x4BlkIdx inside the macroblock is derived by invoking the inverse 4x4 chroma block scanning process in subclause 6.4.7 with chroma4x4BlkIdx as the input and the output being assigned to (xO, yO).

3. the following applies:

$$S'_C[(xP/ 2) + xO + j, (yP / 2) + yO + i] = u_{ij} \quad \text{with } i = 0..nH - 1 \text{ and } j = 0..nW - 1 \tag{8-206}$$

8.6 (void)

8.7 Deblocking filter process

A conditional filtering process is specified in this subclause that is an integral part of the decoding process which shall be applied by decoders.

The conditional filtering process is applied to all 4x4 block edges of a picture, except edges at the boundary of the picture and any edges for which the deblocking filter process is disabled by disable_deblocking_filter_idc, as specified below. This filtering process is performed on a macroblock basis after the completion of the picture construction process prior to deblocking filter process (as specified in subclauses 8.5) for the entire decoded picture, with all macroblocks in a picture processed in order of increasing macroblock addresses.

NOTE – Prior to the operation of the deblocking filter process for each macroblock, the deblocked samples of the macroblock above (if any) and the macroblock to the left (if any) of the current macroblock are always available because the deblocking filter process is performed after the completion of the picture construction process prior to deblocking filter process for the entire decoded picture. However, for purposes of determining which edges are to be filtered when disable_deblocking_filter_idc is equal to 2, macroblocks in different slices are considered not available during specified steps of the operation of the deblocking filter process.

The deblocking filter process is invoked for the luma and chroma components separately. For each macroblock and each component, vertical edges are filtered first, starting with the edge on the left-hand side of the macroblock proceeding through the edges towards the right-hand side of the macroblock in their geometrical order, and then horizontal edges are filtered, starting with the edge on the top of the macroblock proceeding through the edges towards the bottom of the macroblock in their geometrical order. Figure 8-8 shows edges of a macroblock which can be interpreted as luma or chroma edges.

When interpreting the edges in Figure 8-8 as luma edges, the following applies:

– both types, the solid bold and dashed bold luma edges are filtered.

When interpreting the edges in Figure 8-8 as chroma edges, the following applies:

– only the solid bold chroma edges are filtered.

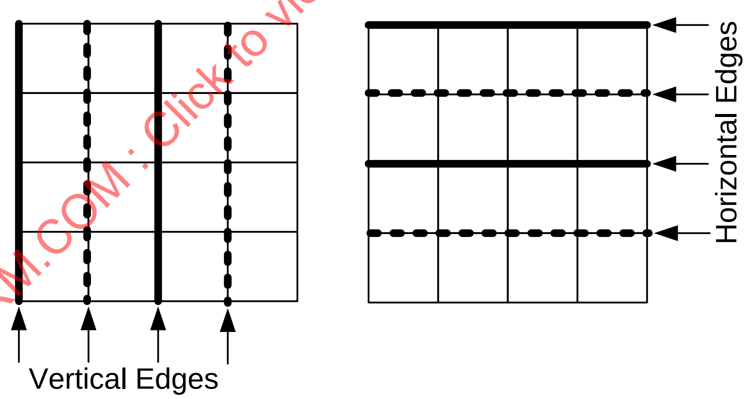

Figure 8-8 – Boundaries in a macroblock to be filtered

For the current macroblock address CurrMbAddr proceeding over values 0..PicSizeInMbs − 1, the following ordered steps are specified:

1. The derivation process for neighbouring macroblocks specified in subclause 6.4.11.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

2. The variables filterInternalEdgesFlag, filterLeftMbEdgeFlag and filterTopMbEdgeFlag are derived as specified by the following ordered steps:

   a. The variable filterInternalEdgesFlag is derived as follows:

      – If disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 1, the variable filterInternalEdgesFlag is set equal to 0.

      – Otherwise (disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is not equal to 1), the variable filterInternalEdgesFlag is set equal to 1.

   b. The variable filterLeftMbEdgeFlag is derived as follows:

      – If any of the following conditions are true, the variable filterLeftMbEdgeFlag is set equal to 0:

        – CurrMbAddr % PicWidthInMbs is equal to 0,

        – disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 1,

        – disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 2 and the macroblock mbAddrA is not available.

      – Otherwise, the variable filterLeftMbEdgeFlag is set equal to 1.

   c. The variable filterTopMbEdgeFlag is derived as follows:

      – If any of the following conditions are true, the variable filterTopMbEdgeFlag is set equal to 0:

        – CurrMbAddr is less than PicWidthInMbs,

        – disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 1,

        – disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 2 and the macroblock mbAddrB is not available.

      – Otherwise, the variable filterTopMbEdgeFlag is set equal to 1.

3. Given the variables filterInternalEdgesFlag, filterLeftMbEdgeFlag and filterTopMbEdgeFlag the deblocking filtering is controlled as follows:

   a. When filterLeftMbEdgeFlag is equal to 1, the left vertical luma edge is filtered by invoking the process specified in subclause 8.7.1 with chromaEdgeFlag = 0, verticalEdgeFlag = 1, and $(xE_k, yE_k) = (0, k)$ with $k = 0..15$ as the inputs and S'_L as the output.

   b. When filterInternalEdgesFlag is equal to 1, the filtering of the internal vertical luma edges is specified by the following ordered steps:

      i. the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 1, and $(xE_k, yE_k) = (4, k)$ with $k = 0..15$ as the inputs and S'_L as the output.

      ii. The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 1, and $(xE_k, yE_k) = (8, k)$ with $k = 0..15$ as the inputs and S'_L as the output.

      iii. the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 1, and $(xE_k, yE_k) = (12, k)$ with $k = 0..15$ as the inputs and S'_L as the output.

   c. When filterTopMbEdgeFlag is equal to 1, the filtering of the top horizontal luma edge is specified as follows:

      – Otherwise, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, and $(xE_k, yE_k) = (k, 0)$ with $k = 0..15$ as the inputs and S'_L as the output.

   d. When filterInternalEdgesFlag is equal to 1, the filtering of the internal horizontal luma edges is specified by the following ordered steps:

i. the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, and $(xE_k, yE_k) = (k, 4)$ with k = 0..15 as the inputs and S'_L as the output.

ii. The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, and $(xE_k, yE_k) = (k, 8)$ with k = 0..15 as the inputs and S'_L as the output.

iii. the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, and $(xE_k, yE_k) = (k, 12)$ with k = 0..15 as the inputs and S'_L as the output.

e. When filtering of both chroma components, with iCbCr = 0 for Cb and iCbCr = 1 for Cr, the following ordered steps are specified:

i. When filterLeftMbEdgeFlag is equal to 1, the left vertical chroma edge is filtered by invoking the process specified in subclause 8.7.1 with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 1, and $(xE_k, yE_k) = (0, k)$ with k = 0..MbHeightC − 1 as the inputs and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as the output.

ii. When filterInternalEdgesFlag is equal to 1, the filtering of the internal vertical chroma edge is specified by the following ordered steps:

(1) the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 1, and $(xE_k, yE_k) = (4, k)$ with k = 0..MbHeightC − 1 as the inputs and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as the output.

iii. When filterTopMbEdgeFlag is equal to 1, the filtering of the top horizontal chroma edge is specified as follows:

– the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, and $(xE_k, yE_k) = (k, 0)$ with k = 0..MbWidthC − 1 as the inputs and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as the output.

iv. When filterInternalEdgesFlag is equal to 1, the filtering of the internal horizontal chroma edge is specified by the following ordered steps:

(1) the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, and $(xE_k, yE_k) = (k, 4)$ with k = 0..MbWidthC − 1 as the inputs and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as the output.

NOTE – For example, in 4:2:0 chroma format, the following applies: 3 horizontal luma edges, 1 horizontal chroma edge for Cb, and 1 horizontal chroma edge for Cr are filtered that are internal to a macroblock.

The arrays S'_L, S'_{Cb}, S'_{Cr} are assigned to the arrays S_L, S_{Cb}, S_{Cr} (which represent the decoded picture), respectively.

8.7.1 Filtering process for block edges

Inputs to this process are chromaEdgeFlag, the chroma component index iCbCr (when chromaEdgeFlag is equal to 1), verticalEdgeFlag, and a set of nE sample locations (xE_k, yE_k), with k = 0..nE − 1, expressed relative to the upper left corner of the macroblock CurrMbAddr. The set of sample locations (xE_k, yE_k) represent the sample locations immediately to the right of a vertical edge (when verticalEdgeFlag is equal to 1) or immediately below a horizontal edge (when verticalEdgeFlag is equal to 0).

The variable nE is derived as follows:

– If chromaEdgeFlag is equal to 0, nE is set equal to 16.

– Otherwise (chromaEdgeFlag is equal to 1), nE is set equal to 8.

Let s′ be a variable specifying a luma or chroma sample array. s′ is derived as follows:

– If chromaEdgeFlag is equal to 0, s′ represents the luma sample array S'_L of the current picture.

– Otherwise, if chromaEdgeFlag is equal to 1 and iCbCr is equal to 0, s′ represents the chroma sample array S'_{Cb} of the chroma component Cb of the current picture.

– Otherwise (chromaEdgeFlag is equal to 1 and iCbCr is equal to 1), s′ represents the chroma sample array S'_{Cr} of the chroma component Cr of the current picture.

The position of the upper-left luma sample of the macroblock CurrMbAddr is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddr = CurrMbAddr as input and the output being assigned to (xI, yI).

The variables xP and yP are derived as follows:

– If chromaEdgeFlag is equal to 0, xP is set equal to xI and yP is set equal to yI.

– Otherwise (chromaEdgeFlag is equal to 1), xP is set equal to xI / 2 and yP is set equal to (yI + 2− 1) / 2.

p_3	p_2	p_1	p_0	q_0	q_1	q_2	q_3

Figure 8-9 – Convention for describing samples across a 4x4 block horizontal or vertical boundary

For each sample location (xE_k, yE_k), k = 0..(nE − 1), the following ordered steps are specified:

1. The filtering process is applied to a set of eight samples across a 4x4 block horizontal or vertical edge denoted as p_i and q_i with i = 0..3 as shown in Figure 8-9 with the edge lying between p_0 and q_0. p_i and q_i with i = 0..3 are specified as follows:

   – If verticalEdgeFlag is equal to 1,

   $$q_i = s'[\ xP + xE_k + i,\ yP + yE_k\ ] \quad (8\text{-}207)$$

   $$p_i = s'[\ xP + xE_k - i - 1,\ yP + yE_k\ ] \quad (8\text{-}208)$$

   – Otherwise (verticalEdgeFlag is equal to 0),

   $$q_i = s'[\ xP + xE_k,\ yP + yE_k + i\ ] \quad (8\text{-}209)$$

   $$p_i = s'[\ xP + xE_k,\ yP + yE_k - i - 1\ ] \quad (8\text{-}210)$$

2. The process specified in subclause 8.7.2 is invoked with the sample values p_i and q_i (i = 0..3), chromaEdgeFlag, and verticalEdgeFlag as the inputs, and the output is assigned to the filtered result sample values p'_i and q'_i with i = 0..2.

3. The input sample values p_i and q_i with i = 0..2 are replaced by the corresponding filtered result sample values p'_i and q'_i with i = 0..2 inside the sample array s′ as follows:

   – If verticalEdgeFlag is equal to 1,

   $$s'[\ xP + xE_k + i,\ yP + yE_k\ ] = q'_i \quad (8\text{-}211)$$

   $$s'[\ xP + xE_k - i - 1,\ yP + yE_k\ ] = p'_i \quad (8\text{-}212)$$

   – Otherwise (verticalEdgeFlag is equal to 0),

   $$s'[\ xP + xE_k,\ yP + yE_k + i\ ] = q'_i \quad (8\text{-}213)$$

   $$s'[\ xP + xE_k,\ yP + yE_k - i - 1\ ] = p'_i \quad (8\text{-}214)$$

8.7.2 Filtering process for a set of samples across a horizontal or vertical block edge

Inputs to this process are the input sample values p_i and q_i with i in the range of 0..3 of a single set of samples across an edge that is to be filtered, chromaEdgeFlag, and verticalEdgeFlag.

Outputs of this process are the filtered result sample values p'_i and q'_i with i in the range of 0..2.

The content dependent boundary filtering strength variable bS is derived as follows:

– If chromaEdgeFlag is equal to 0, the derivation process for the content dependent boundary filtering strength specified in subclause 8.7.2.1 is invoked with p_0, q_0, and verticalEdgeFlag as input, and the output is assigned to bS.

– Otherwise (chromaEdgeFlag is equal to 1), the bS used for filtering a set of samples of a horizontal or vertical chroma edge is set equal to the value of bS for filtering the set of samples of a horizontal or vertical luma edge, respectively, that contains the luma sample at location (2 * x, 2* y) inside the luma array of the frame, where (x, y) is the location of the chroma sample q_0 inside the chroma array for that frame.

Let filterOffsetA and filterOffsetB be the values of FilterOffsetA and FilterOffsetB as specified in subclause 7.4.3 for the slice that contains the macroblock containing sample q_0.

Let qP_p and qP_q be variables specifying quantisation parameter values for the macroblocks containing the samples p_0 and q_0, respectively. The variables qP_z (with z being replaced by p or q) are derived as follows:

– If chromaEdgeFlag is equal to 0, the following applies:

  – If the macroblock containing the sample z_0 is an I_PCM macroblock, qP_z is set to 0.

  – Otherwise (the macroblock containing the sample z_0 is not an I_PCM macroblock), qP_z is set to the value of QP_Y of the macroblock containing the sample z_0.

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

  – If the macroblock containing the sample z_0 is an I_PCM macroblock, qP_z is set equal to the value of QP_C that corresponds to a value of 0 for QP_Y as specified in subclause 8.5.8.

  – Otherwise (the macroblock containing the sample z_0 is not an I_PCM macroblock), qP_z is set equal to the value of QP_C that corresponds to the value QP_Y of the macroblock containing the sample z_0 as specified in subclause 8.5.8.

The process specified in subclause 8.7.2.2 is invoked with p_0, q_0, p_1, q_1, chromaEdgeFlag, bS, filterOffsetA, filterOffsetB, qP_p, and qP_q as inputs, and the outputs are assigned to filterSamplesFlag, indexA, α, and β.

Depending on the variable filterSamplesFlag, the following applies:

– If filterSamplesFlag is equal to 1, the following applies:

  – If bS is less than 4, the process specified in subclause 8.7.2.3 is invoked with p_i and q_i (i = 0..2), chromaEdgeFlag, bS, β, and indexA given as input, and the output is assigned to p'_i and q'_i (i = 0..2).

  – Otherwise (bS is equal to 4), the process specified in subclause 8.7.2.4 is invoked with p_i and q_i (i = 0..3), chromaEdgeFlag, α, and β given as input, and the output is assigned to p'_i and q'_i (i = 0..2).

– Otherwise (filterSamplesFlag is equal to 0), the filtered result samples p'_i and q'_i (i = 0..2) are replaced by the corresponding input samples p_i and q_i:

$$\text{for } i = 0..2, \quad p'_i = p_i \tag{8-215}$$

$$\text{for } i = 0..2, \quad q'_i = q_i \tag{8-216}$$

8.7.2.1 Derivation process for the luma content dependent boundary filtering strength

Inputs to this process are the input sample values p_0 and q_0 of a single set of samples across an edge that is to be filtered and verticalEdgeFlag.

Output of this process is the variable bS.

The variable bS is derived as follows:

– If the block edge is also a macroblock edge and the following condition is true, a value of bS equal to 4 is the output:

– either or both of the samples p_0 or q_0 is in a macroblock coded using an Intra macroblock prediction mode,

Otherwise, if the following condition is true, a value of bS equal to 3 is the output:

– either or both of the samples p_0 or q_0 is in a macroblock coded using an Intra macroblock prediction mode,

– Otherwise, if any of the following conditions are true, a value of bS equal to 2 is the output:

  – the 4x4 luma transform block associated with the 4x4 luma block containing the sample p_0 contains non-zero transform coefficient levels,

  – the 4x4 luma transform block associated with the 4x4 luma block containing the sample q_0 contains non-zero transform coefficient levels.

– Otherwise, if any of the following conditions are true, a value of bS equal to 1 is the output:

  – for the prediction of the macroblock/sub-macroblock partition containing the sample p_0 different reference pictures are used than for the prediction of the macroblock/sub-macroblock partition containing the sample q_0,

    NOTE – The determination of whether the reference pictures used for the two macroblock/sub-macroblock partitions are the same or different is based only on which pictures are referenced, without regard to whether the index position within a reference picture list is different.

  – one motion vector is used to predict the macroblock/sub-macroblock partition containing the sample p_0 and one motion vector is used to predict the macroblock/sub-macroblock partition containing the sample q_0 and the absolute difference between the horizontal or vertical components of the motion vectors used is greater than or equal to 4 in units of quarter luma frame samples,

– Otherwise, a value of bS equal to 0 is the output.

8.7.2.2 Derivation process for the thresholds for each block edge

Inputs to this process are:

– the input sample values p_0, q_0, p_1 and q_1 of a single set of samples across an edge that is to be filtered,

– the variables chromaEdgeFlag and bS, for the set of input samples, as specified in subclause 8.7.2,

– the variables filterOffsetA, filterOffsetB, qP_p, and qP_q.

Outputs of this process are the variable filterSamplesFlag, which indicates whether the input samples are filtered, the value of indexA, and the values of the threshold variables α and β.

Let qP_{av} be a variable specifying an average quantisation parameter. It is derived as:

$$qP_{av} = (qP_p + qP_q + 1) >> 1 \quad (8\text{-}217)$$

Let indexA be a variable that is used to access the α table (Table 8-8) as well as the t_{C0} table (Table 8-9), which is used in filtering of edges with bS less than 4 as specified in subclause 8.7.2.3, and let indexB be a variable that is used to access the β table (Table 8-8). The variables indexA and indexB are derived as:

$$\text{indexA} = \text{Clip3}(0, 51, qP_{av} + \text{filterOffsetA}) \quad (8\text{-}218)$$

$$\text{indexB} = \text{Clip3}(0, 51, qP_{av} + \text{filterOffsetB}) \quad (8\text{-}219)$$

The variables α' and β' depending on the values of indexA and indexB are specified in Table 8-8. Depending on chromaEdgeFlag, the corresponding threshold variables α and β are derived as follows:

– If chromaEdgeFlag is equal to 0,

$$\alpha = \alpha' * (1 << (\text{BitDepth}_Y - 8)) \quad (8\text{-}220)$$

$$\beta = \beta' * (1 << (\text{BitDepth}_Y - 8)) \quad (8\text{-}221)$$

– Otherwise (chromaEdgeFlag is equal to 1),

$$\alpha = \alpha' * (1 << (BitDepth_C - 8)) \qquad (8\text{-}222)$$

$$\beta = \beta' * (1 << (BitDepth_C - 8)) \qquad (8\text{-}223)$$

The variable filterSamplesFlag is derived by:

$$filterSamplesFlag = (bS \mathrel{!=} 0 \;\&\&\; Abs(p_0 - q_0) < \alpha \;\&\&\; Abs(p_1 - p_0) < \beta \;\&\&\; Abs(q_1 - q_0) < \beta) \qquad (8\text{-}224)$$

Table 8-8 – Derivation of offset dependent threshold variables α′ and β′ from indexA and indexB

	indexA (for α′) or indexB (for β′)																									
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
α′	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	5	6	7	8	9	10	12	13
β′	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	3	3	3	3	4	4	4

Table 8-8 (concluded) – Derivation of indexA and indexB from offset dependent threshold variables α′ and β′

	indexA (for α′) or indexB (for β′)																									
	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
α′	15	17	20	22	25	28	32	36	40	45	50	56	63	71	80	90	101	113	127	144	162	182	203	226	255	255
β′	6	6	7	7	8	8	9	9	10	10	11	11	12	12	13	13	14	14	15	15	16	16	17	17	18	18

8.7.2.3 Filtering process for edges with bS less than 4

Inputs to this process are the input sample values p_i and q_i ($i = 0..2$) of a single set of samples across an edge that is to be filtered, chromaEdgeFlag, bS, β, and indexA, for the set of input samples, as specified in subclause 8.7.2.

Outputs of this process are the filtered result sample values p'_i and q'_i ($i = 0..2$) for the set of input sample values.

Depending on the values of indexA and bS, the variable t_{C0} is specified in Table 8-9.

Table 8-9 – Value of variable t_{C0} as a function of indexA and bS

	indexA																									
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
bS = 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
bS = 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
bS = 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1

Table 8-9 (concluded) – Value of variable t_{C0} as a function of indexA and bS

	indexA																									
	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
bS = 1	1	1	1	1	1	1	1	2	2	2	2	3	3	3	4	4	4	5	6	6	7	8	9	10	11	13
bS = 2	1	1	1	1	1	2	2	2	2	3	3	3	4	4	5	5	6	7	8	8	10	11	12	13	15	17
bS = 3	1	2	2	2	2	3	3	3	4	4	4	5	6	6	7	8	9	10	11	13	14	16	18	20	23	25

The threshold variables a_p and a_q are derived by:

$$a_p = \text{Abs}(p_2 - p_0) \quad (8\text{-}225)$$
$$a_q = \text{Abs}(q_2 - q_0) \quad (8\text{-}226)$$

The threshold variable t_C is determined as follows:

– If chromaEdgeFlag is equal to 0,

$$t_C = t_{C0} + ((a_p < \beta) ? 1 : 0) + ((a_q < \beta) ? 1 : 0) \quad (8\text{-}227)$$

– Otherwise (chromaEdgeFlag is equal to 1),

$$t_C = t_{C0} + 1 \quad (8\text{-}228)$$

Let Clip1( ) be a function that is replaced by $\text{Clip1}_Y( )$ when chromaEdgeFlag is equal to 0 and by $\text{Clip1}_C( )$ when chromaEdgeFlag is equal to 1.

The filtered result samples p'_0 and q'_0 are derived by:

$$\Delta = \text{Clip3}(-t_C, t_C, ((((q_0 - p_0) << 2) + (p_1 - q_1) + 4) >> 3)) \quad (8\text{-}229)$$
$$p'_0 = \text{Clip1}(p_0 + \Delta) \quad (8\text{-}230)$$
$$q'_0 = \text{Clip1}(q_0 - \Delta) \quad (8\text{-}231)$$

The filtered result sample p'_1 is derived as follows:

– If chromaEdgeFlag is equal to 0 and a_p is less than β,

$$p'_1 = p_1 + \text{Clip3}(-t_{C0}, t_{C0}, (p_2 + ((p_0 + q_0 + 1) >> 1) - (p_1 << 1)) >> 1) \quad (8\text{-}232)$$

– Otherwise (chromaEdgeFlag is equal to 1 or a_p is greater than or equal to β),

$$p'_1 = p_1 \quad (8\text{-}233)$$

The filtered result sample q'_1 is derived as follows:

– If chromaEdgeFlag is equal to 0 and a_q is less than β,

$$q'_1 = q_1 + \text{Clip3}(-t_{C0}, t_{C0}, (q_2 + ((p_0 + q_0 + 1) >> 1) - (q_1 << 1)) >> 1) \quad (8\text{-}234)$$

– Otherwise (chromaEdgeFlag is equal to 1 or a_q is greater than or equal to β),

$$q'_1 = q_1 \quad (8\text{-}235)$$

The filtered result samples p'_2 and q'_2 are always set equal to the input samples p_2 and q_2:

$$p'_2 = p_2 \quad (8\text{-}236)$$
$$q'_2 = q_2 \quad (8\text{-}237)$$

8.7.2.4 Filtering process for edges for bS equal to 4

Inputs to this process are the input sample values p_i and q_i (i = 0..3) of a single set of samples across an edge that is to be filtered, chromaEdgeFlag, and the values of the threshold variables α and β for the set of samples, as specified in subclause 8.7.2.

Outputs of this process are the filtered result sample values p'_i and q'_i (i = 0..2) for the set of input sample values.

Let a_p and a_q be two threshold variables as specified in Equations 8-225 and 8-226, respectively, in subclause 8.7.2.3.

The filtered result samples p'_i (i = 0..2) are derived as follows:

– If chromaEdgeFlag is equal to 0 and the following condition holds,

$$a_p < \beta \;\&\&\; \text{Abs}(p_0 - q_0) < ((\alpha >> 2) + 2) \quad (8\text{-}238)$$

then the variables p'_0, p'_1, and p'_2 are derived by:

$$p'_0 = (p_2 + 2{*}p_1 + 2{*}p_0 + 2{*}q_0 + q_1 + 4) >> 3 \quad (8\text{-}239)$$

$$p'_1 = (p_2 + p_1 + p_0 + q_0 + 2) >> 2 \quad (8\text{-}240)$$

$$p'_2 = (2{*}p_3 + 3{*}p_2 + p_1 + p_0 + q_0 + 4) >> 3 \quad (8\text{-}241)$$

– Otherwise (chromaEdgeFlag is equal to 1 or the condition in Equation 8-238 does not hold), the variables p'_0, p'_1, and p'_2 are derived by:

$$p'_0 = (2{*}p_1 + p_0 + q_1 + 2) >> 2 \quad (8\text{-}242)$$

$$p'_1 = p_1 \quad (8\text{-}243)$$

$$p'_2 = p_2 \quad (8\text{-}244)$$

The filtered result samples q'_i (i = 0..2) are derived as follows:

– If chromaEdgeFlag is equal to 0 and the following condition holds,

$$a_q < \beta \;\&\&\; \text{Abs}(p_0 - q_0) < ((\alpha >> 2) + 2) \quad (8\text{-}245)$$

then the variables q'_0, q'_1, and q'_2 are derived by

$$q'_0 = (p_1 + 2{*}p_0 + 2{*}q_0 + 2{*}q_1 + q_2 + 4) >> 3 \quad (8\text{-}246)$$

$$q'_1 = (p_0 + q_0 + q_1 + q_2 + 2) >> 2 \quad (8\text{-}247)$$

$$q'_2 = (2{*}q_3 + 3{*}q_2 + q_1 + q_0 + p_0 + 4) >> 3 \quad (8\text{-}248)$$

– Otherwise (chromaEdgeFlag is equal to 1 or the condition in Equation 8-245 does not hold), the variables q'_0, q'_1, and q'_2 are derived by:

$$q'_0 = (2{*}q_1 + q_0 + p_1 + 2) >> 2 \quad (8\text{-}249)$$

$$q'_1 = q_1 \quad (8\text{-}250)$$

$$q'_2 = q_2 \quad (8\text{-}251)$$

9 Parsing process

Inputs to this process are bits from the RBSP.

Outputs of this process are syntax element values.

This process is invoked when the descriptor of a syntax element in the syntax tables in subclause 7.3 is equal to ue(v), me(v), se(v), te(v) (see subclause 9.1), ce(v) (see subclause 9.2)

9.1 Parsing process for Exp-Golomb codes

This process is invoked when the descriptor of a syntax element in the syntax tables in subclause 7.3 is equal to ue(v), me(v), se(v), or te(v). For syntax elements in subclauses 7.3.4 and 7.3.5.

Inputs to this process are bits from the RBSP.

Outputs of this process are syntax element values.

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

```
leadingZeroBits = −1
for( b = 0; !b; leadingZeroBits++ )                                  (9-1)
    b = read_bits( 1 )
```

The variable codeNum is then assigned as follows:

$$\text{codeNum} = 2^{\text{leadingZeroBits}} - 1 + \text{read\_bits}(\ \text{leadingZeroBits}\ ) \quad (9\text{-}2)$$

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

Table 9-1 illustrates the structure of the Exp-Golomb code by separating the bit string into "prefix" and "suffix" bits. The "prefix" bits are those bits that are parsed in the above pseudo-code for the computation of leadingZeroBits, and are shown as either 0 or 1 in the bit string column of Table 9-1. The "suffix" bits are those bits that are parsed in the computation of codeNum and are shown as x_i in Table 9-1, with i being in the range 0 to leadingZeroBits − 1, inclusive. Each x_i can take on values 0 or 1.

Table 9-1 – Bit strings with "prefix" and "suffix" bits and assignment to codeNum ranges (informative)

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

Table 9-2 illustrates explicitly the assignment of bit strings to codeNum values.

Table 9-2 – Exp-Golomb bit strings and codeNum in explicit form and used as ue(v) (informative)

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

Depending on the descriptor, the value of a syntax element is derived as follows:

– If the syntax element is coded as ue(v), the value of the syntax element is equal to codeNum.

– Otherwise, if the syntax element is coded as se(v), the value of the syntax element is derived by invoking the mapping process for signed Exp-Golomb codes as specified in subclause 9.1.1 with codeNum as the input.

– Otherwise, if the syntax element is coded as me(v), the value of the syntax element is derived by invoking the mapping process for coded block pattern as specified in subclause 9.1.2 with codeNum as the input.

– Otherwise (the syntax element is coded as te(v)), the range of possible values for the syntax element is determined first. The range of this syntax element may be between 0 and x, with x being greater than or equal to 1 and the range is used in the derivation of the value of the syntax element value as follows:

  – If x is greater than 1, codeNum and the value of the syntax element is derived in the same way as for syntax elements coded as ue(v).

  – Otherwise (x is equal to 1), the parsing process for codeNum which is equal to the value of the syntax element is given by a process equivalent to:

    b = read_bits(1) (9-3)
    codeNum = !b

9.1.1 Mapping process for signed Exp-Golomb codes

Input to this process is codeNum as specified in subclause 9.1.

Output of this process is a value of a syntax element coded as se(v).

The syntax element is assigned to the codeNum by ordering the syntax element by its absolute value in increasing order and representing the positive value for a given absolute value with the lower codeNum. Table 9-3 provides the assignment rule.

Table 9-3 – Assignment of syntax element to codeNum for signed Exp-Golomb coded syntax elements se(v)

codeNum	syntax element value
0	0
1	1
2	−1
3	2
4	−2
5	3
6	−3
k	$(-1)^{k+1}$ Ceil(k÷2)

9.1.2 Mapping process for coded block pattern

Input to this process is codeNum as specified in subclause 9.1.

Output of this process is a value of the syntax element coded_block_pattern coded as me(v).

Table 9-4 shows the assignment of coded_block_pattern to codeNum depending on whether the macroblock prediction mode is equal to Intra_4x4 or Inter.

Table 9-4 – Assignment of codeNum to values of coded_block_pattern for macroblock prediction modes

codeNum	coded_block_pattern	
	Intra_4x4	Inter
0	47	0
1	31	16
2	15	1
3	0	2
4	23	4
5	27	8
6	29	32
7	30	3
8	7	5
9	11	10
10	13	12
11	14	15

codeNum	coded_block_pattern	
	Intra_4x4	Inter
12	39	47
13	43	7
14	45	11
15	46	13
16	16	14
17	3	6
18	5	9
19	10	31
20	12	35
21	19	37
22	21	42
23	26	44
24	28	33
25	35	34
26	37	36
27	42	40
28	44	39
29	1	43
30	2	45
31	4	46
32	8	17
33	17	18
34	18	20
35	20	24
36	24	19
37	6	21
38	9	26
39	22	28
40	25	23

codeNum	coded_block_pattern	
	Intra_4x4	Inter
41	32	27
42	33	29
43	34	30
44	36	22
45	40	25
46	38	38
47	41	41

9.2 CAVLC parsing process for transform coefficient levels

This process is invoked for the parsing of syntax elements with descriptor equal to ce(v) in subclause 7.3.5.3.2.

Inputs to this process are bits from slice data, a maximum number of non-zero transform coefficient levels maxNumCoeff, the luma block index luma4x4BlkIdx or the chroma block index chroma4x4BlkIdx,cb4x4BlkIdx or cr4x4BlkIdx of the current block of transform coefficient levels.

Output of this process is the list coeffLevel containing transform coefficient levels of the luma block with block index luma4x4BlkIdx or the chroma block with block index chroma4x4BlkIdx, cb4x4BlkIdx or cr4x4BlkIdx.

The process is specified in the following ordered steps:

1. All transform coefficient level values coeffLevel[i], with indices i ranging from 0 to maxNumCoeff − 1, in the list coeffLevel are set equal to 0.

2. The total number of non-zero transform coefficient levels TotalCoeff(coeff_token) and the number of trailing one transform coefficient levels TrailingOnes(coeff_token) are derived by parsing coeff_token as specified in subclause 9.2.1.

3. The following then applies:

   – If the number of non-zero transform coefficient levels TotalCoeff(coeff_token) is equal to 0, the list coeffLevel (in which all transform coefficient level values are equal to 0) is returned and no further steps are carried out.

   – Otherwise, the following steps are carried out:

      a. The non-zero transform coefficient levels are derived by parsing trailing_ones_sign_flag, level_prefix, and level_suffix as specified in subclause 9.2.2.

      b. The runs of zero transform coefficient levels before each non-zero transform coefficient level are derived by parsing total_zeros and run_before as specified in subclause 9.2.3.

      c. The level and run information are combined into the list coeffLevel as specified in subclause 9.2.4.

9.2.1 Parsing process for total number of non-zero transform coefficient levels and number of trailing ones

Inputs to this process are bits from slice data, a maximum number of non-zero transform coefficient levels maxNumCoeff, the luma block index luma4x4BlkIdx or the chroma block index chroma4x4BlkIdx,cb4x4BlkIdx or cr4x4BlkIdx of the current block of transform coefficient levels.

Outputs of this process are TotalCoeff(coeff_token), TrailingOnes(coeff_token), and the variable nC.

The syntax element coeff_token is decoded using one of the six VLCs specified in the six right-most columns of Table 9-5. Each VLC specifies both TotalCoeff(coeff_token) and TrailingOnes(coeff_token) for a given codeword coeff_token. Theselection of the applicable column ofTable 9-5is determined by a variable nC. The value of nCis derived as follows:

– If the CAVLC parsing process is invoked for ChromaDCLevel, nC is set equal to −1,

– Otherwise, the following ordered steps are performed:

1. When the CAVLC parsing process is invoked for Intra16x16DCLevel, luma4x4BlkIdx is set equal to 0.
2. (void)
3. (void)
4. The variables blkA and blkB are derived as follows:
   – If the CAVLC parsing process is invoked for Intra16x16DCLevel, Intra16x16ACLevel, or LumaLevel4x4, the process specified in subclause 6.4.11.4 is invoked with luma4x4BlkIdx as the input, and the output is assigned to mbAddrA, mbAddrB, luma4x4BlkIdxA, and luma4x4BlkIdxB. The 4x4 luma block specified by mbAddrA\luma4x4BlkIdxA is assigned toblkA, and the 4x4 luma block specified by mbAddrB\luma4x4BlkIdxB is assigned to blkB.
   – Otherwise (the CAVLC parsing process is invoked for ChromaACLevel), the process specified in subclause 6.4.11.5 is invoked with chroma4x4BlkIdx as input, and the output is assigned to mbAddrA, mbAddrB, chroma4x4BlkIdxA, and chroma4x4BlkIdxB. The 4x4 chroma block specified by mbAddrA\iCbCr\chroma4x4BlkIdxA is assigned toblkA, and the 4x4 chroma block specified by mbAddrB\iCbCr\chroma4x4BlkIdxB is assigned to blkB.
5. The variable availableFlagN with N being replaced by A and B is derived as follows:
   – If any of the following conditions are true, availableFlagN is set equal to 0:
     – mbAddrN is not available,
   – Otherwise, availableFlagN is set equal to 1.
6. For N being replaced by A and B, when availableFlagN is equal to 1, the variable nN is derived as follows:
   – If any of the following conditions are true, nN is set equal to 0:
     – The macroblock mbAddrN has mb_type equal to P_Skip,
     – The macroblock mbAddrN has mb_type not equal to I_PCM and all AC residual transform coefficient levels of the neighbouring block blkN are equal to 0 due to the corresponding bit of CodedBlockPatternLuma or CodedBlockPatternChroma being equal to 0.
   – Otherwise, if mbAddrN is an I_PCM macroblock, nN is set equal to 16.
   – Otherwise, nN is set equal to the value TotalCoeff(coeff_token) of the neighbouring block blkN.

     NOTE – The values nA and nB that are derived using TotalCoeff(coeff_token) do not include the DC transform coefficient levels in Intra_16x16 macroblocks or DC transform coefficient levels in chroma blocks, because these transform coefficient levels are decoded separately. When the block above or to the left belongs to an Intra_16x16 macroblock, nA ornB is the number of decoded non-zero AC transform coefficient levels for the adjacent 4x4 block in the Intra_16x16 macroblock. When the block above or to the left is a chroma block, nA or nB is the number of decoded non-zero AC transform coefficient levels for the adjacent chroma block.

NOTE – When parsing for Intra16x16DCLevel the values nA and nB are based on the number of non-zero transform coefficient levels in adjacent 4x4 blocks and not on the number of non-zero DC transform coefficient levels in adjacent 16x16 blocks.

7. The variable nC is derived as follows:
   - If availableFlagA is equal to 1 and availableFlagB is equal to 1, the variable nC is set equal to (nA + nB + 1) >> 1.
   - Otherwise, if availableFlagA is equal to 1 (and availableFlagB is equal to 0), the variable nC is set equal to nA.
   - Otherwise, if availableFlagB is equal to 1 (and availableFlagA is equal to 0), the variable nC is set equal to nB.
   - Otherwise (availableFlagA is equal to 0 and availableFlagB is equal to 0), the variable nC is set equal to 0.

When maxNumCoeff is equal to 15, it is a requirement of bitstream conformance that the value of TotalCoeff(coeff_token) resulting from decoding coeff_token shall not be equal to 16.

Table 9-5 – coeff_token mapping to TotalCoeff(coeff_token) and TrailingOnes(coeff_token)

TrailingOnes (coeff_token)	TotalCoeff (coeff_token)	0 <= nC < 2	2 <= nC < 4	4 <= nC < 8	8 <= nC	nC = = −1
0	0	1	11	1111	0000 11	01
0	1	0001 01	0010 11	0011 11	0000 00	0001 11
1	1	01	10	1110	0000 01	1
0	2	0000 0111	0001 11	0010 11	0001 00	0001 00
1	2	0001 00	0011 1	0111 1	0001 01	0001 10
2	2	001	011	1101	0001 10	001
0	3	0000 0011 1	0000 111	0010 00	0010 00	0000 11
1	3	0000 0110	0010 10	0110 0	0010 01	0000 011
2	3	0000 101	0010 01	0111 0	0010 10	0000 010
3	3	0001 1	0101	1100	0010 11	0001 01
0	4	0000 0001 11	0000 0111	0001 111	0011 00	0000 10
1	4	0000 0011 0	0001 10	0101 0	0011 01	0000 0011
2	4	0000 0101	0001 01	0101 1	0011 10	0000 0010
3	4	0000 11	0100	1011	0011 11	0000 000
0	5	0000 0000 111	0000 0100	0001 011	0100 00	-
1	5	0000 0001 10	0000 110	0100 0	0100 01	-
2	5	0000 0010 1	0000 101	0100 1	0100 10	-
3	5	0000 100	0011 0	1010	0100 11	-
0	6	0000 0000 0111 1	0000 0011 1	0001 001	0101 00	-
1	6	0000 0000 110	0000 0110	0011 10	0101 01	-
2	6	0000 0001 01	0000 0101	0011 01	0101 10	-
3	6	0000 0100	0010 00	1001	0101 11	-
0	7	0000 0000 0101 1	0000 0001 111	0001 000	0110 00	-
1	7	0000 0000 0111 0	0000 0011 0	0010 10	0110 01	-
2	7	0000 0000 101	0000 0010 1	0010 01	0110 10	-
3	7	0000 0010 0	0001 00	1000	0110 11	-
0	8	0000 0000 0100 0	0000 0001 011	0000 1111	0111 00	-
1	8	0000 0000 0101 0	0000 0001 110	0001 110	0111 01	-
2	8	0000 0000 0110 1	0000 0001 101	0001 101	0111 10	-

Table 9-5 – coeff_token mapping to TotalCoeff(coeff_token) and TrailingOnes(coeff_token)

TrailingOnes (coeff_token)	TotalCoeff (coeff_token)	0 <= nC < 2	2 <= nC < 4	4 <= nC < 8	8 <= nC	nC = = −1
3	8	0000 0001 00	0000 100	0110 1	0111 11	-
0	9	0000 0000 0011 11	0000 0000 1111	0000 1011	1000 00	-
1	9	0000 0000 0011 10	0000 0001 010	0000 1110	1000 01	-
2	9	0000 0000 0100 1	0000 0001 001	0001 010	1000 10	-
3	9	0000 0000 100	0000 0010 0	0011 00	1000 11	-
0	10	0000 0000 0010 11	0000 0000 1011	0000 0111 1	1001 00	-
1	10	0000 0000 0010 10	0000 0000 1110	0000 1010	1001 01	-
2	10	0000 0000 0011 01	0000 0000 1101	0000 1101	1001 10	-
3	10	0000 0000 0110 0	0000 0001 100	0001 100	1001 11	-
0	11	0000 0000 0001 111	0000 0000 1000	0000 0101 1	1010 00	-
1	11	0000 0000 0001 110	0000 0000 1010	0000 0111 0	1010 01	-
2	11	0000 0000 0010 01	0000 0000 1001	0000 1001	1010 10	-
3	11	0000 0000 0011 00	0000 0001 000	0000 1100	1010 11	-
0	12	0000 0000 0001 011	0000 0000 0111 1	0000 0100 0	1011 00	-
1	12	0000 0000 0001 010	0000 0000 0111 0	0000 0101 0	1011 01	-
2	12	0000 0000 0001 101	0000 0000 0110 1	0000 0110 1	1011 10	-
3	12	0000 0000 0010 00	0000 0000 1100	0000 1000	1011 11	-
0	13	0000 0000 0000 1111	0000 0000 0101 1	0000 0011 01	1100 00	-
1	13	0000 0000 0000 001	0000 0000 0101 0	0000 0011 1	1100 01	-
2	13	0000 0000 0001 001	0000 0000 0100 1	0000 0100 1	1100 10	-
3	13	0000 0000 0001 100	0000 0000 0110 0	0000 0110 0	1100 11	-
0	14	0000 0000 0000 1011	0000 0000 0011 1	0000 0010 01	1101 00	-
1	14	0000 0000 0000 1110	0000 0000 0010 11	0000 0011 00	1101 01	-
2	14	0000 0000 0000 1101	0000 0000 0011 0	0000 0010 11	1101 10	-
3	14	0000 0000 0001 000	0000 0000 0100 0	0000 0010 10	1101 11	-
0	15	0000 0000 0000 0111	0000 0000 0010 01	0000 0001 01	1110 00	-
1	15	0000 0000 0000 1010	0000 0000 0010 00	0000 0010 00	1110 01	-
2	15	0000 0000 0000 1001	0000 0000 0010 10	0000 0001 11	1110 10	-
3	15	0000 0000 0000 1100	0000 0000 0000 1	0000 0001 10	1110 11	-

Table 9-5 – coeff_token mapping to TotalCoeff(coeff_token) and TrailingOnes(coeff_token)

TrailingOnes (coeff_token)	TotalCoeff (coeff_token)	0 <= nC < 2	2 <= nC < 4	4 <= nC < 8	8 <= nC	nC = = −1
0	16	0000 0000 0000 0100	0000 0000 0001 11	0000 0000 01	1111 00	-
1	16	0000 0000 0000 0110	0000 0000 0001 10	0000 0001 00	1111 01	-
2	16	0000 0000 0000 0101	0000 0000 0001 01	0000 0000 11	1111 10	-
3	16	0000 0000 0000 1000	0000 0000 0001 00	0000 0000 10	1111 11	-

9.2.2 Parsing process for level information

Inputs to this process are bits from slice data, the number of non-zero transform coefficient levels TotalCoeff(coeff_token), and the number of trailing one transform coefficient levels TrailingOnes(coeff_token).

Output of this process is a list with name levelVal containing transform coefficient levels.

Initially an index i is set equal to 0. Then, when TrailingOnes(coeff_token) is not equal to 0, the following ordered steps are applied TrailingOnes(coeff_token) times to decode the trailing one transform coefficient levels:

1. A 1-bit syntax element trailing_ones_sign_flag is decoded and evaluated as follows:
   – If trailing_ones_sign_flag is equal to 0, levelVal[i] is set equal to 1.
   – Otherwise (trailing_ones_sign_flag is equal to 1), levelVal[i] is set equal to −1.
2. The index i is incremented by 1.

Then, the variable suffixLength is initialised as follows:

– If TotalCoeff(coeff_token) is greater than 10 and TrailingOnes(coeff_token) is less than 3, suffixLength is set equal to 1.

– Otherwise (TotalCoeff(coeff_token) is less than or equal to 10 or TrailingOnes(coeff_token) is equal to 3), suffixLength is set equal to 0.

Then, when TotalCoeff(coeff_token) − TrailingOnes(coeff_token) is not equal to 0, the following ordered steps are applied TotalCoeff(coeff_token) − TrailingOnes(coeff_token) times to decode the remaining non-zero level values:

1. The syntax element level_prefix is decoded as specified in subclause 9.2.2.1.
2. The variable levelSuffixSize is set as follows:
   – Iflevel_prefix is equal to 14 and suffixLength is equal to 0, levelSuffixSize is set equal to 4.
   – Otherwise, if level_prefix is equal to 15, levelSuffixSize is set equal tolevel_prefix − 3.
   – Otherwise, levelSuffixSize is set equal to suffixLength.
3. The syntax element level_suffix is decoded as follows:
   – If levelSuffixSize is greater than 0, the syntax element level_suffix is decoded as unsigned integer representation u(v) with levelSuffixSize bits.
   – Otherwise (levelSuffixSize is equal to 0), the syntax element level_suffix is inferred to be equal to 0.

4. The variable levelCode is set equal to (level_prefix << suffixLength) + level_suffix.
5. (void)
6. (void)
7. When the index i is equal to TrailingOnes(coeff_token) and TrailingOnes(coeff_token) is less than 3, levelCode is incremented by 2.
8. The variable levelVal[i] is derived as follows:
   - If levelCode is an even number, levelVal[i] is set equal to (levelCode + 2) >> 1.
   - Otherwise (levelCode is an odd number), levelVal[i] is set equal to (−levelCode − 1) >> 1.
9. When suffixLength is equal to 0, suffixLength is set equal to 1.
10. When the absolute value of levelVal[i] is greater than (3 << (suffixLength − 1)) and suffixLength is less than 6, suffixLength is incremented by 1.
11. The index i is incremented by 1.

9.2.2.1 Parsing process for level_prefix

Inputs to this process are bits from slice data.

Output of this process is level_prefix.

The parsing process for this syntax element consists in reading the bits starting at the current location in the bitstream up to and including the first non-zero bit, and counting the number of leading bits that are equal to 0. This process is specified as follows:

```
leadingZeroBits = −1
for( b = 0; !b; leadingZeroBits++ )                                    (9-4)
       b = read_bits( 1 )
level_prefix = leadingZeroBits
```

Table 9-6 illustrates the codeword table for level_prefix.

Table 9-6 – Codeword table for level_prefix (informative)

level_prefix	bit string
0	1
1	01
2	001
3	0001
4	0000 1
5	0000 01
6	0000 001
7	0000 0001
8	0000 0000 1
9	0000 0000 01
10	0000 0000 001
11	0000 0000 0001
12	0000 0000 0000 1
13	0000 0000 0000 01
14	0000 0000 0000 001
15	0000 0000 0000 0001

9.2.3 Parsing process for run information

Inputs to this process are bits from slice data, the number of non-zero transform coefficient levels TotalCoeff(coeff_token), and the maximum number of non-zero transform coefficient levels maxNumCoeff.

Output of this process is a list of runs of zero transform coefficient levels preceding non-zero transform coefficient levels called runVal.

Initially, an index i is set equal to 0.

The variable zerosLeft is derived as follows:

– If the number of non-zero transform coefficient levels TotalCoeff(coeff_token) is equal to the maximum number of non-zero transform coefficient levels maxNumCoeff, a variable zerosLeft is set equal to 0.

– Otherwise (the number of non-zero transform coefficient levels TotalCoeff(coeff_token) is less than the maximum number of non-zero transform coefficient levels maxNumCoeff), total_zeros is decoded and zerosLeft is set equal to its value.

The variable tzVlcIndex is setequal to TotalCoeff(coeff_token).

The VLC used to decode total_zeros is derived as follows:

– If maxNumCoeff is equal to 4, one of the VLCs specified in Table 9-9 is used.

– Otherwise (maxNumCoeff is not equal to 4), VLCs from Tables 9-7 and 9-8 are used.

The following ordered steps arethen performed TotalCoeff(coeff_token) − 1 times:

1. The variable runVal[i] is derived as follows:

   – If zerosLeft is greater than zero, a value run_before is decoded based on Table 9-10 and zerosLeft. runVal[i] is set equal to run_before.

   – Otherwise (zerosLeft is equal to 0), runVal[i] is set equal to 0.

2. The value of runVal[i] is subtracted from zerosLeft and the result is assigned to zerosLeft. It is a requirement of bitstream conformance that the result of the subtraction shall be greater than or equal to 0.

3. The index i is incremented by 1.

Finally the value of zerosLeft is assigned to runVal[i].

Table 9-7 – total_zeros tables for 4x4 blocks with tzVlcIndex 1 to 7

total_zeros	tzVlcIndex					
	1	2	3	4	5	6	7
0	1	111	0101	0001 1	0101	0000 01	0000 01
1	011	110	111	111	0100	0000 1	0000 1
2	010	101	110	0101	0011	111	101
3	0011	100	101	0100	111	110	100
4	0010	011	0100	110	110	101	011
5	0001 1	0101	0011	101	101	100	11
6	0001 0	0100	100	100	100	011	010
7	0000 11	0011	011	0011	011	010	0001
8	0000 10	0010	0010	011	0010	0001	001
9	0000 011	0001 1	0001 1	0010	0000 1	001	0000 00
10	0000 010	0001 0	0001 0	0001 0	0001	0000 00	-
11	0000 0011	0000 11	0000 01	0000 1	0000 0	-	-
12	0000 0010	0000 10	0000 1	0000 0	-	-	-
13	0000 0001 1	0000 01	0000 00	-	-	-	-
14	0000 0001 0	0000 00	-	-	-	-	-
15	0000 0000 1	-	-	-	-	-	-

Table 9-8 – total_zeros tables for 4x4 blocks with tzVlcIndex 8 to 15

total_zeros	tzVlcIndex							
	8	9	10	11	12	13	14	15
0	0000 01	0000 01	0000 1	0000	0000	000	00	0
1	0001	0000 00	0000 0	0001	0001	001	01	1
2	0000 1	0001	001	001	01	1	1	-
3	011	11	11	010	1	01	-	-
4	11	10	10	1	001	-	-	-
5	10	001	01	011	-	-	-	-
6	010	01	0001	-	-	-	-	-
7	001	0000 1	-	-	-	-	-	-
8	0000 00	-	-	-	-	-	-	-

Table 9-9 – total_zeros tables for chroma DC 2x2 (4:2:0 chroma sampling)

total_zeros	tzVlcIndex		
	1	2	3
0	1	1	1
1	01	01	0
2	001	00	-
3	000	-	-

Table 9-10 – Tables for run_before

run_before	zerosLeft						
	1	2	3	4	5	6	>6
0	1	1	11	11	11	11	111
1	0	01	10	10	10	000	110
2	-	00	01	01	011	001	101
3	-	-	00	001	010	011	100
4	-	-	-	000	001	010	011
5	-	-	-	-	000	101	010
6	-	-	-	-	-	100	001
7	-	-	-	-	-	-	0001
8		-	-	-	-	-	00001
9	-	-	-	-	-	-	000001
10	-	-	-	-	-	-	0000001
11	-	-	-	-	-	-	00000001
12	-	-	-	-	-	-	000000001
13	-	-	-	-	-	-	0000000001
14	-	-	-	-	-	-	00000000001

9.2.4 Combining level and run information

Input to this process are a list of transform coefficient levels called levelVal, a list of runs called runVal, and the number of non-zero transform coefficient levels TotalCoeff(coeff_token).

Output of this process is an list coeffLevel of transform coefficient levels.

A variable coeffNum is set equal to −1 and an index i is set equal to TotalCoeff(coeff_token) − 1. The following ordered steps are then applied TotalCoeff(coeff_token) times:

1. coeffNum is incremented by runVal[i] + 1.
2. coeffLevel[coeffNum] is set equal to levelVal[i].
3. The index i is decremented by 1.

Annex A
(normative)
Profiles and levels

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

NOTE – This International Standard 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. All specifications that are not specified by the profile in this Annex shall be considered informative.

NOTE – Encoders are not required to make use of any particular subset of features supported in a profile. This text is derived from ISO/IEC 14996-10 and is intended to only specify Constrained Baseline profile.

Each level specifies a set of limits on the values that may be taken by the syntax elements of this International Standard. 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, levels generally correspond to decoder processing load and memory capability.

A.1 Requirements on video decoder capability

Capabilities of video decoders conforming to this International Standard are specified in terms of the ability to decode video streams conforming to the constraints of profiles and levels specified in this annex. For each such profile, the level supported for that profile shall also be expressed.

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

NOTE – Decoders should not infer that when a reserved value of profile_idc or level_idc falls between the values specified in this International Standard that this indicates intermediate capabilities between the specified profiles or levels, as there are no restrictions on the method to be chosen by ITU-T | ISO/IEC for the use of such future reserved values.

A.2 Profiles

All constraints for picture parameter sets that are specified in subclause A.2.1 are constraints for picture parameter sets that are activated in the bitstream. All constraints for sequence parameter sets that are specified in subclause A.2.1 are constraints for sequence parameter sets that are activated in the bitstream.

A.2.1 Constrained Baseline profile

Bitstreams conforming to the Constrained Baseline profile shall obey the following constraints:

– Only I and P slice types may be present.
– NAL unit streams shall not contain nal_unit_type values in the range of 2 to 4, inclusive.
– Arbitrary slice order is not allowed.
– Sequence parameter sets shall have frame_mbs_only_flag equal to 1.
– Picture parameter sets shall have num_slice_groups_minus1 equal to 0 only.
– Picture parameter sets shall have redundant_pic_cnt_present_flag equal to 0 only.
– Picture parameter sets shall have weighted_pred_flag and weighted_bipred_idc both equal to 0.
– Picture parameter sets shall have entropy_coding_mode_flag equal to 0.
– The syntax element level_prefix shall not be greater than 15 (when present).

– The syntax elements pcm_sample_luma[i], with i = 0..255, and pcm_sample_chroma[i], with i = 0..2 * MbWidthC * MbHeightC − 1, shall not be equal to 0 (when present).

– The level constraints specified for the Constrained Baseline profile in subclause A.3.1 shall be fulfilled.

Conformance of a bitstream to the Constrained Baseline profile is indicated by profile_idc being equal to 66 with constraint_set1_flag being equal to 1. Alternatively, bitstreams conforming to this Specification may indicate conformance to the Constrained Baseline profile by profile_idc being equal to 77 with constraint_set0_flag being equal to 1 or by profile_idc being equal to 88 with both constraint_set0_flag and constraint_set1_flag being equal to 1.

Decoders conforming to the Constrained Baseline profile at a specific level shall be capable of decoding all bitstreams in which all of the following are true:

– profile_idc is equal to 66 or constraint_set0_flag is equal to 1,

– constraint_set1_flag is equal to 1,

– level_idc and constraint_set3_flag represent a level less than or equal to the specified level.

As defined here, a conforming Web Video Coding decoder is able to decode all bitstreams that can be decoded by a conforming AVC Constrained Baseline Profile decoder as defined in ISO/IEC 14496-10.

All parts of this International Standard that are not part of the set of technical specifications that are necessary for conformance to Constrained Baseline Profile shall be considered informative.

A.3 Levels

A.3.1 General

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 primary coded picture or the corresponding decoded picture of access unit n.

Let the variable fR be derived as follows:

– fR is set equal to 1 ÷ 172.

Bitstreams conforming to the Constrained Baseline profile at a specified level shall obey the following constraints:

a) The nominal removal time of access unit n with n > 0 from the CPB as specified in subclause C.1.2, satisfies the constraint that $t_{r,n}(n) - t_r(n - 1)$ is greater than or equal to Max(PicSizeInMbs ÷ MaxMBPS, fR), where MaxMBPS is the value specified in Table A-1 that applies to picture n − 1 and PicSizeInMbs is the number of macroblocks in picture n − 1.

b) The difference between consecutive output times of pictures from the DPB as specified in subclause C.2.2, satisfies the constraint that $\Delta t_{o,dpb}(n)$ >= Max(PicSizeInMbs ÷ MaxMBPS, fR), where MaxMBPS is the value specified in Table A-1 for picture n and PicSizeInMbs is the number of macroblocks of picture n, provided that picture n is a picture that is output and is not the last picture of the bitstream that is output.

c) The sum of the NumBytesInNALunit variables for access unit 0 is less than or equal to 384 *(Max(PicSizeInMbs, fR * MaxMBPS) + MaxMBPS * ($t_r(0) - t_{r,n}(0)$)) ÷ MinCR, where MaxMBPS and MinCR are the values specified in Table A-1 that apply to picture 0 and PicSizeInMbs is the number of macroblocks in picture 0.

d) The sum of the NumBytesInNALunit variables for access unit n with n > 0 is less than or equal to 384 * MaxMBPS * ($t_r(n) - t_r(n - 1)$) ÷ MinCR, where MaxMBPS and MinCR are the values specified in Table A-1 that apply to picture n.

e) PicWidthInMbs * FrameHeightInMbs <= MaxFS, where MaxFS is specified in Table A-1

f) PicWidthInMbs <= Sqrt(MaxFS * 8)

g) FrameHeightInMbs <= Sqrt(MaxFS * 8)

h) max_dec_frame_buffering <= MaxDpbFrames, where MaxDpbFrames is equal to Min(MaxDpbMbs / (PicWidthInMbs * FrameHeightInMbs), 16) and MaxDpbMbs is given in Table A-1.

i) For the VCL HRD parameters, BitRate[SchedSelIdx] <= 1000 * MaxBR and CpbSize[SchedSelIdx] <= 1000 * MaxCPB for at least one value of SchedSelIdx, where BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are given as follows:

   – If vcl_hrd_parameters_present_flag is equal to 1, BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are given by Equations E-37 and E-38, respectively, using the syntax elements of the hrd_parameters( ) syntax structure that immediately follows vcl_hrd_parameters_present_flag.

   – Otherwise (vcl_hrd_parameters_present_flag is equal to 0), BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are inferred as specified in subclause E.2.2 for VCL HRD parameters.

   MaxBR and MaxCPB are specified in Table A-1 in units of 1000 bits/s and 1000 bits, respectively. The bitstream shall satisfy these conditions for at least one value of SchedSelIdx in the range 0 to cpb_cnt_minus1, inclusive.

j) For the NAL HRD parameters, BitRate[SchedSelIdx] <= 1200 * MaxBR and CpbSize[SchedSelIdx] <= 1200 * MaxCPB for at least one value of SchedSelIdx, where BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are given as follows:

   – If nal_hrd_parameters_present_flag is equal to 1, BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are given by Equations E-37 and E-38, respectively, using the syntax elements of the hrd_parameters( ) syntax structure that immediately follows nal_hrd_parameters_present_flag.

   – Otherwise (nal_hrd_parameters_present_flag is equal to 0), BitRate[SchedSelIdx] and CpbSize[SchedSelIdx] are inferred as specified in subclause E.2.2 for NAL HRD parameters.

   MaxBR and MaxCPB are specified in Table A-1 in units of 1200 bits/s and 1200 bits, respectively. The bitstream shall satisfy these conditions for at least one value of SchedSelIdx in the range 0 to cpb_cnt_minus1.

k) The vertical motion vector component range for luma motion vectors does not exceed MaxVmvR in units of luma frame samples, where MaxVmvR is specified in Table A-1

l) The horizontal motion vector range does not exceed the range of −2048 to 2047.75, inclusive, in units of luma samples

m) Let setOf2Mb be the set of unsorted pairs of macroblocks that contains the unsorted pairs of macroblocks (mbA, mbB) of a coded video sequence for which any of the following conditions are true:

   – mbA and mbB are macroblocks that belong to the same slice and are consecutive in decoding order,

   – mbA is the last macroblock (in decoding order) of aslice, and mbB is the first macroblock (in decoding order) of the next slice in decoding order,

      NOTE – The macroblocks mbA and mbB can belong to different pictures.

   For each unsorted pair of macroblocks (mbA, mbB) of the set setOf2Mb, the total number of motion vectors (given by the sum of the number of motion vectors for macroblock mbA and the number of motion vectors for macroblock mbB) does not exceed MaxMvsPer2Mb, where MaxMvsPer2Mb is specified in Table A-1. The number of motion vectors for each macroblock is the value of the variable MvCnt after the completion of the intra or inter prediction process for the macroblock.

      NOTE – The constraint specifies that the total number of motion vectors for two consecutive macroblocks in decoding order must not exceed MaxMvsPer2Mb.

n) The number of bits of macroblock_layer( ) data for any macroblock is not greater than 3200. The number of bits of macroblock_layer( ) data is given by the number of bits in the macroblock_layer( ) syntax structure for a macroblock.

Table A-1 specifies the limits for each level. A definition of all levels identified in the "Level number" column of Table A-1 is specified for the Constrained Baseline profile. Each entry in Table A-1 indicates, for the level corresponding to the row of the table, the absence or value of a limit that is imposed by the variable corresponding to the column of the table, as follows:

– If the table entry is marked as "-", no limit is imposed by the value of the variable as a requirement of bitstream conformance to the profile at the specified level.

– Otherwise, the table entry specifies the value of the variable for the associated limit that is imposed as a requirement of bitstream conformance to the profile at the specified level.

For purposes of comparison of level capabilities, a level shall be considered to be a lower (higher) level than some other level if the level appears nearer to the top (bottom) row of Table A-1 than the other level.

In bitstreams conforming to the Constrained Baseline profile, the conformance of the bitstream to a specified level is indicated by the syntax elements level_idc and constraint_set3_flag as follows:

– If level_idc is equal to 11 and constraint_set3_flag is equal to 1, the indicated level is level 1b.

– Otherwise (level_idc is not equal to 11 or constraint_set3_flag is not equal to 1), level_idc is equal to a value of ten times the level number (of the indicated level) specified in Table A-1.

Table A-1 – Level limits

Level number	Max macroblock processing rate MaxMBPS (MB/s)	Max frame size MaxFS (MBs)	Max decoded picture buffer size MaxDpbMbs (MBs)	Max video bit rate MaxBR (1000 bits/s or 1200 bits/s)	Max CPB size MaxCPB (1000 bits or 1200 bits)	Vertical MV component range MaxVmvR (luma frame samples)	Min compression ratio MinCR	Max number of motion vectors per two consecutive MBs MaxMvsPer2Mb
1	1 485	99	396	64	175	[−64,+63.75]	2	-
1b	1 485	99	396	128	350	[−64,+63.75]	2	-
1.1	3 000	396	900	192	500	[−128,+127.75]	2	-
1.2	6 000	396	2 376	384	1 000	[−128,+127.75]	2	-
1.3	11 880	396	2 376	768	2 000	[−128,+127.75]	2	-
2	11 880	396	2 376	2 000	2 000	[−128,+127.75]	2	-
2.1	19 800	792	4 752	4 000	4 000	[−256,+255.75]	2	-
2.2	20 250	1 620	8 100	4 000	4 000	[−256,+255.75]	2	-
3	40 500	1 620	8 100	10 000	10 000	[−256,+255.75]	2	32
3.1	108 000	3 600	18 000	14 000	14 000	[−512,+511.75]	4	16
3.2	216 000	5 120	20 480	20 000	20 000	[−512,+511.75]	4	16
4	245 760	8 192	32 768	20 000	25 000	[−512,+511.75]	4	16
4.1	245 760	8 192	32 768	50 000	62 500	[−512,+511.75]	2	16
4.2	522 240	8 704	34 816	50 000	62 500	[−512,+511.75]	2	16
5	589 824	22 080	110 400	135 000	135 000	[−512,+511.75]	2	16
5.1	983 040	36 864	184 320	240 000	240 000	[−512,+511.75]	2	16

Levels with non-integer level numbers in Table A-1 are referred to as "intermediate levels".

NOTE – All levels have the same status, but some applications may choose to use only the integer-numbered levels.

Informative subclause A.3.2 shows the effect of these limits on frame rates for several example picture formats.

In bitstreams conforming to the Constrained Baseline profile, $(xInt_{max} - xInt_{min} + 6) * (yInt_{max} - yInt_{min} + 6) <=$ MaxSubMbRectSize in macroblocks coded with mb_type equal to P_8x8 or P_8x8ref0 for all invocations of the process specified in subclause 8.4.2.2.1 used to generate the predicted luma sample array for a single reference picture list (reference picture list 0) for each 8x8 sub-macroblock with the macroblock partition index mbPartIdx, where NumSubMbPart(sub_mb_type[mbPartIdx]) > 1, where MaxSubMbRectSize is specified in Table A-2 for the Constrained Baseline profile and

– $xInt_{min}$ is the minimum value of $xInt_L$ among all luma sample predictions for the sub-macroblock
– $xInt_{max}$ is the maximum value of $xInt_L$ among all luma sample predictions for the sub-macroblock
– $yInt_{min}$ is the minimum value of $yInt_L$ among all luma sample predictions for the sub-macroblock
– $yInt_{max}$ is the maximum value of $yInt_L$ among all luma sample predictions for the sub-macroblock

Table A-2 specifies limits for each level that are specific to bitstreams conforming to the Constrained Baseline profile. Each entryin Table A-2 indicates, for the level corresponding to the row of the table, the absence or value of a limit that is imposed by the variable corresponding to the column of the table, as follows:

– If the table entry is marked as "-", no limit is imposed by the value of the variable as a requirement of bitstream conformance to the profile at the specified level.

– Otherwise, the table entry specifies the value of the variable for the associated limit that is imposed as a requirement of bitstream conformance to the profile at the specified level.

Table A-2 –Constrained Baseline profile level limits

Level number	MaxSubMbRectSize
1	576
1b	576
1.1	576
1.2	576
1.3	576
2	576
2.1	576
2.2	576
3	576
3.1	-
3.2	-
4	-
4.1	-
4.2	-
5	-
5.1	-

A.3.2 Effect of level limits on frame rate (informative)

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

Table A-3 – Maximum frame rates (frames per second) for some example frame sizes

Level:					**1**	**1b**	**1.1**	**1.2**	**1.3**	**2**	**2.1**
Max frame size (macroblocks):					**99**	**99**	**396**	**396**	**396**	**396**	**792**
Max macroblocks/second:					**1 485**	**1 485**	**3 000**	**6 000**	**11 880**	**11 880**	**19 800**
Max frame size (samples):					25 344	25 344	101 376	101 376	101 376	101 376	202 752
Max samples/second:					380 160	380 160	768 000	1 536 000	3 041 280	3 041 280	5 068 800
Format	**Luma Width**	**Luma Height**	**MBs Total**	**Luma Samples**							
SQCIF	**128**	**96**	48	12 288	30.9	30.9	62.5	125.0	172.0	172.0	172.0
QCIF	**176**	**144**	99	25 344	15.0	15.0	30.3	60.6	120.0	120.0	172.0
QVGA	**320**	**240**	300	76 800	-	-	10.0	20.0	39.6	39.6	66.0
525 SIF	**352**	**240**	330	84 480	-	-	9.1	18.2	36.0	36.0	60.0
CIF	**352**	**288**	396	101 376	-	-	7.6	15.2	30.0	30.0	50.0
525 HHR	**352**	**480**	660	168 960	-	-	-	-	-	-	30.0
625 HHR	**352**	**576**	792	202 752	-	-	-	-	-	-	25.0
VGA	**640**	**480**	1 200	307 200	-	-	-	-	-	-	-
525 4SIF	**704**	**480**	1 320	337 920	-	-	-	-	-	-	-
525 SD	**720**	**480**	1 350	345 600	-	-	-	-	-	-	-
4CIF	**704**	**576**	1 584	405 504	-	-	-	-	-	-	-
625 SD	**720**	**576**	1 620	414 720	-	-	-	-	-	-	-
SVGA	**800**	**600**	1 900	486 400	-	-	-	-	-	-	-
XGA	**1024**	**768**	3 072	786 432	-	-	-	-	-	-	-
720p HD	**1280**	**720**	3 600	921 600	-	-	-	-	-	-	-
4VGA	**1280**	**960**	4 800	1 228 800	-	-	-	-	-	-	-
SXGA	**1280**	**1024**	5 120	1 310 720	-	-	-	-	-	-	-
525 16SIF	**1408**	**960**	5 280	1 351 680	-	-	-	-	-	-	-
16CIF	**1408**	**1152**	6 336	1 622 016	-	-	-	-	-	-	-
4SVGA	**1600**	**1200**	7 500	1 920 000	-	-	-	-	-	-	-
1080 HD	**1920**	**1088**	8 160	2 088 960	-	-	-	-	-	-	-
2Kx1K	**2048**	**1024**	8 192	2 097 152	-	-	-	-	-	-	-
2Kx1080	**2048**	**1088**	8 704	2 228 224	-	-	-	-	-	-	-
4XGA	**2048**	**1536**	12 288	3 145 728	-	-	-	-	-	-	-
16VGA	**2560**	**1920**	19 200	4 915 200	-	-	-	-	-	-	-
3616x1536 (2.35:1)	**3616**	**1536**	21 696	5 554 176	-	-	-	-	-	-	-
3672x1536 (2.39:1)	**3680**	**1536**	22 080	5 652 480	-	-	-	-	-	-	-
4Kx2K	**4096**	**2048**	32 768	8 388 608	-	-	-	-	-	-	-
4096x2304 (16:9)	**4096**	**2304**	36 864	9 437 184	-	-	-	-	-	-	-

Table A-3 (continued) – Maximum frame rates (frames per second) for some example frame sizes

Level:					**2.2**	**3**	**3.1**	**3.2**	**4**	**4.1**	**4.2**
Max frame size (macroblocks):					**1 620**	**1 620**	**3 600**	**5 120**	**8 192**	**8 192**	**8 704**
Max macroblocks/second:					**20 250**	**40 500**	**108 000**	**216 000**	**245 760**	**245 760**	**522 240**
Max frame size (samples):					414 720	414 720	921 600	1 310 720	2 097 152	2 097 152	2 228 224
Max samples/second:					5 184 000	10 368 000	27 648 000	55 296 000	62 914 560	62 914 560	133 693 440
Format	**Luma Width**	**Luma Height**	**MBs Total**	**Luma Samples**							
SQCIF	**128**	**96**	48	12 288	172.0	172.0	172.0	172.0	172.0	172.0	172.0
QCIF	**176**	**144**	99	25 344	172.0	172.0	172.0	172.0	172.0	172.0	172.0
QVGA	**320**	**240**	300	76 800	67.5	135.0	172.0	172.0	172.0	172.0	172.0
525 SIF	**352**	**240**	330	84 480	61.4	122.7	172.0	172.0	172.0	172.0	172.0
CIF	**352**	**288**	396	101 376	51.1	102.3	172.0	172.0	172.0	172.0	172.0
525 HHR	**352**	**480**	660	168 960	30.7	61.4	163.6	172.0	172.0	172.0	172.0
625 HHR	**352**	**576**	792	202 752	25.6	51.1	136.4	172.0	172.0	172.0	172.0
VGA	**640**	**480**	1 200	307 200	16.9	33.8	90.0	172.0	172.0	172.0	172.0
525 4SIF	**704**	**480**	1 320	337 920	15.3	30.7	81.8	163.6	172.0	172.0	172.0
525 SD	**720**	**480**	1 350	345 600	15.0	30.0	80.0	160.0	172.0	172.0	172.0
4CIF	**704**	**576**	1 584	405 504	12.8	25.6	68.2	136.4	155.2	155.2	172.0
625 SD	**720**	**576**	1 620	414 720	12.5	25.0	66.7	133.3	151.7	151.7	172.0
SVGA	**800**	**600**	1 900	486 400	-	-	56.8	113.7	129.3	129.3	172.0
XGA	**1024**	**768**	3 072	786 432	-	-	35.2	70.3	80.0	80.0	172.0
720p HD	**1280**	**720**	3 600	921 600	-	-	30.0	60.0	68.3	68.3	145.1
4VGA	**1280**	**960**	4 800	1 228 800	-	-	-	45.0	51.2	51.2	108.8
SXGA	**1280**	**1024**	5 120	1 310 720	-	-	-	42.2	48.0	48.0	102.0
525 16SIF	**1408**	**960**	5 280	1 351 680	-	-	-	-	46.5	46.5	98.9
16CIF	**1408**	**1152**	6 336	1 622 016	-	-	-	-	38.8	38.8	82.4
4SVGA	**1600**	**1200**	7 500	1 920 000	-	-	-	-	32.8	32.8	69.6
1080 HD	**1920**	**1088**	8 160	2 088 960	-	-	-	-	30.1	30.1	64.0
2Kx1K	**2048**	**1024**	8 192	2 097 152	-	-	-	-	30.0	30.0	63.8
2Kx1080	**2048**	**1088**	8 704	2 228 224	-	-	-	-	-	-	60.0
4XGA	**2048**	**1536**	12 288	3 145 728	-	-	-	-	-	-	-
16VGA	**2560**	**1920**	19 200	4 915 200	-	-	-	-	-	-	-
3616x1536 (2.35:1)	**3616**	**1536**	21 696	5 554 176	-	-	-	-	-	-	-
3672x1536 (2.39:1)	**3680**	**1536**	22 080	5 652 480	-	-	-	-	-	-	-
4Kx2K	**4096**	**2048**	32 768	8 388 608	-	-	-	-	-	-	-
4096x2304 (16:9)	**4096**	**2304**	36 864	9 437 184	-	-	-	-	-	-	-

Table A-3 (concluded) – Maximum frame rates (frames per second) for some example frame sizes

Level:					**5**	**5.1**
Max frame size (macroblocks):					**22 080**	**36 864**
Max macroblocks/second:					**589 824**	**983 040**
Max frame size (samples):					5 652 480	9 437 184
Max samples/second:					150 994 944	251 658 240
Format	**Luma Width**	**Luma Height**	**MBs Total**	**Luma Samples**		
SQCIF	**128**	**96**	48	12 288	172.0	172.0
QCIF	**176**	**144**	99	25 344	172.0	172.0
QVGA	**320**	**240**	300	76 800	172.0	172.0
525 SIF	**352**	**240**	330	84 480	172.0	172.0
CIF	**352**	**288**	396	101 376	172.0	172.0
525 HHR	**352**	**480**	660	168 960	172.0	172.0
625 HHR	**352**	**576**	792	202 752	172.0	172.0
VGA	**640**	**480**	1 200	307 200	172.0	172.0
525 4SIF	**704**	**480**	1 320	337 920	172.0	172.0
525 SD	**720**	**480**	1 350	345 600	172.0	172.0
4CIF	**704**	**576**	1 584	405 504	172.0	172.0
625 SD	**720**	**576**	1 620	414 720	172.0	172.0
SVGA	**800**	**600**	1 900	486 400	172.0	172.0
XGA	**1024**	**768**	3 072	786 432	172.0	172.0
720p HD	**1280**	**720**	3 600	921 600	163.8	172.0
4VGA	**1280**	**960**	4 800	1 228 800	122.9	172.0
SXGA	**1280**	**1024**	5 120	1 310 720	115.2	172.0
525 16SIF	**1408**	**960**	5 280	1 351 680	111.7	172.0
16CIF	**1408**	**1152**	6 336	1 622 016	93.1	155.2
4SVGA	**1600**	**1200**	7 500	1 920 000	78.6	131.1
1080 HD	**1920**	**1088**	8 160	2 088 960	72.3	120.5
2Kx1K	**2048**	**1024**	8 192	2 097 152	72.0	120.0
2Kx1080	**2048**	**1088**	8 704	2 228 224	67.8	112.9
4XGA	**2048**	**1536**	12 288	3 145 728	48.0	80.0
16VGA	**2560**	**1920**	19 200	4 915 200	30.7	51.2
3616x1536 (2.35:1)	**3616**	**1536**	21 696	5 554 176	27.2	45.3
3672x1536 (2.39:1)	**3680**	**1536**	22 080	5 652 480	26.7	44.5
4Kx2K	**4096**	**2048**	32 768	8 388 608	-	30.0
4096x2304 (16:9)	**4096**	**2304**	36 864	9 437 184	-	26.7