INTERNATIONAL STANDARD

ISO/IEC 14496-10

Second edition
2004-10-01

Information technology — Coding of audio-visual objects —

Part 10: **Advanced Video Coding**

Technologies de l'information — Codage des objets audiovisuels —

Partie 10: Codage visuel avancé

Reference number
ISO/IEC 14496-10:2004(E)

Contents

Page

LIST OF FIGURES

LIST OF TABLES

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

ISO/IEC 14496-10 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information.*

This second edition cancels and replaces the first edition (ISO/IEC 14496-10:2003) which has been technically revised.

This part of ISO/IEC 14496 is technically aligned with ITU-T Rec. H.264 but is not published as identical text.

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*

0 Introduction

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

0.1 Prologue

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

As the costs for both processing power and memory have reduced, network support for coded video data has diversified, and advances in video coding technology have progressed, the need has arisen for an industry standard for compressed video representation with substantially increased coding efficiency and enhanced robustness to network environments. Toward these ends the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG) formed a Joint Video Team (JVT) in 2001 for development of a new Recommendation | International Standard.

0.2 Purpose

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

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

0.3 Applications

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

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

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

0.4 Profiles and levels

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

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

Considering the practicality of implementing the full syntax of this Specification, however, a limited number of subsets of the syntax are also stipulated by means of "profiles" and "levels". These and other related terms are formally defined in clause 3.

A "profile" is a subset of the entire bitstream syntax that is specified by this Recommendation | International Standard. Within the bounds imposed by the syntax of a given profile it is still possible to require a very large variation in the performance of encoders and decoders depending upon the values taken by syntax elements in the bitstream such as the specified size of the decoded pictures. In many applications, it is currently neither practical nor economic to implement a decoder capable of dealing with all hypothetical uses of the syntax within a particular profile.

In order to deal with this problem, "levels" are specified within each profile. A level is a specified set of constraints imposed on values of the syntax elements in the bitstream. These constraints may be simple limits on values. Alternatively they may take the form of constraints on arithmetic combinations of values (e.g. picture width multiplied by picture height multiplied by number of pictures decoded per second).

Coded video content conforming to this Recommendation | International Standard uses a common syntax. In order to achieve a subset of the complete syntax, flags, parameters, and other syntax elements are included in the bitstream that signal the presence or absence of syntactic elements that occur later in the bitstream.

0.5 Overview of the design characteristics

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

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

0.5.1 Predictive coding

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

Because of the conflicting requirements of random access and highly efficient compression, two main coding types are specified. Intra coding is done without reference to other pictures. Intra coding may provide access points to the coded sequence where decoding can begin and continue correctly, but typically also shows only moderate compression efficiency. Inter coding (predictive or bi-predictive) is more efficient using inter prediction of each block of sample values from some previously decoded picture selected by the encoder. In contrast to some other video coding standards, pictures coded using bi-predictive inter prediction may also be used as references for inter coding of other pictures.

The application of the three coding types to pictures in a sequence is flexible, and the order of the decoding process is generally not the same as the order of the source picture capture process in the encoder or the output order from the decoder for display. The choice is left to the encoder and will depend on the requirements of the application. The decoding order is specified such that the decoding of pictures that use inter-picture prediction follows later in decoding order than other pictures that are referenced in the decoding process.

0.5.2 Coding of progressive and interlaced video

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

This Recommendation | International Standard specifies a syntax and decoding process for video that originated in either progressive-scan or interlaced-scan form, which may be mixed together in the same sequence. The two fields of an interlaced frame are separated in capture time while the two fields of a progressive frame share the same capture time. Each field may be coded separately or the two fields may be coded together as a frame. Progressive frames are typically coded as a frame. For interlaced video, the encoder can choose between frame coding and field coding. Frame coding or field coding can be adaptively selected on a picture-by-picture basis and also on a more localized basis within a coded frame. Frame coding is typically preferred when the video scene contains significant detail with limited motion. Field coding typically works better when there is fast picture-to-picture motion.

0.5.3 Picture partitioning into macroblocks and smaller partitions

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

As in previous video coding Recommendations and International Standards, a macroblock, consisting of a 16x16 block of luma samples and two corresponding blocks of chroma samples, is used as the basic processing unit of the video decoding process.

A macroblock can be further partitioned for inter prediction. The selection of the size of inter prediction partitions is a result of a trade-off between the coding gain provided by using motion compensation with smaller blocks and the quantity of data needed to represent the data for motion compensation. In this Recommendation | International Standard the inter prediction process can form segmentations for motion representation as small as 4x4 luma samples in size, using motion vector accuracy of one-quarter of the luma sample grid spacing displacement. The process for inter prediction of a sample block can also involve the selection of the picture to be used as the reference picture from a number of stored previously-decoded pictures. Motion vectors are encoded differentially with respect to predicted values formed from nearby encoded motion vectors.

Typically, the encoder calculates appropriate motion vectors and other data elements represented in the video data stream. This motion estimation process in the encoder and the selection of whether to use inter prediction for the representation of each region of the video content is not specified in this Recommendation | International Standard.

0.5.4 Spatial redundancy reduction

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

Both source pictures and prediction residuals have high spatial redundancy. This Recommendation | International Standard is based on the use of a block-based transform method for spatial redundancy removal. After inter prediction from previously-decoded samples in other pictures or spatial-based prediction from previously-decoded samples within the current picture, the resulting prediction residual is split into 4x4 blocks. These are converted into the transform domain where they are quantised. After quantisation many of the transform coefficients are zero or have low amplitude and can thus be represented with a small amount of encoded data. The processes of transformation and quantisation in the encoder are not specified in this Recommendation | International Standard.

0.6 How to read this specification

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

It is suggested that the reader starts with clause 1 (Scope) and moves on to clause 3 (Definitions). Clause 6 should be read for the geometrical relationship of the source, input, and output of the decoder. Clause 7 (Syntax and semantics) specifies the order to parse syntax elements from the bitstream. See subclauses 7.1-7.3 for syntactical order and see subclause 7.4 for semantics; i.e., the scope, restrictions, and conditions that are imposed on the syntax elements. The actual parsing for most syntax elements is specified in clause 9 (Parsing process). Finally, clause 8 (Decoding process) specifies how the syntax elements are mapped into decoded samples. Throughout reading this specification, the reader should refer to clauses 2 (Normative references), 4 (Abbreviations), and 5 (Conventions) as needed. Annexes A through E also form an integral part of this Recommendation | International Standard.

Annex A defines three profiles (Baseline, Main, and Extended), each being tailored to certain application domains, and defines the so-called levels of the profiles. Annex B specifies syntax and semantics of a byte stream format for delivery of coded video as an ordered stream of bytes. Annex C specifies the hypothetical reference decoder and its use to check bitstream and decoder conformance. Annex D specifies syntax and semantics for supplemental enhancement information message payloads. Finally, Annex E specifies syntax and semantics of the video usability information parameters of the sequence parameter set.

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

Information technology — Coding of audio-visual objects —

Part 10: Advanced Video Coding

1 Scope

This document specifies ITU-T Recommendation H.264 | ISO/IEC International Standard ISO/IEC 14496-10 video coding.

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.

– ITU-T Recommendation T.35 (2000), *Procedure for the allocation of ITU-T defined codes for non-standard facilities*

– ISO/IEC 11578:1996, *Information technology — Open Systems Interconnection — Remote Procedure Call (RPC)*

– ISO/CIE 10527:1991, *CIE standard colorimetric observers*

3 Definitions

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

3.1 **access unit**: A set of *NAL units* always containing exactly one *primary coded picture*. In addition to the *primary coded picture*, an access unit may also contain one or more *redundant coded pictures* or other *NAL units* not containing *slices* or *slice data partitions* 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 **adaptive binary arithmetic decoding process**: An entropy *decoding process* that derives the values of *bins* from a *bitstream* produced by an *adaptive binary arithmetic encoding process*.

3.4 **adaptive binary arithmetic encoding process**: An entropy *encoding process*, not normatively specified in this Recommendation | International Standard, that codes a sequence of *bins* and produces a *bitstream* that can be decoded using the *adaptive binary arithmetic decoding process*.

3.5 **arbitrary slice order**: A *decoding order* of *slices* in which the *macroblock address* of the first *macroblock* of some *slice* of a *picture* may be smaller than the *macroblock address* of the first *macroblock* of some other preceding *slice* of the same *coded picture*.

3.6 **B slice**: A *slice* that may be decoded using *intra prediction* from decoded samples within the same *slice* or *inter prediction* from previously-decoded *reference pictures*, using at most two *motion vectors* and *reference indices* to *predict* the sample values of each *block*.

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

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

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

3.10 **bin string**: A string of *bins*. A bin string is an intermediate binary representation of values of *syntax elements* from the *binarization* of the *syntax element*.

3.11 **bi-predictive slice:** See *B slice***.**

3.12 **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*.

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

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

3.15 **bottom macroblock (of a macroblock pair)**: The *macroblock* within a *macroblock pair* that contains the samples in the bottom row of samples for the *macroblock pair*. For a *field macroblock pair*, the bottom macroblock represents the samples from the region of the *bottom field* of the *frame* that lie within the spatial region of the *macroblock pair*. For a *frame macroblock pair*, the bottom macroblock represents the samples of the *frame* that lie within the bottom half of the spatial region of the *macroblock pair*.

3.16 **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*.

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

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

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

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

3.20 **category**: A number associated with each *syntax element*. The category is used to specify the allocation of *syntax elements* to *NAL units* for *slice data partitioning*. It may also be used in a manner determined by the application to refer to classes of *syntax elements* in a manner not specified in this Recommendation | International Standard.

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

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

3.22 **coded field**: A *coded representation* of a *field*.

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

3.24 **coded picture**: A *coded representation* of a *picture*. A coded picture may be either a *coded field* or a *coded frame*. Coded picture is a collective term referring to a *primary coded picture* or a *redundant coded picture*, but not to both together.

3.25 **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.

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

3.27 **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 *access units* including all subsequent *access units* up to but not including any subsequent *IDR access unit*.

3.28 **component**: An array or single sample from one of the three arrays (*luma* and two *chroma*) that make up a *field* or *frame*.

3.29 **complementary field pair:** A collective term for a *complementary reference field pair* or a *complementary non-reference field pair*.

3.30 **complementary non-reference field pair**: Two *non-reference fields* that are in consecutive *access units* in *decoding order* as two *coded fields* of opposite parity where the first *field* is not already a paired *field*.

3.31 **complementary reference field pair**: Two *reference fields* that are in consecutive *access units* in *decoding order* as two *coded fields* and share the same value of the frame_num *syntax element*, where the second *field* in *decoding order* is not an *IDR picture* and does not include a memory_management_control_operation *syntax element* equal to 5.

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

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

3.34 **decoded picture**: A *decoded picture* is derived by decoding a *coded picture*. A *decoded picture* is either a decoded *frame*, or a decoded *field*. A decoded *field* is either a decoded *top field* or a decoded *bottom field*.

3.35 **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.

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

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

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

3.39 **direct prediction**: An *inter prediction* for a *block* for which no *motion vector* is decoded. Two direct *prediction* modes are specified that are referred to as spatial direct *prediction* and temporal *prediction* mode.

3.40 **decoder under test (DUT)**: A *decoder* that is tested for conformance to this Recommendation | 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*.

3.41 **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*.

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

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

3.44 **field**: An assembly of alternate rows of a *frame*. A *frame* is composed of two *fields*, a *top field* and a *bottom field*.

3.45 **field macroblock**: A macroblock containing samples from a single *field*. All *macroblocks* of a *coded field* are field macroblocks. When *macroblock-adaptive frame/field decoding* is in use, some *macroblocks* of a *coded frame* may be field macroblocks.

3.46 **field macroblock pair**: A *macroblock pair* decoded as two *field macroblocks.*

3.47 **field scan**: A specific sequential ordering of *transform coefficients* that differs from the *zig-zag scan* by scanning columns more rapidly than rows. Field scan is used for *transform coefficients* in *field macroblocks.*

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

3.49 **frame**: A *frame* contains an array of *luma* samples and two corresponding arrays of *chroma* samples. A *frame* consists of two *fields*, a *top field* and a *bottom field*.

3.50 **frame macroblock**: A *macroblock* representing samples from the two *fields* of a *coded frame*. When *macroblock-adaptive frame/field decoding* is not in use, all macroblocks of a *coded frame* are frame macroblocks. When *macroblock-adaptive frame/field decoding* is in use, some macroblocks of a *coded frame* may be frame macroblocks.

3.51 **frame macroblock pair**: A *macroblock pair* decoded as two *frame macroblocks.*

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

3.53 **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.

3.54 **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*.

3.55 **I slice**: A *slice* that is not an *SI slice* that is decoded using *prediction* only from decoded samples within the same *slice*.

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

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

3.57 **instantaneous decoding refresh (IDR) picture**: A *coded picture* containing only *slices* with *I* or *SI slice types* that causes the *decoding process* to mark all *reference pictures* as "unused for reference" immediately after decoding the IDR picture. After the decoding of an IDR picture all following *coded pictures* in *decoding order* can be decoded without *inter prediction* from any *picture* decoded prior to the IDR picture. The first *picture* of each *coded video sequence* is an IDR picture.

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

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

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

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

3.62 **intra slice**: See I slice.

3.63 **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*.

3.64 **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.

3.65 **level**: A defined set of constraints on the values that may be taken by the *syntax elements* and variables of this Recommendation | 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, level is the value of a *transform coefficient* prior to *scaling*.

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

3.67 **list 0 (list 1) prediction**: *Inter prediction* of the content of a *slice* using a *reference index* pointing into *reference picture list 0* (*list 1*).

3.68 **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.

3.69 **macroblock**: A 16x16 *block* of *luma* samples and two corresponding *blocks* of *chroma* samples. The division of a *slice* or a *macroblock pair* into macroblocks is a *partitioning*.

3.70 **macroblock-adaptive frame/field decoding**: A *decoding process* for *coded frames* in which some *macroblocks* may be decoded as *frame macroblocks* and others may be decoded as *field macroblocks.*

3.71 **macroblock address**: When *macroblock-adaptive frame/field decoding* is not in use, 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*. When *macroblock-adaptive frame/field decoding* is in use, the macroblock address of the *top macroblock* of a *macroblock pair* is two times the index of the *macroblock pair* in a *macroblock pair raster scan* of the *picture*, and the macroblock address of the *bottom macroblock* of a *macroblock pair* is the macroblock address of the corresponding *top macroblock* plus 1. The macroblock address of the *top macroblock* of each *macroblock pair* is an even number and the macroblock address of the *bottom macroblock* of each *macroblock pair* is an odd number.

3.72 **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. When *macroblock-adaptive frame/field decoding* is not in use, y is incremented by 1 for each *macroblock* row from top to bottom. When *macroblock-adaptive frame/field decoding* is in use, y is incremented by 2 for each *macroblock pair* row from top to bottom, and is incremented by an additional 1 when a macroblock is a *bottom macroblock*.

3.73 **macroblock pair**: A pair of vertically contiguous *macroblocks* in a *frame* that is coupled for use in *macroblock-adaptive frame/field decoding*. The division of a *slice* into macroblock pairs is a *partitioning*.

3.74 **macroblock partition**: A *block* of *luma* samples and two corresponding *blocks* of *chroma* samples resulting from a *partitioning* of a *macroblock* for *inter prediction*.

3.75 **macroblock to slice group map**: A means of mapping *macroblocks* of a *picture* into *slice groups*. The macroblock to slice group map consists of a list of numbers, one for each coded *macroblock*, specifying the *slice group* to which each coded *macroblock* belongs.

3.76 **map unit to slice group map**: A means of mapping *slice group map units* of a *picture* into *slice groups*. The map unit to slice group map consists of a list of numbers, one for each *slice group map unit*, specifying the *slice group* to which each coded *slice group map unit* belongs.

3.76.1 **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.

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

3.78 **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*.

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

3.79 **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*.

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

3.81 **non-paired field**: A collective term for a *non-paired reference field* or a *non-paired non-reference field*.

3.82 **non-paired non-reference field:** A decoded *non-reference field* that is not part of a *complementary non-reference field pair*.

3.83 **non-paired reference field**: A decoded *reference field* that is not part of a *complementary reference field pair*.

3.84 **non-reference field**: A *field* coded with nal_ref_idc equal to 0.

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

3.86 **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*.

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

3.87 **opposite parity**: The *opposite parity* of *top* is *bottom*, and vice versa.

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

3.89 **P slice**: A *slice* that may be decoded using *intra prediction* from decoded samples within the same *slice* or *inter prediction* from previously-decoded *reference pictures*, using at most one *motion vector* and *reference index* to *predict* the sample values of each *block*.

3.90 **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*.

3.91 **parity**: The parity of a *field* can be *top* or *bottom*.

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

3.93 **picture**: A collective term for a *field* or a *frame*.

3.93.1 **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*.

3.94 **picture order count**: A variable having a value that is non-decreasing with increasing *picture* position in output order relative to the previous *IDR picture* in *decoding order* or relative to the previous *picture* containing the *memory management control operation* that marks all *reference pictures* as "unused for reference".

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

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

3.97 **predictive slice**: See P slice.

3.98 **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.

3.99 **primary coded picture**: The coded representation of a *picture* to be used by the *decoding process* for a bitstream conforming to this Recommendation | 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. See also *redundant coded picture*.

3.100 **profile**: A specified subset of the syntax of this Recommendation | International Standard.

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

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

3.103 **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.

3.104 **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.

3.105 **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*.

3.106 **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*.

3.107 **redundant coded picture**: A coded representation of a *picture* or a part of a *picture*. The content of a redundant coded picture shall not be used by the *decoding process* for a *bitstream* conforming to this Recommendation | International Standard. A *redundant coded picture* is not required to contain all *macroblocks* in the *primary coded picture*. Redundant coded pictures have no normative effect on the *decoding process*. See also *primary coded picture*.

3.108 **reference field**: A *reference field* may be used for *inter prediction* when *P*, *SP*, and *B slices* of a *coded field* or *field macroblocks* of a *coded frame* are decoded. See also *reference picture*.

3.109 **reference frame**: A *reference frame* may be used for *inter prediction* when *P*, *SP*, and *B slices* of a *coded frame* are decoded. See also *reference picture*.

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

3.111 **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*.

3.112 **reference picture list**: A list of short-term *picture* numbers and long-term *picture* numbers that are assigned to *reference pictures*.

3.113 **reference picture list 0**: A *reference picture list* used for *inter prediction* of a *P*, *B*, or *SP slice*. All *inter prediction* used for *P* and *SP* slices uses reference picture list 0. Reference picture list 0 is one of two *reference picture lists* used for *inter prediction* for a *B slice*, with the other being *reference picture list 1*.

3.114 **reference picture list 1**: A *reference picture list* used for *inter prediction* of a *B slice*. Reference picture list 1 is one of two lists of *reference picture lists* used for *inter prediction* for a *B slice*, with the other being *reference picture list 0*.

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

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

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

3.118 **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* or a *field scan*. In other contexts, run refers to a number of *macroblocks*.

3.119 **sample aspect ratio**: Specifies, for assisting the display process, which is not specified in this Recommendation | 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).

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

3.120.1 **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*.

3.120.2 **shall**: A term used to express mandatory requirements for conformance to this Recommendation | 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 Recommendation | International Standard.

3.120.3 **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 Recommendation | International Standard.

3.121 **SI slice**: A *slice* that is coded using *prediction* only from decoded samples within the same *slice* and using quantisation of the *prediction* samples. An SI slice can be coded such that its decoded samples can be constructed identically to an *SP slice*.

3.122 **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*.

3.123 **slice**: An integer number of *macroblocks* or *macroblock pairs* ordered consecutively in the *raster scan* within a particular *slice group*. For the *primary coded picture*, the division of each *slice group* into slices is a *partitioning*. Although a slice contains *macroblocks* or *macroblock pairs* that are consecutive in the raster scan within a slice group, these *macroblocks* or *macroblock pairs* are not necessarily consecutive in the raster scan within the *picture*. The addresses of the *macroblocks* are derived from the address of the first *macroblock* in a slice (as represented in the *slice header*) and the *macroblock to slice group map.*

3.124 **slice data partitioning**: A method of *partitioning* selected *syntax elements* into *syntax structures* based on a *category* associated with each *syntax element*.

3.125 **slice group**: A subset of the *macroblocks* or *macroblock pairs* of a *picture*. The division of the *picture* into slice groups is a *partitioning* of the *picture.* The partitioning is specified by the *macroblock to slice group map*.

3.126 **slice group map units**: The units of the *map unit to slice group map.*

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

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

3.129 **SP slice**: A *slice* that is coded using *inter prediction* from previously-decoded *reference pictures*, using at most one *motion vector* and *reference index* to *predict* the sample values of each *block.* An SP slice can be coded such that its decoded samples can be constructed identically to another SP slice or an *SI slice*.

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

3.131 **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.

3.132 **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*.

3.133 **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*.

3.134 **switching I slice**: See SI slice.

3.135 **switching P slice**: See SP slice.

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

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

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

3.139 **top macroblock (of a macroblock pair)**: The *macroblock* within a *macroblock pair* that contains the samples in the top row of samples for the *macroblock pair.* For a *field macroblock pair*, the top macroblock represents the samples from the region of the *top field* of the *frame* that lie within the spatial region of the *macroblock pair.* For a *frame macroblock pair*, the top macroblock represents the samples of the *frame* that lie within the top half of the spatial region of the *macroblock pair.*

3.140 **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*.

3.141 **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.

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

3.143 **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 Recommendation | International Standard and will not have a specified meaning in the future as an integral part of this Recommendation | International Standard.

3.144 **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.

3.145 **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

4.1 **CABAC**: Context-based Adaptive Binary Arithmetic Coding

4.2 **CAVLC**: Context-based Adaptive Variable Length Coding

4.3 **CBR**: Constant Bit Rate

4.4 **CPB**: Coded Picture Buffer

4.5 **DPB**: Decoded Picture Buffer

4.6 **DUT**: Decoder under test

4.7 **FIFO**: First-In, First-Out

4.8 **HRD**: Hypothetical Reference Decoder

4.9 **HSS**: Hypothetical Stream Scheduler

4.10 **IDR**: Instantaneous Decoding Refresh

4.11 **LSB**: Least Significant Bit

4.12 **MB**: Macroblock

4.13 **MBAFF**: Macroblock-Adaptive Frame-Field Coding

4.14 **MSB**: Most Significant Bit

4.15 **NAL**: Network Abstraction Layer

4.16 **RBSP**: Raw Byte Sequence Payload

4.17 **SEI**: Supplemental Enhancement Information

4.18 **SODB**: String Of Data Bits

4.19 **UUID**: Universal Unique Identifier

4.20 **VBR**: Variable Bit Rate

4.21 **VCL**: Video Coding Layer

4.22 **VLC**: Variable Length Coding

4.23 **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

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.

When order of precedence is not indicated explicitly by use of parenthesis, the following rules apply

– multiplication and division operations are considered to take place before addition and subtraction

– multiplication and division operations in sequence are evaluated sequentially from left to right

– addition and subtraction operations in sequence are evaluated sequentially from left to right

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

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.
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 shall 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

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

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

$$\text{Clip1}(\ x\ ) = \text{Clip3}(\ 0, 255, x\ ) \qquad (5\text{-}3)$$

$$\text{Clip3}(\ x, y, z\ ) = \begin{cases} x & ; \quad z < x \\ y & ; \quad z > y \\ z & ; \quad \text{otherwise} \end{cases} \qquad (5\text{-}4)$$

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

$$\text{InverseRasterScan}(\ a, b, c, d, e\ ) = \begin{cases} (a\%(d/b)) * b; & e == 0 \\ (a/(d/b)) * c; & e == 1 \end{cases} \qquad (5\text{-}6)$$

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

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

$$\text{Luma4x4BlkScan}(\ x, y\ ) = (\ x / 2\ ) * 4 + (\ y / 2\ ) * 8 + \text{RasterScan}(\ x \% 2, y \% 2, 2\ ) \qquad (5\text{-}9)$$

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

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

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

$$\text{RasterScan}(\ x,\ y,\ n_x\ ) = x + y * n_x \tag{5-13}$$

$$\text{Round}(\ x\ ) = \text{Sign}(\ x\ ) * \text{Floor}(\ \text{Abs}(\ x\ ) + 0.5\ ) \tag{5-14}$$

$$\text{Sign}(\ x\ ) = \begin{cases} 1 & ;\quad x >= 0 \\ -1 & ;\quad x < 0 \end{cases} \tag{5-15}$$

$$\text{Sqrt}(\ x\ ) = \sqrt{x} \tag{5-16}$$

5.8 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 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 are described by their names, which are constructed as syntax element names, 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).

Square parentheses are used for indexing in lists or arrays. Lists or arrays can either be syntax elements or variables. Two-dimensional arrays are sometimes also specified using matrix notation using subscripts for indexing.

NOTE – The index order for two-dimensional arrays using square parentheses and subscripts is interchanged. A sample at horizontal position x and vertical position y in a two-dimensional sample array denoted as s[x, y] would, in matrix notation, be referred to 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 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 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 other value different than zero.

5.9 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

5.10 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.

The assignment of variables is specified as follows.

- If invoking a process, variables are explicitly assigned to lower case input or output variables of the process specification in case these do not have the same name.
- Otherwise (when the variables at the invoking and 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 Recommendation | 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 and fields that is given via the bitstream.

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

The source and decoded pictures (frames or fields) are each comprised of three sample arrays, one luma and two chroma sample arrays.

The variable ChromaFormatFactor is specified in Table 6-1, depending on the chroma format sampling structure. The value of ChromaFormatFactor shall be inferred equal to 1.5, indicating 4:2:0 sampling. In monochrome sampling there is only one sample array, which may nominally be considered a luma array. In 4:2:0 sampling, each of the two chroma arrays has half the height and half the width of the luma array. In 4:2:2 sampling, each of the two chroma arrays has the same height and half the width of the luma array. In 4:4:4 sampling, each of the two chroma arrays has the same height and width as the luma array.

NOTE – Other values may be valid for future versions of this Recommendation | International Standard.

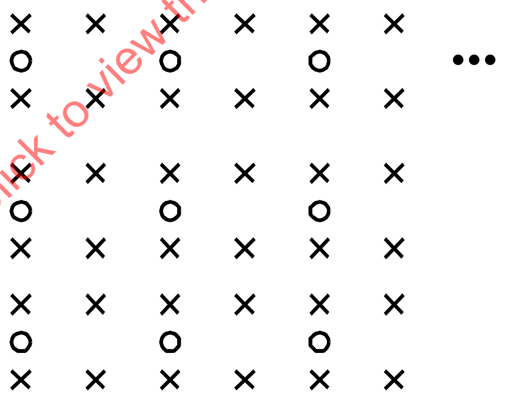
Table 6-1 – ChromaFormatFactor values

Chroma Format	ChromaFormatFactor
monochrome	1
4:2:0	1.5
4:2:2	2
4:4:4	3

This Recommendation | International Standard represents colour sequences using 4:2:0 chroma sampling. The width of the luma sample array of each picture is an integer multiple of 16. The width of the chroma sample arrays of each picture are an integer multiple of 8. The height of the luma sample array of each coded picture (whether it is a coded frame or a coded field) is an integer multiple of 16 and the height of each chroma array for these pictures is an integer multiple of 8. If any coded frames are present within a coded video sequence that contains coded fields or contains coded frames that use macroblock-adaptive frame-field coding, the height of the luma sample array of all coded frames in the coded video sequence is an integer multiple of 32 and the height of each chroma sample array for these frames is an integer multiple of 16. 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 width of fields coded referring to a specific sequence parameter set is the same as that of frames coded referring to the same sequence parameter set (see below). The height of fields coded referring to a specific sequence parameter set is half that of frames coded referring to the same sequence parameter set (see below).

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).



Guide:

× = Location of luma sample

○ = Location of chroma sample

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

A frame consists of two fields as described below. A coded picture may represent a coded frame or an individual coded field. A coded video sequence conforming to this Recommendation | International Standard may contain arbitrary combinations of coded frames and coded fields. The decoding process is also specified in a manner that allows smaller regions of a coded frame to be coded either as a frame or field region, by use of macroblock-adaptive frame-field coding.

Source and decoded fields are one of two types: top field or bottom field. When two fields are output at the same time, or are combined to be used as a reference frame (see below), the two fields (which shall be of opposite parity) are interleaved. The first (i.e., top), third, fifth, etc. rows of a decoded frame are the top field rows. The second, fourth, sixth, etc. rows of a decoded frame are the bottom field rows. The first (i.e., top) row is numbered as row number 0; the second row is numbered as row number 1; the third row is numbered as row number 2, etc. A top field consists of only the top field rows of a frame, and a bottom field consists of only the bottom field rows of a frame. When the top field or bottom field of a decoded frame is used as a reference field (see below) only the even-numbered rows (for a top field) or the odd-numbered rows (for a bottom field) of the decoded frame are used.

The nominal vertical and horizontal relative locations of luma and chroma samples in top and bottom fields are shown in Figure 6-2. The nominal vertical sampling relative locations of the chroma samples in a top field are specified as shifted up by one-quarter luma sample height relative to the field-sampling grid. The vertical sampling locations of the chroma samples in a bottom field are specified as shifted down by one-quarter luma sample height relative to the field-sampling grid. Alternative chroma sample relative locations may be indicated in the video usability information (see Annex E).

NOTE – The shifting of the chroma samples is in order for these samples to align vertically to the usual location relative to the full-frame sampling grid as shown in Figure 6-1.

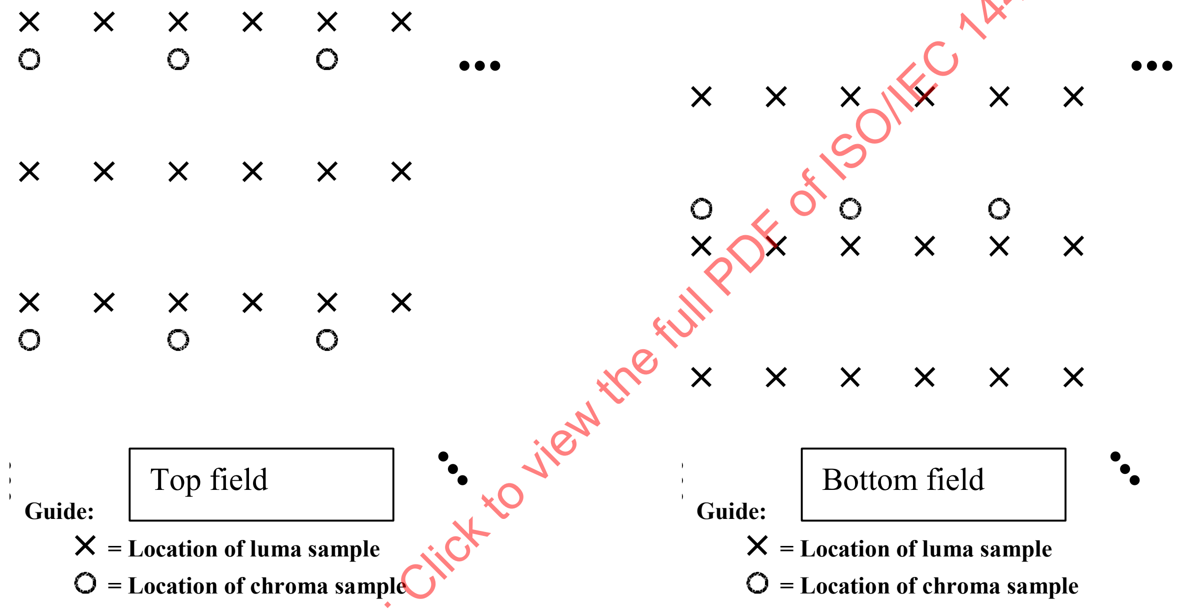

Top field
Guide:
× = Location of luma sample
○ = Location of chroma sample
Bottom field
Guide:
× = Location of luma sample
○ = Location of chroma sample

Figure 6-2 – Nominal vertical and horizontal sampling locations of samples top and bottom fields.

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, or, when macroblock-adaptive frame/field decoding is in use, a sequence of macroblock pairs.

Each macroblock is comprised of one 16x16 luma and two 8x8 chroma sample arrays. When macroblock-adaptive frame/field decoding is not in use, each macroblock represents a spatial rectangular region of the picture. For example, a picture may be divided into two slices as shown in Figure 6-3.

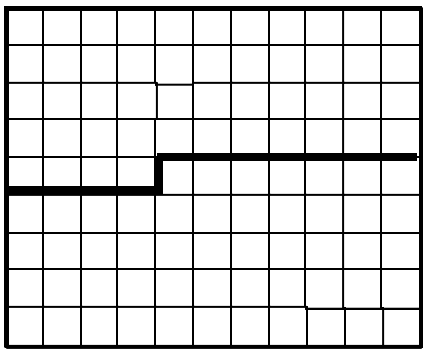


Figure 6-3 – A picture with 11 by 9 macroblocks that is partitioned into two slices

When macroblock-adaptive frame/field decoding is in use, the picture is partitioned into slices containing an integer number of macroblock pairs as shown in Figure 6-4. Each macroblock pair consists of two macroblocks.

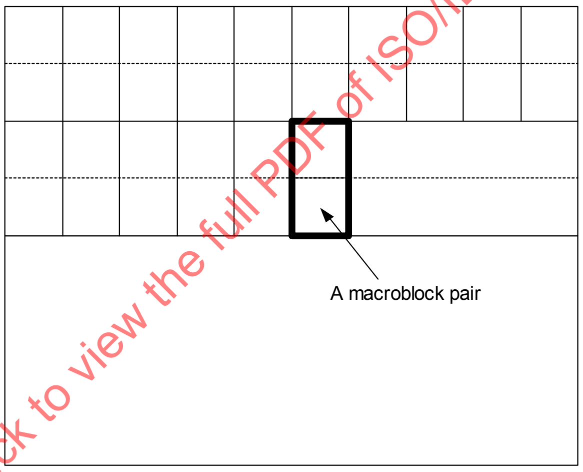


Figure 6-4 – Partitioning of the decoded frame into macroblock pairs.

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.

- If MbaffFrameFlag is equal to 0,

$$x = \text{InverseRasterScan}(\ \text{mbAddr}, 16, 16, \text{PicWidthInSamples}_L, 0\ ) \qquad (6\text{-}1)$$

$$y = \text{InverseRasterScan}(\ \text{mbAddr}, 16, 16, \text{PicWidthInSamples}_L, 1\ ) \qquad (6\text{-}2)$$

- Otherwise (MbaffFrameFlag is equal to 1), the following applies.

$$xO = \text{InverseRasterScan}(\ mbAddr / 2, 16, 32, PicWidthInSamples_L, 0\ ) \qquad (6\text{-}3)$$

$$yO = \text{InverseRasterScan}(\ mbAddr / 2, 16, 32, PicWidthInSamples_L, 1\ ) \qquad (6\text{-}4)$$

Depending on the current macroblock the following applies.

- If the current macroblock is a frame macroblock

$$x = xO \qquad (6\text{-}5)$$

$$y = yO + (\ mbAddr \% 2\ ) * 16 \qquad (6\text{-}6)$$

- Otherwise (the current macroblock is a field macroblock),

$$x = xO \qquad (6\text{-}7)$$

$$y = yO + (\ mbAddr \% 2\ ) \qquad (6\text{-}8)$$

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 6-5. 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 Table 7-10, Table 7-11, Table 7-14, and Table 7-15. 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, P_8x8ref0, or B_8x8, depending on the sub-macroblock type.

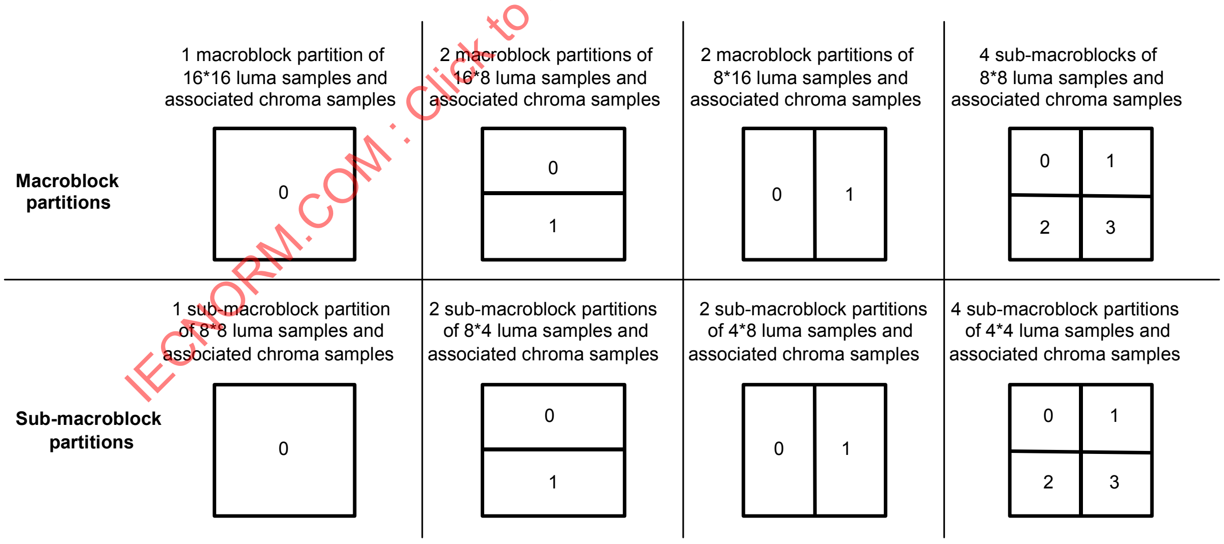

1 macroblock partition of 16*16 luma samples and associated chroma samples
2 macroblock partitions of 16*8 luma samples and associated chroma samples
2 macroblock partitions of 8*16 luma samples and associated chroma samples
4 sub-macroblocks of 8*8 luma samples and associated chroma samples
Macroblock partitions
0
0
1
0
1
0
1
2
3
1 sub-macroblock partition of 8*8 luma samples and associated chroma samples
2 sub-macroblock partitions of 8*4 luma samples and associated chroma samples
2 sub-macroblock partitions of 4*8 luma samples and associated chroma samples
4 sub-macroblock partitions of 4*4 luma samples and associated chroma samples
Sub-macroblock partitions
0
0
1
0
1
0
1
2
3

Figure 6-5 – 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) (6-9)

y = InverseRasterScan(mbPartIdx, MbPartWidth(mb_type), MbPartHeight(mb_type), 16, 1) (6-10)

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, P_8x8ref0, or B_8x8,

  x = InverseRasterScan(subMbPartIdx, SubMbPartWidth(sub_mb_type[mbPartIdx]),
  SubMbPartHeight(sub_mb_type[mbPartIdx]), 8, 0) (6-11)

  y = InverseRasterScan(subMbPartIdx, SubMbPartWidth(sub_mb_type[mbPartIdx]),
  SubMbPartHeight(sub_mb_type[mbPartIdx]), 8, 1) (6-12)

- Otherwise,

  x = InverseRasterScan(subMbPartIdx, 4, 4, 8, 0) (6-13)

  y = InverseRasterScan(subMbPartIdx, 4, 4, 8, 1) (6-14)

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 6-6 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 6-6 – 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) (6-15)

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

6.4.4 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 one of the following conditions is true in which case the macroblock shall be marked as not available:

- mbAddr < 0
- mbAddr > CurrMbAddr
- the macroblock with address mbAddr belongs to a different slice than the current slice

6.4.5 Derivation process for neighbouring macroblock addresses and their availability

This process can only be invoked when MbaffFrameFlag is equal to 0.

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 6-7 shows the relative spatial locations of the macroblocks with mbAddrA, mbAddrB, mbAddrC, and mbAddrD relative to the current macroblock with CurrMbAddr.

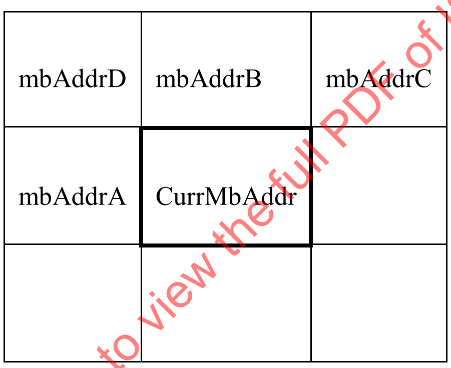


Figure 6-7 – Neighbouring macroblocks for a given macroblock

Input to the process in subclause 6.4.4 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 6.4.4 is mbAddrB = CurrMbAddr – PicWidthInMbs and the output is whether the macroblock mbAddrB is available.

Input to the process in subclause 6.4.4 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 6.4.4 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.

6.4.6 Derivation process for neighbouring macroblock addresses and their availability in MBAFF frames

This process can only be invoked when MbaffFrameFlag is equal to 1.

The outputs of this process are

- mbAddrA: the address and availability status of the top macroblock of the macroblock pair to the left of the current macroblock pair.
- mbAddrB: the address and availability status of the top macroblock of the macroblock pair above the current macroblock pair.

- mbAddrC: the address and availability status of the top macroblock of the macroblock pair above-right of the current macroblock pair.
- mbAddrD: the address and availability status of the top macroblock of the macroblock pair above-left of the current macroblock pair.

Figure 6-8 shows the relative spatial locations of the macroblocks with mbAddrA, mbAddrB, mbAddrC, and mbAddrD relative to the current macroblock with CurrMbAddr.

mbAddrA, mbAddrB, mbAddrC, and mbAddrD have identical values regardless whether the current macroblock is the top or the bottom macroblock of a macroblock pair.

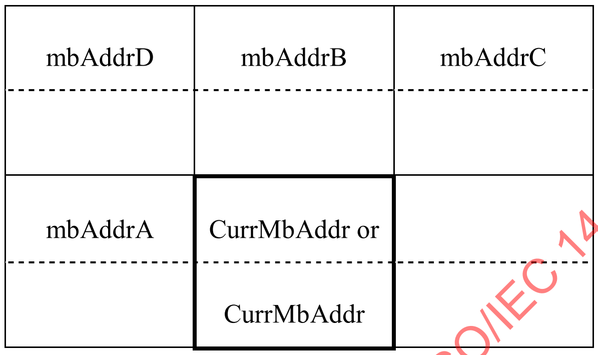


Figure 6-8 – Neighbouring macroblocks for a given macroblock in MBAFF frames

Input to the process in subclause 6.4.4 is mbAddrA =2 * (CurrMbAddr / 2 – 1) and the output is whether the macroblock mbAddrA is available. In addition, mbAddrA is marked as not available when (CurrMbAddr / 2) % PicWidthInMbs is equal to 0.

Input to the process in subclause 6.4.4 is mbAddrB =2 * (CurrMbAddr / 2 – PicWidthInMbs) and the output is whether the macroblock mbAddrB is available.

Input to the process in subclause 6.4.4 is mbAddrC = 2 * (CurrMbAddr / 2 – PicWidthInMbs + 1) and the output is whether the macroblock mbAddrC is available. In addition, mbAddrC is marked as not available when (CurrMbAddr / 2 + 1) % PicWidthInMbs is equal to 0.

Input to the process in subclause 6.4.4 is mbAddrD = 2 * (CurrMbAddr / 2 – PicWidthInMbs - 1) and the output is whether the macroblock mbAddrD is available. In addition, mbAddrD is marked as not available when (CurrMbAddr / 2) % PicWidthInMbs is equal to 0.

6.4.7 Derivation processes for neighbouring macroblocks, blocks, and partitions

Subclause 6.4.7.1 specifies the derivation process for neighbouring macroblocks.

Subclause 6.4.7.2 specifies the derivation process for neighbouring 8x8 luma blocks.

Subclause 6.4.7.3 specifies the derivation process for neighbouring 4x4 luma blocks.

Subclause 6.4.7.4 specifies the derivation process for neighbouring 4x4 chroma blocks.

Subclause 6.4.7.5 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, and chroma4x4BlkIdxN for the output. These input and output assignments are used in subclauses 6.4.7.1 to 6.4.7.5. The variable predPartWidth is specified when Table 6-2 is referred to.

Table 6-2 – Specification of input and output assignments for subclauses 6.4.7.1 to 6.4.7.5

N	xD	yD
A	-1	0

B	0	-1
C	predPartWidth	-1
D	-1	-1

Figure 6-9 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.

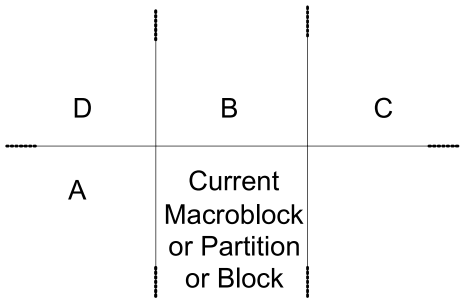


Figure 6-9 – Determination of the neighbouring macroblock, blocks, and partitions (informative)

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

mbAddrN (with N being A or B) is derived as follows.

- The difference of luma location (xD, yD) is set according to Table 6-2.
- The derivation process for neighbouring locations as specified in subclause 6.4.8 is invoked for luma locations with (xN, yN) equal to (xD, yD), and the output is assigned to mbAddrN.

6.4.7.2 Derivation process for neighbouring 8x8 luma block

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

The luma8x8BlkIdx specifies the 8x8 luma blocks of a macroblock in a raster scan.

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,
- luma8x8BlkIdxA: the index of the 8x8 luma block to the left of the 8x8 block with index luma8x8BlkIdx and its availability status,
- mbAddrB: either equal to CurrMbAddr or the address of the macroblock above the current macroblock and its availability status,
- luma8x8BlkIdxB: the index of the 8x8 luma block above the 8x8 block with index luma8x8BlkIdx and its availability status.

mbAddrN and luma8x8BlkIdxN (with N being A or B) are derived as follows.

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

$$xN = (luma8x8BlkIdx \% 2) * 8 + xD \quad (6-17)$$

$$yN = (luma8x8BlkIdx / 2) * 8 + yD \quad (6-18)$$

- The derivation process for neighbouring locations as specified in subclause 6.4.8 is invoked for luma locations with (xN, yN) as the input and the output is assigned to mbAddrN and (xW, yW).
- The variable luma8x8BlkIdxN is derived as follows.
  - If mbAddrN is not available, luma8x8BlkIdxN is marked as not available.
  - Otherwise (mbAddrN is available), the 8x8 luma block in the macroblock mbAddrN covering the luma location (xW, yW) shall be assigned to luma8x8BlkIdxN.

6.4.7.3 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 follows.

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

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

$$yN = y + yD \quad (6-20)$$

- The derivation process for neighbouring locations as specified in subclause 6.4.8 is invoked for luma locations with (xN, yN) as the input and the output is assigned to mbAddrN and (xW, yW).
- The variable luma4x4BlkIdxN is derived as follows.
  - If mbAddrN is not available, luma4x4BlkIdxN is marked as not available.
  - Otherwise (mbAddrN is available), the 4x4 luma block in the macroblock mbAddrN covering the luma location (xW, yW) shall be assigned to luma4x4BlkIdxN.

6.4.7.4 Derivation process for neighbouring 4x4 chroma blocks

Input to this is a current 4x4 chroma block 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 chroma 4x4 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 chroma 4x4 block index chroma4x4BlkIdx and its availability status.

The derivation process for neighbouring 8x8 luma block is invoked with luma8x8BlkIdx = chroma4x4BlkIdx as the input and with mbAddrA, chroma4x4BlkIdxA = luma8x8BlkIdxA, mbAddrB, and chroma4x4BlkIdxB = luma8x8BlkIdxB as the output.

6.4.7.5 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 subMbPartIdx (with N being A, B, C, or D) are derived as follows.

- 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.
- 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, P_8x8ref0 or B_8x8, 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).
- The variable predPartWidth in Table 6-2 is specified as follows.
  - If mb_type is equal to P_Skip, B_Skip, or B_Direct_16x16, predPartWidth = 16.
  - Otherwise, if mb_type is equal to B_8x8, the following applies.
    - If currSubMbType is equal to B_Direct_8x8, predPartWidth = 16.
      NOTE – When currSubMbType is equal to B_Direct_8x8 and direct_spatial_mv_pred_flag is equal to 1, the predicted motion vector is the predicted motion vector for the complete macroblock.
    - Otherwise, predPartWidth = SubMbPartWidth(sub_mb_type[mbPartIdx]).
  - Otherwise, if mb_type is equal to P_8x8 or P_8x8ref0, predPartWidth = SubMbPartWidth(sub_mb_type[mbPartIdx]).
  - Otherwise, predPartWidth = MbPartWidth(mb_type).
- The difference of luma location (xD, yD) is set according to Table 6-2.
- The neighbouring luma location (xN, yN) is specified by

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

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

- The derivation process for neighbouring locations as specified in subclause 6.4.8 is invoked for luma locations with (xN, yN) as the input and the output is assigned to mbAddrN and (xW, yW).
- 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 applies.
    - The macroblock partition in the macroblock mbAddrN covering the luma location (xW, yW) shall be assigned to mbPartIdxN and the sub-macroblock partition inside the macroblock partition mbPartIdxN covering the sample (xW, yW) in the macroblock mbAddrN shall be assigned to subMbPartIdxN.
    - 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.

6.4.8 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 maxWH be a variable specifying a maximum value of the location components xN, yN, xW, and yW. maxWH is derived as follows.

- If this process is invoked for neighbouring luma locations,

$$\text{maxWH} = 16 \qquad (6\text{-}23)$$

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

$$\text{maxWH} = 8 \qquad (6\text{-}24)$$

Depending on the variable MbaffFrameFlag, the neighbouring luma locations are derived as follows.

- If MbaffFrameFlag is equal to 0, the specification for neighbouring luma locations in fields and non-MBAFF frames as described in subclause 6.4.8.1 is applied.
- Otherwise (MbaffFrameFlag is equal to 1), the specification for neighbouring luma locations in MBAFF frames as described in subclause 6.4.8.2 is applied.

6.4.8.1 Specification for neighbouring luma locations in fields and non-MBAFF frames

The specifications in this subclause are applied when MbaffFrameFlag is equal to 0.

The derivation process for neighbouring macroblock addresses and their availability in subclause 6.4.5 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 .. maxWH - 1	mbAddrA
0 .. maxWH - 1	< 0	mbAddrB

0 .. maxWH - 1	0 .. maxWH - 1	CurrMbAddr
> maxWH - 1	< 0	mbAddrC
> maxWH - 1	0 .. maxWH - 1	not available
	> maxWH - 1	not available

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

$$xW = (xN + maxWH) \% maxWH \qquad (6\text{-}25)$$

$$yW = (yN + maxWH) \% maxWH \qquad (6\text{-}26)$$

6.4.8.2 Specification for neighbouring luma locations in MBAFF frames

The specifications in this subclause are applied when MbaffFrameFlag is equal to 1.

The derivation process for neighbouring macroblock addresses and their availability in subclause 6.4.6 is invoked with mbAddrA, mbAddrB, mbAddrC, and mbAddrD as well as their availability status as the output.

Table 6-4 specifies the macroblock addresses mbAddrN and yM in two ordered steps:

1. Specification of a macroblock address mbAddrX depending on (xN, yN) and the following variables:
   - The variable currMbFrameFlag is derived as follows.
     - If the macroblock with address CurrMbAddr is a frame macroblock, currMbFrameFlag is set equal to 1,
     - Otherwise (the macroblock with address CurrMbAddr is a field macroblock), currMbFrameFlag is set equal to 0.
   - The variable mbIsTopMbFlag is derived as follows.
     - If the macroblock with address CurrMbAddr is a top macroblock (CurrMbAddr % 2 is equal to 0), mbIsTopMbFlag is set equal to 1;
     - Otherwise (the macroblock with address CurrMbAddr is a bottom macroblock, CurrMbAddr % 2 is equal to 1), mbIsTopMbFlag is set equal to 0.
2. Depending on the availability of mbAddrX, the following applies.
   - If mbAddrX is not available, mbAddrN is marked as not available.
   - Otherwise (mbAddrX is available), mbAddrN is marked as available and Table 6-4 specifies mbAddrN and yM depending on (xN, yN), currMbFrameFlag, mbIsTopMbFlag, and the variable mbAddrXFrameFlag, which is derived as follows.
     - If the macroblock mbAddrX is a frame macroblock, mbAddrXFrameFlag is set equal to 1,
     - Otherwise (the macroblock mbAddrX is a field macroblock), mbAddrXFrameFlag is set equal to 0.

Unspecified values (na) of the above flags in Table 6-4 indicate that the value of the corresponding flag is not relevant for the current table rows.

Table 6-4 - Specification of mbAddrN and yM

xN	yN	currMbFrameFlag	mbIsTopMbFlag	mbAddrX	mbAddrXFrameFlag	additional condition	mbAddrN	yM
< 0	< 0	1	1	mbAddrD			mbAddrD + 1	yN
			0	mbAddrA	1		mbAddrA	yN
					0		mbAddrA + 1	(yN + maxWH) >> 1
		0	1	mbAddrD	1		mbAddrD + 1	2*yN
					0		mbAddrD	yN
			0	mbAddrD			mbAddrD + 1	yN
< 0	0 .. maxWH - 1	1	1	mbAddrA	1		mbAddrA	yN
					0	yN % 2 = = 0	mbAddrA	yN >> 1
						yN % 2 != 0	mbAddrA + 1	yN >> 1
			0	mbAddrA	1		mbAddrA + 1	yN
					0	yN % 2 = = 0	mbAddrA	(yN + maxWH) >> 1
						yN % 2 != 0	mbAddrA + 1	(yN + maxWH) >> 1
		0	1	mbAddrA	1	yN < (maxWH / 2)	mbAddrA	yN <<1
						yN >= (maxWH / 2)	mbAddrA + 1	(yN <<1) - maxWH
					0		mbAddrA	yN
			0	mbAddrA	1	yN < (maxWH / 2)	mbAddrA	(yN <<1) + 1
						yN >= (maxWH / 2)	mbAddrA + 1	(yN <<1) + 1 – maxWH
					0		mbAddrA + 1	yN
0 .. maxWH - 1	< 0	1	1	mbAddrB			mbAddrB + 1	yN
			0	CurrMbAddr			CurrMbAddr - 1	yN
		0	1	mbAddrB	1		mbAddrB + 1	2 * yN
					0		mbAddrB	yN
			0	mbAddrB			mbAddrB + 1	yN
0 .. maxWH - 1	0 .. maxWH - 1			CurrMbAddr			CurrMbAddr	yN
> maxWH - 1	<0	1	1	mbAddrC			mbAddrC + 1	yN
			0	not available			not available	na
		0	1	mbAddrC	1		mbAddrC + 1	2 * yN
					0		mbAddrC	yN
			0	mbAddrC			mbAddrC + 1	yN
> maxWH - 1	0 .. maxWH - 1			not available			not available	na
	> maxWH - 1			not available			not available	na

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

$$xW = (xN + maxWH) \% maxWH \quad (6\text{-}27)$$

$$yW = (yM + maxWH) \% maxWH \quad (6\text{-}28)$$

7 Syntax and semantics

7.1 Method of describing syntax in tabular form

The syntax tables describe a superset of the syntax of all allowed input bitstreams. Additional constraints on the syntax may be specified in other clauses.

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

The following table lists examples of pseudo code used to describe the syntax. When syntax_element appears, it specifies that a data element is read (extracted) from the bitstream and the bitstream pointer.

	C	Descriptor
/* A statement can be a syntax element with an associated syntax category and 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	3	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 more data in an RBSP before rbsp_trailing_bits( ), 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.

Categories (labelled in the table as C) specify the partitioning of slice data into at most three slice data partitions. Slice data partition A contains all syntax elements of category 2. Slice data partition B contains all syntax elements of category 3. Slice data partition C contains all syntax elements of category 4. The meaning of other category values is not specified. For some syntax elements, two category values, separated by a vertical bar, are used. In these cases, the category value to be applied is further specified in the text. For syntax structures used within other syntax structures, the categories of all syntax elements found within the included syntax structure are listed, separated by a vertical bar. A syntax element or syntax structure with category marked as "All" is present within all syntax structures that include that syntax element or syntax structure. For syntax structures used within other syntax structures, a numeric category value provided in a syntax table at the location of the inclusion of a syntax structure containing a syntax element with category marked as "All" is considered to apply to the syntax elements with category "All".

The following descriptors specify the parsing process of each syntax element. For some syntax elements, two descriptors, separated by a vertical bar, are used. In these cases, the left descriptors apply when entropy_coding_mode_flag is equal to 0 and the right descriptor applies when entropy_coding_mode_flag is equal to 1.

- ae(v): context-adaptive arithmetic entropy-coded syntax element. The parsing process for this descriptor is specified in subclause 9.3.
- 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) {	**C**	**Descriptor**
forbidden_zero_bit	All	f(1)
nal_ref_idc	All	u(2)
nal_unit_type	All	u(5)
NumBytesInRBSP = 0		
for(i = 1; i < NumBytesInNALunit; i++) {		
if(i + 2 < NumBytesInNALunit && next_bits(24) = = 0x000003) {		
rbsp_byte[NumBytesInRBSP++ **]**	All	b(8)
rbsp_byte[NumBytesInRBSP++ **]**	All	b(8)
i += 2		
emulation_prevention_three_byte /* equal to 0x03 */	All	f(8)
} else		
rbsp_byte[NumBytesInRBSP++ **]**	All	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( ) {	**C**	**Descriptor**
profile_idc	0	u(8)
constraint_set0_flag	0	u(1)
constraint_set1_flag	0	u(1)
constraint_set2_flag	0	u(1)
reserved_zero_5bits /* equal to 0 */	0	u(5)
level_idc	0	u(8)
seq_parameter_set_id	0	ue(v)
log2_max_frame_num_minus4	0	ue(v)
pic_order_cnt_type	0	ue(v)
if(pic_order_cnt_type = = 0)		
log2_max_pic_order_cnt_lsb_minus4	0	ue(v)
else if(pic_order_cnt_type = = 1) {		
delta_pic_order_always_zero_flag	0	u(1)
offset_for_non_ref_pic	0	se(v)
offset_for_top_to_bottom_field	0	se(v)
num_ref_frames_in_pic_order_cnt_cycle	0	ue(v)
for(i = 0; i < num_ref_frames_in_pic_order_cnt_cycle; i++)		
offset_for_ref_frame[i **]**	0	se(v)
}		
num_ref_frames	0	ue(v)
gaps_in_frame_num_value_allowed_flag	0	u(1)
pic_width_in_mbs_minus1	0	ue(v)
pic_height_in_map_units_minus1	0	ue(v)
frame_mbs_only_flag	0	u(1)
if(!frame_mbs_only_flag)		
mb_adaptive_frame_field_flag	0	u(1)
direct_8x8_inference_flag	0	u(1)
frame_cropping_flag	0	u(1)
if(frame_cropping_flag) {		
frame_crop_left_offset	0	ue(v)
frame_crop_right_offset	0	ue(v)
frame_crop_top_offset	0	ue(v)
frame_crop_bottom_offset	0	ue(v)
}		
vui_parameters_present_flag	0	u(1)
if(vui_parameters_present_flag)		
vui_parameters( )	0	
rbsp_trailing_bits( )	0	
}		

7.3.2.2 Picture parameter set RBSP syntax

pic_parameter_set_rbsp( ) {	C	Descriptor
pic_parameter_set_id	1	ue(v)
seq_parameter_set_id	1	ue(v)
entropy_coding_mode_flag	1	u(1)
pic_order_present_flag	1	u(1)
num_slice_groups_minus1	1	ue(v)
if(num_slice_groups_minus1 > 0) {		
slice_group_map_type	1	ue(v)
if(slice_group_map_type = = 0)		
for(iGroup = 0; iGroup <= num_slice_groups_minus1; iGroup++)		
run_length_minus1[iGroup **]**	1	ue(v)
else if(slice_group_map_type = = 2)		
for(iGroup = 0; iGroup < num_slice_groups_minus1; iGroup++) {		
top_left[iGroup **]**	1	ue(v)
bottom_right[iGroup **]**	1	ue(v)
}		
else if(slice_group_map_type = = 3 || slice_group_map_type = = 4 || slice_group_map_type = = 5) {		
slice_group_change_direction_flag	1	u(1)
slice_group_change_rate_minus1	1	ue(v)
} else if(slice_group_map_type = = 6) {		
pic_size_in_map_units_minus1	1	ue(v)
for(i = 0; i <= pic_size_in_map_units_minus1; i++)		
slice_group_id[i **]**	1	u(v)
}		
}		
num_ref_idx_l0_active_minus1	1	ue(v)
num_ref_idx_l1_active_minus1	1	ue(v)
weighted_pred_flag	1	u(1)
weighted_bipred_idc	1	u(2)
pic_init_qp_minus26 /* relative to 26 */	1	se(v)
pic_init_qs_minus26 /* relative to 26 */	1	se(v)
chroma_qp_index_offset	1	se(v)
deblocking_filter_control_present_flag	1	u(1)
constrained_intra_pred_flag	1	u(1)
redundant_pic_cnt_present_flag	1	u(1)
rbsp_trailing_bits( )	1	
}		

7.3.2.3 Supplemental enhancement information RBSP syntax

sei_rbsp( ) {	C	Descriptor
do		
sei_message( )	5	
while(more_rbsp_data( ))		
rbsp_trailing_bits( )	5	
}		

7.3.2.3.1 Supplemental enhancement information message syntax

sei_message( ) {	C	Descriptor
payloadType = 0		
while(next_bits(8) = = 0xFF) {		
ff_byte /* equal to 0xFF */	5	f(8)
payloadType += 255		
}		
last_payload_type_byte	5	u(8)
payloadType += last_payload_type_byte		
payloadSize = 0		
while(next_bits(8) = = 0xFF) {		
ff_byte /* equal to 0xFF */	5	f(8)
payloadSize += 255		
}		
last_payload_size_byte	5	u(8)
payloadSize += last_payload_size_byte		
sei_payload(payloadType, payloadSize)	5	
}		

7.3.2.4 Access unit delimiter RBSP syntax

access_unit_delimiter_rbsp( ) {	C	Descriptor
primary_pic_type	6	u(3)
rbsp_trailing_bits( )	6	
}		

7.3.2.5 End of sequence RBSP syntax

end_of_seq_rbsp( ) {	C	Descriptor
}		

7.3.2.6 End of stream RBSP syntax

end_of_stream_rbsp( ) {	**C**	**Descriptor**
}		

7.3.2.7 Filler data RBSP syntax

filler_data_rbsp(NumBytesInRBSP) {	**C**	**Descriptor**
while(next_bits(8) = = 0xFF)		
ff_byte /* equal to 0xFF */	9	f(8)
rbsp_trailing_bits( )	9	
}		

7.3.2.8 Slice layer without partitioning RBSP syntax

slice_layer_without_partitioning_rbsp( ) {	**C**	**Descriptor**
slice_header( )	2	
slice_data( ) /* all categories of slice_data( ) syntax */	2 | 3 | 4	
rbsp_slice_trailing_bits( )	2	
}		

7.3.2.9 Slice data partition RBSP syntax

7.3.2.9.1 Slice data partition A RBSP syntax

slice_data_partition_a_layer_rbsp( ) {	**C**	**Descriptor**
slice_header( )	2	
slice_id	2	ue(v)
slice_data( ) /* only category 2 parts of slice_data( ) syntax */	2	
rbsp_slice_trailing_bits( )	2	
}		

7.3.2.9.2 Slice data partition B RBSP syntax

slice_data_partition_b_layer_rbsp( ) {	**C**	**Descriptor**
slice_id	3	ue(v)
if(redundant_pic_cnt_present_flag)		
redundant_pic_cnt	3	ue(v)
slice_data( ) /* only category 3 parts of slice_data( ) syntax */	3	
rbsp_slice_trailing_bits( )	3	
}		

7.3.2.9.3 Slice data partition C RBSP syntax

slice_data_partition_c_layer_rbsp( ) {	C	Descriptor
slice_id	4	ue(v)
if(redundant_pic_cnt_present_flag)		
redundant_pic_cnt	4	ue(v)
slice_data( ) /* only category 4 parts of slice_data( ) syntax */	4	
rbsp_slice_trailing_bits( )	4	
}		

7.3.2.10 RBSP slice trailing bits syntax

rbsp_slice_trailing_bits( ) {	C	Descriptor
rbsp_trailing_bits( )	All	
if(entropy_coding_mode_flag)		
while(more_rbsp_trailing_data( ))		
cabac_zero_word /* equal to 0x0000 */	All	f(16)
}		

7.3.2.11 RBSP trailing bits syntax

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

7.3.3 Slice header syntax

slice_header( ) {	C	Descriptor
first_mb_in_slice	2	ue(v)
slice_type	2	ue(v)
pic_parameter_set_id	2	ue(v)
frame_num	2	u(v)
if(!frame_mbs_only_flag) {		
field_pic_flag	2	u(1)
if(field_pic_flag)		
bottom_field_flag	2	u(1)
}		
if(nal_unit_type = = 5)		
idr_pic_id	2	ue(v)
if(pic_order_cnt_type = = 0) {		
pic_order_cnt_lsb	2	u(v)
if(pic_order_present_flag && !field_pic_flag)		
delta_pic_order_cnt_bottom	2	se(v)
}		

if(pic_order_cnt_type = = 1 && !delta_pic_order_always_zero_flag) {		
delta_pic_order_cnt[0 **]**	2	se(v)
if(pic_order_present_flag && !field_pic_flag)		
delta_pic_order_cnt[1 **]**	2	se(v)
}		
if(redundant_pic_cnt_present_flag)		
redundant_pic_cnt	2	ue(v)
if(slice_type = = B)		
direct_spatial_mv_pred_flag	2	u(1)
if(slice_type = = P | | slice_type = = SP | | slice_type = = B) {		
num_ref_idx_active_override_flag	2	u(1)
if(num_ref_idx_active_override_flag) {		
num_ref_idx_l0_active_minus1	2	ue(v)
if(slice_type = = B)		
num_ref_idx_l1_active_minus1	2	ue(v)
}		
}		
ref_pic_list_reordering( )	2	
if((weighted_pred_flag && (slice_type = = P | | slice_type = = SP)) | | (weighted_bipred_idc = = 1 && slice_type = = B))		
pred_weight_table( )	2	
if(nal_ref_idc != 0)		
dec_ref_pic_marking( )	2	
if(entropy_coding_mode_flag && slice_type != I && slice_type != SI)		
cabac_init_idc	2	ue(v)
slice_qp_delta	2	se(v)
if(slice_type = = SP | | slice_type = = SI) {		
if(slice_type = = SP)		
sp_for_switch_flag	2	u(1)
slice_qs_delta	2	se(v)
}		
if(deblocking_filter_control_present_flag) {		
disable_deblocking_filter_idc	2	ue(v)
if(disable_deblocking_filter_idc != 1) {		
slice_alpha_c0_offset_div2	2	se(v)
slice_beta_offset_div2	2	se(v)
}		
}		
if(num_slice_groups_minus1 > 0 && slice_group_map_type >= 3 && slice_group_map_type <= 5)		
slice_group_change_cycle	2	u(v)
}		

7.3.3.1 Reference picture list reordering syntax

ref_pic_list_reordering( ) {	C	Descriptor
if(slice_type != I && slice_type != SI) {		
ref_pic_list_reordering_flag_l0	2	u(1)
if(ref_pic_list_reordering_flag_l0)		
do {		
reordering_of_pic_nums_idc	2	ue(v)
if(reordering_of_pic_nums_idc = = 0 || reordering_of_pic_nums_idc = = 1)		
abs_diff_pic_num_minus1	2	ue(v)
else if(reordering_of_pic_nums_idc = = 2)		
long_term_pic_num	2	ue(v)
} while(reordering_of_pic_nums_idc != 3)		
}		
if(slice_type = = B) {		
ref_pic_list_reordering_flag_l1	2	u(1)
if(ref_pic_list_reordering_flag_l1)		
do {		
reordering_of_pic_nums_idc	2	ue(v)
if(reordering_of_pic_nums_idc = = 0 || reordering_of_pic_nums_idc = = 1)		
abs_diff_pic_num_minus1	2	ue(v)
else if(reordering_of_pic_nums_idc = = 2)		
long_term_pic_num	2	ue(v)
} while(reordering_of_pic_nums_idc != 3)		
}		
}		

7.3.3.2 Prediction weight table syntax

pred_weight_table( ) {	**C**	**Descriptor**
luma_log2_weight_denom	2	ue(v)
chroma_log2_weight_denom	2	ue(v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag	2	u(1)
if(luma_weight_l0_flag) {		
luma_weight_l0[i]	2	se(v)
luma_offset_l0[i]	2	se(v)
}		
chroma_weight_l0_flag	2	u(1)
if(chroma_weight_l0_flag)		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
if(slice_type = = B)		
for(i = 0; i <= num_ref_idx_l1_active_minus1; i++) {		
luma_weight_l1_flag	2	u(1)
if(luma_weight_l1_flag) {		
luma_weight_l1[i]	2	se(v)
luma_offset_l1[i]	2	se(v)
}		
chroma_weight_l1_flag	2	u(1)
if(chroma_weight_l1_flag)		
for(j = 0; j < 2; j++) {		
chroma_weight_l1[i][j]	2	se(v)
chroma_offset_l1[i][j]	2	se(v)
}		
}		
}		

7.3.3.3 Decoded reference picture marking syntax

dec_ref_pic_marking( ) {	**C**	**Descriptor**
if(nal_unit_type = = 5) {		
no_output_of_prior_pics_flag	2 | 5	u(1)
long_term_reference_flag	2 | 5	u(1)
} else {		
adaptive_ref_pic_marking_mode_flag	2 | 5	u(1)
if(adaptive_ref_pic_marking_mode_flag)		
do {		
memory_management_control_operation	2 | 5	ue(v)
if(memory_management_control_operation = = 1 | | memory_management_control_operation = = 3)		
difference_of_pic_nums_minus1	2 | 5	ue(v)
if(memory_management_control_operation = = 2)		
long_term_pic_num	2 | 5	ue(v)
if(memory_management_control_operation = = 3 | | memory_management_control_operation = = 6)		
long_term_frame_idx	2 | 5	ue(v)
if(memory_management_control_operation = = 4)		
max_long_term_frame_idx_plus1	2 | 5	ue(v)
} while(memory_management_control_operation != 0)		
}		
}		

7.3.4 Slice data syntax

slice_data( ) {	**C**	**Descriptor**
if(entropy_coding_mode_flag)		
while(!byte_aligned( ))		
cabac_alignment_one_bit	2	f(1)
CurrMbAddr = first_mb_in_slice * (1 + MbaffFrameFlag)		
moreDataFlag = 1		
prevMbSkipped = 0		
do {		
if(slice_type != I && slice_type != SI)		
if(!entropy_coding_mode_flag) {		
mb_skip_run	2	ue(v)
prevMbSkipped = (mb_skip_run > 0)		
for(i=0; i<mb_skip_run; i++)		
CurrMbAddr = NextMbAddress(CurrMbAddr)		
moreDataFlag = more_rbsp_data( )		
} else {		
mb_skip_flag	2	ae(v)
moreDataFlag = !mb_skip_flag		
}		
if(moreDataFlag) {		

if(MbaffFrameFlag && (CurrMbAddr % 2 = = 0 ||
                (CurrMbAddr % 2 = = 1 && prevMbSkipped)))		

mb_field_decoding_flag	2	u(1) | ae(v)
macroblock_layer( )	2 | 3 | 4	
}		
if(!entropy_coding_mode_flag)		
moreDataFlag = more_rbsp_data( )		
else {		
if(slice_type != I && slice_type != SI)		
prevMbSkipped = mb_skip_flag		
if(MbaffFrameFlag && CurrMbAddr % 2 = = 0)		
moreDataFlag = 1		
else {		
end_of_slice_flag	2	ae(v)
moreDataFlag = !end_of_slice_flag		
}		
}		
CurrMbAddr = NextMbAddress(CurrMbAddr)		
} while(moreDataFlag)		
}		

7.3.5 Macroblock layer syntax

macroblock_layer( ) {	**C**	**Descriptor**
mb_type	2	ue(v) | ae(v)
if(mb_type = = I_PCM) {		
while(!byte_aligned( ))		
pcm_alignment_zero_bit	2	f(1)
for(i = 0; i < 256 * ChromaFormatFactor; i++)		
pcm_byte[i **]**	2	u(8)
} else {		
if(MbPartPredMode(mb_type, 0) != Intra_4x4 && MbPartPredMode(mb_type, 0) != Intra_16x16 && NumMbPart(mb_type) = = 4)		
sub_mb_pred(mb_type)	2	
else		
mb_pred(mb_type)	2	
if(MbPartPredMode(mb_type, 0) != Intra_16x16)		
coded_block_pattern	2	me(v) | ae(v)
if(CodedBlockPatternLuma > 0 || CodedBlockPatternChroma > 0 || MbPartPredMode(mb_type, 0) = = Intra_16x16) {		
mb_qp_delta	2	se(v) | ae(v)
residual( )	3 | 4	
}		
}		
}		

7.3.5.1 Macroblock prediction syntax

mb_pred(mb_type) {	C	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 **]**	2	u(1) | ae(v)
if(!prev_intra4x4_pred_mode_flag[luma4x4BlkIdx])		
rem_intra4x4_pred_mode[luma4x4BlkIdx **]**	2	u(3) | ae(v)
}		
intra_chroma_pred_mode	2	ue(v) | ae(v)
} else if(MbPartPredMode(mb_type, 0) != Direct) {		
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if((num_ref_idx_l0_active_minus1 > 0 || mb_field_decoding_flag) && MbPartPredMode(mb_type, mbPartIdx) != Pred_L1)		
ref_idx_l0[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if((num_ref_idx_l1_active_minus1 > 0 || mb_field_decoding_flag) && MbPartPredMode(mb_type, mbPartIdx) != Pred_L0)		
ref_idx_l1[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode (mb_type, mbPartIdx) != Pred_L1)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l0[mbPartIdx **][** 0 **][** compIdx **]**	2	se(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode(mb_type, mbPartIdx) != Pred_L0)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l1[mbPartIdx **][** 0 **][** compIdx **]**	2	se(v) | ae(v)
}		
}		

7.3.5.2 Sub-macroblock prediction syntax

sub_mb_pred(mb_type) {	**C**	**Descriptor**
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
sub_mb_type[mbPartIdx **]**	2	ue(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if((num_ref_idx_l0_active_minus1 > 0 || mb_field_decoding_flag) && mb_type != P_8x8ref0 && sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L1)		
ref_idx_l0[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if((num_ref_idx_l1_active_minus1 > 0 || mb_field_decoding_flag) && sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L0)		
ref_idx_l1[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L1)		
for(subMbPartIdx = 0; subMbPartIdx < NumSubMbPart(sub_mb_type[mbPartIdx]); subMbPartIdx++)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l0[mbPartIdx **][** subMbPartIdx **][** compIdx **]**	2	se(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L0)		
for(subMbPartIdx = 0; subMbPartIdx < NumSubMbPart(sub_mb_type[mbPartIdx]); subMbPartIdx++)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l1[mbPartIdx **][** subMbPartIdx **][** compIdx **]**	2	se(v) | ae(v)
}		

7.3.5.3 Residual data syntax

residual( ) {	C	Descriptor
if(!entropy_coding_mode_flag)		
residual_block = residual_block_cavlc		
else		
residual_block = residual_block_cabac		
if(MbPartPredMode(mb_type, 0) = = Intra_16x16)		
residual_block(Intra16x16DCLevel, 16)	3	
for(i8x8 = 0; i8x8 < 4; i8x8++) /* each luma 8x8 block */		
for(i4x4 = 0; i4x4 < 4; i4x4++) /* each 4x4 sub-block of block */		
if(CodedBlockPatternLuma & (1 << i8x8)) {		
if(MbPartPredMode(mb_type, 0) = = Intra_16x16)		
residual_block(Intra16x16ACLevel[i8x8 * 4 + i4x4], 15)	3	
else		
residual_block(LumaLevel[i8x8 * 4 + i4x4], 16)	3 | 4	
} else {		
if(MbPartPredMode(mb_type, 0) = = Intra_16x16)		
for(i = 0; i < 15; i++)		
Intra16x16ACLevel[i8x8 * 4 + i4x4][i] = 0		
else		
for(i = 0; i < 16; i++)		
LumaLevel[i8x8 * 4 + i4x4][i] = 0		
}		
for(iCbCr = 0; iCbCr < 2; iCbCr++)		
if(CodedBlockPatternChroma & 3) /* chroma DC residual present */		
residual_block(ChromaDCLevel[iCbCr], 4)	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], 15)	3 | 4	
else		
for(i = 0; i < 15; i++)		
ChromaACLevel[iCbCr][i4x4][i] = 0		
}		

7.3.5.3.1 Residual block CAVLC syntax

residual_block_cavlc(coeffLevel, maxNumCoeff) {	**C**	**Descriptor**
for(i = 0; i < maxNumCoeff; i++)		
coeffLevel[i] = 0		
coeff_token	3 | 4	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	3 | 4	u(1)
level[i] = 1 – 2 * trailing_ones_sign_flag		
} else {		
level_prefix	3 | 4	ce(v)
levelCode = (level_prefix << suffixLength)		
if(suffixLength > 0 || level_prefix >= 14) {		
level_suffix	3 | 4	u(v)
levelCode += level_suffix		
}		
if(level_prefix = = 15 && suffixLength = = 0)		
levelCode += 15		

if(i = = TrailingOnes(coeff_token) &&
           TrailingOnes(coeff_token) < 3)
          levelCode += 2		

if(levelCode % 2 = = 0)		
level[i] = (levelCode + 2) >> 1		
else		
level[i] = (–levelCode – 1) >> 1		
if(suffixLength = = 0)		
suffixLength = 1		

if(Abs(level[i]) > (3 << (suffixLength – 1)) &&
           suffixLength < 6)
          suffixLength++		

}		
if(TotalCoeff(coeff_token) < maxNumCoeff) {		
total_zeros	3 | 4	ce(v)
zerosLeft = total_zeros		
} else		
zerosLeft = 0		
for(i = 0; i < TotalCoeff(coeff_token) – 1; i++) {		
if(zerosLeft > 0) {		
run_before	3 | 4	ce(v)
run[i] = run_before		
} else		

run[i] = 0		
zerosLeft = zerosLeft – run[i]		
}		
run[TotalCoeff(coeff_token) – 1] = zerosLeft		
coeffNum = -1		
for(i = TotalCoeff(coeff_token) – 1; i >= 0; i--) {		
coeffNum += run[i] + 1		
coeffLevel[coeffNum] = level[i]		
}		
}		
}		

7.3.5.3.2 Residual block CABAC syntax

residual_block_cabac(coeffLevel, maxNumCoeff) {	**C**	**Descriptor**
coded_block_flag	3 | 4	ae(v)
if(coded_block_flag) {		
numCoeff = maxNumCoeff		
i = 0		
do {		
significant_coeff_flag[i **]**	3 | 4	ae(v)
if(significant_coeff_flag[i]) {		
last_significant_coeff_flag[i **]**	3 | 4	ae(v)
if(last_significant_coeff_flag[i]) {		
numCoeff = i + 1		
for(j = numCoeff; j < maxNumCoeff; j++)		
coeffLevel[j] = 0		
}		
}		
i++		
} while(i < numCoeff-1)		
coeff_abs_level_minus1[numCoeff-1 **]**	3 | 4	ae(v)
coeff_sign_flag[numCoeff-1 **]**	3 | 4	ae(v)
coeffLevel[numCoeff-1] = (coeff_abs_level_minus1[numCoeff – 1] + 1) * (1 – 2 * coeff_sign_flag[numCoeff – 1])		
for(i = numCoeff-2; i >= 0; i--) {		
if(significant_coeff_flag[i]) {		
coeff_abs_level_minus1[i **]**	3 | 4	ae(v)
coeff_sign_flag[i **]**	3 | 4	ae(v)
coeffLevel[i] = (coeff_abs_level_minus1[i] + 1) * (1 – 2 * coeff_sign_flag[i])		
} else		
coeffLevel[i] = 0		
}		
} else		
for(i = 0; i < maxNumCoeff; i++)		
coeffLevel[i] = 0		
}		

7.4 Semantics

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 Recommendation | 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 or a picture parameter set or a slice of a reference picture or a slice data partition of a reference picture.

nal_ref_idc equal to 0 for a NAL unit containing a slice or slice data partition indicates that the slice or slice data partition 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 slice or slice data partition NAL unit of a particular picture, it shall be equal to 0 for all slice and slice data partition NAL units of the picture.

nal_ref_idc shall not be equal to 0 for IDR NAL units, i.e., 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. VCL NAL units are specified as those NAL units having nal_unit_type equal to 1 to 5, inclusive. All remaining NAL units are called non-VCL NAL units.

The column marked "C" in Table 7-1 lists the categories of the syntax elements that may be present in the NAL unit. In addition, syntax elements with syntax category "All" may be present, as determined by the syntax and semantics of the RBSP data structure. The presence or absence of any syntax elements of a particular listed category is determined from the syntax and semantics of the associated RBSP data structure. nal_unit_type shall not be equal to 3 or 4 unless at least one syntax element is present in the RBSP data structure having a syntax element category value equal to the value of nal_unit_type and not categorized as "All".

Table 7-1 – NAL unit type codes

nal_unit_type	Content of NAL unit and RBSP syntax structure	C
0	Unspecified	
1	Coded slice of a non-IDR picture slice_layer_without_partitioning_rbsp( )	2, 3, 4
2	Coded slice data partition A slice_data_partition_a_layer_rbsp( )	2
3	Coded slice data partition B slice_data_partition_b_layer_rbsp( )	3
4	Coded slice data partition C slice_data_partition_c_layer_rbsp( )	4
5	Coded slice of an IDR picture slice_layer_without_partitioning_rbsp( )	2, 3
6	Supplemental enhancement information (SEI) sei_rbsp( )	5
7	Sequence parameter set seq_parameter_set_rbsp( )	0
8	Picture parameter set pic_parameter_set_rbsp( )	1
9	Access unit delimiter access_unit_delimiter_rbsp( )	6
10	End of sequence end_of_seq_rbsp( )	7
11	End of stream end_of_stream_rbsp( )	8
12	Filler data filler_data_rbsp( )	9
13..23	Reserved	
24..31	Unspecified	

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 Recommendation | 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 Recommendation | International Standard.

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 Recommendation | 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.

When the value of nal_unit_type is equal to 5 for a NAL unit containing a slice of a coded picture, the value of nal_unit_type shall be 5 in all other VCL NAL units of the same coded picture. Such a picture is referred to as an IDR picture.

NOTE – Slice data partitioning cannot be used for IDR pictures.

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 shall contain 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, and
     iii) When the rbsp_stop_one_bit is not the last bit of a byte-aligned byte, one or more rbsp_alignment_zero_bit is present to result in byte alignment.
  3) One or more cabac_zero_word 16-bit syntax elements equal to 0x0000 may be present in some RBSPs after the rbsp_trailing_bits( ) at the end of the RBSP.

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

7.4.1.1 Encapsulation of an SODB within an RBSP (informative)

This subclause does not form an integral part of this Recommendation | 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, and

– to enable a NAL unit to have a size larger than that of the SODB under some circumstances (using one or more cabac_zero_word).

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

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',

and finally, when the last byte of the RBSP data is equal to 0x00 (which can only occur when the RBSP ends in a cabac_zero_word), a final byte equal to 0x03 is appended to the end of the data.

The resulting sequence of bytes is then prefixed with the first byte of the NAL unit containing the indication of the type of RBSP data structure it contains. This 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, and

– 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, D.1, and E.1 specifies the decoding order of syntax elements. Decoders conforming to this Recommendation | International Standard 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

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 or coded slice data partition A 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 or coded slice data partition A 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 shall have the same content as that of the active picture parameter set RBSP unless it follows the last VCL NAL unit of a coded picture and precedes the first VCL NAL unit of another coded picture.

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

When a sequence parameter set RBSP (with a particular value of 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 shall have the same content as that of the active sequence parameter set RBSP unless it follows the last access unit of a 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 Recommendation | International Standard), they must be available to the decoding process in a timely fashion such that these constraints are obeyed.

All constraints that are expressed on the relationship between the values of the syntax elements (and the values of variables derived from those syntax elements) in 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. For interpretation of SEI messages, the values of the parameters of the picture parameter set and sequence parameter set that are active for the operation of the decoding process for the VCL NAL units of the primary coded picture in the same access unit shall be considered in effect unless otherwise specified in the SEI message semantics.

7.4.1.2.2 Order of access units and association to coded video sequences

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

A coded video sequence consists of one or more access units. The order of NAL units and coded pictures and their association to access units is described in subclause 7.4.1.2.3.

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.

The values of picture order count for the coded pictures in consecutive access units in decoding order containing non-reference pictures shall be non-decreasing.

When present, an access unit following an access unit that contains an end of sequence NAL unit shall be an IDR access unit.

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.

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

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

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 13 to 18, inclusive
- 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 is present, the buffering period SEI message shall be the first SEI message payload of the first SEI NAL unit in the access unit
- The primary coded picture shall precede the corresponding redundant coded pictures.
- When redundant coded pictures are present, they shall be ordered in ascending order of the value of redundant_pic_cnt.
- When an end of sequence NAL unit is present, it shall follow the primary coded picture and all redundant coded pictures (if any).
- 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 19 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 31, inclusive, is shown in Figure 7-1.

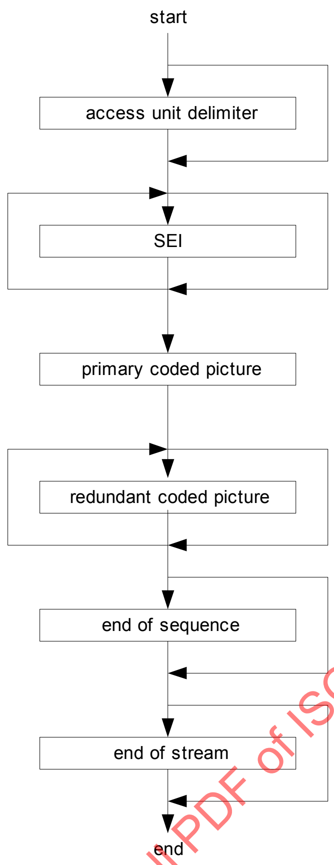


Figure 7-1 – The structure of an access unit not containing any NAL units with nal_unit_type equal to 0, 7, 8, or in the range of 12 to 31, inclusive

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.

Any coded slice NAL unit or coded slice data partition A NAL unit of the primary coded picture of the current access unit shall be different from any coded slice NAL unit or coded slice data partition A 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.
- field_pic_flag differs in value.
- bottom_field_flag is present in both and 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 either pic_order_cnt_lsb differs in value, or delta_pic_order_cnt_bottom 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.

- nal_unit_type differs in value with one of the nal_unit_type values being equal to 5.
- nal_unit_type is equal to 5 for both and idr_pic_id differs in value.

NOTE – Some of the VCL NAL units in redundant coded pictures or some 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

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.

- If arbitrary slice order is allowed as specified in Annex A, coded slice of an IDR picture NAL units may have any order relative to each other.
- Otherwise (arbitrary slice order is not allowed), the order of coded slice of an IDR picture NAL units shall be in the order of increasing macroblock address for the first macroblock of each coded slice of an IDR picture NAL unit.

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

- If arbitrary slice order is allowed as specified in Annex A, coded slice of a non-IDR picture NAL units or coded slice data partition A NAL units may have any order relative to each other. A coded slice data partition A NAL unit with a particular value of slice_id shall precede any present coded slice data partition B NAL unit with the same value of slice_id. A coded slice data partition A NAL unit with a particular value of slice_id shall precede any present coded slice data partition C NAL unit with the same value of slice_id. When a coded slice data partition B NAL unit with a particular value of slice_id is present, it shall precede any present coded slice data partition C NAL unit with the same value of slice_id.
- Otherwise (arbitrary slice order is not allowed), the order of coded slice of a non-IDR picture NAL units or coded slice data partition A NAL units shall be in the order of increasing macroblock address for the first macroblock of each coded slice of a non-IDR picture NAL unit or coded slice data partition A NAL unit. A coded slice data partition A NAL unit with a particular value of slice_id shall immediately precede any present coded slice data partition B NAL unit with the same value of slice_id. A coded slice data partition A NAL unit with a particular value of slice_id shall immediately precede any present coded slice data partition C NAL unit with the same value of slice_id, when a coded slice data partition B NAL unit with the same value of slice_id is not present. When a coded slice data partition B NAL unit with a particular value of slice_id is present, it shall immediately precede any present coded slice data partition C NAL unit with the same value of slice_id.

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 19 to 23, inclusive, which are reserved, shall not precede the first VCL NAL unit of the primary coded picture within the access unit (when specified in the future by ITU-T | ISO/IEC).

7.4.2 Raw byte sequence payloads and RBSP trailing bits semantics

7.4.2.1 Sequence parameter set RBSP semantics

profile_idc and **level_idc** indicate the profile and level to which the bitstream conforms, as specified in Annex A.

constraint_set0_flag equal to 1 indicates that the bitstream obeys all constraints specified in subclause A.2.1. constraint_set0_flag equal to 0 indicates that the bitstream may or may not obey all constraints specified in subclause A.2.1.

constraint_set1_flag equal to 1 indicates that the bitstream obeys all constraints specified in subclause A.2.2. constraint_set1_flag equal to 0 indicates that the bitstream may or may not obey all constraints specified in subclause A.2.2.

constraint_set2_flag equal to 1 indicates that the bitstream obeys all constraints specified in subclause A.2.3. constraint_set2_flag equal to 0 indicates that the bitstream may or may not obey all constraints specified in subclause A.2.3.

NOTE – When more than one of constraint_set0_flag, constraint_set1_flag, or constraint_set2_flag are equal to 1, the bitstream obeys the constraints of all of the indicated subclauses of subclause A.2.

reserved_zero_5bits shall be equal to 0 in bitstreams conforming to this Recommendation | International Standard. Other values of reserved_zero_5bits may be specified in the future by ITU-T | ISO/IEC. Decoders shall ignore the value of reserved_zero_5bits.

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:

$$\text{MaxFrameNum} = 2^{(\text{ log2\_max\_frame\_num\_minus4} + 4\text{ })} \tag{7-1}$$

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 any of the following

- an access unit containing a non-reference frame followed immediately by an access unit containing a non-reference picture
- two access units each containing a field with the two fields together forming a complementary non-reference field pair followed immediately by an access unit containing a non-reference picture
- an access unit containing a non-reference field followed immediately by an access unit containing another non-reference picture that does not form a complementary non-reference field pair with the first of the two access units

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:

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

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 8.2.1. The value of offset_for_non_ref_pic shall be in the range of -2^{31} to $2^{31} - 1$, inclusive.

offset_for_top_to_bottom_field is used to calculate the picture order count of the bottom field in a frame as specified in 8.2.1. The value of offset_for_top_to_bottom_field shall be in the range of -2^{31} 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} to $2^{31} - 1$, inclusive.

num_ref_frames specifies the maximum number of short-term and long-term reference frames, complementary reference field pairs, and non-paired reference fields that may be used by the decoding process for inter prediction of any picture in the sequence. num_ref_frames also determines the size of the sliding window operation as specified in subclause 8.2.5.3. The value of num_ref_frames shall be in the range of 0 to MaxDpbSize (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 follows

$$\text{PicWidthInMbs} = \text{pic\_width\_in\_mbs\_minus1} + 1 \quad (7\text{-}3)$$

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

$$\text{PicWidthInSamples}_L = \text{PicWidthInMbs} * 16 \quad (7\text{-}4)$$

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

$$\text{PicWidthInSamples}_C = \text{PicWidthInMbs} * 8 \quad (7\text{-}5)$$

pic_height_in_map_units_minus1 plus 1 specifies the height in slice group map units of a decoded frame or field.

The variables PicHeightInMapUnits and PicSizeInMapUnits are derived as follows

$$\text{PicHeightInMapUnits} = \text{pic\_height\_in\_map\_units\_minus1} + 1 \quad (7\text{-}6)$$

$$\text{PicSizeInMapUnits} = \text{PicWidthInMbs} * \text{PicHeightInMapUnits} \quad (7\text{-}7)$$

frame_mbs_only_flag equal to 0 specifies that coded pictures of the coded video sequence may either be coded fields or coded frames. frame_mbs_only_flag equal to 1 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, and frame_mbs_only_flag is specified by constraints in Annex A.

Depending on frame_mbs_only_flag, semantics are assigned to pic_height_in_map_units_minus1 as follows.

- If frame_mbs_only_flag is equal to 0, pic_height_in_map_units_minus1 plus 1 is the height of a field in units of macroblocks.
- Otherwise (frame_mbs_only_flag is equal to 1), pic_height_in_map_units_minus1 plus 1 is the height of a frame in units of macroblocks.

The variable FrameHeightInMbs is derived as follows

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

mb_adaptive_frame_field_flag equal to 0 specifies no switching between frame and field macroblocks within a picture. mb_adaptive_frame_field_flag equal to 1 specifies the possible use of switching between frame and field macroblocks within frames. When mb_adaptive_frame_field_flag is not present, it shall be inferred to be equal to 0.

direct_8x8_inference_flag specifies the method used in the derivation process for luma motion vectors for B_Skip, B_Direct_16x16 and B_Direct_8x8 as specified in subclause 8.4.1.2. When frame_mbs_only_flag is equal to 0, direct_8x8_inference_flag shall be equal to 1.

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 a frame within a rectangle as follows.

– If frame_mbs_only_flag is equal to 1, the cropping rectangle contains luma samples with horizontal coordinates from 2 * frame_crop_left_offset to $\text{PicWidthInSamples}_L$ - (2 * frame_crop_right_offset + 1) and vertical coordinates from 2 * frame_crop_top_offset to (FrameHeightInMbs * 16) - (2 * frame_crop_bottom_offset + 1), inclusive. In this case, the value of frame_crop_left_offset shall be in the range of 0 to 8 * PicWidthInMbs - (frame_crop_right_offset + 1), inclusive; and the value of frame_crop_top_offset shall be in the range of 0 to 8 * FrameHeightInMbs - (frame_crop_bottom_offset + 1), inclusive.

– Otherwise (frame_mbs_only_flag is equal to 0), the cropping rectangle contains luma samples with horizontal coordinates from 2 * frame_crop_left_offset to $\text{PicWidthInSamples}_L$ - (2 * frame_crop_right_offset + 1) and vertical coordinates from 4 * frame_crop_top_offset to (FrameHeightInMbs * 16) - (4 * frame_crop_bottom_offset + 1), inclusive. In this case the value of frame_crop_left_offset shall be in the range of 0 to 8 * PicWidthInMbs - (frame_crop_right_offset + 1), inclusive; and the value of frame_crop_top_offset shall be in the range of 0 to 4 * FrameHeightInMbs - (frame_crop_bottom_offset + 1), inclusive.

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

The 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.

For decoded fields, the specified samples of the decoded field are the samples that fall within the rectangle specified in frame coordinates.

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

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 for which two descriptors appear in the syntax tables as follows.

- If entropy_coding_mode_flag is equal to 0, 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).
- Otherwise (entropy_coding_mode_flag is equal to 1), the method specified by the right descriptor in the syntax table is applied (CABAC, see subclause 9.3).

pic_order_present_flag equal to 1 specifies that the picture order count related syntax elements are present in the slice headers as specified in subclause 7.3.3. pic_order_present_flag equal to 0 specifies that the picture order count related syntax elements are not present in the slice headers.

num_slice_groups_minus1 plus 1 specifies the number of slice groups for a picture. When num_slice_groups_minus1 is equal to 0, all slices of the picture belong to the same slice group. The allowed range of num_slice_groups_minus1 is specified in Annex A.

slice_group_map_type specifies how the mapping of slice group map units to slice groups is coded. The value of slice_group_map_type shall be in the range of 0 to 6, inclusive.

slice_group_map_type equal to 0 specifies interleaved slice groups.

slice_group_map_type equal to 1 specifies a dispersed slice group mapping.

slice_group_map_type equal to 2 specifies one or more "foreground" slice groups and a "leftover" slice group.

slice_group_map_type values equal to 3, 4, and 5 specify changing slice groups. When num_slice_groups_minus1 is not equal to 1, slice_group_map_type shall not be equal to 3, 4, or 5.

slice_group_map_type equal to 6 specifies an explicit assignment of a slice group to each slice group map unit.

Slice group map units are specified as follows.

– If frame_mbs_only_flag is equal to 0 and mb_adaptive_frame_field_flag is equal to 1 and the coded picture is a frame, the slice group map units are macroblock pair units.
– Otherwise, if frame_mbs_only_flag is equal to 1 or a coded picture is a field, the slice group map units are units of macroblocks.
– Otherwise (frame_mbs_only_flag is equal to 0 and mb_adaptive_frame_field_flag is equal to 0 and the coded picture is a frame), the slice group map units are units of two macroblocks that are vertically contiguous as in a frame macroblock pair of an MBAFF frame.

run_length_minus1[i **]** is used to specify the number of consecutive slice group map units to be assigned to the i-th slice group in raster scan order of slice group map units. The value of run_length_minus1[i] shall be in the range of 0 to PicSizeInMapUnits - 1, inclusive.

top_left[i **]** and **bottom_right[** i **]** specify the top-left and bottom-right corners of a rectangle, respectively. top_left[i] and bottom_right[i] are slice group map unit positions in a raster scan of the picture for the slice group map units. For each rectangle i, all of the following constraints shall be obeyed by the values of the syntax elements top_left[i] and bottom_right[i]

- top_left[i] shall be less than or equal to bottom_right[i] and bottom_right[i] shall be less than PicSizeInMapUnits.
- (top_left[i] % PicWidthInMbs) shall be less than or equal to the value of (bottom_right[i] % PicWidthInMbs).

slice_group_change_direction_flag is used with slice_group_map_type to specify the refined map type when slice_group_map_type is 3, 4, or 5.

slice_group_change_rate_minus1 is used to specify the variable SliceGroupChangeRate. SliceGroupChangeRate specifies the multiple in number of slice group map units by which the size of a slice group can change from one picture to the next. The value of slice_group_change_rate_minus1 shall be in the range of 0 to PicSizeInMapUnits – 1, inclusive. The SliceGroupChangeRate variable is specified as follows:

$$\text{SliceGroupChangeRate} = \text{slice\_group\_change\_rate\_minus1} + 1 \qquad (7\text{-}9)$$

pic_size_in_map_units_minus1 is used to specify the number of slice group map units in the picture. pic_size_in_map_units_minus1 shall be equal to PicSizeInMapUnits - 1.

slice_group_id[i **]** identifies a slice group of the i-th slice group map unit in raster scan order. The size of the slice_group_id[i] syntax element is Ceil(Log2(num_slice_groups_minus1 + 1)) bits. The value of slice_group_id[i] shall be in the range of 0 to num_slice_groups_minus1, inclusive.

num_ref_idx_l0_active_minus1 specifies the maximum reference index for reference picture list 0 that shall be used to decode each slice of the picture in which list 0 is used when num_ref_idx_active_override_flag is equal to 0 for the slice. When MbaffFrameFlag is equal to 1, num_ref_idx_l0_active_minus1 is the maximum index value for the decoding of frame macroblocks and 2 * num_ref_idx_l0_active_minus1 + 1 is the maximum index value for the decoding of field macroblocks. The value of num_ref_idx_l0_active_minus1 shall be in the range of 0 to 31, inclusive.

num_ref_idx_l1_active_minus1 has the same semantics as num_ref_idx_l0_active_minus1 with l0 and list 0 replaced by l1 and list 1, respectively.

weighted_pred_flag equal to 0 specifies that weighted prediction shall not be applied to P and SP slices. weighted_pred_flag equal to 1 specifies that weighted prediction shall be applied to P and SP slices.

weighted_bipred_idc equal to 0 specifies that the default weighted prediction shall be applied to B slices. weighted_bipred_idc equal to 1 specifies that explicit weighted prediction shall be applied to B slices. weighted_bipred_idc equal to 2 specifies that implicit weighted prediction shall be applied to B slices. The value of weighted_bipred_idc shall be in the range of 0 to 2, inclusive.

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 to +25, inclusive.

pic_init_qs_minus26 specifies the initial value minus 26 of SliceQS_Y for all macroblocks in SP or SI slices. The initial value is modified at the slice layer when a non-zero value of slice_qs_delta is decoded. The value of pic_init_qs_minus26 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. 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 or SI macroblock types.

redundant_pic_cnt_present_flag equal to 0 specifies that the redundant_pic_cnt syntax element is not present in slice headers, data partitions B, and data partitions C that refer (either directly or by association with a corresponding data partition A) to the picture parameter set. redundant_pic_cnt_present_flag equal to 1 specifies that the redundant_pic_cnt syntax element is present in all slice headers, data partitions B, and data partitions C that refer (either directly or by association with a corresponding data partition A) 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 NAL unit 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 are specified in Annex D. The derived SEI payload size payloadSize is specified in bytes and shall be equal to the number of bytes in the 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 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.

Table 7-2 – Meaning of primary_pic_type

primary_pic_type	slice_type values that may be present in the primary coded picture
0	I
1	I, P
2	I, P, B
3	SI
4	SI, SP
5	I, SI
6	I, SI, P, SP
7	I, SI, P, SP, B

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.

7.4.2.7 Filler data RBSP semantics

The filler data RBSP contains bytes whose value shall be equal to 0xFF. No normative decoding process is specified for a filler data RBSP.

ff_byte is a byte equal to 0xFF.

7.4.2.8 Slice layer without partitioning RBSP semantics

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

7.4.2.9 Slice data partition RBSP semantics

7.4.2.9.1 Slice data partition A RBSP semantics

When slice data partitioning is in use, the coded data for a single slice is divided into three separate partitions. Partition A contains all syntax elements of category 2.

Category 2 syntax elements include all syntax elements in the slice header and slice data syntax structures other than the syntax elements in the residual( ) syntax structure.

slice_id identifies the slice associated with the data partition. Each slice shall have a unique slice_id value within the coded picture that contains the slice. When arbitrary slice order is not allowed as specified in Annex A, the first slice of a coded picture, in decoding order, shall have slice_id equal to 0 and the value of slice_id shall be incremented by one for each subsequent slice of the coded picture in decoding order.

The range of slice_id is specified as follows.

– If MbaffFrameFlag is equal to 0, slice_id shall be in the range of 0 to PicSizeInMbs - 1, inclusive.

– Otherwise (MbaffFrameFlag is equal to 1), slice_id shall be in the range of 0 to PicSizeInMbs / 2 - 1, inclusive.

7.4.2.9.2 Slice data partition B RBSP semantics

When slice data partitioning is in use, the coded data for a single slice is divided into one to three separate partitions. Slice data partition B contains all syntax elements of category 3.

Category 3 syntax elements include all syntax elements in the residual( ) syntax structure and in syntax structures used within that syntax structure for collective macroblock types I and SI as specified in Table 7-7.

slice_id has the same semantics as specified in subclause 7.4.2.9.1.

redundant_pic_cnt shall be equal to 0 for slices and slice data partitions belonging to the primary coded picture. The redundant_pic_cnt shall be greater than 0 for coded slices and coded slice data partitions in redundant coded pictures. When redundant_pic_cnt is not present, its value shall be inferred to be equal to 0. The value of redundant_pic_cnt shall be in the range of 0 to 127, inclusive.

The presence of a slice data partition B RBSP is specified as follows.

- If the syntax elements of a slice data partition A RBSP indicate the presence of any syntax elements of category 3 in the slice data for a slice, a slice data partition B RBSP shall be present having the same value of slice_id and redundant_pic_cnt as in the slice data partition A RBSP.

- Otherwise (the syntax elements of a slice data partition A RBSP do not indicate the presence of any syntax elements of category 3 in the slice data for a slice), no slice data partition B RBSP shall be present having the same value of slice_id and redundant_pic_cnt as in the slice data partition A RBSP.

7.4.2.9.3 Slice data partition C RBSP semantics

When slice data partitioning is in use, the coded data for a single slice is divided into three separate partitions. Slice data partition C contains all syntax elements of category 4.

Category 4 syntax elements include all syntax elements in the residual( ) syntax structure and in syntax structures used within that syntax structure for collective macroblock types P and B as specified in Table 7-7.

slice_id has the same semantics as specified in subclause 7.4.2.9.1.

redundant_pic_cnt has the same semantics as specified in subclause 7.4.2.9.2.

The presence of a slice data partition C RBSP is specified as follows.

- If the syntax elements of a slice data partition A RBSP indicate the presence of any syntax elements of category 4 in the slice data for a slice, a slice data partition C RBSP shall be present having the same value of slice_id and redundant_pic_cnt as in the slice data partition A RBSP.
- Otherwise (the syntax elements of a slice data partition A RBSP do not indicate the presence of any syntax elements of category 4 in the slice data for a slice), no slice data partition C RBSP shall be present having the same value of slice_id and redundant_pic_cnt as in the slice data partition A RBSP.

7.4.2.10 RBSP slice trailing bits semantics

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

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

Let BinCountsInNALunits be the number of times that the parsing process function DecodeBin(), specified in subclause 9.3.3.2, is invoked to decode the contents of all VCL NAL units of a coded picture. When entropy_coding_mode_flag is equal to 1, BinCountsInNALunits shall not exceed (32 ÷ 3) * NumBytesInVclNALunits + 96 * PicSizeInMbs.

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

7.4.2.11 RBSP trailing bits semantics

rbsp_stop_one_bit is a single bit equal to 1.

rbsp_alignment_zero_bit is a single bit 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, field_pic_flag, bottom_field_flag, idr_pic_id, pic_order_cnt_lsb, delta_pic_order_cnt_bottom, delta_pic_order_cnt[0], delta_pic_order_cnt[1], sp_for_switch_flag, and slice_group_change_cycle 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. When arbitrary slice order is not allowed as specified in Annex A, 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.

– If MbaffFrameFlag is equal to 0, 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.

– Otherwise (MbaffFrameFlag is equal to 1), first_mb_in_slice * 2 is the macroblock address of the first macroblock in the slice, which is the top macroblock of the first macroblock pair in the slice, and first_mb_in_slice shall be in the range of 0 to PicSizeInMbs / 2 - 1, inclusive.

slice_type specifies the coding type of the slice according to Table 7-3.

Table 7-3 – Name association to slice_type

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

slice_type values in the range 5..9 specify, in addition to the coding type of the current slice, 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 – 5.

When nal_unit_type is equal to 5 (IDR picture), slice_type shall be equal to 2, 4, 7, or 9.

When num_ref_frames is equal to 0, slice_type shall be equal to 2, 4, 7, or 9.

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.
  - the current picture and the preceding reference picture belong to consecutive access units in decoding order
  - the current picture and the preceding reference picture are reference fields having opposite parity
  - 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, the following applies.

– There shall not be any previous field or 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 )                                    (7-10)
   UnusedShortTermFrameNum = ( UnusedShortTermFrameNum + 1 ) % MaxFrameNum
```

– 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.

NOTE – When the primary coded picture is not an IDR picture and does not contain memory_management_control_operation syntax element equal to 5, the value of frame_num of a corresponding redundant coded picture is the same as the value of frame_num in the primary coded picture. Alternatively, the redundant coded picture includes a memory_management_control_operation syntax element equal to 5 and the corresponding primary coded picture is an IDR picture.

field_pic_flag equal to 1 specifies that the slice is a slice of a coded field. field_pic_flag equal to 0 specifies that the slice is a slice of a coded frame. When field_pic_flag is not present it shall be inferred to be equal to 0.

The variable MbaffFrameFlag is derived as follows.

$$\text{MbaffFrameFlag} = (\text{mb\_adaptive\_frame\_field\_flag} \ \&\& \ !\text{field\_pic\_flag}) \qquad (7\text{-}11)$$

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

$$\text{PicHeightInMbs} = \text{FrameHeightInMbs} \, / \, (1 + \text{field\_pic\_flag}) \qquad (7\text{-}12)$$

The variable for picture height for the luma component is derived as follows

$$\text{PicHeightInSamples}_L = \text{PicHeightInMbs} * 16 \qquad (7\text{-}13)$$

The variable for picture height for the chroma component is derived as follows

$$\text{PicHeightInSamples}_C = \text{PicHeightInMbs} * 8 \quad (7\text{-}14)$$

The variable PicSizeInMbs for the current picture is derived according to:

$$\text{PicSizeInMbs} = \text{PicWidthInMbs} * \text{PicHeightInMbs} \quad (7\text{-}15)$$

The variable MaxPicNum is derived as follows.

- If field_pic_flag is equal to 0, MaxPicNum is set equal to MaxFrameNum.
- Otherwise (field_pic_flag is equal to 1), MaxPicNum is set equal to 2*MaxFrameNum.

The variable CurrPicNum is derived as follows.

- If field_pic_flag is equal to 0, CurrPicNum is set equal to frame_num.
- Otherwise (field_pic_flag is equal to 1), CurrPicNum is set equal to 2 * frame_num + 1.

bottom_field_flag equal to 1 specifies that the slice is part of a coded bottom field. bottom_field_flag equal to 0 specifies that the picture is a coded top field. When this syntax element is not present for the current slice, it shall be inferred to be equal to 0.

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.

pic_order_cnt_lsb specifies the picture order count modulo MaxPicOrderCntLsb for the top field of a coded frame or for a coded field. The size 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_bottom specifies the picture order count difference between the bottom field and the top field of a coded frame as follows.

- If the current picture includes a memory_management_control_operation equal to 5, the value of delta_pic_order_cnt_bottom shall be in the range of (1 – MaxPicOrderCntLsb) to 2^{31} - 1, inclusive.
- Otherwise (the current picture does not include a memory_management_control_operation equal to 5), the value of delta_pic_order_cnt_bottom shall be in the range of -2^{31} 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.

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} 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.

delta_pic_order_cnt[1 **]** specifies the picture order count difference from the expected picture order count for the bottom field of a coded frame specified in subclause 8.2.1. The value of delta_pic_order_cnt[1] shall be in the range of -2^{31} 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 for slices and slice data partitions belonging to the primary coded picture. The value of redundant_pic_cnt shall be greater than 0 for coded slices or coded slice data partitions of a redundant coded picture. When redundant_pic_cnt is not present in the bitstream, its value shall be inferred to be equal to 0. The value of redundant_pic_cnt shall be in the range of 0 to 127, inclusive.

NOTE - There should be no noticeable difference between any area of the decoded primary picture and a corresponding area that would result from application of the decoding process specified in clause 8 for any redundant picture in the same access unit.

The value of pic_parameter_set_id in a coded slice or coded slice data partition of a redundant coded picture shall be such that the value of pic_order_present_flag in the picture parameter set in use in a redundant coded picture is equal to the value of pic_order_present_flag in the picture parameter set in use in the corresponding primary coded picture.

When present in the primary coded picture and any redundant coded picture, the following syntax elements shall have the same value: field_pic_flag, bottom_field_flag, and idr_pic_id.

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.

NOTE – The above constraint also has the following implications. If the value of nal_ref_idc for the VCL NAL units of the primary coded picture is equal to 0, the value of nal_ref_idc for the VCL NAL units of any corresponding redundant coded picture are equal to 0; otherwise (the value of nal_ref_idc for the VCL NAL units of the primary coded picture is greater than 0), the value of nal_ref_idc for the VCL NAL units of any corresponding redundant coded picture are also greater than 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 or any redundant coded picture of the same access unit shall be identical regardless whether the primary coded picture or any redundant coded picture (instead of the primary coded picture) of the access unit would be decoded.

NOTE – The above constraint also has the following implications.

If a primary coded picture is not an IDR picture, the contents of the dec_ref_pic_marking( ) syntax structure must be identical in all slice headers of the primary coded picture and all redundant coded pictures corresponding to the primary coded picture.

Otherwise (a primary coded picture is an IDR picture), the following applies.

If a redundant coded picture corresponding to the primary coded picture is an IDR picture, the contents of the dec_ref_pic_marking( ) syntax structure must be identical in all slice headers of the primary coded picture and the redundant coded picture corresponding to the primary coded picture.

Otherwise (a redundant picture corresponding to the primary coded picture is not an IDR picture), all slice headers of the redundant picture must contain a dec_ref_pic_marking syntax( ) structure including a memory_management_control_operation syntax element equal to 5, and the following applies.

If the value of long_term_reference_flag in the primary coded picture is equal to 0, the dec_ref_pic_marking syntax structure of the redundant coded picture must not include a memory_management_control_operation syntax element equal to 6.

Otherwise (the value of long_term_reference_flag in the primary coded picture is equal to 1), the dec_ref_pic_marking syntax structure of the redundant coded picture must include memory_management_control_operation syntax elements equal to 5, 4, and 6 in decoding order, and the value of max_long_term_frame_idx_plus1 must be equal to 1, and the value of long_term_frame_idx must be equal to 0.

The values of TopFieldOrderCnt and BottomFieldOrderCnt (if applicable) that result after completion of the decoding process for any redundant coded picture or the primary coded picture of the same access unit shall be identical regardless whether the primary coded picture or any redundant coded picture (instead of the primary coded picture) of the access unit would be decoded.

There is no required decoding process for a coded slice or coded slice data partition of a redundant coded picture. When the redundant_pic_cnt in the slice header of a coded slice is greater than 0, the decoder may discard the coded slice. However, a coded slice or coded slice data partition of any redundant coded picture shall obey the same constraints as a coded slice or coded slice data partition of a primary picture.

NOTE – When some of the samples in the decoded primary picture cannot be correctly decoded due to errors or losses in transmission of the sequence and a coded redundant slice can be correctly decoded, the decoder should replace the samples of the decoded primary picture with the corresponding samples of the decoded redundant slice. When more than one redundant slice covers the relevant region of the primary picture, the redundant slice having the lowest value of redundant_pic_cnt should be used.

Redundant slices and slice data partitions having the same value of redundant_pic_cnt belong to the same redundant picture. Decoded slices within the same redundant picture need not cover the entire picture area and shall not overlap.

direct_spatial_mv_pred_flag specifies the method used in the decoding process to derive motion vectors and reference indices for inter prediction as follows.

- If direct_spatial_mv_pred_flag is equal to 1, the derivation process for luma motion vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8 in subclause 8.4.1.2 shall use spatial direct mode prediction as specified in subclause 8.4.1.2.2.

- Otherwise (direct_spatial_mv_pred_flag is equal to 0), the derivation process for luma motion vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8 in subclause 8.4.1.2 shall use temporal direct mode prediction as specified in subclause 8.4.1.2.3.

num_ref_idx_active_override_flag equal to 0 specifies that the values of the syntax elements num_ref_idx_l0_active_minus1 and num_ref_idx_l1_active_minus1 specified in the referred picture parameter set are in effect. num_ref_idx_active_override_flag equal to 1 specifies that the num_ref_idx_l0_active_minus1 and num_ref_idx_l1_active_minus1 specified in the referred picture parameter set are overridden for the current slice (and only for the current slice) by the following values in the slice header.

When the current slice is a P, SP, or B slice and field_pic_flag is equal to 0 and the value of num_ref_idx_l0_active_minus1 in the picture parameter set exceeds 15, num_ref_idx_active_override_flag shall be equal to 1.

When the current slice is a B slice and field_pic_flag is equal to 0 and the value of num_ref_idx_l1_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_minus1 specifies the maximum reference index for reference picture list 0 that shall be used to decode the slice.

The range of num_ref_idx_l0_active_minus1 is specified as follows.

– If field_pic_flag is equal to 0, num_ref_idx_l0_active_minus1 shall be in the range of 0 to 15, inclusive. When MbaffFrameFlag is equal to 1, num_ref_idx_l0_active_minus1 is the maximum index value for the decoding of frame macroblocks and 2 * num_ref_idx_l0_active_minus1 + 1 is the maximum index value for the decoding of field macroblocks.

– Otherwise (field_pic_flag is equal to 1), num_ref_idx_l0_active_minus1 shall be in the range of 0 to 31, inclusive.

num_ref_idx_l1_active_minus1 has the same semantics as num_ref_idx_l0_active_minus1 with l0 and list 0 replaced by l1 and list 1, respectively.

cabac_init_idc specifies the index for determining the initialisation table used in the initialisation process for context variables. The value of cabac_init_idc shall be in the range of 0 to 2, 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:

$$\text{SliceQP}_Y = 26 + \text{pic\_init\_qp\_minus26} + \text{slice\_qp\_delta} \tag{7-16}$$

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

sp_for_switch_flag specifies the decoding process to be used to decode P macroblocks in an SP slice as follows.

- If sp_for_switch_flag is equal to 0, the P macroblocks in the SP slice shall be decoded using the SP decoding process for non-switching pictures as specified in subclause 8.6.1.

- Otherwise (sp_for_switch_flag is equal to 1), the P macroblocks in the SP slice shall be decoded using the SP and SI decoding process for switching pictures as specified in subclause 8.6.2.

slice_qs_delta specifies the value of QS_Y for all the macroblocks in SP and SI slices. The QS_Y quantisation parameter for the slice is computed as:

$$QS_Y = 26 + \text{pic\_init\_qs\_minus26} + \text{slice\_qs\_delta} \tag{7-17}$$

The value of slice_qs_delta shall be limited such that QS_Y is in the range of 0 to 51, inclusive. This value of QS_Y is used for the decoding of all macroblocks in SI slices with mb_type equal to SI and all macroblocks in SP slices with prediction mode equal to inter.

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:

$$\text{FilterOffsetA} = \text{slice\_alpha\_c0\_offset\_div2} \ll 1 \tag{7-18}$$

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:

$$\text{FilterOffsetB} = \text{slice\_beta\_offset\_div2} \ll 1 \tag{7-19}$$

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.

slice_group_change_cycle is used to derive the number of slice group map units in slice group 0 when slice_group_map_type is equal to 3, 4, or 5, as specified by

$$\text{MapUnitsInSliceGroup0} = \text{Min}(\ \text{slice\_group\_change\_cycle} * \text{SliceGroupChangeRate},\ \text{PicSizeInMapUnits}\ ) \tag{7-20}$$

The value of slice_group_change_cycle is represented in the bitstream by the following number of bits

$$\text{Ceil}(\ \text{Log2}(\ \text{PicSizeInMapUnits} \div \text{SliceGroupChangeRate} + 1\ )\ ) \tag{7-21}$$

The value of slice_group_change_cycle shall be in the range of 0 to Ceil(PicSizeInMapUnits÷SliceGroupChangeRate), inclusive.

7.4.3.1 Reference picture list reordering semantics

The syntax elements reordering_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_reordering_flag_l0 equal to 1 specifies that the syntax element reordering_of_pic_nums_idc is present for specifying reference picture list 0. ref_pic_list_reordering_flag_l0 equal to 0 specifies that this syntax element is not present.

When ref_pic_list_reordering_flag_l0 is equal to 1, the number of times that reordering_of_pic_nums_idc is not equal to 3 following ref_pic_list_reordering_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_reordering_flag_l0 shall be equal to 1 and reordering_of_pic_nums_idc shall not be equal to 3 until RefPicList0[num_ref_idx_l0_active_minus1] in the reordered list produced as specified in subclause 8.2.4.3 is not equal to "no reference picture".

ref_pic_list_reordering_flag_l1 equal to 1 specifies that the syntax element reordering_of_pic_nums_idc is present for specifying reference picture list 1. ref_pic_list_reordering_flag_l1 equal to 0 specifies that this syntax element is not present.

When ref_pic_list_reordering_flag_l1 is equal to 1, the number of times that reordering_of_pic_nums_idc is not equal to 3 following ref_pic_list_reordering_flag_l1 shall not exceed num_ref_idx_l1_active_minus1 + 1.

When decoding a B slice and RefPicList1[num_ref_idx_l1_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_reordering_flag_l1 shall be equal to 1 and reordering_of_pic_nums_idc shall not be equal to 3 until RefPicList1[num_ref_idx_l1_active_minus1] in the reordered list produced as specified in subclause 8.2.4.3 is not equal to "no reference picture".

reordering_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 reordering_of_pic_nums_idc are specified in Table 7-4. The value of the first reordering_of_pic_nums_idc that follows immediately after ref_pic_list_reordering_flag_l0 or ref_pic_list_reordering_flag_l1 shall not be equal to 3.

Table 7-4 – reordering_of_pic_nums_idc operations for reordering of reference picture lists

reordering_of_pic_nums_idc	Reordering 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 reordering 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 or complementary reference field pairs marked as "used for long-term reference". When decoding a coded field, long_term_pic_num shall be equal to a LongTermPicNum assigned to one of the reference fields marked as "used for long-term reference".

7.4.3.2 Prediction weight table semantics

luma_log2_weight_denom is the base 2 logarithm of the denominator for all luma weighting factors. The value of luma_log2_weight_denom shall be in the range of 0 to 7, inclusive.

chroma_log2_weight_denom is the base 2 logarithm of the denominator for all chroma weighting factors. The value of chroma_log2_weight_denom shall be in the range of 0 to 7, inclusive.

luma_weight_l0_flag equal to 1 specifies that weighting factors for the luma component of list 0 prediction are present. luma_weight_l0_flag equal to 0 specifies that these weighting factors are not present.

luma_weight_l0[i] is the weighting factor applied to the luma prediction value for list 0 prediction using RefPicList0[i]. When luma_weight_l0_flag is equal to 1, the value of luma_weight_l0[i] shall be in the range of –128 to 127, inclusive. When luma_weight_l0_flag is equal to 0, luma_weight_l0[i] shall be inferred to be equal to $2^{\text{luma\_log2\_weight\_denom}}$ for RefPicList0[i].

luma_offset_l0[i] is the additive offset applied to the luma prediction value for list 0 prediction using RefPicList0[i]. The value of luma_offset_l0[i] shall be in the range of –128 to 127, inclusive. When luma_weight_l0_flag is equal to 0, luma_offset_l0[i] shall be inferred as equal to 0 for RefPicList0[i].

chroma_weight_l0_flag equal to 1 specifies that weighting factors for the chroma prediction values of list 0 prediction are present. chroma_weight_l0_flag equal to 0 specifies that these weighting factors are not present.

chroma_weight_l0[i][j] is the weighting factor applied to the chroma prediction values for list 0 prediction using RefPicList0[i] with j equal to 0 for Cb and j equal to 1 for Cr. When chroma_weight_l0_flag is equal to 1, the value of chroma_weight_l0[i][j] shall be in the range of –128 to 127, inclusive. When chroma_weight_l0_flag is equal to 0, chroma_weight_l0[i][j] shall be inferred to be equal to $2^{\text{chroma\_log2\_weight\_denom}}$ for RefPicList0[i].

chroma_offset_l0[i][j] is the additive offset applied to the chroma prediction values for list 0 prediction using RefPicList0[i] with j equal to 0 for Cb and j equal to 1 for Cr. The value of chroma_offset_l0[i][j] shall be in the range of -128 to 127, inclusive. When chroma_weight_l0_flag is equal to 0, chroma_offset_l0[i][j] shall be inferred to be equal to 0 for RefPicList0[i].

luma_weight_l1_flag, luma_weight_l1, luma_offset_l1, chroma_weight_l1_flag, chroma_weight_l1, chroma_offset_l1 have the same semantics as luma_weight_l0_flag, luma_weight_l0, luma_offset_l0, chroma_weight_l0_flag, chroma_weight_l0, chroma_offset_l0, respectively, with l0, list 0, and List0 replaced by l1, list 1, and List1, respectively.

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 have the marking "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.

The syntax element adaptive_ref_pic_marking_mode_flag and the content of the decoded reference picture marking syntax structure shall be identical for all coded slices of a coded picture.

The syntax category of the decoded reference picture marking syntax structure shall be inferred as follows.

- If the decoded reference picture marking syntax structure is in a slice header, the syntax category of the decoded reference picture marking syntax structure shall be inferred to be equal to 2.
- Otherwise (the decoded reference picture marking syntax structure is in a decoded reference picture marking repetition SEI message as specified in Annex D), the syntax category of the decoded reference picture marking syntax structure shall be inferred to be equal to 5.

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 sequence, no_output_of_prior_pics_flag equal to 1 may 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 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, complementary field pairs, and non-paired fields that are currently marked as "used for long-term reference" is equal to 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 manage 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.

memory_management_control_operation shall not be equal to 1 in a slice header unless the specified short-term picture is currently marked as "used for reference" when the memory_management_control_operation is processed by the decoding process, and the specified short-term picture has not been assigned to a long-term frame index and is not assigned to a long-term frame index in the same decoded reference picture marking syntax structure.

memory_management_control_operation shall not be equal to 2 in a slice header unless the specified long-term picture number refers to a frame or field that is currently marked as "used for 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 short-term reference picture is currently marked as "used for reference" when the memory_management_control_operation is processed by the decoding process and the specified short-term reference picture has not previously been assigned a long-term frame index and is not assigned to any other long-term frame index within the same decoded reference picture marking syntax structure.

memory_management_control_operation shall not be equal to 3 or 6 when 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.

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.

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

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

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, or 4 that specifies the same reference picture to subsequently be marked as "unused for reference".

No more than one memory_management_control_operation shall be present in a slice header that specifies the same action to be taken.

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 loop
1	Mark a short-term picture as "unused for reference"
2	Mark a frame or field having a long-term picture number as "unused for reference"
3	Assign a long-term frame index to a short-term picture
4	Specify the maximum long-term frame index
5	Mark all reference pictures as "unused for reference" and set the MaxLongTermFrameIdx variable to "no long-term frame indices"
6	Assign a long-term frame index to the current decoded picture

When decoding a field and a memory_management_control_operation command equal to 3 assigns a long-term frame index to a field that is part of a short-term reference frame or a short-term complementary reference field pair, another memory_management_control_operation command to assign the same long-term frame index to the other field of the same frame or complementary reference field pair shall be present in the same decoded reference picture marking syntax structure.

When the first field (in decoding order) of a complementary reference field pair includes a long_term_reference_flag equal to 1 or a memory_management_control_operation command equal to 6, the decoded reference picture marking syntax structure for the other field of the complementary reference field pair shall contain a memory_management_control_operation command equal to 6 that assigns the same long-term frame index to the other field.

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 as follows.

- If field_pic_flag is equal to 0, the resulting picture number shall be one of the set of picture numbers assigned to reference frames or complementary reference field pairs.

  NOTE – When field_pic_flag is equal to 0, the resulting picture number must be a picture number assigned to a complementary reference field pair in which both fields are marked as "used for reference" or a frame in which both fields are marked as "used for reference". In particular, when field_pic_flag is equal to 0, the marking of a non-paired field or a frame in which a single field is marked as "used for reference" cannot be affected by a memory_management_control_operation equal to 1.

- Otherwise (field_pic_flag is equal to 1), the resulting picture number shall be one of the set of picture numbers assigned to reference fields.

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 as follows.

- If field_pic_flag is equal to 0, the resulting long-term picture number shall be one of the set of long-term picture numbers assigned to reference frames or complementary reference field pairs.

  NOTE – When field_pic_flag is equal to 0, the resulting long-term picture number must be a long-term picture number assigned to a complementary reference field pair in which both fields are marked as "used for reference" or a frame in which both fields are marked as "used for reference". In particular, when field_pic_flag is equal to 0, the marking of a non-paired field or a frame in which a single field is marked as "used for reference" cannot be affected by a memory_management_control_operation equal to 2.

- Otherwise (field_pic_flag is equal to 1), the resulting long-term picture number shall be one of the set of long-term picture numbers assigned to reference fields.

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 num_ref_frames, inclusive.

7.4.4 Slice data semantics

cabac_alignment_one_bit is a bit equal to 1.

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

mb_skip_flag equal to 1 specifies that for the current macroblock, when decoding a P or SP slice, mb_type shall be inferred to be P_Skip and the macroblock type is collectively referred to as P macroblock type, or for which, when decoding a B slice, mb_type shall be inferred to be B_Skip and the macroblock type is collectively referred to as B macroblock type. mb_skip_flag equal to 0 specifies that the current macroblock is not skipped.

mb_field_decoding_flag equal to 0 specifies that the current macroblock pair is a frame macroblock pair. mb_field_decoding_flag equal to 1 specifies that the macroblock pair is a field macroblock pair. Both macroblocks of a frame macroblock pair are referred to in the text as frame macroblocks, whereas both macroblocks of a field macroblock pair are referred to in the text as field macroblocks.

When mb_field_decoding_flag is not present for either macroblock of a macroblock pair, the value of mb_field_decoding_flag is derived as follows.

– If there is a neighbouring macroblock pair immediately to the left of the current macroblock pair in the same slice, the value of mb_field_decoding_flag shall be inferred to be equal to the value of mb_field_decoding_flag for the neighbouring macroblock pair immediately to the left of the current macroblock pair,

– Otherwise, if there is no neighbouring macroblock pair immediately to the left of the current macroblock pair in the same slice and there is a neighbouring macroblock pair immediately above the current macroblock pair in the same slice, the value of mb_field_decoding_flag shall be inferred to be equal to the value of mb_field_decoding_flag for the neighbouring macroblock pair immediately above the current macroblock pair,

– Otherwise (there is no neighbouring macroblock pair either immediately to the left or immediately above the current macroblock pair in the same slice), the value of mb_field_decoding_flag shall be inferred to be equal to 0.

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( ) used in the slice data syntax table is specified in subclause 8.2.2.

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, SI, P, SP, and B 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 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.

NOTE – There are some macroblock types with Pred_L0 prediction mode that are classified as B macroblock types.

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 7-10) and I (see Table 7-8) (macroblock types)
B (slice)	B (see Table 7-11) and I (see Table 7-8) (macroblock types)
SI (slice)	SI (see Table 7-9) and I (see Table 7-8) (macroblock types)
SP (slice)	P (see Table 7-10) 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 7-22	Equation 7-22
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_4x4: the macroblock is coded as an Intra_4x4 prediction macroblock.

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 mode macroblock.

To each Intra_16x16 prediction macroblock, an Intra16x16PredMode is assigned, which specifies the Intra_16x16 prediction mode. CodedBlockPatternChroma contains the coded block pattern value for chroma as specified in Table 7-12. CodedBlockPatternLuma specifies whether, for the luma component, non-zero AC transform coefficient levels are present. CodedBlockPatternLuma equal to 0 specifies that all AC transform coefficient levels in the luma component of the macroblock are equal to 0. CodedBlockPatternLuma equal to 15 specifies that at least one AC transform coefficient levels in the luma component of the macroblock is non-zero, requiring scanning of AC transform coefficient levels for all 16 of the 4x4 blocks in the 16x16 block.

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 8.3.2. 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.

A macroblock type that may be referred to as SI macroblock type is specified in Table 7-9.

The macroblock types for SI slices are specified in Table 7-9 and Table 7-8. The mb_type value 0 is specified in Table 7-9 and the mb_type values 1 to 26 are specified in Table 7-8, indexed by subtracting 1 from the value of mb_type.

Table 7-9 – Macroblock type with value 0 for SI slices

mb_type	Name of mb_type	MbPartPredMode (mb_type, 0)	Intra16x16PredMode	CodedBlockPatternChroma	CodedBlockPatternLuma
0	SI	Intra_4x4	na	Equation 7-22	Equation 7-22

The following semantics are assigned to the macroblock type in Table 7-9. The SI macroblock is coded as Intra_4x4 prediction macroblock.

Macroblock types that may be collectively referred to as P macroblock types are specified in Table 7-10.

The macroblock types for P and SP slices are specified in Table 7-10 and Table 7-8. mb_type values 0 to 4 are specified in Table 7-10 and mb_type values 5 to 30 are specified in Table 7-8, indexed by subtracting 5 from the value of mb_type.

Table 7-10 – 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 7-10.

– 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) 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) 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 equal to 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 (MbPartPredMode( )) in Table 7-10.

– Pred_L0: specifies that the inter prediction process is invoked using list 0 prediction. Pred_L0 is an Inter macroblock prediction mode.

Macroblock types that may be collectively referred to as B macroblock types are specified in Table 7-11.

The macroblock types for B slices are specified in Table 7-11 and Table 7-8. The mb_type values 0 to 22 are specified in Table 7-11 and the mb_type values 23 to 48 are specified in Table 7-8, indexed by subtracting 23 from the value of mb_type.

Table 7-11 – Macroblock type values 0 to 22 for B 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	B_Direct_16x16	na	Direct	na	8	8
1	B_L0_16x16	1	Pred_L0	na	16	16
2	B_L1_16x16	1	Pred_L1	na	16	16
3	B_Bi_16x16	1	BiPred	na	16	16
4	B_L0_L0_16x8	2	Pred_L0	Pred_L0	16	8
5	B_L0_L0_8x16	2	Pred_L0	Pred_L0	8	16
6	B_L1_L1_16x8	2	Pred_L1	Pred_L1	16	8
7	B_L1_L1_8x16	2	Pred_L1	Pred_L1	8	16
8	B_L0_L1_16x8	2	Pred_L0	Pred_L1	16	8
9	B_L0_L1_8x16	2	Pred_L0	Pred_L1	8	16
10	B_L1_L0_16x8	2	Pred_L1	Pred_L0	16	8
11	B_L1_L0_8x16	2	Pred_L1	Pred_L0	8	16
12	B_L0_Bi_16x8	2	Pred_L0	BiPred	16	8
13	B_L0_Bi_8x16	2	Pred_L0	BiPred	8	16
14	B_L1_Bi_16x8	2	Pred_L1	BiPred	16	8
15	B_L1_Bi_8x16	2	Pred_L1	BiPred	8	16
16	B_Bi_L0_16x8	2	BiPred	Pred_L0	16	8
17	B_Bi_L0_8x16	2	BiPred	Pred_L0	8	16
18	B_Bi_L1_16x8	2	BiPred	Pred_L1	16	8
19	B_Bi_L1_8x16	2	BiPred	Pred_L1	8	16
20	B_Bi_Bi_16x8	2	BiPred	BiPred	16	8
21	B_Bi_Bi_8x16	2	BiPred	BiPred	8	16
22	B_8x8	4	na	na	8	8
inferred	B_Skip	na	Direct	na	8	8

The following semantics are assigned to the macroblock types in Table 7-11:

– B_Direct_16x16: no motion vector differences or reference indices are present for the macroblock in the bitstream. The functions MbPartWidth(B_Direct_16x16), and MbPartHeight(B_Direct_16x16) are used in the derivation process for motion vectors and reference frame indices in subclause 8.4.1 for direct mode prediction.

– B_X_16x16 with X being replaced by L0, L1, or Bi: the samples of the macroblock are predicted with one luma macroblock partition of size 16x16 luma samples and associated chroma samples. For a macroblock with type B_X_16x16 with X being replaced by either L0 or L1, one motion vector difference and one reference index is present in the bitstream for the macroblock. For a macroblock with type B_X_16x16 with X being replaced by Bi, two motion vector differences and two reference indices are present in the bitstream for the macroblock.

– B_X0_X1_MxN, with X0, X1 referring to the first and second macroblock partition and being replaced by L0, L1, or Bi, and 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. For a macroblock partition X0 or X1 with X0 or X1 being replaced by either L0 or L1, one motion vector difference and one reference index is present in the bitstream. For a macroblock partition X0 or X1 with X0 or X1 being replaced by Bi, two motion vector differences and two reference indices are present in the bitstream for the macroblock partition.

– B_8x8: for each sub-macroblock an additional syntax element (sub_mb_type) is present in the bitstream that specifies the type of the corresponding sub-macroblock (see subclause 7.4.5.2).

– B_Skip: no further data is present for the macroblock in the bitstream. The functions MbPartWidth(B_Skip), and MbPartHeight(B_Skip) are used in the derivation process for motion vectors and reference frame indices in subclause 8.4.1 for direct mode prediction.

The following semantics are assigned to the macroblock prediction modes (MbPartPredMode( )) in Table 7-11.

– Direct: no motion vector differences or reference indices are present for the macroblock (in case of B_Skip or B_Direct_16x16) in the bitstream. Direct is an Inter macroblock prediction mode.

– Pred_L0: see semantics for Table 7-10.

– Pred_L1: specifies that the Inter prediction process is invoked using list 1 prediction. Pred_L1 is an Inter macroblock prediction mode.

– BiPred: specifies that the Inter prediction process is invoked using list 0 and list 1 prediction. BiPred is an Inter macroblock prediction mode.

pcm_alignment_zero_bit is a bit equal to 0.

pcm_byte[i] is a sample value. pcm_byte[i] shall not be equal to 0. The first 256 pcm_byte[i] values represent luma sample values in the raster scan within the macroblock. The next (256 * (ChromaFormatFactor - 1)) / 2 pcm_byte[i] values represent Cb sample values in the raster scan within the macroblock. The last (256 * (ChromaFormatFactor - 1)) / 2 pcm_byte[i] values represent Cr sample values in the raster scan within the macroblock.

coded_block_pattern specifies which of the six 8x8 blocks - luma and chroma – may contain non-zero transform coefficient levels. For macroblocks with prediction mode not equal to Intra_16x16, coded_block_pattern is present in the bitstream and the variables CodedBlockPatternLuma and CodedBlockPatternChroma are derived as follows.

CodedBlockPatternLuma = coded_block_pattern % 16
CodedBlockPatternChroma = coded_block_pattern / 16 (7-22)

When coded_block_pattern is present, CodedBlockPatternLuma specifies, for each of the four 8x8 luma blocks of the macroblock, one of the following cases.

- All transform coefficient levels of the four 4x4 luma blocks in the 8x8 luma block are equal to zero

- One or more transform coefficient levels of one or more of the 4x4 luma blocks in the 8x8 luma block shall be non-zero valued.

The meaning of CodedBlockPatternChroma is specified in Table 7-12.

Table 7-12 – 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 to +25, inclusive. mb_qp_delta shall be inferred to be equal to 0 when it is not present for any macroblock (including P_Skip and B_Skip macroblock types).

The value of QP_Y is derived as

$$QP_Y = (QP_{Y,PREV} + \text{mb\_qp\_delta} + 52) \% 52 \qquad (7\text{-}23)$$

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 7-16 at the start of each slice.

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 7-13.

Table 7-13 – 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 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, and, if applicable, the parity of the field within the reference picture used for prediction are specified as follows.

- If MbaffFrameFlag is equal to 0 or mb_field_decoding_flag is equal to 0, the value of ref_idx_l0[mbPartIdx] shall be in the range of 0 to num_ref_idx_l0_active_minus1, inclusive.
- Otherwise (MbaffFrameFlag is equal to 1 and mb_field_decoding_flag is equal to 1), the value of ref_idx_l0[mbPartIdx] shall be in the range of 0 to 2 * num_ref_idx_l0_active_minus1 + 1, 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.

ref_idx_l1[mbPartIdx **]** has the same semantics as ref_idx_l0, with l0 and list 0 replaced by l1 and list 1, respectively.

mvd_l0[mbPartIdx **][** 0 **][** compIdx **]** specifies the difference between a 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.

mvd_l1[mbPartIdx **][** 0 **][** compIdx **]** has the same semantics as mvd_l0, with l0 and L0 replaced by l1 and L1, respectively.

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, SP, and B slices. Each table presents the value of sub_mb_type, the name of sub_mb_type, the number of sub-macroblock partitions used (given by the NumSubMbPart(sub_mb_type) function), and the prediction mode of the sub-macroblock (given by the SubMbPredMode(sub_mb_type) function). In the text, the value of sub_mb_type may be referred to by "sub-macroblock type". In the text, the value of SubMbPredMode( ) may be referred to by "sub-macroblock prediction mode".

The sub-macroblock types for P macroblock types are specified in Table 7-14.

Table 7-14 – 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])
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 7-14.

– 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 (SubMbPredMode( )) in Table 7-14.

– Pred_L0: see semantics for Table 7-10.

The sub-macroblock types for B macroblock types are specified in Table 7-15.

Table 7-15 – Sub-macroblock types in B 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])
na	B_Skip	na	Direct	4	4
na	B_Direct_16x16	na	Direct	4	4
0	B_Direct_8x8	na	Direct	4	4
1	B_L0_8x8	1	Pred_L0	8	8
2	B_L1_8x8	1	Pred_L1	8	8
3	B_Bi_8x8	1	BiPred	8	8
4	B_L0_8x4	2	Pred_L0	8	4
5	B_L0_4x8	2	Pred_L0	4	8
6	B_L1_8x4	2	Pred_L1	8	4
7	B_L1_4x8	2	Pred_L1	4	8
8	B_Bi_8x4	2	BiPred	8	4
9	B_Bi_4x8	2	BiPred	4	8
10	B_L0_4x4	4	Pred_L0	4	4
11	B_L1_4x4	4	Pred_L1	4	4
12	B_Bi_4x4	4	BiPred	4	4

The following semantics are assigned to the macroblock types in Table 7-15:

– B_Skip and B_Direct_16x16: no motion vector differences or reference indices are present for the sub-macroblock in the bitstream. The functions SubMbPartWidth( ) and SubMbPartHeight( ) are used in the derivation process for motion vectors and reference frame indices in subclause 8.4.1 for direct mode prediction.

– B_Direct_8x8: no motion vector differences or reference indices are present for the sub-macroblock in the bitstream. The functions SubMbPartWidth(B_Direct_8x8) and SubMbPartHeight(B_Direct_8x8) are used in the derivation process for motion vectors and reference frame indices in subclause 8.4.1 for direct mode prediction.

– B_X_MxN, with X being replaced by L0, L1, or Bi, and 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, or the samples of the sub-macroblock are predicted using two luma partitions of size MxN equal to 8x4, or the samples of the sub-macroblock are predicted using two luma partitions of size MxN equal to 4x8, or the samples of the sub-macroblock are predicted using four luma partitions of size MxN equal to 4x4, and associated chroma samples, respectively. All sub-macroblock partitions share the same reference index. For an MxN sub-macroblock partition in a sub-macroblock with sub_mb_type being B_X_MxN with X being replaced by either L0 or L1, one motion vector difference is present in the bitstream. For an MxN sub-macroblock partition in a sub-macroblock with sub_mb_type being B_Bi_MxN, two motion vector difference are present in the bitstream.

The following semantics are assigned to the sub-macroblock prediction modes (SubMbPredMode( )) in Table 7-15.

– Direct: see semantics for Table 7-11.

– Pred_L0: see semantics for Table 7-10.

– Pred_L1: see semantics for Table 7-11.

– BiPred: see semantics for Table 7-11.

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

ref_idx_l1[mbPartIdx **]** has the same semantics as ref_idx_l1 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.

mvd_l1[mbPartIdx **][** subMbPartIdx **][** compIdx **]** has the same semantics as mvd_l1 in subclause 7.4.5.1.

7.4.5.3 Residual data semantics

The syntax structure residual_block( ), which is used for parsing the transform coefficient levels, is assigned as follows.

- If entropy_coding_mode_flag is equal to 0, residual_block is set equal to residual_block_cavlc, which is used for parsing the syntax elements for transform coefficient levels.
- Otherwise (entropy_coding_mode_flag is equal to 1), residual_block is set equal to residual_block_cabac, which is used for parsing the syntax elements for transform coefficient levels.

Depending on mb_type, luma or chroma, the syntax structure residual_block(coeffLevel, maxNumCoeff) is used with the arguments coeffLevel, which is a list containing the maxNumCoeff transform coefficient levels that are parsed in residual_block( ), 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 Intra16x16DCLevel and into the 16 lists Intra16x16ACLevel[i]. Intra16x16DCLevel 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 Intra16x16ACLevel[i].
    - Otherwise (MbPartPredMode(mb_type, 0) is not equal to Intra_16x16), 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 LumaLevel[i].
- For each chroma component, indexed by iCbCr = 0..1, the 4 DC transform coefficient levels of the 4x4 chroma blocks are parsed into iCbCr-th list ChromaDCLevel[iCbCr].
- For each of the 4x4 chroma blocks, indexed by i4x4 = 0..3, of each chroma component, indexed by iCbCr = 0..1, the 15 AC transform coefficient levels are parsed into the i4x4-th list of the iCbCr-th chroma component ChromaACLevel[iCbCr][i4x4].

7.4.5.3.1 Residual block CAVLC semantics

The function TotalCoeff(coeff_token) that is used in subclause 7.3.5.3.1 returns the number of non-zero transform coefficient levels derived from coeff_token.

The function TrailingOnes(coeff_token) that is used in subclause 7.3.5.3.1 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.

7.4.5.3.2 Residual block CABAC semantics

coded_block_flag specifies whether the block contains non-zero transform coefficient levels as follows.

- If coded_block_flag is equal to 0, the block contains no non-zero transform coefficient levels.
- Otherwise (coded_block_flag is equal to 1), the block contains at least one non-zero transform coefficient level.

significant_coeff_flag[i **]** specifies whether the transform coefficient level at scanning position i is non-zero as follows.

- If significant_coeff_flag[i] is equal to 0, the transform coefficient level at scanning position i is set equal to 0;
- Otherwise (significant_coeff_flag[i] is equal to 1), the transform coefficient level at scanning position i has a non-zero value.

last_significant_coeff_flag[i **]** specifies for the scanning position i whether there are non-zero transform coefficient levels for subsequent scanning positions i + 1 to maxNumCoeff – 1 as follows.

- If last_significant_coeff_flag[i] is equal to 1, all following transform coefficient levels (in scanning order) of the block have value equal to 0..
- Otherwise (last_significant_coeff_flag[i] is equal to 0), there are further non-zero transform coefficient levels along the scanning path.

coeff_abs_level_minus1[i **]** is the absolute value of a transform coefficient level minus 1. The value of coeff_abs_level_minus1 is constrained by the limits in subclause 8.5.

coeff_sign_flag[i **]** specifies the sign of a transform coefficient level as follows.

- If coeff_sign_flag is equal to 0, the corresponding transform coefficient level has a positive value.
- Otherwise (coeff_sign_flag is equal to 1), the corresponding transform coefficient level has a negative value.

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).

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 Recommendation | International Standard.

Each picture referred to in this clause is a primary picture. Each slice referred to in this clause is a slice of a primary picture. Each slice data partition referred to in this clause is a slice data partition of a primary picture.

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)
  - Variables and functions relating to the macroblock to slice group map are derived in subclause 8.2.2. (only needed to be invoked for one slice of a picture)
  - The method of combining the various partitions when slice data partitioning is used is described in subclause 8.2.3.
  - 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.
  - Prior to decoding each slice, the derivation of reference picture lists as described in 8.2.4 is necessary for inter prediction.
  - 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 and SI 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 the constructed samples prior to the deblocking filter process.
  - The inter prediction process for P and B 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 and B macroblocks and for P macroblocks in P slices. The output are constructed samples prior to the deblocking filter process.
  - The decoding process for P macroblocks in SP slices or SI macroblocks is specified in subclause 8.6. That process derives samples for P macroblocks in SP slices and for SI macroblocks. 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 8.7 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.

Subclauses 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, 2, 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 and 2.

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 3 to 5.

Subclause 8.6 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1 and 3 to 5.

Subclause 8.7 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1 to 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 TopFieldOrderCnt (if applicable) and BottomFieldOrderCnt (if applicable).

Picture order counts are used to determine initial picture orderings for reference pictures in the decoding of B slices (see subclauses 8.2.4.2.3 and 8.2.4.2.4), to represent picture order differences between frames or fields for motion vector derivation in temporal direct mode (see subclause 8.4.1.2.3), for implicit mode weighted prediction in B slices (see subclause 8.4.2.3.2), and for decoder conformance checking (see subclause C.4).

Picture order count information is derived for every frame, field (whether decoded from a coded field or as a part of a decoded frame), or complementary field pair as follows:

- Each coded frame is associated with two picture order counts, called TopFieldOrderCnt and BottomFieldOrderCnt for its top field and bottom field, respectively.

- Each coded field is associated with a picture order count, called TopFieldOrderCnt for a coded top field and BottomFieldOrderCnt for a bottom field.

- Each complementary field pair is associated with two picture order counts, which are the TopFieldOrderCnt for its coded top field and the BottomFieldOrderCnt for its coded bottom field, respectively.

TopFieldOrderCnt and BottomFieldOrderCnt indicate the picture order of the corresponding top field or bottom field relative to the first output field of the previous IDR picture or the previous reference picture including a memory_management_control_operation equal to 5 in decoding order.

TopFieldOrderCnt and BottomFieldOrderCnt are 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), TopFieldOrderCnt of the current picture (if any) is set equal to TopFieldOrderCnt - tempPicOrderCnt, and BottomFieldOrderCnt of the current picture (if any) is set equal to BottomFieldOrderCnt - tempPicOrderCnt.

The bitstream shall not contain data that results in Min(TopFieldOrderCnt, BottomFieldOrderCnt) not equal to 0 for a coded IDR frame, TopFieldOrderCnt not equal to 0 for a coded IDR top field, or BottomFieldOrderCnt not equal to 0 for a coded IDR bottom field. Thus, at least one of TopFieldOrderCnt and BottomFieldOrderCnt shall be equal to 0 for the fields of a coded IDR frame.

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

- Consider the list variable listD containing as elements the TopFieldOrderCnt and BottomFieldOrderCnt values associated with the list of pictures including all of the following
  - 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
  - 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 and 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.

- Consider the list variable listO which contains the elements of listD sorted in ascending order. listO shall not contain any of the following.
  - a pair of TopFieldOrderCnt and BottomFieldOrderCnt for a frame or complementary field pair that are not at consecutive positions in listO.
  - a TopFieldOrderCnt that has a value equal to another TopFieldOrderCnt.
  - a BottomFieldOrderCnt that has a value equal to another BottomFieldOrderCnt.
  - a BottomFieldOrderCnt that has a value equal to a TopFieldOrderCnt unless the BottomFieldOrderCnt and TopFieldOrderCnt belong to the same coded frame or complementary field pair.

The bitstream shall not contain data that results in values of TopFieldOrderCnt, BottomFieldOrderCnt, 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:

```
if( picX is a frame or a complementary field pair )
    PicOrderCnt( picX ) = Min( TopFieldOrderCnt, BottomFieldOrderCnt ) of the frame or complementary field
pair picX
else if( picX is a top field )
    PicOrderCnt( picX ) = TopFieldOrderCnt of field picX                                    (8-1)
else if( picX is a bottom field )
    PicOrderCnt( picX ) = BottomFieldOrderCnt of field picX
```

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

$$\text{DiffPicOrderCnt(picA, picB) = PicOrderCnt(picA) - PicOrderCnt(picB)} \qquad (8\text{-}2)$$

The bitstream shall contain data that results in values of DiffPicOrderCnt(picA, picB) used in the decoding process that are in 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.

NOTE – Many applications assign PicOrderCnt(X) proportional to the sampling time of the picture X relative to the sampling time of an IDR picture.

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 are either or both TopFieldOrderCnt or BottomFieldOrderCnt.

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.

  - If the previous reference picture in decoding order is not a bottom field, prevPicOrderCntMsb is set equal to 0 and prevPicOrderCntLsb is set equal to the value of TopFieldOrderCnt for the previous reference picture in decoding order.
  - Otherwise (the previous reference picture in decoding order is a bottom field), prevPicOrderCntMsb is set equal to 0 and prevPicOrderCntLsb is set equal to 0.
- 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 follows:

```
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
```

When the current picture is not a bottom field, TopFieldOrderCnt is derived as follows:

```
if( !field_pic_flag || !bottom_field_flag )
    TopFieldOrderCnt = PicOrderCntMsb + pic_order_cnt_lsb                      (8-4)
```

When the current picture is not a top field, BottomFieldOrderCnt is derived as follows:

```
if( !field_pic_flag )
    BottomFieldOrderCnt = TopFieldOrderCnt + delta_pic_order_cnt_bottom
else if( bottom_field_flag )                                                  (8-5)
    BottomFieldOrderCnt = PicOrderCntMsb + pic_order_cnt_lsb
```

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 are either or both TopFieldOrderCnt or BottomFieldOrderCnt.

The values of TopFieldOrderCnt and BottomFieldOrderCnt are 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 derivation proceeds in the following ordered steps.

1. The variable FrameNumOffset is derived as follows:

```
if( nal_unit_type = = 5 )
    FrameNumOffset = 0
else if( prevFrameNum > frame_num )                                           (8-6)
    FrameNumOffset = prevFrameNumOffset + MaxFrameNum
else
    FrameNumOffset = prevFrameNumOffset
```

2. The variable absFrameNum is derived as follows:

```
if( num_ref_frames_in_pic_order_cnt_cycle  !=  0 )
    absFrameNum = FrameNumOffset + frame_num
else                                                                    (8-7)
    absFrameNum = 0
if( nal_ref_idc  = =  0  &&  absFrameNum  >  0 )
    absFrameNum = absFrameNum – 1
```

3. When absFrameNum > 0, picOrderCntCycleCnt and frameNumInPicOrderCntCycle are derived as follows:

```
if( absFrameNum  >  0 ) {
    picOrderCntCycleCnt = ( absFrameNum – 1 ) / num_ref_frames_in_pic_order_cnt_cycle
    frameNumInPicOrderCntCycle = ( absFrameNum – 1 ) % num_ref_frames_in_pic_order_cnt_cycle      (8-8)
}
```

4. The variable expectedDeltaPerPicOrderCntCycle is derived as follows:

```
expectedDeltaPerPicOrderCntCycle = 0
for( i = 0;  i < num_ref_frames_in_pic_order_cnt_cycle;  i++ )
    expectedDeltaPerPicOrderCntCycle += offset_for_ref_frame[ i ]                 (8-9)
```

5. The variable expectedPicOrderCnt is derived as follows:

```
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 )                                                    (8-10)
    expectedPicOrderCnt = expectedPicOrderCnt + offset_for_non_ref_pic
```

6. The variables TopFieldOrderCnt or BottomFieldOrderCnt are derived as follows:

```
if( !field_pic_flag ) {
    TopFieldOrderCnt = expectedPicOrderCnt + delta_pic_order_cnt[ 0 ]
    BottomFieldOrderCnt = TopFieldOrderCnt +
            offset_for_top_to_bottom_field + delta_pic_order_cnt[ 1 ]            (8-11)
} else if( !bottom_field_flag )
    TopFieldOrderCnt = expectedPicOrderCnt + delta_pic_order_cnt[ 0 ]
else
    BottomFieldOrderCnt = expectedPicOrderCnt + offset_for_top_to_bottom_field + delta_pic_order_cnt[ 0 ]
```

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 are either or both TopFieldOrderCnt or BottomFieldOrderCnt.

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 follows.

```
if( nal_unit_type  = =  5 )
    FrameNumOffset = 0
else if( prevFrameNum  >  frame_num )                                    (8-12)
    FrameNumOffset = prevFrameNumOffset + MaxFrameNum
else
    FrameNumOffset = prevFrameNumOffset
```

The variable tempPicOrderCnt is derived as follows:

```
if( nal_unit_type  = =  5 )
    tempPicOrderCnt = 0
else if( nal_ref_idc  = =  0 )                                           (8-13)
    tempPicOrderCnt = 2 * ( FrameNumOffset + frame_num ) – 1
else
    tempPicOrderCnt = 2 * ( FrameNumOffset + frame_num )
```

The variables TopFieldOrderCnt or BottomFieldOrderCnt are derived as follows:

```
if( !field_pic_flag ) {
    TopFieldOrderCnt = tempPicOrderCnt
    BottomFieldOrderCnt = tempPicOrderCnt                                (8-14)
} else if( bottom_field_flag )
    BottomFieldOrderCnt = tempPicOrderCnt
else
    TopFieldOrderCnt = tempPicOrderCnt
```

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 TopFieldOrderCnt or more than one of these pictures having the same value of BottomFieldOrderCnt.

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

8.2.2 Decoding process for macroblock to slice group map

Inputs to this process are the active picture parameter set and the slice header of the slice to be decoded.

Output of this process is a macroblock to slice group map MbToSliceGroupMap.

This process is invoked at the start of every slice.

NOTE – The output of this process is equal for all slices of a picture.

When num_slice_groups_minus1 is equal to 1 and slice_group_map_type is equal to 3, 4, or 5, slice groups 0 and 1 have a size and shape determined by slice_group_change_direction_flag as shown in Table 8-1 and specified in subclauses 8.2.2.4-8.2.2.6.

Table 8-1 – Refined slice group map type

slice_group_map_type	slice_group_change_direction_flag	refined slice group map type
3	0	Box-out clockwise
3	1	Box-out counter-clockwise
4	0	Raster scan
4	1	Reverse raster scan
5	0	Wipe right
5	1	Wipe left

In such a case, MapUnitsInSliceGroup0 slice group map units in the specified growth order are allocated for slice group 0 and the remaining PicSizeInMapUnits – MapUnitsInSliceGroup0 slice group map units of the picture are allocated for slice group 1.

When num_slice_groups_minus1 is equal to 1 and slice_group_map_type is equal to 4 or 5, the variable sizeOfUpperLeftGroup is defined as follows:

```
sizeOfUpperLeftGroup = ( slice_group_change_direction_flag ?
                          ( PicSizeInMapUnits – MapUnitsInSliceGroup0 ) : MapUnitsInSliceGroup0 )   (8-15)
```

The variable mapUnitToSliceGroupMap is derived as follows.

- If num_slice_groups_minus1 is equal to 0, the map unit to slice group map is generated for all i ranging from 0 to PicSizeInMapUnits – 1, inclusive, as specified by:

```
mapUnitToSliceGroupMap[ i ] = 0                                                            (8-16)
```

- Otherwise (num_slice_groups_minus1 is not equal to 0), mapUnitToSliceGroupMap is derived as follows.
  – If slice_group_map_type is equal to 0, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.1 applies.
  – Otherwise, if slice_group_map_type is equal to 1, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.2 applies.
  – Otherwise, if slice_group_map_type is equal to 2, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.3 applies.
  – Otherwise, if slice_group_map_type is equal to 3, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.4 applies.
  – Otherwise, if slice_group_map_type is equal to 4, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.5 applies.
  – Otherwise, if slice_group_map_type is equal to 5, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.6 applies.
  – Otherwise (slice_group_map_type is equal to 6), the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.7 applies.

After derivation of the mapUnitToSliceGroupMap, the process specified in subclause 8.2.2.8 is invoked to convert the map unit to slice group map mapUnitToSliceGroupMap to the macroblock to slice group map MbToSliceGroupMap. After derivation of the macroblock to slice group map as specified in subclause 8.2.2.8, the function NextMbAddress(n) is defined as the value of the variable nextMbAddress derived as specified by:

```
i = n + 1
while( i < PicSizeInMbs  &&  MbToSliceGroupMap[ i ]  !=  MbToSliceGroupMap[ n ] )
    i++;
nextMbAddress = i                                                                          (8-17)
```

8.2.2.1 Specification for interleaved slice group map type

The specifications in this subclause apply when slice_group_map_type is equal to 0.

The map unit to slice group map is generated as specified by:

```
i = 0
do
    for( iGroup = 0; iGroup <= num_slice_groups_minus1 && i < PicSizeInMapUnits;
         i += run_length_minus1[ iGroup++ ] + 1 )
        for( j = 0; j <= run_length_minus1[ iGroup ] && i + j < PicSizeInMapUnits; j++ )
            mapUnitToSliceGroupMap[ i + j ] = iGroup                                       (8-18)
while( i < PicSizeInMapUnits )
```

8.2.2.2 Specification for dispersed slice group map type

The specifications in this subclause apply when slice_group_map_type is equal to 1.

The map unit to slice group map is generated as specified by:

```
for( i = 0; i < PicSizeInMapUnits; i++ )
    mapUnitToSliceGroupMap[ i ] = ( ( i % PicWidthInMbs ) +
                                  ( ( ( i / PicWidthInMbs ) * ( num_slice_groups_minus1 + 1 ) ) / 2 ) )
                                  % ( num_slice_groups_minus1 + 1 )                    (8-19)
```

8.2.2.3 Specification for foreground with left-over slice group map type

The specifications in this subclause apply when slice_group_map_type is equal to 2.

The map unit to slice group map is generated as specified by:

```
for( i = 0; i < PicSizeInMapUnits; i++ )
    mapUnitToSliceGroupMap[ i ] = num_slice_groups_minus1
for( iGroup = num_slice_groups_minus1 – 1; iGroup >= 0; iGroup-- ) {
    yTopLeft = top_left[ iGroup ] / PicWidthInMbs
    xTopLeft = top_left[ iGroup ] % PicWidthInMbs
    yBottomRight = bottom_right[ iGroup ] / PicWidthInMbs
    xBottomRight = bottom_right[ iGroup ] % PicWidthInMbs
    for( y = yTopLeft; y <= yBottomRight; y++ )
        for( x = xTopLeft; x <= xBottomRight; x++ )
            mapUnitToSliceGroupMap[ y * PicWidthInMbs + x ] = iGroup                    (8-20)
}
```

After application of the process specified in Equation 8-20, there shall be at least one value of i from 0 to PicSizeInMapUnits – 1, inclusive, for which mapUnitToSliceGroupMap[i] is equal to iGroup for each value of iGroup from 0 to num_slice_groups_minus1, inclusive (i.e., each slice group shall contain at least one slice group map unit).

NOTE – The rectangles may overlap. Slice group 0 contains the macroblocks that are within the rectangle specified by top_left[0] and bottom_right[0]. A slice group having slice group ID greater than 0 and less than num_slice_groups_minus1 contains the macroblocks that are within the specified rectangle for that slice group that are not within the rectangle specified for any slice group having a smaller slice group ID. The slice group with slice group ID equal to num_slice_groups_minus1 contains the macroblocks that are not in the other slice groups.

8.2.2.4 Specification for box-out slice group map types

The specifications in this subclause apply when slice_group_map_type is equal to 3.

The map unit to slice group map is generated as specified by:

```
for( i = 0; i < PicSizeInMapUnits; i++ )
    mapUnitToSliceGroupMap[ i ] = 1
x = ( PicWidthInMbs – slice_group_change_direction_flag ) / 2
y = ( PicHeightInMapUnits – slice_group_change_direction_flag ) / 2
( leftBound, topBound ) = ( x, y )
( rightBound, bottomBound ) = ( x, y )
( xDir, yDir ) = ( slice_group_change_direction_flag – 1, slice_group_change_direction_flag )
for( k = 0; k < MapUnitsInSliceGroup0; k += mapUnitVacant ) {
    mapUnitVacant = ( mapUnitToSliceGroupMap[ y * PicWidthInMbs + x ]  = =  1 )
    if( mapUnitVacant )
        mapUnitToSliceGroupMap[ y * PicWidthInMbs + x ] = 0                    (8-21)
    if( xDir  = =  –1  &&  x  = =  leftBound ) {
        leftBound = Max( leftBound – 1, 0 )
        x = leftBound
        ( xDir, yDir ) = ( 0, 2 * slice_group_change_direction_flag – 1 )
    } else if( xDir  = =  1  &&  x  = =  rightBound ) {
        rightBound = Min( rightBound + 1, PicWidthInMbs – 1 )
        x = rightBound
        ( xDir, yDir ) = ( 0, 1 – 2 * slice_group_change_direction_flag )
    } else if( yDir  = =  –1  &&  y  = =  topBound ) {
        topBound = Max( topBound – 1, 0 )
        y = topBound
        ( xDir, yDir ) = ( 1 – 2 * slice_group_change_direction_flag, 0 )
    } else if( yDir  = =  1  &&  y  = =  bottomBound ) {
        bottomBound  =  Min( bottomBound + 1, PicHeightInMapUnits – 1 )
```

```
            y = bottomBound
            ( xDir, yDir ) = ( 2 * slice_group_change_direction_flag – 1, 0 )
        } else
            ( x, y ) = ( x + xDir, y + yDir )
    }
```

8.2.2.5 Specification for raster scan slice group map types

The specifications in this subclause apply when slice_group_map_type is equal to 4.

The map unit to slice group map is generated as specified by:

```
    for( i = 0; i < PicSizeInMapUnits; i++ )
        if( i < sizeOfUpperLeftGroup )
            mapUnitToSliceGroupMap[ i ] = slice_group_change_direction_flag
        else                                                                        (8-22)
            mapUnitToSliceGroupMap[ i ] = 1 – slice_group_change_direction_flag
```

8.2.2.6 Specification for wipe slice group map types

The specifications in this subclause apply when slice_group_map_type is equal to 5.

The map unit to slice group map is generated as specified by:

```
    k = 0;
    for( j = 0; j < PicWidthInMbs; j++ )
        for( i = 0; i < PicHeightInMapUnits; i++ )
            if( k++ < sizeOfUpperLeftGroup )
                mapUnitToSliceGroupMap[ i * PicWidthInMbs + j ] = slice_group_change_direction_flag
            else                                                                    (8-23)
                mapUnitToSliceGroupMap[ i * PicWidthInMbs + j ] = 1 – slice_group_change_direction_flag
```

8.2.2.7 Specification for explicit slice group map type

The specifications in this subclause apply when slice_group_map_type is equal to 6.

The map unit to slice group map is generated as specified by:

mapUnitToSliceGroupMap[i] = slice_group_id[i] (8-24)

for all i ranging from 0 to PicSizeInMapUnits – 1, inclusive.

8.2.2.8 Specification for conversion of map unit to slice group map to macroblock to slice group map

For each value of i ranging from 0 to PicSizeInMbs – 1, inclusive, the macroblock to slice group map is specified as follows.

– If frame_mbs_only_flag is equal to 1 or field_pic_flag is equal to 1, the macroblock to slice group map is specified by:

  MbToSliceGroupMap[i] = mapUnitToSliceGroupMap[i] (8-25)

– Otherwise, if MbaffFrameFlag is equal to 1, the macroblock to slice group map is specified by:

  MbToSliceGroupMap[i] = mapUnitToSliceGroupMap[i / 2] (8-26)

– Otherwise (frame_mbs_only_flag is equal to 0 and mb_adaptive_frame_field_flag is equal to 0 and field_pic_flag is equal to 0), the macroblock to slice group map is specified by:

  MbToSliceGroupMap[i] = mapUnitToSliceGroupMap[(i / (2 * PicWidthInMbs)) * PicWidthInMbs
  + (i % PicWidthInMbs)] (8-27)

8.2.3 Decoding process for slice data partitioning

Inputs to this process are

– a slice data partition A layer RBSP,

– when syntax elements of category 3 are present in the slice data, a slice data partition B layer RBSP having the same slice_id as in the slice data partition A layer RBSP, and

– when syntax elements of category 4 are present in the slice data, a slice data partition C layer RBSP having the same slice_id as in the slice data partition A layer RBSP.

NOTE – The slice data partition B layer RBSP and slice data partition C layer RBSP need not be present.

Output of this process is a coded slice.

When slice data partitioning is not used, coded slices are represented by a slice layer without partitioning RBSP that contains a slice header followed by a slice data syntax structure that contains all the syntax elements of categories 2, 3, and 4 (see category column in subclause 7.3) of the macroblock data for the macroblocks of the slice.

When slice data partitioning is used, the macroblock data of a slice is partitioned into one to three partitions contained in separate NAL units. Partition A contains a slice data partition A header, and all syntax elements of category 2. Partition B, when present, contains a slice data partition B header and all syntax elements of category 3. Partition C, when present, contains a slice data partition C header and all syntax elements of category 4.

When slice data partitioning is used, the syntax elements of each category are parsed from a separate NAL unit, which need not be present when no symbols of the respective category exist. The decoding process shall process the slice data partitions of a coded slice in a manner equivalent to processing a corresponding slice layer without partitioning RBSP by extracting each syntax element from the slice data partition in which the syntax element appears depending on the slice data partition assignment in the syntax tables in subclause 7.3.

NOTE - Syntax elements of category 3 are relevant to the decoding of residual data of I and SI macroblock types. Syntax elements of category 4 are relevant to the decoding of residual data of P and B macroblock types. Category 2 encompasses all other syntax elements related to the decoding of macroblocks, and their information is often denoted as header information. The slice data partition A header contains all the syntax elements of the slice header, and additionally a slice_id that are used to associate the slice data partitions B and C with the slice data partition A. The slice data partition B and C headers contain the slice_id syntax element that establishes their association with the slice data partition A of the slice.

8.2.4 Decoding process for reference picture lists construction

This process is invoked at the beginning of decoding of each P, SP, or B slice.

Outputs of this process are a reference picture list RefPicList0 and, when decoding a B slice, a second reference picture list RefPicList1.

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 decoded reference pictures are identified by the value of frame_num. Long-term decoded 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 list of variables PicNum and LongTermPicNum, which is called a reference picture list. When decoding a P or SP slice, there is a single reference picture list RefPicList0. When decoding a B slice, there is a second independent reference picture list RefPicList1 in addition to RefPicList0.

At the beginning of decoding of each slice, reference picture list RefPicList0, and for B slices RefPicList1, are derived as follows.

- An initial reference picture list RefPicList0 and for B slices RefPicList1 are derived as specified in subclause 8.2.4.2.

- The initial reference picture list RefPicList0 and for B slices RefPicList1 are modified as specified in subclause 8.2.4.3.

The number of entries in the modified reference picture list RefPicList0 is num_ref_idx_l0_active_minus1 + 1, and for B slices the number of entries in the modified reference picture list RefPicList1 is num_ref_idx_l1_active_minus1 + 1. A reference picture may appear at more than one index in the modified reference picture lists RefPicList0 or RefPicList1.

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 or the decoded reference picture marking process specified in subclause 8.2.5 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, and for the decoded reference picture marking process in subclause 8.2.5.

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                    (8-28)
else
    FrameNumWrap = FrameNum
```

where the value of frame_num used in Equation 8-28 is the frame_num in the slice header(s) for the current picture.

To each long-term reference picture the variable LongTermFrameIdx is assigned 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. The values of these variables depend on the value of field_pic_flag and bottom_field_flag for the current picture and they are set as follows.

- If field_pic_flag is equal to 0, the following applies.
  - For each short-term reference frame or complementary reference field pair:

$$\text{PicNum} = \text{FrameNumWrap} \tag{8-29}$$

  - For each long-term reference frame or long-term complementary reference field pair:

$$\text{LongTermPicNum} = \text{LongTermFrameIdx} \tag{8-30}$$

    NOTE – When decoding a frame the value of MbaffFrameFlag has no influence on the derivations in subclauses 8.2.4.2, 8.2.4.3, and 8.2.5.

- Otherwise (field_pic_flag is equal to 1), the following applies.
  - For each short-term reference field the following applies.
    - If the reference field has the same parity as the current field

$$\text{PicNum} = 2 * \text{FrameNumWrap} + 1 \tag{8-31}$$

    - Otherwise (the reference field has the opposite parity of the current field),

$$\text{PicNum} = 2 * \text{FrameNumWrap} \tag{8-32}$$

  - For each long-term reference field the following applies.
    - If the reference field has the same parity as the current field

$$\text{LongTermPicNum} = 2 * \text{LongTermFrameIdx} + 1 \tag{8-33}$$

    - Otherwise (the reference field has the opposite parity of the current field),

$$\text{LongTermPicNum} = 2 * \text{LongTermFrameIdx} \tag{8-34}$$

8.2.4.2 Initialisation process for reference picture lists

This initialisation process is invoked when decoding a P, SP, or B slice header.

Outputs of this process are initial reference picture list RefPicList0, and when decoding a B slice, initial reference picture list RefPicList1.

RefPicList0 and RefPicList1 have initial entries of the variables PicNum and LongTermPicNum as specified in subclauses 8.2.4.2.1 through 8.2.4.2.5.

When the number of entries in the initial RefPicList0 or RefPicList1 produced as specified in subclauses 8.2.4.2.1 through 8.2.4.2.5 is greater than num_ref_idx_l0_active_minus1 + 1 or num_ref_idx_l1_active_minus1 + 1, respectively, the extra entries past position num_ref_idx_l0_active_minus1 or num_ref_idx_l1_active_minus1 are discarded from the initial reference picture list.

When the number of entries in the initial RefPicList0 or RefPicList1 produced as specified in subclauses 8.2.4.2.1 through 8.2.4.2.5 is less than num_ref_idx_l0_active_minus1 + 1 or num_ref_idx_l1_active_minus1 + 1, respectively, 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 and SP slices in frames

This initialisation process is invoked when decoding a P or SP slice in a coded frame.

Output of this process is the initial reference picture list RefPicList0.

When this process is invoked, there shall be at least one reference frame or complementary reference field pair that is currently marked as "used for short-term reference" or "used for long-term reference".

The reference picture list RefPicList0 is ordered so that short-term reference frames and short-term complementary reference field pairs have lower indices than long-term reference frames and long-term complementary reference field pairs.

The short-term reference frames and complementary reference field pairs are ordered starting with the frame or complementary field pair with the highest PicNum value and proceeding through in descending order to the frame or complementary field pair with the lowest PicNum value.

The long-term reference frames and complementary reference field pairs are ordered starting with the frame or complementary field pair with the lowest LongTermPicNum value and proceeding through in ascending order to the frame or complementary field pair with the highest LongTermPicNum value.

NOTE – A non-paired reference field is not used for inter prediction for decoding a frame, regardless of the value of MbaffFrameFlag.

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, and
- RefPicList0[4] is set equal to the long-term reference picture with LongTermPicNum = 3.

8.2.4.2.2 Initialisation process for the reference picture list for P and SP slices in fields

This initialisation process is invoked when decoding a P or SP slice in a coded field.

Output of this process is initial reference picture list RefPicList0.

When decoding a field, each field included in the reference picture list has a separate index in the list.

NOTE - When decoding a field, there are effectively at least twice as many pictures available for referencing as there would be when decoding a frame at the same position in decoding order.

Two ordered lists of reference frames, refFrameList0ShortTerm and refFrameList0LongTerm, are derived as follows. For purposes of the formation of this list of frames, decoded frames, complementary reference field pairs, non-paired reference fields and reference frames in which a single field is marked "used for short-term reference" or "used for long-term reference" are all considered reference frames.

- The FrameNumWrap of all frames having one or more fields marked "used for short-term reference" are included in the list of short-term reference frames refFrameList0ShortTerm. When the current field is the second field (in decoding order) of a complementary reference field pair and the first field is marked as "used for short-term reference", the FrameNumWrap of the first field is included in the list refFrameList0ShortTerm. refFrameList0ShortTerm is ordered starting with the frame with the highest FrameNumWrap value and proceeding through in descending order to the frame with the lowest FrameNumWrap value.
- The LongTermFrameIdx of all frames having one or more fields marked "used for long-term reference" are included in the list of long-term reference frames refFrameList0LongTerm. When the current field is the second field (in decoding order) of a complementary reference field pair and the first field is marked as "used for long-term reference", the LongTermFrameIdx of the first field is included in the list refFrameList0LongTerm. refFrameList0LongTerm is ordered starting with the frame with the lowest LongTermFrameIdx value and proceeding through in ascending order to the frame with the highest LongTermFrameIdx value.

The process specified in subclause 8.2.4.2.5 is invoked with refFrameList0ShortTerm and refFrameList0LongTerm given as input and the output is assigned to RefPicList0.

8.2.4.2.3 Initialisation process for reference picture lists for B slices in frames

This initialisation process is invoked when decoding a B slice in a coded frame.

Outputs of this process are the initial reference picture lists RefPicList0 and RefPicList1.

When this process is invoked, there shall be at least one reference frame or complementary reference field pair that is currently marked as "used for short-term reference" or "used for long-term reference".

For B slices, the order of short-term reference pictures in the reference picture lists RefPicList0 and RefPicList1 depends on output order, as given by PicOrderCnt( ). When pic_order_cnt_type is equal to 0, reference pictures that are marked as "non-existing" as specified in subclause 8.2.5.2 are not included in either RefPicList0 or RefPicList1.

NOTE – When gaps_in_frame_num_value_allowed_flag is equal to 1, encoders should use reference picture list reordering to ensure proper operation of the decoding process (particularly when pic_order_cnt_type is equal to 0, in which case PicOrderCnt( ) is not inferred for "non-existing" frames).

The reference picture list RefPicList0 is ordered such that short-term reference frames and short-term complementary reference field pairs have lower indices than long-term reference frames and long-term complementary reference field pairs. It is ordered as follows.

- Short-term reference frames and short-term complementary reference field pairs are ordered starting with the short-term reference frame or complementary reference field pair frm0 with the largest value of PicOrderCnt(frm0) less than the value of PicOrderCnt(CurrPic) and proceeding through in descending order to the short-term reference frame or complementary reference field pair frm1 that has the smallest value of PicOrderCnt(frm1), and then continuing with the short-term reference frame or complementary reference field pair frm2 with the smallest value of PicOrderCnt(frm2) greater than the value of PicOrderCnt(CurrPic) of the current frame and proceeding through in ascending order to the short-term reference frame or complementary reference field pair frm3 that has the largest value of PicOrderCnt(frm3).
- The long-term reference frames and long-term complementary reference field pairs are ordered starting with the long-term reference frame or complementary reference field pair that has the lowest LongTermPicNum value and proceeding through in ascending order to the long-term reference frame or complementary reference field pair that has the highest LongTermPicNum value.

The reference picture list RefPicList1 is ordered so that short-term reference frames and short-term complementary reference field pairs have lower indices than long-term reference frames and long-term complementary reference field pairs. It is ordered as follows.

- Short-term reference frames and short-term complementary reference field pairs are ordered starting with the short-term reference frame or complementary reference field pair frm4 with the smallest value of PicOrderCnt(frm4) greater than the value of PicOrderCnt(CurrPic) of the current frame and proceeding through in ascending order to the short-term reference frame or complementary reference field pair frm5 that has the largest value of PicOrderCnt(frm5), and then continuing with the short-term reference frame or complementary reference field pair frm6 with the largest value of PicOrderCnt(frm6) less than the value of PicOrderCnt(CurrPic) of the current frame and proceeding through in descending order to the short-term reference frame or complementary reference field pair frm7 that has the smallest value of PicOrderCnt(frm7).

- Long-term reference frames and long-term complementary reference field pairs are ordered starting with the long-term reference frame or complementary reference field pair that has the lowest LongTermPicNum value and proceeding through in ascending order to the long-term reference frame or complementary reference field pair that has the highest LongTermPicNum value.
- When the reference picture list RefPicList1 has more than one entry and RefPicList1 is identical to the reference picture list RefPicList0, the first two entries RefPicList1[0] and RefPicList1[1] are switched.

  NOTE – A non-paired reference field is not used for inter prediction of frames independent of the value of MbaffFrameFlag.

8.2.4.2.4 Initialisation process for reference picture lists for B slices in fields

This initialisation process is invoked when decoding a B slice in a coded field.

Outputs of this process are the initial reference picture lists RefPicList0 and RefPicList1.

When decoding a field, each field of a stored reference frame is identified as a separate reference picture with a unique index. The order of short-term reference pictures in the reference picture lists RefPicList0 and RefPicList1 depend on output order, as given by PicOrderCnt( ). When pic_order_cnt_type is equal to 0, reference pictures that are marked as "non-existing" as specified in subclause 8.2.5.2 are not included in either RefPicList0 or RefPicList1.

NOTE – When gaps_in_frame_num_value_allowed_flag is equal to 1, encoders should use reference picture list reordering to ensure proper operation of the decoding process (particularly when pic_order_cnt_type is equal to 0, in which case PicOrderCnt( ) is not inferred for "non-existing" frames).

NOTE – When decoding a field, there are effectively at least twice as many pictures available for referencing as there would be when decoding a frame at the same position in decoding order.

Three ordered lists of reference frames, refFrameList0ShortTerm, refFrameList1ShortTerm and refFrameListLongTerm, are derived as follows. For purposes of the formation of these lists of frames the term reference entry refers in the following to decoded reference frames, complementary reference field pairs, or non-paired reference fields. When pic_order_cnt_type is equal to 0, the term reference entry does not refer to frames that are marked as "non-existing" as specified in subclause 8.2.5.2.

- refFrameList0ShortTerm is ordered starting with the reference entry f0 with the largest value of PicOrderCnt(f0) less than or equal to the value of PicOrderCnt(CurrPic) of the current field and proceeding through in descending order to the short-term reference entry f1 that has the smallest value of PicOrderCnt(f1), and then continuing with the reference entry f2 with the smallest value of PicOrderCnt(f2) greater than the value of PicOrderCnt(CurrPic) of the current field and proceeding through in ascending order to the short-term reference entry f3 that has the largest value of PicOrderCnt(f3).

  NOTE - When the current field follows in decoding order a coded field fldPrev with which together it forms a complementary reference field pair, fldPrev is included into the list refFrameList0ShortTerm using PicOrderCnt(fldPrev) and the ordering method described in the previous sentence is applied.

- refFrameList1ShortTerm is ordered starting with the reference entry f4 with the smallest value of PicOrderCnt(f4) greater than the value of PicOrderCnt(CurrPic) of the current field and proceeding through in ascending order to the short-term reference entry f5 that has the largest value of PicOrderCnt(f5), and then continuing with the reference entry f6 with the largest value of PicOrderCnt(f6) less than or equal to the value of PicOrderCnt(CurrPic) of the current field and proceeding through in descending order to the short-term reference entry f7 that has the smallest value of PicOrderCnt(f7).

  NOTE - When the current field follows in decoding order a coded field fldPrev with which together it forms a complementary reference field pair, fldPrev is included into the list refFrameList1ShortTerm using PicOrderCnt(fldPrev) and the ordering method described in the previous sentence is applied.

- refFrameListLongTerm is ordered starting with the reference entry having the lowest LongTermFrameIdx value and proceeding through in ascending order to the reference entry having highest LongTermFrameIdx value.

  NOTE - When the complementary field of the current picture is marked "used for long-term reference" it is included into the list refFrameListLongTerm. A reference entry in which only one field is marked as “used for long-term reference” is included into the list refFrameListLongTerm.

The process specified in subclause 8.2.4.2.5 is invoked with refFrameList0ShortTerm and refFrameListLongTerm given as input and the output is assigned to RefPicList0.

The process specified in subclause 8.2.4.2.5 is invoked with refFrameList1ShortTerm and refFrameListLongTerm given as input and the output is assigned to RefPicList1.

When the reference picture list RefPicList1 has more than one entry and RefPicList1 is identical to the reference picture list RefPicList0, the first two entries RefPicList1[0] and RefPicList1[1] are switched.

8.2.4.2.5 Initialisation process for reference picture lists in fields

Inputs of this process are the reference frame lists refFrameListXShortTerm (with X may be 0 or 1) and refFrameListLongTerm.

Output of this process is reference picture list RefPicListX (which may be RefPicList0 or RefPicList1).

The reference picture list RefPicListX is a list ordered such that short-term reference fields have lower indices than long-term reference fields. Given the reference frame lists refFrameListXShortTerm and refFrameListLongTerm, it is derived as follows.

- Short-term reference fields are ordered by selecting reference fields from the ordered list of frames refFrameListXShortTerm by alternating between fields of differing parity, starting with a field that has the same parity as the current field (when present). When one field of a reference frame was not decoded or is not marked as "used for short-term reference", the missing field is ignored and instead the next available stored reference field of the chosen parity from the ordered list of frames refFrameListXShortTerm is inserted into RefPicListX. When there are no more short-term reference fields of the alternate parity in the ordered list of frames refFrameListXShortTerm, the next not yet indexed fields of the available parity are inserted into RefPicListX in the order in which they occur in the ordered list of frames refFrameListXShortTerm.
- Long-term reference fields are ordered by selecting reference fields from the ordered list of frames refFrameListLongTerm by alternating between fields of differing parity, starting with a field that has the same parity as the current field (when present). When one field of a reference frame was not decoded or is not marked as "used for long-term reference", the missing field is ignored and instead the next available stored reference field of the chosen parity from the ordered list of frames refFrameListLongTerm is inserted into RefPicListX. When there are no more long-term reference fields of the alternate parity in the ordered list of frames refFrameListLongTerm, the next not yet indexed fields of the available parity are inserted into RefPicListX in the order in which they occur in the ordered list of frames refFrameListLongTerm.

8.2.4.3 Reordering process for reference picture lists

Input to this process is reference picture list RefPicList0 and, when decoding a B slice, also reference picture list RefPicList1.

Outputs of this process are a possibly modified reference picture list RefPicList0 and, when decoding a B slice, also a possibly modified reference picture list RefPicList1.

When ref_pic_list_reordering_flag_l0 is equal to 1, the following applies.

- Let refIdxL0 be an index into the reference picture list RefPicList0. It is initially set equal to 0.
- The corresponding syntax elements reordering_of_pic_nums_idc are processed in the order they occur in the bitstream. For each of these syntax elements, the following applies.
  - If reordering_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 RefPicList0 and refIdxL0 given as input, and the output is assigned to RefPicList0 and refIdxL0.
  - Otherwise, if reordering_of_pic_nums_idc is equal to 2, the process specified in subclause 8.2.4.3.2 is invoked with RefPicList0 and refIdxL0 given as input, and the output is assigned to RefPicList0 and refIdxL0.
  - Otherwise (reordering_of_pic_nums_idc is equal to 3), the reordering process for reference picture list RefPicList0 is finished.

When ref_pic_list_reordering_flag_l1 is equal to 1, the following applies.

- Let refIdxL1 be an index into the reference picture list RefPicList1. It is initially set equal to 0.
- The corresponding syntax elements reordering_of_pic_nums_idc are processed in the order they occur in the bitstream. For each of these syntax elements, the following applies.
  - If reordering_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 RefPicList1 and refIdxL1 given as input, and the output is assigned to RefPicList1 and refIdxL1.
  - Otherwise, if reordering_of_pic_nums_idc is equal to 2, the process specified in subclause 8.2.4.3.2 is invoked with RefPicList1 and refIdxL1 given as input, and the output is assigned to RefPicList1 and refIdxL1.
  - Otherwise (reordering_of_pic_nums_idc is equal to 3), the reordering process for reference picture list RefPicList1 is finished.

8.2.4.3.1 Reordering process of reference picture lists for short-term pictures

Inputs to this process are reference picture list RefPicListX (with X being 0 or 1) and an index refIdxLX into this list.

Outputs of this process are a possibly modified reference picture list RefPicListX and the incremented index refIdxLX.

The variable picNumLXNoWrap is derived as follows.

- If reordering_of_pic_nums_idc is equal to 0

```
if( picNumLXPred – ( abs_diff_pic_num_minus1 + 1 ) < 0 )
    picNumLXNoWrap = picNumLXPred – ( abs_diff_pic_num_minus1 + 1 ) + MaxPicNum        (8-35)
else
    picNumLXNoWrap = picNumLXPred – ( abs_diff_pic_num_minus1 + 1 )
```

- Otherwise (reordering_of_pic_nums_idc is equal to 1),

```
if( picNumLXPred + ( abs_diff_pic_num_minus1 + 1 ) >= MaxPicNum )
    picNumLXNoWrap = picNumLXPred + ( abs_diff_pic_num_minus1 + 1 ) – MaxPicNum        (8-36)
else
    picNumLXNoWrap = picNumLXPred + ( abs_diff_pic_num_minus1 + 1 )
```

picNumLXPred is the prediction value for the variable picNumLXNoWrap. When the process specified in this subclause is invoked the first time for a slice (that is, for the first occurrence of reordering_of_pic_nums_idc equal to 0 or 1 in the ref_pic_list_reordering( ) syntax), picNumL0Pred and picNumL1Pred are initially set equal to CurrPicNum. After each assignment of picNumLXNoWrap, the value of picNumLXNoWrap is assigned to picNumLXPred.

The variable picNumLX is derived as follows

```
if( picNumLXNoWrap > CurrPicNum )
    picNumLX = picNumLXNoWrap – MaxPicNum        (8-37)
else
    picNumLX = picNumLXNoWrap
```

picNumLX 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 shall be conducted to place the picture with short-term picture number picNumLX into the index position refIdxLX, shift the position of any other remaining pictures to later in the list, and increment the value of refIdxLX.

```
for( cIdx = num_ref_idx_lX_active_minus1 + 1; cIdx > refIdxLX; cIdx-- )
    RefPicListX[ cIdx ] = RefPicListX[ cIdx – 1]
RefPicListX[ refIdxLX++ ] = short-term reference picture with PicNum equal to picNumLX
nIdx = refIdxLX
for( cIdx = refIdxLX; cIdx <= num_ref_idx_lX_active_minus1 + 1; cIdx++ )        (8-38)
    if( PicNumF( RefPicListX[ cIdx ] ) != picNumLX )
        RefPicListX[ nIdx++ ] = RefPicListX[ cIdx ]
```

where the function PicNumF(RefPicListX[cIdx]) is derived as follows:

– If the picture RefPicListX[cIdx] is marked as "used for short-term reference", PicNumF(RefPicListX[cIdx]) is the PicNum of the picture RefPicListX[cIdx].

– Otherwise (the picture RefPicListX[cIdx] is not marked as "used for short-term reference"), PicNumF(RefPicListX[cIdx]) is equal to MaxPicNum.

NOTE – A value of MaxPicNum can never be equal to picNumLX.

NOTE – Within this pseudo-code procedure, the length of the list RefPicListX 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_lX_active_minus1 of the list need to be retained.

8.2.4.3.2 Reordering process of reference picture lists for long-term pictures

Inputs to this process are reference picture list RefPicListX (with X being 0 or 1) and an index refIdxLX into this list.

Outputs of this process are a possibly modified reference picture list RefPicListX and the incremented index refIdxLX.

The following procedure shall be conducted to place the picture with long-term picture number long_term_pic_num into the index position refIdxLX, shift the position of any other remaining pictures to later in the list, and increment the value of refIdxLX.

```
for( cIdx = num_ref_idx_lX_active_minus1 + 1; cIdx > refIdxLX; cIdx-- )
    RefPicListX[ cIdx ] = RefPicListX[ cIdx – 1]
RefPicListX[ refIdxLX++ ] = long-term reference picture with LongTermPicNum equal to long_term_pic_num
nIdx = refIdxLX
for( cIdx = refIdxLX; cIdx <= num_ref_idx_lX_active_minus1 + 1; cIdx++ )                    (8-39)
    if( LongTermPicNumF( RefPicListX[ cIdx ] ) != long_term_pic_num )
        RefPicListX[ nIdx++ ] = RefPicListX[ cIdx ]
```

where the function LongTermPicNumF(RefPicListX[cIdx]) is derived as follows:

– If the picture RefPicListX[cIdx] is marked as "used for long-term reference", LongTermPicNumF(RefPicListX[cIdx]) is the LongTermPicNum of the picture RefPicListX[cIdx].

– Otherwise (the picture RefPicListX[cIdx] is not marked as "used for long-term reference"), LongTermPicNumF(RefPicListX[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 RefPicListX 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_lX_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". For a decoded reference frame, both of its fields are marked the same as the frame. For a complementary reference field pair, the pair is marked the same as both of its fields. A picture that is marked as "used for short-term reference" is identified by its FrameNum and, when it is a field, by its parity. A picture that is marked as "used for long-term reference" is identified by its LongTermFrameIdx and, when it is a field, by its parity.

Frames or complementary field pairs 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, the complementary field pair, or one of its constituent fields is marked as “unused for reference”. A field 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 field until 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 applies.

        - All reference pictures shall be marked as "unused for reference"

        - Depending on long_term_reference_flag, the following applies.

- If long_term_reference_flag is equal to 0, the IDR picture shall be marked as "used for short-term reference" and MaxLongTermFrameIdx shall be set equal to "no long-term frame indices".
- Otherwise (long_term_reference_flag is equal to 1), the IDR picture shall be marked as "used for long-term reference", the LongTermFrameIdx for the IDR picture shall be set equal to 0, and MaxLongTermFrameIdx shall be 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".

After marking the current decoded reference picture, the total number of frames with at least one field marked as "used for reference", plus the number of complementary field pairs with at least one field marked as "used for reference", plus the number of non-paired fields marked as "used for reference" shall not be greater than 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 Recommendation | 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 7-10 except the value of frame_num for the current picture.

The decoding process shall generate and mark a frame 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 7-10, using the "sliding window" picture marking process as specified in subclause 8.2.5.3. The generated frames shall also be marked as "non-existing" and "used for short-term reference". The sample values of the generated frames may be set to any value. These generated frames which are marked as "non-existing" shall not be referred to in the inter prediction process, shall not be referred to in the reordering commands for reference picture lists for short-term pictures (subclause 8.2.4.3.1), and shall not be referred to in the assignment process of a LongTermFrameIdx to a short-term picture (subclause 8.2.5.4.3).

When pic_order_cnt_type is not equal to 0, TopFieldOrderCnt and BottomFieldOrderCnt are derived for each of the "non-existing" frames by invoking the decoding process for picture order count in subclause 8.2.1. When invoking the process in subclause 8.2.1 for a particular "non-existing" frame, the current picture is considered to be a picture considered having frame_num inferred to be equal to UnusedShortTermFrameNum, nal_ref_idc inferred to be not equal to 0, nal_unit_type inferred to be not equal to 5, field_pic_flag inferred to be equal to 0, adaptive_ref_pic_marking_mode_flag inferred to be equal to 0, delta_pic_order_cnt[0] (if needed) inferred to be equal to 0, and delta_pic_order_cnt[1] (if needed) inferred to be equal to 0.

NOTE - The decoding process should infer an unintentional picture loss when any of these values of frame_num pertaining to "non-existing" pictures is referred to in the inter prediction process, is referred to in the reordering commands for reference picture lists for short-term pictures (subclause 8.2.4.3.1), or is referred to in the assignment process of a LongTermFrameIdx to a short-term picture (subclause 8.2.5.4.3). The decoding process should not infer an unintentional picture loss when a memory management control operation not equal to 3 is applied to a frame marked as "non-existing".

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.

- If the current picture is a coded field that is the second field in decoding order of a complementary reference field pair, and the first field has been marked as "used for short-term reference", the current picture is also marked as "used for short-term reference".
- Otherwise, the following applies.

- Let numShortTerm be the total number of reference frames, complementary reference field pairs and non-paired reference fields for which at least one field is marked as "used for short-term reference". Let numLongTerm be the total number of reference frames, complementary reference field pairs and non-paired reference fields for which at least one field is marked as "used for long-term reference".
- When numShortTerm + numLongTerm is equal to Max(num_ref_frames, 1), the condition that numShortTerm is greater than 0 shall be fulfilled, and the short-term reference frame, complementary reference field pair or non-paired reference field that has the smallest value of FrameNumWrap is marked as "unused for reference". When it is a frame or a complementary field pair, both of its fields are also 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.

- If field_pic_flag is equal to 0, memory_management_control_operation commands are applied to the frames or complementary reference field pairs specified.
- Otherwise (field_pic_flag is equal to 1), memory_management_control_operation commands are applied to the individual reference fields specified.

8.2.5.4.1 Marking process of a short-term picture as "unused for reference"

This process is invoked when memory_management_control_operation is equal to 1.

Let picNumX be specified by

$$\text{picNumX} = \text{CurrPicNum} - (\,\text{difference\_of\_pic\_nums\_minus1} + 1\,). \qquad (8\text{-}40)$$

Depending on field_pic_flag the value of picNumX is used to mark a short-term picture as "unused for reference" as follows.

- If field_pic_flag is equal to 0, the short-term reference frame or short-term complementary reference field pair specified by picNumX and both of its fields are marked as "unused for reference".
- Otherwise (field_pic_flag is equal to 1), the short-term reference field specified by picNumX is marked as "unused for reference". When that reference field is part of a reference frame or a complementary reference field pair, the frame or complementary field pair is also marked as "unused for reference", but the marking of the other field is not changed.

8.2.5.4.2 Marking process of a long-term picture as "unused for reference"

This process is invoked when memory_management_control_operation is equal to 2.

Depending on field_pic_flag the value of LongTermPicNum is used to mark a long-term picture as "unused for reference" as follows.

- If field_pic_flag is equal to 0, the long-term reference frame or long-term complementary reference field pair having LongTermPicNum equal to long_term_pic_num and both of its fields are marked as "unused for reference".
- Otherwise (field_pic_flag is equal to 1), the long-term reference field specified by LongTermPicNum equal to long_term_pic_num is marked as "unused for reference". When that reference field is part of a reference frame or a complementary reference field pair, the frame or complementary field pair is also marked as "unused for reference", but the marking of the other field is not changed.

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 or complementary reference field pair or non-paired reference field 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 or a long-term complementary reference field pair, that frame or complementary field pair and both of its fields are marked as "unused for reference". When LongTermFrameIdx is already assigned to a non-paired reference field, and the field is not the complementary field of the picture specified by picNumX, that field is marked as “unused for reference”.

Depending on field_pic_flag the value of LongTermFrameIdx is used to mark a picture from "used for short-term reference" to "used for long-term reference" as follows.

- If field_pic_flag is equal to 0, the marking of the short-term reference frame or short-term complementary reference field pair specified by picNumX and both of its fields are changed from "used for short-term reference" to "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.
- Otherwise (field_pic_flag is equal to 1), the marking of the short-term reference field 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" shall be marked as “unused for reference”.

The variable MaxLongTermFrameIdx is derived as follows.

- If max_long_term_frame_idx_plus1 is equal to 0, MaxLongTermFrameIdx shall be set equal to “no long-term frame indices”.
- Otherwise (max_long_term_frame_idx_plus1 is greater than 0), MaxLongTermFrameIdx shall be 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 Recommendation | 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 or a long-term complementary reference field pair, that frame or complementary field pair and both of its fields are marked as "unused for reference". When LongTermFrameIdx is already assigned to a non-paired reference field, and the field is not the complementary field of the current picture, that field 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.

When field_pic_flag is equal to 0, both its fields are also marked as "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.

When field_pic_flag is equal to 1 and the current picture is a second (in decoding order) field of a complementary reference field pair, the pair is also marked as "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.

After marking the current decoded reference picture, the total number of frames with at least one field marked as “used for reference”, plus the number of complementary field pairs with at least one field marked as “used for reference”, plus the number of non-paired fields marked as “used for reference” shall not be greater than Max(num_ref_frames, 1).

NOTE – Under some circumstances, the above statement may impose a constraint on the order in which a memory_management_control_operation syntax element equal to 6 can appear in the decoded reference picture marking syntax relative to a memory_management_control_operation syntax element equal to 1, 2, or 4.

8.3 Intra prediction process

This process is invoked for I and SI macroblock types.

Inputs to this process are constructed samples prior to the deblocking filter process from neighbouring macroblocks and for Intra_4x4 prediction mode, the associated values of Intra4x4PredMode from neighbouring macroblocks.

Outputs of this process are specified as follows.

- If mb_type is not equal to I_PCM, the Intra prediction samples of components of the macroblock or in case of the Intra_4x4 prediction process for luma samples, the outputs are 4x4 luma sample arrays as part of the 16x16 luma array of prediction samples of the macroblock.
- Otherwise (mb_type is equal to I_PCM), constructed macroblock 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 process specified in subclause 8.3.4 is invoked.
- Otherwise (mb_type is not equal to I_PCM), the following applies.
  - 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 specification in subclause 8.3.1 applies.
    - Otherwise (the macroblock prediction mode is equal to Intra_16x16), the specification in subclause 8.3.2 applies.
  - The decoding processes for Intra prediction modes for the chroma components are described in subclause 8.3.3.

Samples used in the Intra prediction process shall be sample values prior to alteration by any deblocking filter operations.

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 constructed luma samples prior to the deblocking filter process from neighbouring macroblocks and the associated values of Intra4x4PredMode from the neighbouring macroblocks or macroblock pairs.

Outputs of this process are 4x4 luma sample arrays as part of the 16x16 luma array of prediction samples of the macroblock pred_L.

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 variable Intra4x4PredMode[luma4x4BlkIdx] is derived as specified in subclause 8.3.1.1.

For each luma block of 4x4 samples indexed using luma4x4BlkIdx = 0..15,

1. The Intra_4x4 sample prediction process in subclause 8.3.1.2 is invoked with luma4x4BlkIdx and constructed samples prior (in decoding order) to the deblocking filter process from adjacent luma blocks as the input and the output are the Intra_4x4 luma prediction samples $\text{pred4x4}_L[\ x, y\ ]$ 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) and x, y = 0..3.

$$\text{pred}_L[\ xO + x, yO + y\ ] = \text{pred4x4}_L[\ x, y\ ] \qquad (8\text{-}41)$$

3. 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 the Intra4x4PredMode

Inputs to this process are the index of the 4x4 luma block luma4x4BlkIdx and variable arrays Intra4x4PredMode that are previously (in decoding order) derived for adjacent macroblocks.

Output of this process is the variable Intra4x4PredMode[luma4x4BlkIdx].

Table 8-2 specifies the values for Intra4x4PredMode[luma4x4BlkIdx] and the associated names.

Table 8-2 – 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.

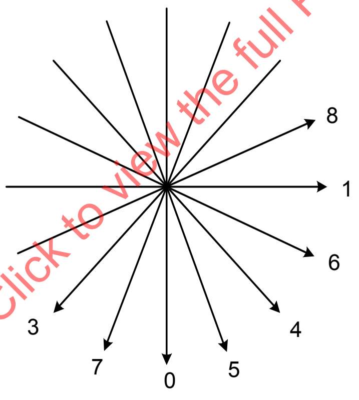


Figure 8-1 – Intra_4x4 prediction mode directions (informative)

Let intra4x4PredModeA and intra4x4PredModeB be variables that specify the intra prediction modes of neighbouring 4x4 luma blocks.

Intra4x4PredMode[luma4x4BlkIdx] is derived as follows.

- The process specified in subclause 6.4.7.3 is invoked with luma4x4BlkIdx given as input and the output is assigned to mbAddrA, luma4x4BlkIdxA, mbAddrB, and luma4x4BlkIdxB.
- The variable dcOnlyPredictionFlag is derived as follows.
  - If one of the following conditions is true, dcOnlyPredictionFlag 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 Inter prediction mode and constrained_intra_pred_flag is equal to 1

- the macroblock with address mbAddrB is available and coded in Inter prediction mode and constrained_intra_pred_flag is equal to 1

- Otherwise, dcOnlyPredictionFlag is set equal to 0.

- For N being either replaced by A or B, the variables intra4x4PredModeN are derived as follows.
  - If dcOnlyPredictionFlag is equal to 1 or the macroblock with address mbAddrN is not coded in Intra_4x4 macroblock prediction mode, intra4x4PredModeN is set equal to 2 (Intra_4x4_DC prediction mode).
  - Otherwise (dcOnlyPredictionFlag is equal to 0 and the macroblock with address mbAddrN is coded in Intra_4x4 macroblock prediction mode), intra4x4PredModeN is set equal to Intra4x4PredMode[luma4x4BlkIdxN], where Intra4x4PredMode is the variable array assigned to the macroblock mbAddrN.
- Intra4x4PredMode[luma4x4BlkIdx] is derived by applying the following procedure.

```
predIntra4x4PredMode = Min( intra4x4PredModeA, intra4x4PredModeB )
if( prev_intra4x4_pred_mode_flag[ luma4x4BlkIdx ] )
    Intra4x4PredMode[ luma4x4BlkIdx ] = predIntra4x4PredMode
else                                                                              (8-42)
    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 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 the 4x4 luma block with index luma4x4BlkIdx and constructed samples prior (in decoding order) to the deblocking filter process from adjacent luma blocks.

Output of this process are the prediction samples $\text{pred4x4}_L[\, x, y \,]$, 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 follows.

– The luma location (xN, yN) is specified by

$$xN = xO + x \qquad (8\text{-}43)$$

$$yN = yO + y \qquad (8\text{-}44)$$

– The derivation process for neighbouring locations in subclause 6.4.8 is invoked for luma locations with (xN, yN) as input and mbAddrN and (xW, yW) as output.

– 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 is 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 Inter 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 luma sample at luma location (xW, yW) inside the macroblock mbAddrN is assigned to p[x, y].

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 frame 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 shall be used.

8.3.1.2.1 Specification of Intra_4x4_Vertical prediction mode

This Intra_4x4 prediction mode shall be used 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

$$\text{pred4x4}_L[x, y] = p[x, -1], \text{ with } x, y = 0..3 \qquad (8\text{-}45)$$

8.3.1.2.2 Specification of Intra_4x4_Horizontal prediction mode

This Intra_4x4 prediction mode shall be used 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

$$\text{pred4x4}_L[x, y] = p[-1, y], \text{ with } x,y = 0..3 \qquad (8\text{-}46)$$

8.3.1.2.3 Specification of Intra_4x4_DC prediction mode

This Intra_4x4 prediction mode shall be used 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

$$\text{pred4x4}_L[x, y] = (p[0, -1] + p[1, -1] + p[2, -1] + p[3, -1] + p[-1, 0] + p[-1, 1] + p[-1, 2] + p[-1, 3] + 4) >> 3 \qquad (8\text{-}47)$$

- 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

$$\text{pred4x4}_L[x, y] = (p[-1, 0] + p[-1, 1] + p[-1, 2] + p[-1, 3] + 2) >> 2 \qquad (8\text{-}48)$$

- 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

$$\text{pred4x4}_L[x, y] = (p[0, -1] + p[1, -1] + p[2, -1] + p[3, -1] + 2) >> 2 \qquad (8\text{-}49)$$

- 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

$$\text{pred4x4}_L[x, y] = 128 \qquad (8\text{-}50)$$

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 shall be used 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,

$$\text{pred4x4}_L[\,x, y\,] = (\,p[\,6, -1\,] + 3 * p[\,7, -1\,] + 2\,) >> 2 \quad (8\text{-}51)$$

- Otherwise (x is not equal to 3 or y is not equal to 3),

$$\text{pred4x4}_L[\,x, y\,] = (\,p[\,x + y, -1\,] + 2 * p[\,x + y + 1, -1\,] + p[\,x + y + 2, -1\,] + 2\,) >> 2 \quad (8\text{-}52)$$

8.3.1.2.5 Specification of Intra_4x4_Diagonal_Down_Right prediction mode

This Intra_4x4 prediction mode shall be used 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,

$$\text{pred4x4}_L[\,x, y\,] = (\,p[\,x - y - 2, -1] + 2 * p[\,x - y - 1, -1\,] + p[\,x - y, -1\,] + 2\,) >> 2 \quad (8\text{-}53)$$

- Otherwise if x is less than y,

$$\text{pred4x4}_L[\,x, y\,] = (\,p[\,-1, y - x - 2\,] + 2 * p[\,-1, y - x - 1\,] + p[\,-1, y - x\,] + 2\,) >> 2 \quad (8\text{-}54)$$

- Otherwise (x is equal to y),

$$\text{pred4x4}_L[\,x, y\,] = (\,p[\,0, -1\,] + 2 * p[\,-1, -1\,] + p[\,-1, 0\,] + 2\,) >> 2 \quad (8\text{-}55)$$

8.3.1.2.6 Specification of Intra_4x4_Vertical_Right prediction mode

This Intra_4x4 prediction mode shall be used 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,

$$\text{pred4x4}_L[\,x, y\,] = (\,p[\,x - (\,y >> 1\,) - 1, -1\,] + p[\,x - (\,y >> 1\,), -1\,] + 1\,) >> 1 \quad (8\text{-}56)$$

- Otherwise, if zVR is equal to 1, 3, or 5,

$$\text{pred4x4}_L[\,x, y\,] = (\,p[\,x - (\,y >> 1\,) - 2, -1] + 2 * p[\,x - (\,y >> 1\,) - 1, -1\,] + p[\,x - (\,y >> 1\,), -1\,] + 2\,) >> 2 \quad (8\text{-}57)$$

- Otherwise, if zVR is equal to -1,

$$\text{pred4x4}_L[\,x, y\,] = (\,p[\,-1, 0\,] + 2 * p[\,-1, -1\,] + p[\,0, -1\,] + 2\,) >> 2 \quad (8\text{-}58)$$

- Otherwise (zVR is equal to -2 or -3),

$$\text{pred4x4}_L[\,x, y\,] = (\,p[\,-1, y - 1\,] + 2 * p[\,-1, y - 2\,] + p[\,-1, y - 3\,] + 2\,) >> 2 \quad (8\text{-}59)$$

8.3.1.2.7 Specification of Intra_4x4_Horizontal_Down prediction mode

This Intra_4x4 prediction mode shall be used 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,

$$\text{pred4x4}_L[\,x, y\,] = (\,p[\,-1, y -(\,x >> 1\,) - 1\,] + p[\,-1, y - (\,x >> 1\,)\,] + 1\,) >> 1 \qquad (8\text{-}60)$$

- Otherwise, if zHD is equal to 1, 3, or 5,

$$\text{pred4x4}_L[\,x, y\,] = (\,p[\,-1, y - (\,x >> 1\,) - 2\,] + 2 * p[\,-1, y - (\,x >> 1\,) - 1\,] + p[\,-1, y - (\,x >> 1\,)\,] + 2\,) >> 2 \qquad (8\text{-}61)$$

- Otherwise, if zHD is equal to -1,

$$\text{pred4x4}_L[\,x, y\,] = (\,p[\,-1, 0\,] + 2 * p[\,-1, -1\,] + p[\,0, -1\,] + 2\,) >> 2 \qquad (8\text{-}62)$$

- Otherwise (zHD is equal to -2 or -3),

$$\text{pred4x4}_L[\,x, y\,] = (\,p[\,x - 1, -1\,] + 2 * p[\,x - 2, -1\,] + p[\,x - 3, -1\,] + 2\,) >> 2 \qquad (8\text{-}63)$$

8.3.1.2.8 Specification of Intra_4x4_Vertical_Left prediction mode

This Intra_4x4 prediction mode shall be used 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,

$$\text{pred4x4}_L[\,x, y\,] = (\,p[\,x + (\,y >> 1\,), -1\,] + p[\,x + (\,y >> 1\,) + 1, -1\,] + 1) >> 1 \qquad (8\text{-}64)$$

- Otherwise (y is equal to 1 or 3),

$$\text{pred4x4}_L[\,x, y\,] = (\,p[\,x + (\,y >> 1\,), -1\,] + 2 * p[\,x + (\,y >> 1\,) + 1, -1\,] + p[\,x + (\,y >> 1\,) + 2, -1\,] + 2\,) >> 2 \qquad (8\text{-}65)$$

8.3.1.2.9 Specification of Intra_4x4_Horizontal_Up prediction mode

This Intra_4x4 prediction mode shall be used 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

$$\text{pred4x4}_L[\,x, y\,] = (\,p[\,-1, y + (\,x >> 1\,)\,] + p[\,-1, y + (\,x >> 1\,) + 1\,] + 1\,) >> 1 \qquad (8\text{-}66)$$

- Otherwise, if zHU is equal to 1 or 3

$$\text{pred4x4}_L[\,x, y\,] = (\,p[\,-1, y + (\,x >> 1\,)\,] + 2 * p[\,-1, y + (\,x >> 1\,) + 1\,] + p[\,-1, y + (\,x >> 1\,) + 2\,] + 2\,) >> 2 \qquad (8\text{-}67)$$

- Otherwise, if zHU is equal to 5,

$$\text{pred4x4}_L[\,x, y\,] = (\,p[\,-1, 2\,] + 3 * p[\,-1, 3\,] + 2\,) >> 2 \qquad (8\text{-}68)$$

- Otherwise (zHU is greater than 5),

$$\text{pred4x4}_L[\,x, y\,] = p[\,-1, 3\,] \qquad (8\text{-}69)$$

8.3.2 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 are constructed samples prior to the deblocking process from neighbouring luma blocks (if available).

Outputs of this process are Intra prediction luma samples for the current macroblock $\text{pred}_L[\ x, y\ ]$.

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 follows.

– The derivation process for neighbouring locations in subclause 6.4.8 is invoked for luma locations with (x, y) assigned to (xN, yN) as input and mbAddrN and (xW, yW) as output.

– 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 is 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 Inter 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 luma sample at luma location (xW, yW) inside the macroblock mbAddrN is assigned to p[x, y].

Let $\text{pred}_L[\ x, y\ ]$ with x, y = 0..15 denote the prediction samples for the 16x16 luma block samples.

Intra_16x16 prediction modes are specified in Table 8-3.

Table 8-3 – 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 8.3.2.1 to 8.3.2.4 shall be used.

8.3.2.1 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".

$$\text{pred}_L[\ x, y\ ] = p[\ x, -1\ ], \text{ with } x, y = 0..15 \qquad (8\text{-}70)$$

8.3.2.2 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".

$$\text{pred}_L[\ x, y\ ] = p[\ -1, y\ ], \text{ with } x, y = 0..15 \qquad (8\text{-}71)$$

8.3.2.3 Specification of Intra_16x16_DC prediction mode

This Intra_16x16 prediction mode shall be used 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:

$$\text{pred}_L[\,x, y\,] = (\sum_{x'=0}^{15} p[x',-1] + \sum_{y'=0}^{15} p[-1,y'] + 16) >> 5\text{, with } x, y = 0..15 \tag{8-72}$$

- 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:

$$\text{pred}_L[\,x, y\,] = (\sum_{y'=0}^{15} p[-1,y'] + 8) >> 4\text{, with } x, y = 0..15 \tag{8-73}$$

- 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:

$$\text{pred}_L[\,x, y\,] = (\sum_{x'=0}^{15} p[x',-1] + 8) >> 4\text{, with } x, y = 0..15 \tag{8-74}$$

- 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:

$$\text{pred}_L[\,x, y\,] = 128\text{, with } x, y = 0..15 \tag{8-75}$$

8.3.2.4 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”.

$$\text{pred}_L[\,x, y\,] = \text{Clip1}(\,(\,a + b * (\,x - 7\,) + c * (\,y - 7\,) + 16\,) >> 5\,)\text{, with } x, y = 0..15, \tag{8-76}$$

where:

$$a = 16 * (\,p[\,-1, 15\,] + p[\,15, -1\,]\,) \tag{8-77}$$

$$b = (\,5 * H + 32\,) >> 6 \tag{8-78}$$

$$c = (\,5 * V + 32\,) >> 6 \tag{8-79}$$

and H and V are specified in Equations 8-80 and 8-81.

$$H = \sum_{x'=0}^{7} (\,x'+1\,) * (\,p[\,8 + x', -1\,] - p[\,6 - x', -1\,]\,) \tag{8-80}$$

$$V = \sum_{y'=0}^{7} (\,y'+1\,) * (\,p[\,-1, 8 + y'\,] - p[\,-1, 6 - y'\,]\,) \tag{8-81}$$

8.3.3 Intra prediction process for chroma samples

This process is invoked for I and SI macroblock types. It specifies how the Intra prediction chroma samples for the current macroblock are derived.

Inputs to this process are constructed samples prior to the deblocking process from neighbouring chroma blocks (if available).

Outputs of this process are Intra prediction chroma samples for the current macroblock $\text{pred}_{Cb}[\,x, y\,]$ and $\text{pred}_{Cr}[\,x, y\,]$.

Both chroma blocks (Cb and Cr) of the macroblock shall 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 17 neighbouring samples p[x, y] that are constructed chroma samples prior to the deblocking filter process, with x = -1, y = -1..7 and with x = 0..7, y = -1, are derived as follows.

– The derivation process for neighbouring locations in subclause 6.4.8 is invoked for chroma locations with (x, y) assigned to (xN, yN) as input and mbAddrN and (xW, yW) as output.

– Each sample p[x, y] is derived as follows.

  – If any of the following conditions is 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 Inter 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 chroma sample of component C at chroma location (xW, yW) inside the macroblock mbAddrN is assigned to p[x, y].

Let pred_C[x, y] with x, y = 0..7 denote the prediction samples for the chroma block samples.

Intra chroma prediction modes are specified in Table 8-4.

Table 8-4 – 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 8.3.3.1 to 8.3.3.4 shall be used.

8.3.3.1 Specification of Intra_Chroma_DC prediction mode

The values of the prediction samples pred_C[x, y] with x = 0..3 and y = 0..3 are derived as follows.

- If all samples p[x, –1], with x = 0..3 and all samples p[–1, y], with y = 0..3 are marked as “available for Intra chroma prediction”,

$$\text{pred}_C[\,x, y\,] = \left(\sum_{x'=0}^{3} p[x', -1] + \sum_{y'=0}^{3} p[-1, y'] + 4 \right) >> 3, \text{ with } x = 0..3 \text{ and } y = 0..3 \qquad (8\text{-}82)$$

- Otherwise, if all samples p[x, –1], with x = 0..3 are marked as “available for Intra chroma prediction” and any samples p[–1, y], with y = 0..3 are marked as “not available for Intra chroma prediction”,

$$\text{pred}_C[\,x, y\,] = \left(\sum_{x'=0}^{3} p[x', -1] + 2 \right) >> 2, \text{ with } x = 0..3 \text{ and } y = 0..3 \qquad (8\text{-}83)$$

- Otherwise, if any samples p[x, –1], with x = 0..3 are marked as “not available for Intra chroma prediction” and all samples p[-1, y], with y = 0..3 are marked as “available for Intra chroma prediction”,

$$\text{pred}_C[\,x, y\,] = \left(\sum_{y'=0}^{3} p[-1, y'] + 2\right) >> 2,\ \text{with } x = 0..3 \text{ and } y = 0..3 \qquad (8\text{-}84)$$

- 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 chroma prediction"),

$$\text{pred}_C[\,x, y\,] = 128,\ \text{with } x = 0..3 \text{ and } y = 0..3 \qquad (8\text{-}85)$$

The values of the prediction samples pred$_C$[x, y], with x = 4..7 and y = 0..3 are derived as follows.

- If all samples p[x, -1], with x = 4..7 are marked as "available for Intra chroma prediction",

$$\text{pred}_C[\,x, y\,] = \left(\sum_{x'=4}^{7} p[x', -1] + 2\right) >> 2,\ \text{with } x = 4..7 \text{ and } y = 0..3 \qquad (8\text{-}86)$$

- Otherwise, if all samples p[-1, y], with y = 0..3 are marked as "available for Intra chroma prediction",

$$\text{pred}_C[\,x, y\,] = \left(\sum_{y'=0}^{3} p[-1, y'] + 2\right) >> 2,\ \text{with } x = 4..7 \text{ and } y = 0..3 \qquad (8\text{-}87)$$

- Otherwise (some samples p[x, –1], with x = 4..7 and some samples p[-1, y], with y = 0..3 are marked as "not available for Intra chroma prediction"),

$$\text{pred}_C[\,x, y\,] = 128,\ \text{with } x = 4..7 \text{ and } y = 0..3 \qquad (8\text{-}88)$$

The values of the prediction samples pred$_C$[x, y], with x = 0..3 and y = 4..7 are derived as follows.

- If all samples p[-1, y], with y = 4..7 are marked as "available for Intra chroma prediction",

$$\text{pred}_C[\,x, y\,] = \left(\sum_{y'=4}^{7} p[-1, y'] + 2\right) >> 2,\ \text{with } x = 0..3 \text{ and } y = 4..7 \qquad (8\text{-}89)$$

- Otherwise, if all samples p[x, -1], with x = 0..3 are marked as "available for Intra chroma prediction",

$$\text{pred}_C[\,x, y\,] = \left(\sum_{x'=0}^{3} p[x', -1] + 2\right) >> 2,\ \text{with } x = 0..3 \text{ and } y = 4..7 \qquad (8\text{-}90)$$

- Otherwise (some samples p[x, –1], with x = 0..3 and some samples p[-1, y], with y = 4..7 are marked as "not available for Intra chroma prediction"),

$$\text{pred}_C[\,x, y\,] = 128,\ \text{with } x = 0..3 \text{ and } y = 4..7 \qquad (8\text{-}91)$$

The values of the prediction samples pred$_C$[x, y] with x = 4..7 and y = 4..7 are derived as follows.

- If all samples p[x, –1], with x = 4..7 and all samples p[–1, y], with y = 4..7 are marked as "available for Intra chroma prediction",

$$\text{pred}_C[\,x, y\,] = \left(\sum_{x'=4}^{7} p[x', -1] + \sum_{y'=4}^{7} p[-1, y'] + 4\right) >> 3,\ \text{with } x = 4..7 \text{ and } y = 4..7 \qquad (8\text{-}92)$$

- Otherwise, if all samples p[x, –1], with x = 4..7 are marked as "available for Intra chroma prediction" and any samples p[–1, y], with y = 4..7 are marked as "not available for Intra chroma prediction",

$$\text{pred}_C[\,x, y\,] = \left(\sum_{x'=4}^{7} p[x', -1] + 2\right) >> 2,\ \text{with } x = 4..7 \text{ and } y = 4..7 \qquad (8\text{-}93)$$

- Otherwise, if any samples p[x, –1], with x = 4..7 are marked as "not available for Intra chroma prediction" and all samples p[-1, y], with y = 4..7 are marked as "available for Intra chroma prediction",

$$\text{pred}_C[\,x, y\,] = \left(\sum_{y'=4}^{7} p[-1, y'] + 2 \right) >> 2, \text{ with } x = 4..7 \text{ and } y = 4..7 \qquad (8\text{-}94)$$

- Otherwise (some samples p[x, –1], with x = 4..7 and some samples p[-1, y], with y = 4..7 are marked as "not available for Intra chroma prediction"),

$$\text{pred}_C[\,x, y\,] = 128, \text{ with } x = 4..7 \text{ and } y = 4..7 \qquad (8\text{-}95)$$

8.3.3.2 Specification of Intra_Chroma_Horizontal prediction mode

This mode shall be used only when the samples p[-1, y], with y = 0..7 are marked as "available for Intra chroma prediction".

The values of the prediction samples pred_C[x, y] are derived as follows.

$$\text{pred}_C[\,x, y\,] = p[\,-1, y\,], \text{ with } x, y = 0..7 \qquad (8\text{-}96)$$

8.3.3.3 Specification of Intra_Chroma_Vertical prediction mode

This mode shall be used only when the samples p[x, -1], with x = 0..7 are marked as "available for Intra chroma prediction".

The values of the prediction samples pred_C[x, y] are derived as follows.

$$\text{pred}_C[\,x, y\,] = p[\,x, -1\,], \text{ with } x, y = 0..7 \qquad (8\text{-}97)$$

8.3.3.4 Specification of Intra_Chroma_Plane prediction mode

This mode shall be used only when the samples p[x, -1], with x = 0..7 and p[-1, y], with y = -1..7 are marked as "available for Intra chroma prediction".

The values of the prediction samples pred_C[x, y] are derived as follows.

$$\text{pred}_C[\,x, y\,] = \text{Clip1}(\,(\,a + b * (\,x - 3\,) + c * (\,y - 3\,) + 16\,) >> 5\,), \text{ with } x, y = 0..7 \qquad (8\text{-}98)$$

where:

$$a = 16 * (\,p[\,-1, 7\,] + p[\,7, -1\,]\,) \qquad (8\text{-}99)$$

$$b = (\,17 * H + 16\,) >> 5 \qquad (8\text{-}100)$$

$$c = (\,17 * V + 16\,) >> 5 \qquad (8\text{-}101)$$

and H and V are specified as follows.

$$H = \sum_{x'=0}^{3} (x'+1) * (p[4 + x', -1] - p[2 - x', -1]) \qquad (8\text{-}102)$$

$$V = \sum_{y'=0}^{3} (y'+1) * (p[-1, 4 + y'] - p[-1, 2 - y']) \qquad (8\text{-}103)$$

8.3.4 Sample construction process for I_PCM macroblocks

This process is invoked when mb_type is equal to I_PCM.

Outputs of this process are constructed macroblock samples S'_L, S'_{Cb}, and S'_{Cr} prior to the deblocking filter process.

The variable dy is derived as follows.

- If MbaffFrameFlag is equal to 1 and the current macroblock is a field macroblock, dy is set equal to 2.
- Otherwise (MbaffFrameFlag is equal to 0 or the current macroblock is a frame macroblock), dy is set equal to 1.

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 samples prior to the deblocking process are generated as specified by:

```
for( i = 0; i < 256; i++ )
    S'_L[ xP + ( i % 16 ), yP + dy * ( i / 16 ) ) ] = pcm_byte[ i ]                                  (8-104)

for( i = 0; i < 64; i++ ) {
    S'_Cb[ ( xP >> 1 ) + ( i % 8 ), ( ( yP + 1 ) >> 1 ) + dy * ( i / 8 ) ] = pcm_byte[ i + 256 ]       (8-105)
    S'_Cr[ ( xP >> 1 ) + ( i % 8 ), ( ( yP + 1 ) >> 1 ) + dy * ( i / 8 ) ] = pcm_byte[ i + 320 ]
}
```

8.4 Inter prediction process

This process is invoked when decoding P and B 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 8x8 arrays pred_{Cr} and pred_{Cb} 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. 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 Table 7-10, Table 7-11, Table 7-14, and Table 7-15.

The variables partWidth and partHeight are derived as follows.

- If mb_type is not equal to P_8x8 or P_8x8ref0 or B_8x8, the following applies.

  partWidth = MbPartWidth(mb_type) (8-106)

  partHeight = MbPartHeight(mb_type) (8-107)

- Otherwise (mb_type is equal to P_8x8 or P_8x8ref0 or B_8x8),

  partWidth = SubMbPartWidth(sub_mb_type[mbPartIdx]) (8-108)

  partHeight = SubMbPartHeight(sub_mb_type[mbPartIdx]). (8-109)

When mb_type is equal to B_Skip or B_Direct_16x16 or sub_mb_type[mbPartIdx] is equal to B_Direct_8x8, the Inter prediction process is specified for

  partWidth = 4 (8-110)

  partHeight = 4 (8-111)

with mbPartIdx proceeding over values 0..3. For each sub-macroblock indexed by mbPartIdx, subMbPartIdx proceeds over values 0..3.

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. Derivation process for motion vector components and reference indices as specified in subclause 8.4.1.

   Inputs to this process are

   - a macroblock partition mbPartIdx,
   - a sub-macroblock partition subMbPartIdx.

   Outputs of this process are

   - luma motion vectors mvL0 and mvL1 and the chroma motion vectors mvCL0 and mvCL1
   - reference indices refIdxL0 and refIdxL1
   - prediction list utilization flags predFlagL0 and predFlagL1
   - the sub-macroblock partition motion vector count subMvCnt.

2. The variable MvCnt is incremented by subMvCnt.

3. Decoding process for Inter prediction samples as specified in subclause 8.4.2.

   Inputs to this process are

   - a macroblock partition mbPartIdx,
   - a sub-macroblock partition subMbPartIdx.
   - variables specifying partition width and height, partWidth, and partHeight
   - luma motion vectors mvL0 and mvL1 and the chroma motion vectors mvCL0 and mvCL1
   - reference indices refIdxL0 and refIdxL1
   - prediction list utilization flags predFlagL0 and predFlagL1

   Outputs of this process are

   - inter prediction samples (pred); which are a (partWidth)x(partHeight) array predPart_L of prediction luma samples and two (partWidth/2)x(partHeight/2) 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:

$$\text{MvL0[mbPartIdx][subMbPartIdx]} = \text{mvL0} \quad (8\text{-}112)$$

$$\text{MvL1[mbPartIdx][subMbPartIdx]} = \text{mvL1} \quad (8\text{-}113)$$

$$\text{RefIdxL0[mbPartIdx]} = \text{refIdxL0} \quad (8\text{-}114)$$

$$\text{RefIdxL1[mbPartIdx]} = \text{refIdxL1} \quad (8\text{-}115)$$

$$\text{PredFlagL0[mbPartIdx]} = \text{predFlagL0} \quad (8\text{-}116)$$

$$\text{PredFlagL1[mbPartIdx]} = \text{predFlagL1} \quad (8\text{-}117)$$

The location of the upper-left sample of the 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 macroblock sub-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 partition or sub-macroblock partition prediction samples in their correct relative positions in the macroblock, as follows.

The variable $\text{pred}_L[\text{ xP + xS + x, yP + yS + y }]$ with x = 0 .. partWidth – 1, y = 0 .. partHeight – 1 is derived by

$$\text{pred}_L[\text{ xP + xS + x, yP + yS + y }] = \text{predPart}_L[\text{ x, y }] \quad (8\text{-}118)$$

The variable $\text{pred}_C[\text{ xP / 2 + xS / 2 + x, yP / 2 + yS / 2 + y }]$ with x = 0 .. partWidth/2 – 1, y = 0 .. partHeight/2 – 1, and C being replaced by Cb or Cr is derived by

$$\text{pred}_C[\text{ xP / 2 + xS / 2 + x, yP / 2 + yS / 2 + y }] = \text{predPart}_C[\text{ x, y }] \quad (8\text{-}119)$$

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 vectors mvL0 and mvL1 as well as the chroma motion vectors mvCL0 and mvCL1
- reference indices refIdxL0 and refIdxL1

- prediction list utilization flags predFlagL0 and predFlagL1
- a sub-partition macroblock motion vector count variable subMvCnt

For the derivation of the variables mvL0 and mvL1 as well as refIdxL0 and refIdxL1, the following applies.

- If mb_type is equal to P_Skip, the derivation process for luma motion vectors for skipped macroblocks in P and SP slices in subclause 8.4.1.1 is invoked with the output being the luma motion vectors mvL0 and reference indices refIdxL0, and predFlagL0 is set equal to 1. mvL1 and refIdxL1 are marked as not available and predFlagL1 is set equal to 0. The sub-partition motion vector count variable subMvCnt is set equal to 1.
- Otherwise, if mb_type is equal to B_Skip or B_Direct_16x16 or sub_mb_type[mbPartIdx] is equal to B_Direct_8x8, the derivation process for luma motion vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8 in B slices in subclause 8.4.1.2 is invoked with mbPartIdx and subMbPartIdx as the input and the output being the luma motion vectors mvL0, mvL1, the reference indices refIdxL0, refIdxL1, the sub-partition motion vector count subMvCnt, and the prediction utilization flags predFlagL0 and predFlagL1.
- Otherwise, for X being replaced by either 0 or 1 in the variables predFlagLX, mvLX, refIdxLX, and in Pred_LX and in the syntax elements ref_idx_lX and mvd_lX, and the following applies.

- The variables refIdxLX and predFlagLX are derived as follows.
    - If MbPartPredMode(mb_type, mbPartIdx) or SubMbPredMode(sub_mb_type[mbPartIdx]) is equal to Pred_LX or to BiPred,

refIdxLX = ref_idx_lX[mbPartIdx] (8-120)

predFlagLX = 1 (8-121)

    - Otherwise, the variables refIdxLX and predFlagLX are specified by

refIdxLX = -1 (8-122)

predFlagLX = 0 (8-123)

- The variable subMvCnt for sub-partition motion vector count is set equal to predFlagL0 + predFlagL1.
- When predFlagLX is equal to 1, the derivation process for luma motion vector prediction in subclause 8.4.1.3 is invoked with mbPartIdx subMbPartIdx, refIdxLX, and list suffix LX as the input and the output being mvpLX. The luma motion vectors are derived by

mvLX[0] = mvpLX[0] + mvd_lX[mbPartIdx][subMbPartIdx][0] (8-124)

mvLX[1] = mvpLX[1] + mvd_lX[mbPartIdx][subMbPartIdx][1] (8-125)

For the derivation of the variables for the chroma motion vectors, the following applies. When predFlagLX (with X being either 0 or 1) is equal to 1, the derivation process for chroma motion vectors in subclause 8.4.1.4 is invoked with mvLX and refIdxLX as input and the output being mvCLX.

8.4.1.1 Derivation process for luma motion vectors for skipped macroblocks in P and SP 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 follows.

refIdxL0 = 0. (8-126)

For the derivation of the motion vector mvL0 of a P_Skip macroblock type, the following applies.

- Let currSubMbType be set equal to sub_mb_type[0]. The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx set equal to 0, subMbPartIdx set equal to 0, currSubMbType, and list suffix L0 as input and the output is assigned to mbAddrA, mbAddrB, mvL0A, mvL0B, refIdxL0A, and refIdxL0B.
- The variable mvL0 is specified as follows.

- If any one of the following conditions is 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 list suffix L0 as input 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 Derivation process for luma motion vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8

This process is invoked when mb_type is equal to B_Skip or B_Direct_16x16, or sub_mb_type[mbPartIdx] is equal to B_Direct_8x8.

Inputs to this process are mbPartIdx and subMbPartIdx.

Outputs of this process are the reference indices refIdxL0, refIdxL1, the motion vectors mvL0 and mvL1, the sub-partition motion vector count subMvCnt, and the prediction list utilization flags, predFlagL0 and predFlagL1.

The derivation process depends on the value of direct_spatial_mv_pred_flag, which is present in the bitstream in the slice header syntax as specified in subclause 7.3.3, and is specified as follows.

- If direct_spatial_mv_pred_flag is equal to 1, the mode in which the outputs of this process are derived is referred to as spatial direct prediction mode.
- Otherwise (direct_spatial_mv_pred_flag is equal to 0), mode in which the outputs of this process are derived is referred to as temporal direct prediction mode.

Both spatial and temporal direct prediction mode use the co-located motion vectors and reference indices as specified in subclause 8.4.1.2.1.

The motion vectors and reference indices are derived as follows.

- If spatial direct prediction mode is used, the direct motion vector and reference index prediction mode specified in subclause 8.4.1.2.2 is used, with subMvCnt being an output.
- Otherwise (temporal direct prediction mode is used), the direct motion vector and reference index prediction mode specified in subclause 8.4.1.2.3 is used and the variable subMvCnt is derived as follows.
  - If subMbPartIdx is equal to 0, subMvCnt is set equal to 2.
  - Otherwise (subMbPartIdx is not equal to 0), subMvCnt is set equal to 0.

8.4.1.2.1 Derivation process for the co-located 4x4 sub-macroblock partitions

Inputs to this process are mbPartIdx and subMbPartIdx.

Outputs of this process are the picture colPic, the co-located macroblock mbAddrCol, the motion vector mvCol, the reference index refIdxCol, and the variable vertMvScale (which can be One_To_One, Frm_To_Fld or Fld_To_Frm).

Let firstRefPicL1 be the reference picture referred by RefPicList1[0].

When firstRefPicL1 is a frame or a complementary field pair, let firstRefPicL1Top and firstRefPicL1Bottom be the top and bottom fields of firstRefPicL1, and let the following variables be specified as

$$\text{topAbsDiffPOC} = \text{Abs}(\ \text{DiffPicOrderCnt}(\ \text{firstRefPicL1Top},\ \text{CurrPic}\ )\ ) \qquad (8\text{-}127)$$

$$\text{bottomAbsDiffPOC} = \text{Abs}(\ \text{DiffPicOrderCnt}(\ \text{firstRefPicL1Bottom},\ \text{CurrPic}\ )\ ) \qquad (8\text{-}128)$$

The variable colPic specifies the picture that contains the co-located macroblock as specified in Table 8-5.

Table 8-5 – Specification of the variable colPic

field_pic_flag	The first entry in RefPicList1 is …	mb_field_decoding_flag	additional condition	colPic
1	a field of a decoded frame			the frame containing firstRefPicL1
	a decoded field			firstRefPicL1
0	a decoded frame			firstRefPicL1
	a complementary field pair	0	topAbsDiffPOC < bottomAbsDiffPOC	the top field of firstRefPicL1
			topAbsDiffPOC >= bottomAbsDiffPOC	the bottom field of firstRefPicL1
		1	(CurrMbAddr & 1) = = 0	the top field of firstRefPicL1
			(CurrMbAddr & 1) != 0	the bottom field of firstRefPicL1

When direct_8x8_inference_flag is equal to 1, subMbPartIdx is set as follows.

subMbPartIdx = mbPartIdx (8-129)

Let PicCodingStruct(X) be a function with the argument X being either CurrPic or colPic. It is specified in Table 8-6.

Table 8-6 – Specification of PicCodingStruct(X)

X is coded with field_pic_flag equal to …	mb_adaptive_frame_field_flag	PicCodingStruct(X)
1		FLD
0	0	FRM
0	1	AFRM

With luma4x4BlkIdx = mbPartIdx * 4 + subMbPartIdx, the inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with luma4x4BlkIdx as the input and (x, y) assigned to (xCol, yCol) as the output.

Table 8-7 specifies the co-located macroblock address mbAddrCol, yM, and the variable vertMvScale in two steps:

1. Specification of a macroblock address mbAddrX depending on PicCodingStruct(CurrPic), and PicCodingStruct(colPic).

   NOTE - It is not possible for CurrPic and colPic picture coding types to be either (FRM, AFRM) or (AFRM, FRM) because these picture coding types must be separated by an IDR picture.

2. Specification of mbAddrCol, yM, and vertMvScale depending on mb_field_decoding_flag and the variable fieldDecodingFlagX, which is derived as follows.

   - If the macroblock mbAddrX in the picture colPic is a field macroblock, fieldDecodingFlagX is set equal to 1
   - Otherwise (the macroblock mbAddrX in the picture colPic is a frame macroblock), fieldDecodingFlagX is set equal to 0.

Unspecified values in Table 8-7 indicate that the value of the corresponding variable is not relevant for the current table row.

mbAddrCol is set equal to CurrMbAddr or to one of the following values.

$$\text{mbAddrCol1} = 2 * \text{PicWidthInMbs} * (\text{CurrMbAddr} / \text{PicWidthInMbs}) + (\text{CurrMbAddr} \% \text{PicWidthInMbs}) + \text{PicWidthInMbs} * (\text{yCol} / 8) \qquad (8\text{-}130)$$

$$\text{mbAddrCol2} = 2 * \text{CurrMbAddr} + (\text{yCol} / 8) \qquad (8\text{-}131)$$

$$\text{mbAddrCol3} = 2 * \text{CurrMbAddr} + \text{bottom\_field\_flag} \qquad (8\text{-}132)$$

$$\text{mbAddrCol4} = \text{PicWidthInMbs} * (\text{CurrMbAddr} / (2 * \text{PicWidthInMbs})) + (\text{CurrMbAddr} \% \text{PicWidthInMbs}) \qquad (8\text{-}133)$$

$$\text{mbAddrCol5} = \text{CurrMbAddr} / 2 \qquad (8\text{-}134)$$

$$\text{mbAddrCol6} = 2 * (\text{CurrMbAddr} / 2) + ((\text{topAbsDiffPOC} < \text{bottomAbsDiffPOC}) \;?\; 0 : 1) \qquad (8\text{-}135)$$

$$\text{mbAddrCol7} = 2 * (\text{CurrMbAddr} / 2) + (\text{yCol} / 8) \qquad (8\text{-}136)$$

Table 8-7 – Specification of mbAddrCol, yM, and vertMvScale

PicCodingStruct(CurrPic)	PicCodingStruct(colPic)	mbAddrX	mb_field_decoding_flag	fieldDecodingFlagX	mbAddrCol	yM	vertMvScale
FLD	FLD				CurrMbAddr	yCol	One_To_One
	FRM				mbAddrCol1	(2 * yCol) % 16	Frm_To_Fld
	AFRM	2*CurrMbAddr		0	mbAddrCol2	(2 * yCol) % 16	Frm_To_Fld
				1	mbAddrCol3	yCol	One_To_One
FRM	FLD				mbAddrCol4	8 * ((CurrMbAddr / PicWidthInMbs) % 2) + 4 * (yCol / 8)	Fld_To_Frm
	FRM				CurrMbAddr	yCol	One_To_One
AFRM	FLD		0		mbAddrCol5	8 * (CurrMbAddr % 2) +4 * (yCol / 8)	Fld_To_Frm
			1		mbAddrCol5	yCol	One_To_One
	AFRM	CurrMbAddr	0	0	CurrMbAddr	yCol	One_To_One
				1	mbAddrCol6	8 * (CurrMbAddr % 2) + 4 * (yCol / 8)	Fld_To_Frm
		CurrMbAddr	1	0	mbAddrCol7	(2 * yCol) % 16	Frm_To_Fld
				1	CurrMbAddr	yCol	One_To_One

Let mbPartIdxCol be the macroblock partition index of the co-located partition and subMbPartIdxCol the sub-macroblock partition index of the co-located sub-macroblock partition. The partition in the macroblock mbAddrCol inside the picture colPic covering the sample (xCol, yM) shall be assigned to mbPartIdxCol and the sub-macroblock partition inside the partition mbPartIdxCol covering the sample (xCol, yM) in the macroblock mbAddrCol inside the picture colPic shall be assigned to subMbPartIdxCol.

The prediction utilization flags predFlagL0Col and predFlagL1Col are set equal to PredFlagL0[mbPartIdxCol] and PredFlagL1[mbPartIdxCol], respectively, which are the prediction utilization flags that have been assigned to the macroblock partition mbAddrCol\mbPartIdxCol inside the picture colPic.

The motion vector mvCol and the reference index refIdxCol are derived as follows.

- If the macroblock mbAddrCol is coded in Intra macroblock prediction mode or both prediction utilization flags, predFlagL0Col and predFlagL1Col are equal to 0, both components of mvCol are set equal to 0 and refIdxCol is set equal to –1.
- Otherwise, the following applies.
  - If predFlagL0Col is equal to 1, the motion vector mvCol and the reference index refIdxCol are set equal to MvL0[mbPartIdxCol][subMbPartIdxCol] and RefIdxL0[mbPartIdxCol], respectively, which are the motion vector mvL0 and the reference index refIdxL0 that have been assigned to the (sub-)macroblock partition mbAddrCol\mbPartIdxCol\subMbPartIdxCol inside the picture colPic.
  - Otherwise (predFlagL0Col is equal to 0 and predFlagL1Col is equal to 1), the motion vector mvCol and the reference index refIdxCol are set equal to MvL1[mbPartIdxCol][subMbPartIdxCol] and RefIdxL1[mbPartIdxCol], respectively, which are the motion vector mvL1 and the reference index refIdxL1 that have been assigned to the (sub-)macroblock partition mbAddrCol\mbPartIdxCol\subMbPartIdxCol inside the picture colPic.

8.4.1.2.2 Derivation process for spatial direct luma motion vector and reference index prediction mode

This process is invoked when direct_spatial_mv_pred_flag is equal to 1 and any of the following conditions is true.

- mb_type is equal to B_Skip
- mb_type is equal to B_Direct_16x16
- sub_mb_type[mbPartIdx] is equal to B_Direct_8x8.

Inputs to this process are mbPartIdx, subMbPartIdx.

Outputs of this process are the reference indices refIdxL0, refIdxL1, the motion vectors mvL0 and mvL1, the sub-partition motion vector count subMvCnt, and the prediction list utilization flags, predFlagL0 and predFlagL1.

The reference indices refIdxL0 and refIdxL1 and the variable directZeroPredictionFlag are derived by applying the following ordered steps.

1. Let the variable currSubMbType be set equal to sub_mb_type[mbPartIdx].
2. The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx = 0, subMbPartIdx = 0, currSubMbType, and list suffix L0 as input and the output is assigned to the motion vectors mvL0N and the reference indices refIdxL0N with N being replaced by A, B, or C.
3. The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx = 0, subMbPartIdx = 0, currSubMbType, and list suffix L1 as input and the output is assigned to the motion vectors mvL1N and the reference indices refIdxL1N with N being replaced by A, B, or C.

   NOTE – The motion vectors mvL0N, mvL1N and the reference indices refIdxL0N, refIdxL1N are identical for all 4x4 sub-macroblock partitions of a macroblock.
4. The reference indices refIdxL0, refIdxL1, and directZeroPredictionFlag are derived by

$$\text{refIdxL0} = \text{MinPositive}(\ \text{refIdxL0A},\ \text{MinPositive}(\ \text{refIdxL0B},\ \text{refIdxL0C}\ )\ ) \quad (8\text{-}137)$$
$$\text{refIdxL1} = \text{MinPositive}(\ \text{refIdxL1A},\ \text{MinPositive}(\ \text{refIdxL1B},\ \text{refIdxL1C}\ )\ ) \quad (8\text{-}138)$$
$$\text{directZeroPredictionFlag} = 0 \quad (8\text{-}139)$$

   where

$$\text{MinPositive}(\ x, y\ ) = \begin{cases} \text{Min}(x, y) & \text{if } x >= 0 \text{ and } y >= 0 \\ \text{Max}(x, y) & \text{otherwise} \end{cases} \quad (8\text{-}140)$$

5. When both reference indices refIdxL0 and refIdxL1 are less than 0,

$$\text{refIdxL0} = 0 \quad (8\text{-}141)$$
$$\text{refIdxL1} = 0 \quad (8\text{-}142)$$
$$\text{directZeroPredictionFlag} = 1 \quad (8\text{-}143)$$

The process specified in subclause 8.4.1.2.1 is invoked with mbPartIdx, subMbPartIdx given as input and the output is assigned to refIdxCol and mvCol.

The variable colZeroFlag is derived as follows.

- If all of the following conditions are true, colZeroFlag is set equal to 1.
  - the reference picture referred by RefPicList1[0] is a short-term reference picture
  - refIdxCol is equal to 0
  - both motion vector components mvCol[0] and mvCol[1] lie in the range of -1 to 1 in units specified as follows.
    - If the colocated macroblock is a frame macroblock, the units of mvCol[0] and mvCol[1] are units of quarter luma frame samples.
    - Otherwise (the colocated macroblock is a field macroblock), the units of mvCol[0] and mvCol[1] are units of quarter luma field samples.

  NOTE – For purposes of determining the condition above, the value mvCol[1] is not scaled to use the units of a motion vector for the current macroblock in cases when the current macroblock is a frame macroblock and the colocated macroblock is a field macroblock or when the current macroblock is a field macroblock and the colocated macroblock is a frame macroblock. This aspect differs from the use of mvCol[1] in the temporal direct mode as specified in subclause 8.4.1.2.3, which applies scaling to the motion vector of the colocated macroblock to use the same units as the units of a motion vector for the current macroblock, using Equation 8-146 or Equation 8-147 in these cases.

- Otherwise, colZeroFlag is set equal to 0.

The motion vectors mvLX (with X being 0 or 1) are derived as follows.

- If any of the following conditions is true, both components of the motion vector mvLX are set equal to 0.
  - directZeroPredictionFlag is equal to 1
  - refIdxLX is less than 0
  - refIdxLX is equal to 0 and colZeroFlag is equal to 1
- Otherwise, the process specified in subclause 8.4.1.3 is invoked with mbPartIdx = 0, subMbPartIdx = 0, refIdxLX, and list suffix LX as the input and the output is assigned to mvLX.

  NOTE – In the immediately above case, the returned motion vector mvLX is identical for all 4x4 sub-macroblock partitions of a macroblock.

The prediction utilization flags predFlagL0 and predFlagL1 shall be derived as specified using Table 8-8.

Table 8-8 – Assignment of prediction utilization flags

refIdxL0	**refIdxL1**	**predFlagL0**	**predFlagL1**
>= 0	>= 0	1	1
>= 0	< 0	1	0
< 0	>= 0	0	1

The variable subMvCnt is derived as follows.

- If subMbPartIdx is not equal to 0 or direct_8x8_inference_flag is equal to 0, subMvCnt is set equal to 0.
- Otherwise (subMbPartIdx is equal to 0 and direct_8x8_inference_flag is equal to 1), subMvCnt is set equal to predFlagL0 + predFLagL1.

8.4.1.2.3 Derivation process for temporal direct luma motion vector and reference index prediction mode

This process is invoked when direct_spatial_mv_pred_flag is equal to 0 and any of the following conditions is true.

- mb_type is equal to B_Skip
- mb_type is equal to B_Direct_16x16
- sub_mb_type[mbPartIdx] is equal to B_Direct_8x8.

Inputs to this process are mbPartIdx and subMbPartIdx.

Outputs of this process are the motion vectors mvL0 and mvL1, the reference indices refIdxL0 and refIdxL1, and the prediction list utilization flags, predFlagL0 and predFlagL1.

The process specified in subclause 8.4.1.2.1 is invoked with mbPartIdx, subMbPartIdx given as input and the output is assigned to colPic, mbAddrCol, mvCol, refIdxCol, and vertMvScale.

The reference indices refIdxL0 and refIdxL1 are derived as follows.

$$\text{refIdxL0} = ((\text{refIdxCol} < 0) \ ? \ 0 : \text{MapColToList0}(\text{refIdxCol})) \qquad (8\text{-}144)$$

$$\text{refIdxL1} = 0 \qquad (8\text{-}145)$$

NOTE - If the current macroblock is a field macroblock, refIdxL0 and refIdxL1 index a list of fields; otherwise (the current macroblock is a frame macroblock), refIdxL0 and refIdxL1 index a list of frames or complementary reference field pairs.

Let refPicCol be a frame, a field, or a complementary field pair that was referred by the reference index refIdxCol when decoding the co-located macroblock mbAddrCol inside the picture colPic. The function MapColToList0(refIdxCol) is specified as follows.

- If vertMvScale is equal to One_To_One, the following applies.
  - If field_pic_flag is equal to 0 and the current macroblock is a field macroblock, the following applies.
    - Let refIdxL0Frm be the lowest valued reference index in the current reference picture list RefPicList0 that references the frame or complementary field pair that contains the field refPicCol. RefPicList0 shall contain a variable PicNum or LongTermPicNum that references the frame or complementary field pair that contains refPicCol. The return value of MapColToList0( ) is specified as follows.
      - If the field referred to by refIdxCol has the same parity as the current macroblock, MapColToList0(refIdxCol) returns the reference index (refIdxL0Frm << 1).
      - Otherwise (the field referred by refIdxCol has the opposite parity of the current macroblock), MapColToList0(refIdxCol) returns the reference index ((refIdxL0Frm << 1) + 1).
  - Otherwise (field_pic_flag is equal to 1 or the current macroblock is a frame macroblock), MapColToList0(refIdxCol) returns the lowest valued reference index refIdxL0 in the current reference picture list RefPicList0 that references refPicCol. RefPicList0 shall contain a variable PicNum or LongTermPicNum that references refPicCol.
- Otherwise, if vertMvScale is equal to Frm_To_Fld, the following applies.
  - If field_pic_flag is equal to 0, let refIdxL0Frm be the lowest valued reference index in the current reference index list RefPicList0 that references refPicCol. MapColToList0(refIdxCol) returns the reference index (refIdxL0Frm << 1). RefPicList0 shall contain a variable PicNum or LongTermPicNum that references refPicCol.
  - Otherwise (field_pic_flag is equal to 1), MapColToList0(refIdxCol) returns the lowest valued reference index refIdxL0 in the current reference picture list RefPicList0 that references the field of refPicCol with the same parity as the current picture CurrPic. RefPicList0 shall contain a variable PicNum or LongTermPicNum that references the field of refPicCol with the same parity as the current picture CurrPic.
- Otherwise (vertMvScale is equal to Fld_To_Frm), MapColToList0(refIdxCol) returns the lowest valued reference index refIdxL0 in the current reference picture list RefPicList0 that references the frame or complementary field pair that contains refPicCol. RefPicList0 shall contain a variable PicNum or LongTermPicNum that references the frame or complementary field pair that contains refPicCol.

  NOTE – A decoded reference picture that was marked as "used for short-term reference" when it was referenced in the decoding process of the picture containing the co-located macroblock may have been modified to be marked as "used for long-term reference" before being used for reference for inter prediction using the direct prediction mode for the current macroblock.

Depending on the value of vertMvScale the vertical component of mvCol is modified as follows.

- If vertMvScale is equal to Frm_To_Fld

$$\text{mvCol}[1] = \text{mvCol}[1] / 2 \qquad (8\text{-}146)$$

- Otherwise, if vertMvScale is equal to Fld_To_Frm

$$\text{mvCol}[1] = \text{mvCol}[1] * 2 \qquad (8\text{-}147)$$

- Otherwise (vertMvScale is equal to One_To_One), mvCol[1] remains unchanged.

The variables currPicOrField, pic0, and pic1, are derived as follows.

– If field_pic_flag is equal to 0 and the current macroblock is a field macroblock, the following applies.
  – currPicOrField is the field of the current picture CurrPic that has the same parity as the current macroblock.
  – pic1 is the field of RefPicList1[0] that has the same parity as the current macroblock.
  – Let frame0 be the frame or complementary field pair that is referred to by RefPicList0[refIdxL0 /2].
  – The variable pic0 is derived as follows.
    – If refIdxL0 % 2 is equal to 0, pic0 is the field of frame0 that has the same parity as the current macroblock.
    – Otherwise (refIdxL0 % 2 is not equal to 0), pic0 is the field of frame0 that has the opposite parity of the current macroblock.
– Otherwise (field_pic_flag is equal to 1 or the current macroblock is a frame macroblock), currPicOrField is the current picture CurrPic, pic1 is the decoded reference picture referred to by RefPicList1[0], and pic0 is the decoded reference picture referred to by RefPicList0[refIdxL0].

The two motion vectors mvL0 and mvL1 for each 4x4 sub-macroblock partition of the current macroblock are derived as follows:

NOTE – It is often the case that many of the 4x4 sub-macroblock partitions share the same motion vectors and reference pictures. In these cases, temporal direct mode motion compensation can calculate the inter prediction sample values in larger units than 4x4 luma sample blocks. For example, when direct_8x8_inference_flag is equal to 1, at least each 8x8 luma sample quadrant of the macroblock shares the same motion vectors and reference pictures.

– If the reference index refIdxL0 refers to a long-term picture, or DiffPicOrderCnt(pic1, pic0) is equal to 0, the motion vectors mvL0, mvL1 for the direct mode partition are derived by

$$mvL0 = mvCol \tag{8-148}$$

$$mvL1 = 0 \tag{8-149}$$

– Otherwise, the motion vectors mvL0, mvL1 are derived as scaled versions of the motion vector mvCol of the co-located sub-macroblock partition as specified below (see Figure 8-2)

$$tx = (16\,384 + Abs(td / 2)) / td \tag{8-150}$$

$$DistScaleFactor = Clip3(-1024, 1023, (tb * tx + 32) >> 6) \tag{8-151}$$

$$mvL0 = (DistScaleFactor * mvCol + 128) >> 8 \tag{8-152}$$

$$mvL1 = mvL0 – mvCol \tag{8-153}$$

where tb and td are derived as follows.

$$tb = Clip3(-128, 127, DiffPicOrderCnt(currPicOrField, pic0)) \tag{8-154}$$

$$td = Clip3(-128, 127, DiffPicOrderCnt(pic1, pic0)) \tag{8-155}$$

NOTE - mvL0 and mvL1 cannot exceed the ranges specified in Annex A.

The prediction utilization flags predFlagL0 and predFlagL1 are both set equal to 1.

Figure 8-2 illustrates the temporal direct-mode motion vector inference when the current picture is temporally between the list 0 reference picture and the list 1 reference picture.

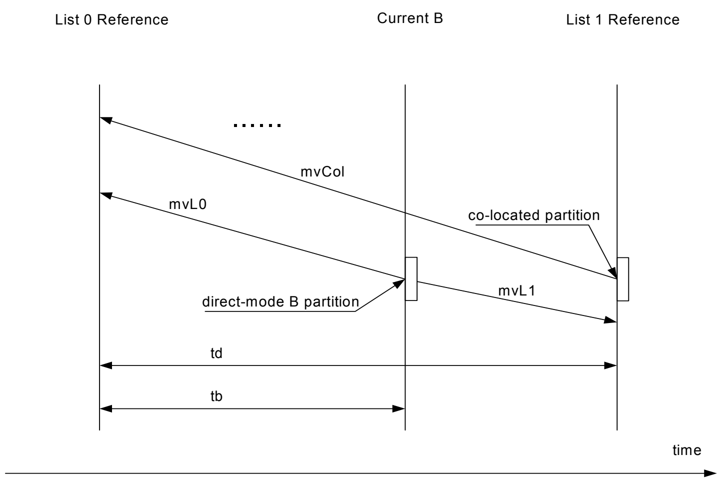


Figure 8-2 –Example for temporal direct-mode motion vector inference (informative)

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,
- list suffix LX,
- the reference index of the current partition refIdxLX.

Output of this process is the prediction mvpLX of the motion vector mvLX.

Let currSubMbType be set equal to sub_mb_type[mbPartIdx].

The derivation process for the neighbouring blocks for motion data in subclause 8.4.1.3.2 is invoked with mbPartIdx, subMbPartIdx, currSubMbType, and list suffix LX as the input and with mbAddrN\mbPartIdxN\subMbPartIdxN, reference indices refIdxLXN and the motion vectors mvLXN with N being replaced by A, B, or C as the output.

The derivation process for median luma motion vector prediction in subclause 8.4.1.3.1 is invoked with mbAddrN\mbPartIdxN\subMbPartIdxN, mvLXN, refIdxLXN with N being replaced by A, B, or C and refIdxLX as the input and mvpLX as the output, unless one of the following is true.

- MbPartWidth(mb_type) is equal to 16, MbPartHeight(mb_type) is equal to 8, mbPartIdx is equal to 0, and refIdxLXB is equal to refIdxLX,

$$\text{mvpLX} = \text{mvLXB} \qquad (8\text{-}156)$$

- MbPartWidth(mb_type) is equal to 16, MbPartHeight(mb_type) is equal to 8, mbPartIdx is equal to 1, and refIdxLXA is equal to refIdxLX,

$$\text{mvpLX} = \text{mvLXA} \qquad (8\text{-}157)$$

- MbPartWidth(mb_type) is equal to 8, MbPartHeight(mb_type) is equal to 16, mbPartIdx is equal to 0, and refIdxLXA is equal to refIdxLX,

$$\text{mvpLX} = \text{mvLXA} \qquad (8\text{-}158)$$

- MbPartWidth(mb_type) is equal to 8, MbPartHeight(mb_type) is equal to 16, mbPartIdx is equal to 1, and refIdxLXC is equal to refIdxLX,

$$\text{mvpLX} = \text{mvLXC} \qquad (8\text{-}159)$$

Figure 8-3 illustrates the non-median prediction as described above.

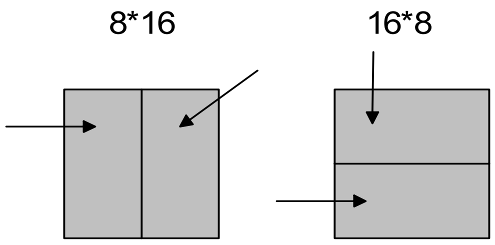


Figure 8-3 – Directional segmentation prediction (informative)

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 mvLXN (with N being replaced by A, B, or C) of the neighbouring partitions,
- the reference indices refIdxLXN (with N being replaced by A, B, or C) of the neighbouring partitions, and
- the reference index refIdxLX of the current partition.

Output of this process is the motion vector prediction mvpLX.

The variable mvpLX is derived as follows:

- When both partitions mbAddrB\mbPartIdxB\subMbPartIdxB and mbAddrC\mbPartIdxC\subMbPartIdxC are not available and mbAddrA\mbPartIdxA\subMbPartIdxA is available,

  mvLXB = mvLXA (8-160)

  mvLXC = mvLXA (8-161)

  refIdxLXB = refIdxLXA (8-162)

  refIdxLXC = refIdxLXA (8-163)

- Depending on reference indices refIdxLXA, refIdxLXB, or refIdxLXC, the following applies.
  - If one and only one of the reference indices refIdxLXA, refIdxLXB, or refIdxLXC is equal to the reference index refIdxLX of the current partition, the following applies. Let refIdxLXN be the reference index that is equal to refIdxLX, the motion vector mvLXN is assigned to the motion vector prediction mvpLX:

    mvpLX = mvLXN (8-164)

  - Otherwise, each component of the motion vector prediction mvpLX is given by the median of the corresponding vector components of the motion vector mvLXA, mvLXB, and mvLXC:

    mvpLX[0] = Median(mvLXA[0], mvLXB[0], mvLXC[0]) (8-165)

    mvpLX[1] = Median(mvLXA[1], mvLXB[1], mvLXC[1]) (8-166)

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 LX

Outputs of this process are (with N being replaced by A, B, or C)

- mbAddrN\mbPartIdxN\subMbPartIdxN specifying neighbouring partitions,
- the motion vectors mvLXN of the neighbouring partitions, and
- the reference indices refIdxLXN of the neighbouring partitions.

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 6.4.7.5 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

$$\text{mbAddrC} = \text{mbAddrD} \tag{8-167}$$

$$\text{mbPartIdxC} = \text{mbPartIdxD} \tag{8-168}$$

$$\text{subMbPartIdxC} = \text{subMbPartIdxD} \tag{8-169}$$

The motion vectors mvLXN and reference indices refIdxLXN (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 Intra prediction mode or predFlagLX of mbAddrN\mbPartIdxN\subMbPartIdxN is equal to 0, both components of mvLXN are set equal to 0 and refIdxLXN is set equal to –1.
- Otherwise, the following applies.
  - The motion vector mvLXN and reference index refIdxLXN are set equal to MvLX[mbPartIdxN][subMbPartIdxN] and RefIdxLX[mbPartIdxN], respectively, which are the motion vector mvLX and reference index refIdxLX that have been assigned to the (sub-)macroblock partition mbAddrN\mbPartIdxN\subMbPartIdxN.
  - The variables mvLXN[1] and refIdxLXN are further processed as follows.
    - If the current macroblock is a field macroblock and the macroblock mbAddrN is a frame macroblock

$$\text{mvLXN}[\ 1\ ] = \text{mvLXN}[\ 1\ ]\ /\ 2 \tag{8-170}$$

$$\text{refIdxLXN} = \text{refIdxLXN} * 2 \tag{8-171}$$

    - Otherwise, if the current macroblock is a frame macroblock and the macroblock mbAddrN is a field macroblock

$$\text{mvLXN}[\ 1\ ] = \text{mvLXN}[\ 1\ ] * 2 \tag{8-172}$$

$$\text{refIdxLXN} = \text{refIdxLXN}\ /\ 2 \tag{8-173}$$

    - Otherwise, the vertical motion vector component mvLXN[1] and the reference index refIdxLXN remain unchanged.

8.4.1.4 Derivation process for chroma motion vectors

Inputs to this process are a luma motion vector mvLX and a reference index refIdxLX.

Outputs of this process are a chroma motion vector mvCLX.

A chroma motion vector is derived from the corresponding luma motion vector. Since the accuracy of luma motion vectors is one-quarter sample and chroma has half resolution compared to luma, the accuracy of chroma motion vectors is one-eighth sample, i.e., a value of 1 for the chroma motion vector refers to a one-eighth sample displacement.

NOTE - For example when the luma vector applies to 8x16 luma samples, the corresponding chroma vector applies to 4x8 chroma samples and when the luma vector applies to 4x4 luma samples, the corresponding chroma vector applies to 2x2 chroma samples.

For the derivation of the motion vector mvCLX, the following applies.

- If the current macroblock is a frame macroblock, the horizontal and vertical components of the chroma motion vector mvCLX are derived by multiplying the corresponding components of luma motion vector mvLX by 2, through mapping one-quarter sample mvLX units to one-eighth sample mvCLX units

$$\text{mvCLX}[\,0\,] = \text{mvLX}[\,0\,] \tag{8-174}$$
$$\text{mvCLX}[\,1\,] = \text{mvLX}[\,1\,] \tag{8-175}$$

- Otherwise (the current macroblock is a field macroblock), only the horizontal component of the chroma motion vector mvCLX[0] is derived using Equation 8-174. The vertical component of the chroma motion vector mvCLX[1] is dependent on the parity of the current field or the current macroblock and the reference picture, which is referred by the reference index refIdxLX. mvCLX[1] is derived from mvLX[1] according to Table 8-9.

Table 8-9 – Derivation of the vertical component of the chroma vector in field coding mode

Parity conditions		**mvCLX[1]**
Reference picture (refIdxLX)	Current field (picture/macroblock)	
Top field	Bottom field	mvLX[1] + 2
Bottom field	Top field	mvLX[1] – 2
Otherwise		mvLX[1]

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, partWidth and partHeight
- luma motion vectors mvL0 and mvL1 and chroma motion vectors mvCL0 and mvCL1
- reference indices refIdxL0 and refIdxL1
- prediction list utilization flags, predFlagL0 and predFlagL1

Outputs of this process are

- the Inter prediction samples predPart, which are a (partWidth)x(partHeight) array predPart_L of prediction luma samples, and two (partWidth/2)x(partHeight/2) arrays predPart_{Cb}, predPart_{Cr} of prediction chroma samples, one for each of the chroma components Cb and Cr.

Let predPartL0_L and predPartL1_L be (partWidth)x(partHeight) arrays of predicted luma sample values and predPartL0_{Cb}, predPartL1_{Cb}, predPartL0_{Cr}, and predPartL1_{Cr} be (partWidth/2)x(partHeight/2) arrays of predicted chroma sample values.

For LX being replaced by either L0 or L1 in the variables predFlagLX, RefPicListX, refIdxLX, refPicLX, predPartLX, the following is specified.

When predFlagLX is equal to 1, the following applies.

- The reference frame consisting of an ordered two-dimensional array refPicLX_L of luma samples and two ordered two-dimensional arrays refPicLX_{Cb} and refPicLX_{Cr} of chroma samples is derived by invoking the process specified in subclause 8.4.2.1 with refIdxLX and RefPicListX given as input.

- The arrays predPartLX_L, predPartLX_{Cb}, and predPartLX_{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 mvLX, mvCLX, and the reference arrays with refPicLX_L, refPicLX_{Cb}, and refPicLX_{Cr} given as input.

For C being replaced by L, Cb, or Cr, the array $predPart_C$ of the prediction samples of component C is derived by invoking the process specified in subclause 8.4.2.3 with the current partition specified by mbPartIdx and subMbPartIdx and the array $predPartL0_C$ and $predPartL1_C$ as well as predFlagL0 and predFlagL1 given as input.

8.4.2.1 Reference picture selection process

Input to this process is a reference index refIdxLX.

Output of this process is a reference picture consisting of a two-dimensional array of luma samples $refPicLX_L$ and two two-dimensional arrays of chroma samples $refPicLX_{Cb}$ and $refPicLX_{Cr}$.

Reference picture list RefPicListX is a list of variables PicNum (for short-term reference pictures) and LongTermPicNum (for long-term reference pictures) of previously decoded reference frames, complementary reference field pairs, or non-paired reference fields that have been marked as "used for reference" as specified in subclause 8.2.5.

Depending on field_pic_flag, the meaning of PicNum and LongTermPicNum is specified as follows.

- If field_pic_flag is equal to 1, all entries of the RefPicListX are variables PicNum and LongTermPicNum of decoded reference fields or fields of decoded reference frames.
- Otherwise (field_pic_flag is equal to 0), all entries of RefPicListX are variables PicNum and LongTermPicNum of decoded reference frames or complementary reference field pairs.

The reference picture list RefPicListX is derived as specified in subclause 8.2.4.

For the derivation of the reference picture, the following applies.

- If field_pic_flag is equal to 1, the reference field or field of a reference frame referred by PicNum = RefPicListX[refIdxLX] or LongTermPicNum = RefPicListX[refIdxLX] shall be the output. The output reference field or field of a reference frame consists of a $(PicWidthInSamples_L)x(PicHeightInSamples_L)$ array of luma samples $refPicLX_L$ and two $(PicWidthInSamples_C)x(PicHeightInSamples_C)$ arrays of chroma samples $refPicLX_{Cb}$ and $refPicLX_{Cr}$.
- Otherwise (field_pic_flag is equal to 0), the following applies.
  - If the current macroblock is a frame macroblock, the reference frame or complementary reference field pair referred by PicNum = RefPicListX[refIdxLX] or LongTermPicNum = RefPicListX[refIdxLX] shall be the output. The output reference frame or complementary reference field pair consists of a $(PicWidthInSamples_L)x(PicHeightInSamples_L)$ array of luma samples $refPicLX_L$ and two $(PicWidthInSamples_C)x(PicHeightInSamples_C)$ arrays of chroma samples $refPicLX_{Cb}$ and $refPicLX_{Cr}$.
  - Otherwise (the current macroblock is a field macroblock), the following applies.
    - Let refFrame be the reference frame or complementary reference field pair that is referred by PicNum = RefPicListX[refIdxLX / 2] or LongTermPicNum = RefPicListX[refIdxLX / 2].
    - The field of refFrame is selected as follows.
      - If refIdxLX % 2 is equal to 0, the field of refFrame that has the same parity as the current macroblock shall be the output.
      - Otherwise (refIdxLX % 2 is equal to 1), the field of refFrame that has the opposite parity as the current macroblock shall be the output.
    - The output reference field or field of a reference frame consists of a $(PicWidthInSamples_L)x(PicHeightInSamples_L/2)$ array of luma samples $refPicLX_L$ and two $(PicWidthInSamples_C)x(PicHeightInSamples_C/2)$ arrays of chroma samples $refPicLX_{Cb}$ and $refPicLX_{Cr}$.

The reference picture sample arrays $refPicLX_L$, $refPicLX_{Cb}$, $refPicLX_{Cr}$ correspond to decoded sample arrays S_L, S_{Cb}, S_{Cr} derived in subclause 8.7 for previous decoded pictures.

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 mvLX given in quarter-luma-sample units,
- a chroma motion vector mvCLX given in eighth-chroma-sample units, and
- the selected reference picture sample arrays refPicLX_L, refPicLX_{Cb}, and refPicLX_{Cb}

Outputs of this process are

- a (partWidth)x(partHeight) array predPartLX_L of prediction luma sample values and
- two (partWidth/2)x(partHeight/2) arrays predPartLX_{Cb}, and predPartLX_{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 refPicLX_L, refPicLX_{Cb}, and refPicLX_{Cr}.

For each luma sample location ($0 <= x_L < \text{partWidth}$, $0 <= y_L < \text{partHeight}$) inside the prediction luma sample array predLX_L, the corresponding predicted luma sample value $\text{predLX}_L[x_L, y_L]$ is derived as follows:

$$\text{xInt}_L = xA_L + (\text{mvLX}[0] >> 2) + x_L \quad (8\text{-}176)$$
$$\text{yInt}_L = yA_L + (\text{mvLX}[1] >> 2) + y_L \quad (8\text{-}177)$$

$$\text{xFrac}_L = \text{mvLX}[0] \,\&\, 3 \quad (8\text{-}178)$$
$$\text{yFrac}_L = \text{mvLX}[1] \,\&\, 3 \quad (8\text{-}179)$$

- The prediction sample value $\text{predLX}_L[x_L, y_L]$ is derived by invoking the process specified in subclause 8.4.2.2.1 with (xInt_L, yInt_L), (xFrac_L, yFrac_L) and refPicLX_L given as input.

Let (xInt_C, yInt_C) be a chroma location given in full-sample units and (xFrac_C, yFrac_C) be an offset given in one-eighth sample units. These variables are used only inside this subclause for specifying general fractional-sample locations inside the reference sample arrays refPicLX_{Cb}, and refPicLX_{Cr}.

For each chroma sample location ($0 <= x_C < \text{partWidth}/2$, $0 <= y_C < \text{partHeight}/2$) inside the prediction chroma sample arrays predPartLX_{Cb} and predPartLX_{Cr}, the corresponding prediction chroma sample values $\text{predPartLX}_{Cb}[x_C, y_C]$ and $\text{predPartLX}_{Cr}[x_C, y_C]$ are derived as follows:

$$\text{xInt}_C = (xA_L >> 1) + (\text{mvCLX}[0] >> 3) + x_C \quad (8\text{-}180)$$
$$\text{yInt}_C = (yA_L >> 1) + (\text{mvCLX}[1] >> 3) + y_C \quad (8\text{-}181)$$

$$\text{xFrac}_C = \text{mvCLX}[0] \,\&\, 7 \quad (8\text{-}182)$$
$$\text{yFrac}_C = \text{mvCLX}[1] \,\&\, 7 \quad (8\text{-}183)$$

- The prediction sample value $\text{predPartLX}_{Cb}[x_C, y_C]$ is derived by invoking the process specified in subclause 8.4.2.2.2 with (xInt_C, yInt_C), (xFrac_C, yFrac_C) and refPicLX_{Cb} given as input.
- The prediction sample value $\text{predPartLX}_{Cr}[x_C, y_C]$ is derived by invoking the process specified in subclause 8.4.2.2.2 with (xInt_C, yInt_C), (xFrac_C, yFrac_C) and refPicLX_{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), and
- the luma sample array of the selected reference picture refPicLX_L

Output of this process is a predicted luma sample value $\text{predPartLX}_L[x_L, y_L]$.

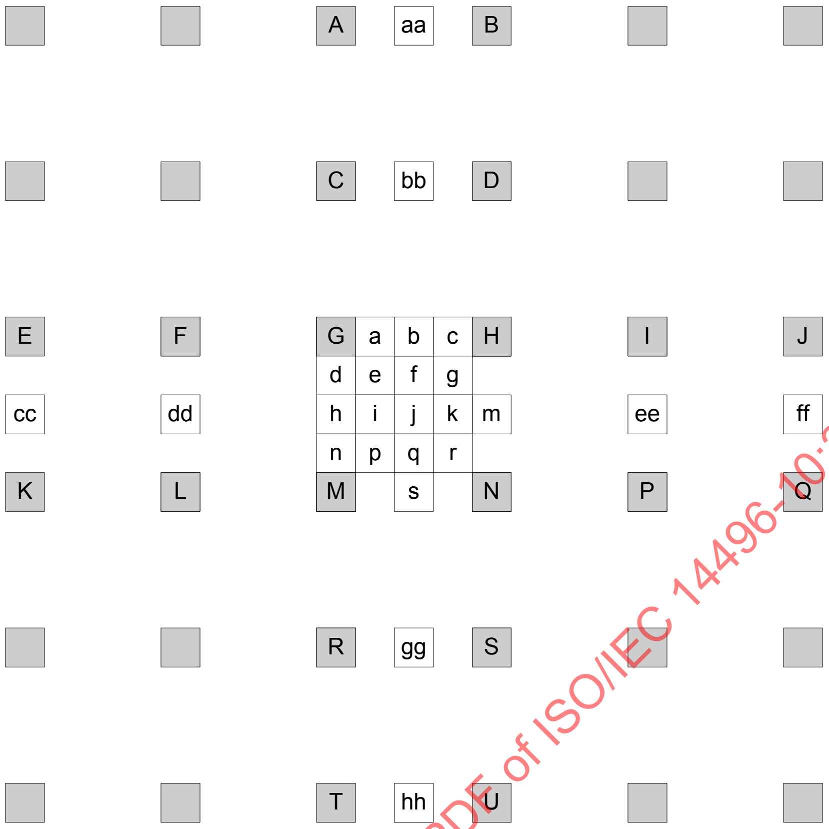


Figure 8-4 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for quarter sample luma interpolation.

The variable refPicHeightEffective$_L$, which is the height of the effective reference picture luma array, is derived as follows.

– If MbaffFrameFlag is equal to 0 or mb_field_decoding_flag is equal to 0, refPicHeightEffective$_L$ is set equal to PicHeightInSamples$_L$.

– Otherwise (MbaffFrameFlag is equal to 1 and mb_field_decoding_flag is equal to 1), refPicHeightEffective$_L$ is set equal to PicHeightInSamples$_L$ / 2.

In Figure 8-4, the positions labelled with upper-case letters within shaded blocks represent luma samples at full-sample locations inside the given two-dimensional array refPicLX$_L$ of luma samples. These samples may be used for generating the predicted luma sample value predPartLX$_L$[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 refPicLX$_L$ of luma samples are derived as follows:

$$\begin{aligned} xZ_L &= \text{Clip3}(\ 0,\ \text{PicWidthInSamples}_L - 1,\ \text{xInt}_L + xDZ_L\ ) \\ yZ_L &= \text{Clip3}(\ 0,\ \text{refPicHeightEffective}_L - 1,\ \text{yInt}_L + yDZ_L\ ) \end{aligned} \quad (8\text{-}184)$$

Table 8-10 specifies (xDZ$_L$, yDZ$_L$) for different replacements of Z.

Table 8-10 – 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 shall be derived by applying a 6-tap filter with tap values (1, -5, 20, 20, -5, 1). The luma prediction values at quarter sample positions shall be 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 shall be 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 shall be 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:

$$b_1 = (E - 5 * F + 20 * G + 20 * H - 5 * I + J) \quad (8\text{-}185)$$
$$h_1 = (A - 5 * C + 20 * G + 20 * M - 5 * R + T) \quad (8\text{-}186)$$

The final prediction values b and h shall be derived using:

$$b = \text{Clip1}((b_1 + 16) >> 5) \quad (8\text{-}187)$$
$$h = \text{Clip1}((h_1 + 16) >> 5) \quad (8\text{-}188)$$

– The samples at half sample position labelled as j shall be 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.

$$j_1 = cc - 5 * dd + 20 * h_1 + 20 * m_1 - 5 * ee + ff\text{, or} \quad (8\text{-}189)$$
$$j_1 = aa - 5 * bb + 20 * b_1 + 20 * s_1 - 5 * gg + hh \quad (8\text{-}190)$$

where intermediate values denoted as aa, bb, gg, s_1 and hh shall be 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 shall be derived by applying the 6-tap filter vertically in the same manner as the derivation of h_1. The final prediction value j shall be derived using:

$$j = \text{Clip1}((j_1 + 512) >> 10) \quad (8\text{-}191)$$

– The final prediction values s and m shall be derived from s_1 and m_1 in the same manner as the derivation of b and h, as given by:

$$s = \text{Clip1}((s_1 + 16) >> 5) \quad (8\text{-}192)$$
$$m = \text{Clip1}((m_1 + 16) >> 5) \quad (8\text{-}193)$$

– The samples at quarter sample positions labelled as a, c, d, n, f, i, k, and q shall be derived by averaging with upward rounding of the two nearest samples at integer and half sample positions using:

$$a = (G + b + 1) >> 1 \quad (8\text{-}194)$$
$$c = (H + b + 1) >> 1 \quad (8\text{-}195)$$
$$d = (G + h + 1) >> 1 \quad (8\text{-}196)$$
$$n = (M + h + 1) >> 1 \quad (8\text{-}197)$$
$$f = (b + j + 1) >> 1 \quad (8\text{-}198)$$
$$i = (h + j + 1) >> 1 \quad (8\text{-}199)$$
$$k = (j + m + 1) >> 1 \quad (8\text{-}200)$$
$$q = (j + s + 1) >> 1. \quad (8\text{-}201)$$

– The samples at quarter sample positions labelled as e, g, p, and r shall be derived by averaging with upward rounding of the two nearest samples at half sample positions in the diagonal direction using

$$e = (b + h + 1) >> 1 \quad (8\text{-}202)$$
$$g = (b + m + 1) >> 1 \quad (8\text{-}203)$$
$$p = (h + s + 1) >> 1 \quad (8\text{-}204)$$
$$r = (m + s + 1) >> 1. \quad (8\text{-}205)$$

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 $predPartLX_L[x_L, y_L]$. This assignment is done according to Table 8-11. The value of $predPartLX_L[x_L, y_L]$ shall be the output.

Table 8-11 – Assignment of the luma prediction sample predPartLX$_L$[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
predPartLX$_L$[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$_C$), and
- chroma component samples from the selected reference picture refPicLX$_C$.

Output of this process is a predicted chroma sample value predPartLX$_C$[x$_C$, y$_C$].

In Figure 8-5, the positions labelled with A, B, C, and D represent chroma samples at full-sample locations inside the given two-dimensional array refPicLX$_C$ of chroma samples.

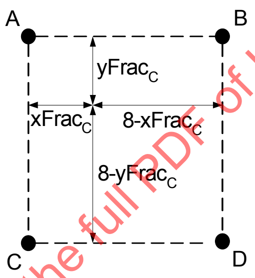


Figure 8-5 – Fractional sample position dependent variables in chroma interpolation and surrounding integer position samples A, B, C, and D.

The variable refPicHeightEffective$_C$, which is the height of the effective reference picture chroma array, is derived as follows.

– If MbaffFrameFlag is equal to 0 or mb_field_decoding_flag is equal to 0, refPicHeightEffective$_C$ is set equal to PicHeightInSamples$_C$.

– Otherwise (MbaffFrameFlag is equal to 1 and mb_field_decoding_flag is equal to 1), refPicHeightEffective$_C$ is set equal to PicHeightInSamples$_C$ / 2.

The sample coordinates specified in Equations 8-206 through 8-213 are used for generating the predicted chroma sample value predPartLX$_C$[x$_C$, y$_C$].

$$xA_C = \text{Clip3}(\ 0, \text{PicWidthInSamples}_C - 1, \text{xInt}_C\ ) \quad (8\text{-}206)$$
$$xB_C = \text{Clip3}(\ 0, \text{PicWidthInSamples}_C - 1, \text{xInt}_C + 1\ ) \quad (8\text{-}207)$$
$$xC_C = \text{Clip3}(\ 0, \text{PicWidthInSamples}_C - 1, \text{xInt}_C\ ) \quad (8\text{-}208)$$
$$xD_C = \text{Clip3}(\ 0, \text{PicWidthInSamples}_C - 1, \text{xInt}_C + 1\ ) \quad (8\text{-}209)$$

$$yA_C = \text{Clip3}(\ 0, \text{refPicHeightEffective}_C - 1, \text{yInt}_C\ ) \quad (8\text{-}210)$$
$$yB_C = \text{Clip3}(\ 0, \text{refPicHeightEffective}_C - 1, \text{yInt}_C\ ) \quad (8\text{-}211)$$
$$yC_C = \text{Clip3}(\ 0, \text{refPicHeightEffective}_C - 1, \text{yInt}_C + 1\ ) \quad (8\text{-}212)$$
$$yD_C = \text{Clip3}(\ 0, \text{refPicHeightEffective}_C - 1, \text{yInt}_C + 1\ ) \quad (8\text{-}213)$$

Given the chroma samples A, B, C, and D at full-sample locations specified in Equations 8-206 through 8-213, the predicted chroma sample value predPartLX$_C$[x$_C$, y$_C$] is derived as follows:

$$\text{predPartLX}_C[\,x_C, y_C\,] = (\,(\,8 - \text{xFrac}_C\,) * (\,8 - \text{yFrac}_C\,) * A + \text{xFrac}_C * (\,8 - \text{yFrac}_C\,) * B + (\,8 - \text{xFrac}_C\,) * \text{yFrac}_C * C + \text{xFrac}_C * \text{yFrac}_C * D + 32\,) >> 6 \quad (8\text{-}214)$$

8.4.2.3 Weighted sample prediction process

Inputs to this process are

- mbPartIdx: the current partition given by the partition index
- subMbPartIdx: the sub-macroblock partition index
- predFlagL0 and predFlagL1: prediction list utilization flags
- predPartLX_L: a (partWidth)x(partHeight) array of prediction luma samples (with LX being replaced by L0 or L1 depending on predFlagL0 and predFlagL1)
- predPartLX_{Cb} and predPartLX_{Cr}: (partWidth/2)x(partHeight/2) arrays of prediction chroma samples, one for each of the chroma components Cb and Cr (with LX being replaced by L0 or L1 depending on predFlagL0 and predFlagL1)

Outputs of this process are

- predPart_L: a (partWidth)x(partHeight) array of prediction luma samples and
- predPart_{Cb}, and predPart_{Cr}: (partWidth/2)x(partHeight/2) arrays of prediction chroma samples, one for each of the chroma components Cb and Cr.

For macroblocks or partitions with predFlagL0 equal to 1 in P and SP slices, the following applies.

- If weighted_pred_flag is equal to 0, the default weighted sample prediction process as described in subclause 8.4.2.3.1 is invoked with the same inputs and outputs as the process described in this subclause.
- Otherwise (weighted_pred_flag is equal to 1), the explicit weighted prediction process as described in subclause 8.4.2.3.2 is invoked with the same inputs and outputs as the process described in this subclause.

For macroblocks or partitions with predFlagL0 or predFlagL1 equal to 1 in B slices, the following applies.

- If weighted_bipred_idc is equal to 0, the default weighted sample prediction process as described in subclause 8.4.2.3.1 is invoked with the same inputs and outputs as the process described in this subclause.
- Otherwise, if weighted_bipred_idc is equal to 1, the explicit weighted sample prediction process as described in subclause 8.4.2.3.2, for macroblocks or partitions with predFlagL0 or predFlagL1 equal to 1 with the same inputs and outputs as the process described in this subclause.
- Otherwise (weighted_bipred_idc is equal to 2), the following applies.
  - If predFlagL0 is equal to 1 and predFlagL1 is equal to 1, the implicit weighted sample prediction as described in subclause 8.4.2.3.2 is invoked with the same inputs and outputs as the process described in this subclause.
  - Otherwise (predFlagL0 or predFlagL1 are equal to 1 but not both), the default weighted sample prediction process as described in subclause 8.4.2.3.1 is invoked with the same inputs and outputs as the process described in this subclause.

8.4.2.3.1 Default weighted sample prediction process

Input to this process are the same as specified in subclause 8.4.2.3.

Output of this process are the same as specified in subclause 8.4.2.3.

Depending on the component for which the prediction block is derived, the following applies.

- If the luma sample prediction values $\text{predPart}_L[\,x, y\,]$ are derived, the following applies with C set equal to L, x set equal to 0 .. partWidth - 1, and y set equal to 0 .. partHeight - 1.
- Otherwise, if the chroma Cb component sample prediction values $\text{predPart}_{Cb}[\,x, y\,]$ are derived, the following applies with C set equal to Cb, x set equal to 0 .. partWidth / 2 - 1, and y set equal to 0 .. partHeight / 2 - 1.
- Otherwise (the chroma Cr component sample prediction values $\text{predPart}_{Cr}[\,x, y\,]$ are derived), the following applies with C set equal to Cr, x set equal to 0 .. partWidth / 2 - 1, and y set equal to 0 .. partHeight / 2 - 1.

The prediction sample values are derived as follows.

- If predFlagL0 is equal to 1 and predFlagL1 is equal to 0 for the current partition

  $\text{predPart}_C[\,x, y\,] = \text{predPartL0}_C[\,x, y\,]$ (8-215)

- Otherwise, if predFlagL0 is equal to 0 and predFlagL1 is equal to 1 for the current partition

  $\text{predPart}_C[\,x, y\,] = \text{predPartL1}_C[\,x, y\,]$ (8-216)

- Otherwise (predFlagL0 and predFlagL1 are equal to 1 for the current partition),

  $\text{predPart}_C[\,x, y\,] = (\,\text{predPartL0}_C[\,x, y\,] + \text{predPartL1}_C[\,x, y\,] + 1\,) >> 1.$ (8-217)

8.4.2.3.2 Weighted sample prediction process

Input to this process are the same as specified in subclause 8.4.2.3.

Output of this process are the same as specified in subclause 8.4.2.3.

Depending on the component for which the prediction block is derived, the following applies.

- If the luma sample prediction values $\text{predPart}_L[\,x, y\,]$ are derived, the following applies with C set equal to L, x set equal to 0 .. partWidth - 1, and y set equal to 0 .. partHeight - 1.
- Otherwise, if the chroma Cb component sample prediction values $\text{predPart}_{Cb}[\,x, y\,]$ are derived, the following applies with C set equal to Cb, x set equal to 0 .. partWidth / 2 - 1, and y set equal to 0 .. partHeight / 2 - 1.
- Otherwise (the chroma Cr component sample prediction values $\text{predPart}_{Cr}[\,x, y\,]$ are derived), the following applies with C set equal to Cr, x set equal to 0 .. partWidth / 2 - 1, and y set equal to 0 .. partHeight / 2 - 1.

The prediction sample values are derived as follows

- If the partition mbPartIdx\subMbPartIdx has predFlagL0 equal to 1 and predFlagL1 equal to 0, the final predicted sample values $\text{predPart}_C[\,x, y\,]$ are derived by

```
if( logWD >= 1 )
    $\text{predPart}_C[\,x, y\,] = \text{Clip1}(\,(\,(\,\text{predPartL0}_C[\,x, y\,] * w_0 + 2^{\text{logWD} - 1}\,) >> \text{logWD}\,) + o_0\,)$
else                                                                                    (8-218)
    $\text{predPart}_C[\,x, y\,] = \text{Clip1}(\,\text{predPartL0}_C[\,x, y\,] * w_0 + o_0\,)$
```

- Otherwise, if the partition mbPartIdx\subMbPartIdx has predFlagL0 equal to 0 and predFlagL1 equal to 1, the final predicted sample values $\text{predPart}_C[\,x, y\,]$ are derived by

```
if( logWD >= 1 )
    $\text{predPart}_C[\,x, y\,] = \text{Clip1}(\,(\,(\,\text{predPartL1}_C[\,x, y\,] * w_1 + 2^{\text{logWD} - 1}\,) >> \text{logWD}\,) + o_1\,)$
else                                                                                    (8-219)
    $\text{predPart}_C[\,x, y\,] = \text{Clip1}(\,\text{predPartL1}_C[\,x, y\,] * w_1 + o_1\,)$
```

- Otherwise (the partition mbPartIdx\subMbPartIdx has both predFlagL0 and predFlagL1 equal to 1), the final predicted sample values $\text{predPart}_C[\,x, y\,]$ are derived by

$$\text{predPart}_C[\,x, y\,] = \text{Clip1}(\,(\,(\,\text{predPartL0}_C[\,x, y\,] * w_0 + \text{predPartL1}_C[\,x, y\,] * w_1 + 2^{\text{logWD}}\,) >> (\,\text{logWD} + 1\,)\,) + (\,(\,o_0 + o_1 + 1\,) >> 1\,)\,) \quad (8\text{-}220)$$

The variables in the above derivation for the prediction samples are derived as follows.

- If weighted_bipred_idc is equal to 2 and the slice_type is equal to B, implicit mode weighted prediction is used as follows.

  $\text{logWD} = 5$ (8-221)

  $o_0 = 0$ (8-222)

  $o_1 = 0$ (8-223)

  and w_0 and w_1 are derived as follows.

- The variables currPicOrField, pic0, and pic1 are derived as follows:
  - If field_pic_flag is equal to 0 and the current macroblock is a field macroblock, the following applies.
    - currPicOrField is the field of the current picture CurrPic that has the same parity as the current macroblock.
    - Let frame0 be the frame or complementary field pair that is referred to by RefPicList0[refIdxL0 / 2].
    - The variable pic0 is derived as follows.
      - If refIdxL0 % 2 is equal to 0, pic0 is the field of frame0 that has the same parity as the current macroblock.
      - Otherwise (refIdxL0 % 2 is not equal to 0), pic0 is the field of frame0 that has the opposite parity of the current macroblock.
    - Let frame1 be the frame or complementary field pair that is referred to by RefPicList1[refIdxL1 / 2].
    - The variable pic1 is derived as follows.
      - If refIdxL1 % 2 is equal to 0, pic1 is the field of frame1 that has the same parity as the current macroblock.
      - Otherwise (refIdxL1 % 2 is not equal to 0), pic1 is the field of frame1 that has the opposite parity of the current macroblock.
  - Otherwise (field_pic_flag is equal to 1 or the current macroblock is a frame macroblock), currPicOrField is the current picture CurrPic, pic1 is the decoded reference picture referred to by RefPicList1[refIdxL1], and pic0 is the decoded reference picture referred to by RefPicList0[refIdxL0].
- The variables tb, td, tx, and DistScaleFactor are derived from the values of currPicOrField, pic0, pic1 using Equations 8-154, 8-155, 8-150, and 8-151, respectively.
- If DiffPicOrderCnt(pic1, pic0) is equal to 0 or one or both of pic1 and pic0 is a long-term reference picture or (DistScaleFactor >> 2) < -64 or (DistScaleFactor >> 2) > 128, w_0 and w_0 are derived as

  $$w_0 = 32 \quad (8\text{-}224)$$

  $$w_1 = 32 \quad (8\text{-}225)$$

- Otherwise,

  $$w_0 = 64 - (\text{DistScaleFactor} >> 2) \quad (8\text{-}226)$$

  $$w_1 = \text{DistScaleFactor} >> 2 \quad (8\text{-}227)$$

- Otherwise (weighted_pred_flag is equal to 1 in P or SP slices or weighted_bipred_idc equal to 1 in B slices), explicit mode weighted prediction is used as follows.
  - The variables refIdxL0WP and refIdxL1WP are derived as follows.
    - If MbaffFrameFlag is equal to 1 and the current macroblock is a field macroblock

      $$\text{refIdxL0WP} = \text{refIdxL0} >> 1 \quad (8\text{-}228)$$

      $$\text{refIdxL1WP} = \text{refIdxL1} >> 1 \quad (8\text{-}229)$$

    - Otherwise (MbaffFrameFlag is equal to 0 or the current macroblock is a frame macroblock),

      $$\text{refIdxL0WP} = \text{refIdxL0} \quad (8\text{-}230)$$

      $$\text{refIdxL1WP} = \text{refIdxL1} \quad (8\text{-}231)$$

  - The variables logWD, w_0, w_1, o_0, and o_1 are derived as follows.
    - If C in predPart_C[x, y] is replaced by L for luma samples

      $$\text{logWD} = \text{luma\_log2\_weight\_denom} \quad (8\text{-}232)$$

      $$w_0 = \text{luma\_weight\_l0}[\ \text{refIdxL0WP}\ ] \quad (8\text{-}233)$$

      $$w_1 = \text{luma\_weight\_l1}[\ \text{refIdxL1WP}\ ] \quad (8\text{-}234)$$

      $$o_0 = \text{luma\_offset\_l0}[\ \text{refIdxL0WP}\ ] \quad (8\text{-}235)$$

      $$o_1 = \text{luma\_offset\_l1}[\ \text{refIdxL1WP}\ ] \quad (8\text{-}236)$$

- Otherwise (C in $\text{predPart}_C[\ x, y\ ]$ is replaced by Cb or Cr for chroma samples, with iCbCr = 0 for Cb, iCbCr = 1 for Cr),

$$\text{logWD} = \text{chroma\_log2\_weight\_denom} \tag{8-237}$$

$$w_0 = \text{chroma\_weight\_l0}[\ \text{refIdxL0WP}\ ][\ \text{iCbCr}\ ] \tag{8-238}$$

$$w_1 = \text{chroma\_weight\_l1}[\ \text{refIdxL1WP}\ ][\ \text{iCbCr}\ ] \tag{8-239}$$

$$o_0 = \text{chroma\_offset\_l0}[\ \text{refIdxL0WP}\ ][\ \text{iCbCr}\ ] \tag{8-240}$$

$$o_1 = \text{chroma\_offset\_l1}[\ \text{refIdxL1WP}\ ][\ \text{iCbCr}\ ] \tag{8-241}$$

When explicit mode weighted prediction is used and the partition mbPartIdx\subMbPartIdx has both predFlagL0 and predFlagL1 equal to 1, the following constraint shall be obeyed

$$-128 <= w_0 + w_1 <= (\ (\ \text{logWD}\ \ ==\ \ 7\ )\ ?\ 127 : 128\ ) \tag{8-242}$$

NOTE –For implicit mode weighted prediction, weights w_0 and w_1 are each guaranteed to be in the range of -64..128 and the constraint expressed in Equation 8-242, although not explicitly imposed, will always be met. For explicit mode weighted prediction with logWD equal to 7, when one of the two weights w_0 or w_1 is inferred to be equal to 128 (as a consequence of luma_weight_l0_flag, luma_weight_l1_flag, chroma_weight_l0_flag, or chroma_weight_l1_flag equal to 0), the other weight (w_1 or w_0) must have a negative value in order for the constraint expressed in Equation 8-242 to hold (and therefore the other flag luma_weight_l0_flag, luma_weight_l1_flag, chroma_weight_l0_flag, or chroma_weight_l1_flag must be equal to 1).

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), LumaLevel (if available), ChromaDCLevel, ChromaACLevel, and available Inter or Intra prediction sample arrays for the current macroblock for the applicable component pred_L, pred_{Cb}, or pred_{Cr}.

NOTE – When decoding a macroblock in Intra_4x4 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 component S'_L, S'_{Cb}, or S'_{Cr}.

NOTE – When decoding a macroblock in Intra_4x4 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 or B_Skip, all values of LumaLevel, ChromaDCLevel, ChromaACLevel are set equal to 0 for the current macroblock.

8.5.1 Specification of transform decoding process for residual blocks

When the current macroblock prediction mode is not equal to Intra_16x16, the variable LumaLevel 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 transform coefficient scanning process as described in subclause 8.5.4 is invoked with LumaLevel[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 8.5.8 is invoked with c as the input and r as the output.

3. 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).

4. The 4x4 array u with elements u_{ij} for i, j = 0..3 is derived as

$$u_{ij} = \text{Clip1}(\ \text{pred}_L[\ xO + j, yO + i\ ] + r_{ij}\ ) \tag{8-243}$$

5. The picture construction process prior to deblocking filter process in subclause 8.5.9 is invoked with luma4x4BlkIdx, u as the input and S' as the output.

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 transform coefficient scanning process as described in subclause 8.5.4 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 8.5.6 is invoked with c as the input and dcY as the output.

2. For a 4x4 luma block indexed by luma4x4BlkIdx = 0..15, 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 8-6 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.

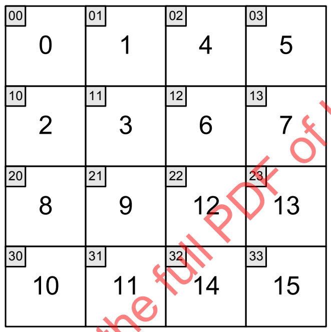


Figure 8-6 – Assignment of the indices of dcY to luma4x4BlkIdx.

   The elements in lumaList with index k = 1..15 are specified as

   lumaList[k] = Intra16x16ACLevel[luma4x4BlkIdx][k - 1] (8-244)

   b. The inverse transform coefficient scanning process as described in subclause 8.5.4 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 8.5.8 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 4x4 array u with elements u_{ij} for i, j = 0..3 is derived as

$$u_{ij} = \mathrm{Clip1}(\mathrm{pred}_L[xO + j, yO + i] + r_{ij}) \quad (8\text{-}245)$$

   f. The picture construction process prior to deblocking filter process in subclause 8.5.9 is invoked with luma4x4BlkIdx, u as the input and S' as the output.

8.5.3 Specification of transform decoding process for chroma samples

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. For each chroma component, the transform decoding proceeds separately in the following ordered steps:

1. The 2x2 chroma DC transform coefficients of the 4x4 chroma blocks of the component indexed by iCbCr of the macroblock are decoded.

a. The 2x2 array c is derived using the inverse raster scanning process applied to ChromaDCLevel as follows

$$c = \begin{bmatrix} \text{ChromaDCLevel}[\,\text{iCbCr}\,][\,0\,] & \text{ChromaDCLevel}[\,\text{iCbCr}\,][\,1\,] \\ \text{ChromaDCLevel}[\,\text{iCbCr}\,][\,2\,] & \text{ChromaDCLevel}[\,\text{iCbCr}\,][\,3\,] \end{bmatrix} \tag{8-246}$$

b. The scaling and transformation process for chroma DC transform coefficients as specified in subclause 8.5.7 is invoked with c as the input and dcC as the output.

2. For each 4x4 chroma block indexed by chroma4x4BlkIdx = 0..3 of the component indexed by iCbCr, 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 8-7 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.

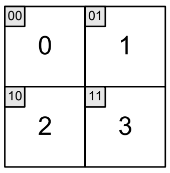


Figure 8-7 – Assignment of the indices of dcC to chroma4x4BlkIdx.

The elements in chromaList with index k = 1..15 are specified as

chromaList[k] = ChromaACLevel[chroma4x4BlkIdx][k - 1] (8-247)

b. The inverse transform coefficient scanning process as described in subclause 8.5.4 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 8.5.8 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 macroblock is derived as follows

xO = InverseRasterScan(chroma4x4BlkIdx, 4, 4, 8, 0) (8-248)

yO = InverseRasterScan(chroma4x4BlkIdx, 4, 4, 8, 1) (8-249)

e. The 4x4 array u with elements u_{ij} for i, j = 0..3 is derived as

$$u_{ij} = \text{Clip1}(\, \text{pred}_C[\, xO + j,\, yO + i\,] + r_{ij}\,) \tag{8-250}$$

f. The picture construction process prior to deblocking filter process in subclause 8.5.9 is invoked with chroma4x4BlkIdx, u as the input and S' as the output.

8.5.4 Inverse scanning process for transform coefficients

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 with level assigned to locations in the transform block.

The decoding process maps the sequence of transform coefficient levels to the transform coefficient level positions. For this mapping, the two inverse scanning patterns shown in Figure 8-8 are used.

The inverse zig-zag scan shall be used for frame macroblocks and the inverse field scan shall be used for field macroblocks.

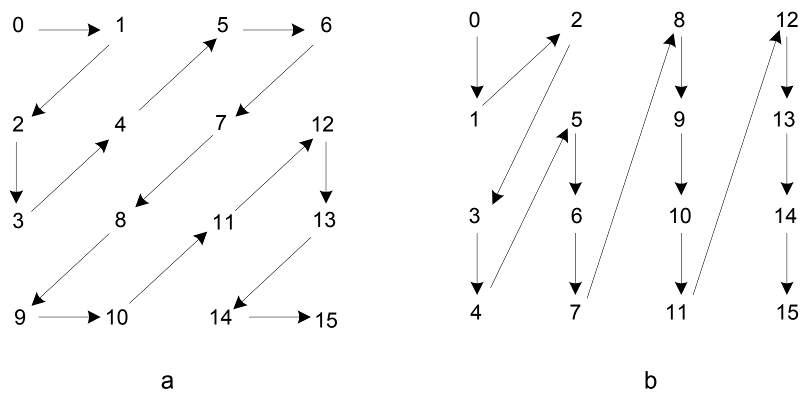


Figure 8-8 – a) Zig-zag scan. b) Field scan

Table 8-12 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 8-12 – Specification of mapping of idx to c_{ij} for zig-zag and field scan

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

8.5.5 Derivation process for the chroma quantisation parameters and scaling function

Outputs of this process are:

– QP_C: the chroma quantisation parameter

– QS_C: the additional chroma quantisation parameter required for decoding SP and SI slices (if applicable)

NOTE – QP quantisation parameter values QP_Y, QP_C, QS_Y, and QS_C are always in the range of 0 to 51, inclusive.

The value of QP_C for chroma 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 shall be determined as specified in Table 8-13 based on the index denoted as qP_I. The value of qP_I shall be derived as follows.

$$qP_I = \text{Clip3}(\ 0,\ 51,\ QP_Y + \text{chroma\_qp\_index\_offset}\ ) \qquad (8\text{-}251)$$

Table 8-13 – 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

When the current slice is an SP or SI slice, QS_C is derived using the above process, substituting QP_Y with QS_Y and QP_C with QS_C.

The function LevelScale(m, i, j) is specified as follows:

$$\text{LevelScale}(m,i,j) = \begin{cases} v_{m0} & \text{for} \quad (i,j) \in \{(0,0),(0,2),(2,0),(2,2)\}, \\ v_{m1} & \text{for} \quad (i,j) \in \{(1,1),(1,3),(3,1),(3,3)\}, \\ v_{m2} & \text{otherwise;} \end{cases} \qquad (8\text{-}252)$$

where the first and second subscripts of v are row and column indices, respectively, of the matrix specified as:

$$v = \begin{bmatrix} 10 & 16 & 13 \\ 11 & 18 & 14 \\ 13 & 20 & 16 \\ 14 & 23 & 18 \\ 16 & 25 & 20 \\ 18 & 29 & 23 \end{bmatrix}. \quad (8\text{-}253)$$

8.5.6 Scaling and transformation process for luma DC transform coefficients for Intra_16x16 macroblock type

Inputs to this process are transform coefficient level values for luma 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 luma 4x4 blocks of Intra_16x16 macroblocks as a 4x4 array dcY with elements dcY_{ij}.

The inverse transform for the 4x4 luma DC transform coefficients is specified by:

$$f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}. \quad (8\text{-}254)$$

A bitstream conforming to this Recommendation | International Standard shall not contain data that results in any element f_{ij} of f that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive.

After the inverse transform, scaling is performed as follows.

- If QP_Y is greater than or equal to 12, the scaled result shall be derived as

$$dcY_{ij} = (f_{ij} * \text{LevelScale}(QP_Y \% 6, 0, 0)) << (QP_Y / 6 - 2), \quad \text{with} \quad i, j = 0..3 . \quad (8\text{-}255)$$

- Otherwise (QP_Y is less than 12), the scaled result shall be derived as

$$dcY_{ij} = (f_{ij} * \text{LevelScale}(QP_Y \% 6, 0, 0) + 2^{1-QP_Y/6}) >> (2 - QP_Y / 6), \quad \text{with} \quad i, j = 0..3 \quad (8\text{-}256)$$

A bitstream conforming to this Recommendation | International Standard shall not contain data that results in any element dcY_{ij} of dcY that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive.

NOTE – When entropy_coding_mode_flag is equal to 0 and QP_Y is less than 10, 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 8-256, a larger range of values must be supported in the decoder prior to the right shift.

8.5.7 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 a 2x2 array c with elements c_{ij}, where i and j form a two-dimensional frequency index.

Outputs of this process are 4 scaled DC values as a 2x2 array dcC with elements dcC_{ij}.

The inverse transform for the 2x2 chroma DC transform coefficients is specified by:

$$f = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (8\text{-}257)$$

A bitstream conforming to this Recommendation | International Standard shall not contain data that results in any element f_{ij} of f that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive.

After the inverse transform, scaling is performed as follows.

- If QP_C is greater than or equal to 6, the scaled result shall be derived as

$$dcC_{ij} = (f_{ij} * \text{LevelScale}(QP_C \,\%\, 6, 0, 0)) << (QP_C \,/\, 6 - 1), \quad \text{with} \quad i, j = 0, 1 \qquad (8\text{-}258)$$

- Otherwise (QP_C is less than 6), the scaled result shall be derived by

$$dcC_{ij} = (f_{ij} * \text{LevelScale}(QP_C \,\%\, 6, 0, 0)) >> 1, \quad \text{with} \quad i, j = 0, 1 \qquad (8\text{-}259)$$

A bitstream conforming to this Recommendation | International Standard shall not contain data that results in any element dcC_{ij} of dcC that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive.

NOTE – When entropy_coding_mode_flag is equal to 0 and QP_C is less than 4, 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 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 8-259, a larger range of values must be supported in the decoder prior to the right shift.

8.5.8 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 sMbFlag is derived as follows.

- If mb_type is equal to SI or the macroblock prediction mode is equal to Inter in an SP slice, sMbFlag is set equal to 1,
- Otherwise (mb_type not equal to SI and the macroblock prediction mode is not equal to Inter in an SP slice), 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

$$qP = QP_Y \qquad (8\text{-}260)$$

- Otherwise, if the input array c relates to a luma residual block and sMbFlag is equal to 1

$$qP = QS_Y \qquad (8\text{-}261)$$

- Otherwise, if the input array c relates to a chroma residual block and sMbFlag is equal to 0

$$qP = QP_C \qquad (8\text{-}262)$$

- Otherwise (the input array c relates to a chroma residual block and sMbFlag is equal to 1),

$$qP = QS_C \qquad (8\text{-}263)$$

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 prediction mode or c relates to a chroma residual block

  the variable d_{00} is derived by

$$d_{00} = c_{00} \qquad (8\text{-}264)$$

- Otherwise,

$$d_{ij} = (c_{ij} * \text{LevelScale}(qP \% 6, i, j)) << (qP / 6), \quad \text{with} \quad i, j = 0..3 \text{ except as noted above} \tag{8-265}$$

The bitstream shall not contain data that results in any element d_{ij} of d with i, j = 0..3 that exceeds the range of integer values from -2^{15} to $2^{15}-1$, 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.

$$e_{i0} = d_{i0} + d_{i2}, \quad \text{with} \quad i = 0..3 \tag{8-266}$$

$$e_{i1} = d_{i0} - d_{i2}, \quad \text{with} \quad i = 0..3 \tag{8-267}$$

$$e_{i2} = (d_{i1} >> 1) - d_{i3}, \quad \text{with} \quad i = 0..3 \tag{8-268}$$

$$e_{i3} = d_{i1} + (d_{i3} >> 1), \quad \text{with} \quad i = 0..3 \tag{8-269}$$

The bitstream shall not contain data that results in any element e_{ij} of e with i, j = 0..3 that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive.

Then, the transformed result is computed from these intermediate values as follows.

$$f_{i0} = e_{i0} + e_{i3}, \quad \text{with} \quad i = 0..3 \tag{8-270}$$

$$f_{i1} = e_{i1} + e_{i2}, \quad \text{with} \quad i = 0..3 \tag{8-271}$$

$$f_{i2} = e_{i1} - e_{i2}, \quad \text{with} \quad i = 0..3 \tag{8-272}$$

$$f_{i3} = e_{i0} - e_{i3}, \quad \text{with} \quad i = 0..3 \tag{8-273}$$

The bitstream shall not contain data that results in any element f_{ij} of f with i, j = 0..3 that exceeds the range of integer values from -2^{15} to $2^{15}-1$, 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.

$$g_{0j} = f_{0j} + f_{2j}, \quad \text{with} \quad j = 0..3 \tag{8-274}$$

$$g_{1j} = f_{0j} - f_{2j}, \quad \text{with} \quad j = 0..3 \tag{8-275}$$

$$g_{2j} = (f_{1j} >> 1) - f_{3j}, \quad \text{with} \quad j = 0..3 \tag{8-276}$$

$$g_{3j} = f_{1j} + (f_{3j} >> 1), \quad \text{with} \quad j = 0..3 \tag{8-277}$$

The bitstream shall not contain data that results in any element g_{ij} of g with i, j = 0..3 that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive.

Then, the transformed result is computed from these intermediate values as follows.

$$h_{0j} = g_{0j} + g_{3j}, \quad \text{with} \quad j = 0..3 \tag{8-278}$$

$$h_{1j} = g_{1j} + g_{2j}, \quad \text{with} \quad j = 0..3 \tag{8-279}$$

$$h_{2j} = g_{1j} - g_{2j}, \quad \text{with} \quad j = 0..3 \tag{8-280}$$

$$h_{3j} = g_{0j} - g_{3j}, \quad \text{with} \quad j = 0..3 \tag{8-281}$$

The bitstream shall not contain data that results in any element h_{ij} of h with i, j = 0..3 that exceeds the range of integer values from -2^{15} to $2^{15}-33$, 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 shall be derived as

$$r_{ij} = (h_{ij} + 2^5) >> 6 \quad \text{with} \quad i, j = 0..3 \tag{8-282}$$

8.5.9 Picture construction process prior to deblocking filter process

Inputs to this process are

- luma4x4BlkIdx or chroma4x4BlkIdx
- a 4x4 sample array u with elements u_{ij} which is either a luma or chroma block

Outputs of this process are constructed sample blocks s' prior to the deblocking filter process.

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 4x4 luma block, the following applies.

- 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).
- Depending on the variable MbaffFrameFlag, the following applies.
  - If MbaffFrameFlag is equal to 1 and the current macroblock is a field macroblock

$$S'_L[\ xP + xO + j,\ yP + 2 * (\ yO + i\ )\ ] = u_{ij} \quad \text{with } i, j = 0..3 \qquad (8\text{-}283)$$

  - Otherwise (MbaffFrameFlag is equal to 0 or the current macroblock is a frame macroblock),

$$S'_L[\ xP + xO + j,\ yP + yO + i\ ] = u_{ij} \quad \text{with } i, j = 0..3 \qquad (8\text{-}284)$$

When u is a chroma block, for each sample u_{ij} of the 4x4 chroma block, the following applies.

- The subscript C in the variables S'_C and pred_C is replaced with Cb for the Cb chroma component and with Cr for the Cr chroma component.
- The position of the upper-left sample of a 4x4 chroma block with index chroma4x4BlkIdx inside the macroblock is derived as follows.

$$xO = \text{InverseRasterScan}(\ \text{chroma4x4BlkIdx}, 4, 4, 8, 0\ ) \qquad (8\text{-}285)$$

$$yO = \text{InverseRasterScan}(\ \text{chroma4x4BlkIdx}, 4, 4, 8, 1\ ) \qquad (8\text{-}286)$$

- Depending on the variable MbaffFrameFlag, the following applies.
  - If MbaffFrameFlag is equal to 1 and the current macroblock is a field macroblock

$$S'_C[\ (\ xP >> 1\ ) + xO + j,\ (\ (\ yP + 1\ ) >> 1\ ) + 2 * (\ yO + i\ )\ ] = u_{ij} \text{ with } i, j = 0..3 \qquad (8\text{-}287)$$

  - Otherwise (MbaffFrameFlag is equal to 0 or the current macroblock is a frame macroblock),

$$S'_C[\ (\ xP >> 1\ ) + xO + j,\ (\ (\ yP + 1\ ) >> 1\ ) + yO + i\ ] = u_{ij} \text{ with } i, j = 0..3 \qquad (8\text{-}288)$$

8.6 Decoding process for P macroblocks in SP slices or SI macroblocks

This process is invoked when decoding P macroblock types in an SP slice type or an SI macroblock type in SI slices.

Inputs to this process are the prediction residual transform coefficient levels and the predicted samples for the current macroblock.

Outputs of this process are the decoded samples of the current macroblock prior to the deblocking filter process.

This subclause specifies the transform coefficient decoding process and picture construction process for P macroblock types in SP slices and SI macroblock type in SI slices.

NOTE – SP slices make use of Inter predictive coding to exploit temporal redundancy in the sequence, in a similar manner to P slice coding. Unlike P slice coding, however, SP slice coding allows identical reconstruction of a slice even when different reference pictures are being used. SI slices make use of spatial prediction, in a similar manner to I slices. SI slice coding allows identical reconstruction to a corresponding SP slice. The properties of SP and SI slices aid in providing functionalities for bitstream switching, splicing, random access, fast-forward, fast reverse, and error resilience/recovery.

An SP slice consists of macroblocks coded either as I macroblock types or P macroblock types.

An SI slice consists of macroblocks coded either as I macroblock types or SI macroblock type.

The transform coefficient decoding process and picture construction process prior to deblocking filter process for I macroblock types in SI slices shall be invoked as specified in subclause 8.5. SI macroblock type shall be decoded as described below.

When the current macroblock is coded as P_Skip, all values of LumaLevel, ChromaDCLevel, ChromaACLevel are set equal to 0 for the current macroblock.

8.6.1 SP decoding process for non-switching pictures

This process is invoked, when decoding P macroblock types in SP slices in which sp_for_switch_flag is equal to 0.

Inputs to this process are Inter prediction samples for the current macroblock from subclause 8.4 and the prediction residual transform coefficient levels.

Outputs of this process are the decoded samples of the current macroblock prior to the deblocking filter process.

This subclause applies to all macroblocks in SP slices in which sp_for_switch_flag is equal to 0, except those with macroblock prediction mode equal to Intra_4x4 or Intra_16x16. It does not apply to SI slices.

8.6.1.1 Luma transform coefficient decoding process

Inputs to this process are Inter prediction luma samples for the current macroblock pred_L from subclause 8.4 and the prediction residual transform coefficient levels, LumaLevel, and the index of the 4x4 luma block luma4x4BlkIdx.

Outputs of this process are the decoded luma samples of the current macroblock prior to the deblocking filter process.

The position of the upper-left sample of the 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 (x, y).

Let the variable p be a 4x4 array of prediction samples with element p_{ij} being derived as follows.

$$p_{ij} = \text{pred}_L[\, x + j,\, y + i \,] \quad \text{with } i, j = 0..3 \tag{8-289}$$

The variable p is transformed producing transform coefficients c^p according to:

$$c^p = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} p_{00} & p_{01} & p_{02} & p_{03} \\ p_{10} & p_{11} & p_{12} & p_{13} \\ p_{20} & p_{21} & p_{22} & p_{23} \\ p_{30} & p_{31} & p_{32} & p_{33} \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \tag{8-290}$$

The inverse transform coefficient scanning process as described in subclause 8.5.4 is invoked with LumaLevel[luma4x4BlkIdx] as the input and the two-dimensional array c^r with elements c_{ij}^r as the output.

The prediction residual transform coefficients c^r are scaled using quantisation parameter QP_Y, and added to the transform coefficients of the prediction block c^p with i, j = 0..3 as follows.

$$c_{ij}^s = c_{ij}^p + (((c_{ij}^r * \text{LevelScale}(QP_Y \,\%\, 6, i, j) * A_{ij}) << (QP_Y / 6)) >> 6) \tag{8-291}$$

where LevelScale(m, i, j) is specified in Equation 8-252, and where A_{ij} is specified as:

$$A_{ij} = \begin{cases} 16 & \text{for} \quad (i,j) \in \{(0,0),(0,2),(2,0),(2,2)\}, \\ 25 & \text{for} \quad (i,j) \in \{(1,1),(1,3),(3,1),(3,3)\}, \\ 20 & \text{otherwise;} \end{cases} \tag{8-292}$$

The function LevelScale2(m, i, j), used in the formulas below, is specified as:

$$\text{LevelScale2}(m,i,j) = \begin{cases} w_{m0} & \text{for} \quad (i,j) \in \{(0,0),(0,2),(2,0),(2,2)\}, \\ w_{m1} & \text{for} \quad (i,j) \in \{(1,1),(1,3),(3,1),(3,3)\}, \\ w_{m2} & \text{otherwise;} \end{cases} \qquad (8\text{-}293)$$

where the first and second subscripts of w are row and column indices, respectively, of the matrix specified as:

$$w = \begin{bmatrix} 13107 & 5243 & 8066 \\ 11916 & 4660 & 7490 \\ 10082 & 4194 & 6554 \\ 9362 & 3647 & 5825 \\ 8192 & 3355 & 5243 \\ 7282 & 2893 & 4559 \end{bmatrix} \qquad (8\text{-}294)$$

The resulting sum, c^s, is quantised with a quantisation parameter QS_Y and with i, j = 0..3 as follows.

$$c_{ij} = (\text{Sign}(c_{ij}^s) * (\text{Abs}(c_{ij}^s) * \text{LevelScale2}(QS_Y \% 6, i, j) + (1 << (14 + QS_Y / 6)))) >> (15 + QS_Y / 6) \qquad (8\text{-}295)$$

The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.8 is invoked with c as the input and r as the output.

The 4x4 array u with elements u_{ij} is derived as follows.

$$u_{ij} = \text{Clip1}(r_{ij}) \text{ with } i, j = 0..3 \qquad (8\text{-}296)$$

The picture construction process prior to deblocking filter process in subclause 8.5.9 is invoked with luma4x4BlkIdx, u as the input and S' as the output.

8.6.1.2 Chroma transform coefficient decoding process

Inputs to this process are Inter prediction chroma samples for the current macroblock from subclause 8.4 and the prediction residual transform coefficient levels, ChromaDCLevel and ChromaACLevel.

Outputs of this process are the decoded chroma samples of the current macroblock prior to the deblocking filter process.

This process is invoked twice: once for the Cb component and once for the Cr component. The component is referred to by replacing C with Cb for the Cb component and C with Cr for the Cr component. Let iCbCr select the current chroma component.

For each 4x4 block of the current chroma component indexed using chroma4x4BlkIdx with chroma4x4BlkIdx equal to 0..3, the following applies.

- The position of the upper-left sample of a 4x4 chroma block with index chroma4x4BlkIdx inside the macroblock is derived as follows.

  $$x = \text{InverseRasterScan}(\text{chroma4x4BlkIdx}, 4, 4, 8, 0) \qquad (8\text{-}297)$$

  $$y = \text{InverseRasterScan}(\text{chroma4x4BlkIdx}, 4, 4, 8, 1) \qquad (8\text{-}298)$$

- Let p be a 4x4 array of prediction samples with elements p_{ij} being derived as follows.

  $$p_{ij} = \text{pred}_C[x + j, y + i] \quad \text{with } i, j = 0..3 \qquad (8\text{-}299)$$

- The 4x4 array p is transformed producing transform coefficients c^p(chroma4x4BlkIdx) using Equation 8-290.

- The variable chromaList, which is a list of 16 entries, is derived. chromaList[0] is set equal to 0. chromaList[k] with index k = 1..15 are specified as follows.

  $$\text{chromaList}[k] = \text{ChromaACLevel}[\text{iCbCr}][\text{chroma4x4BlkIdx}][k - 1] \qquad (8\text{-}300)$$

- The inverse transform coefficient scanning process as described in subclause 8.5.4 is invoked with chromaList as the input and the 4x4 array c^r as the output.

- The prediction residual transform coefficients c^r are scaled using quantisation parameter QP_C, and added to the transform coefficients of the prediction block c^p with i, j = 0..3 except for the combination i = 0, j = 0 as follows.

$$c_{ij}^s = c_{ij}^p(\text{chroma4x4BlkIdx}) + (((c_{ij}^r * \text{LevelScale}(QP_C \,\%\, 6, i, j) * A_{ij}) << (QP_C / 6)) >> 6) \tag{8-301}$$

- The resulting sum, c^s, is quantised with a quantisation parameter QS_C and with i, j = 0..3 except for the combination i = 0, j = 0 as follows. The derivation of c_{00}(chroma4x4BlkIdx) is described below in this subclause.

$$c_{ij}(\text{chroma4x4BlkIdx}) = (\text{Sign}(c_{ij}^s) * (\text{Abs}(c_{ij}^s) * \text{LevelScale2}(QS_C \,\%\, 6, i, j) + (1 << (14 + QS_C / 6)))) >> (15 + QS_C / 6) \tag{8-302}$$

- The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.8 is invoked with c(chroma4x4BlkIdx) as the input and r as the output.

- The 4x4 array u with elements u_{ij} is derived as follows.

$$u_{ij} = \text{Clip1}(r_{ij}) \text{ with } i, j = 0..3 \tag{8-303}$$

- The picture construction process prior to deblocking filter process in subclause 8.5.9 is invoked with chroma4x4BlkIdx and u as the input and S' as the output.

The derivation of the DC transform coefficient level c_{00}(chroma4x4BlkIdx) is specified as follows. The DC transform coefficients of the 4 prediction chroma 4x4 blocks of the current component of the macroblock are assembled into a 2x2 matrix with elements c_{00}^p(chroma4x4BlkIdx) and a 2x2 transform is applied to the DC transform coefficients as follows

$$dc^p = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00}^p(0) & c_{00}^p(1) \\ c_{00}^p(2) & c_{00}^p(3) \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{8-304}$$

The chroma DC prediction residual transform coefficient levels, ChromaDCLevel[iCbCr][k] with k = 0..3 are scaled using quantisation parameter QP, and added to the prediction DC transform coefficients as follows.

$$dc_{ij}^s = dc_{ij}^p + (((\text{ChromaDCLevel}[\text{iCbCr}][j * 2 + i] * \text{LevelScale}(QP_C \,\%\, 6, 0, 0) * A_{00}) << (QP_C / 6)) >>5) \quad \text{with } i, j = 0, 1 \tag{8-305}$$

The 2x2 array dc^s, is quantised using the quantisation parameter QS_C as follows.

$$dc_{ij}^r = (\text{Sign}(dc_{ij}^s) * (\text{Abs}(dc_{ij}^s) * \text{LevelScale2}(QS_C \,\%\, 6, 0, 0) + (1 << (15 + QS_C / 6)))) >> (16 + QS_C / 6) \quad \text{with } i, j = 0, 1 \tag{8-306}$$

The 2x2 array f with elements f_{ij} and i, j = 0..1 is derived as follows.

$$f = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} dc_{00}^r & dc_{01}^r \\ dc_{10}^r & dc_{11}^r \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{8-307}$$

Scaling of the elements f_{ij} of f is performed as follows.

- If QS_C is greater than or equal to 6, the c_{00}( ) are derived by

$$c_{00}(j * 2 + i) = (f_{ij} * \text{LevelScale}(QS_C \,\%\, 6, 0, 0)) << (QS_C / 6 - 1) \quad \text{with } i, j = 0, 1 \tag{8-308}$$

- Otherwise (QS_C is less than 6), the c_{00}( ) are derived by

$$c_{00}(j * 2 + i) = (f_{ij} * \text{LevelScale}(QS_C \,\%\, 6, 0, 0)) >> 1 \quad \text{with } i, j = 0, 1 \tag{8-309}$$

8.6.2 SP and SI slice decoding process for switching pictures

This process is invoked, when decoding P macroblock types in SP slices in which sp_for_switch_flag is equal to 1 and when decoding SI macroblock type in SI slices.

Inputs to this process are the prediction residual transform coefficient levels and the prediction sample arrays $pred_L$, $pred_{Cb}$, $pred_{Cr}$ for the current macroblock.

Outputs of this process are the decoded samples of the current macroblock prior to the deblocking filter process.

8.6.2.1 Luma transform coefficient decoding process

Inputs to this process are prediction luma samples pred_L and the luma prediction residual transform coefficient levels, LumaLevel.

Outputs of this process are the decoded luma samples of the current macroblock prior to the deblocking filter process.

The 4x4 array p with elements p_{ij} with i, j = 0..3 is derived as in subclause 8.6.1.1, is transformed according to Equation 8-290 to produce transform coefficients c^p. These transform coefficients are then quantised with the quantisation parameter QS_Y, as follows:

$$c_{ij}^{s} = (\text{Sign}(c_{ij}^{p}) * (\text{Abs}(c_{ij}^{p}) * \text{LevelScale2}(QS_Y \% 6, i, j) + (1 << (14 + QS_Y / 6)))) >> (15 + QS_Y / 6) \quad \text{with } i, j = 0..3 \qquad (8\text{-}310)$$

The inverse transform coefficient scanning process as described in subclause 8.5.4 is invoked with LumaLevel[luma4x4BlkIdx] as the input and the two-dimensional array c^r with elements c_{ij}^r as the output.

The 4x4 array c with elements c_{ij} with i, j = 0..3 is derived as follows.

$$c_{ij} = c_{ij}^{r} + c_{ij}^{s} \quad \text{with } i, j = 0..3 \qquad (8\text{-}311)$$

The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.8 is invoked with c as the input and r as the output.

The 4x4 array u with elements u_{ij} is derived as follows.

$$u_{ij} = \text{Clip1}(r_{ij}) \text{ with } i, j = 0..3 \qquad (8\text{-}312)$$

The picture construction process prior to deblocking filter process in subclause 8.5.9 is invoked with luma4x4BlkIdx, u as the input and S' as the output.

8.6.2.2 Chroma transform coefficient decoding process

Inputs to this process are predicted chroma samples for the current macroblock from subclause 8.4 and the prediction residual transform coefficient levels, ChromaDCLevel and ChromaACLevel.

Outputs of this process are the decoded chroma samples of the current macroblock prior to the deblocking filter process.

This process is invoked twice: once for the Cb component and once for the Cr component. The component is referred to by replacing C with Cb for the Cb component and C with Cr for the Cr component. Let iCbCr select the current chroma component.

For each 4x4 block of the current chroma component indexed using chroma4x4BlkIdx with chroma4x4BlkIdx equal to 0..3, the following applies.

1. The 4x4 array p with elements p_{ij} with i, j = 0..3 is derived as in subclause 8.6.1.2, is transformed according to Equation 8-290 to produce transform coefficients c^p(chroma4x4BlkIdx). These transform coefficients are then quantised with the quantisation parameter QS_C, with i, j = 0..3 except for the combination i = 0, j = 0 as follows. The processing of c_{00}^p(chroma4x4BlkIdx) is described below in this subclause.

$$c_{ij}^{s} = (\text{Sign}(c_{ij}^{p}(\text{chroma4x4BlkIdx})) * (\text{Abs}(c_{ij}^{p}(\text{chroma4x4BlkIdx})) * \text{LevelScale2}(QS_C \% 6, i, j) + (1 << (14 + QS_C / 6)))) >> (15 + QS_C / 6) \qquad (8\text{-}313)$$

- The variable chromaList, which is a list of 16 entries, is derived. chromaList[0] is set equal to 0. chromaList[k] with index k = 1..15 are specified as follows.

$$\text{chromaList}[k] = \text{ChromaACLevel}[\text{iCbCr}][\text{chroma4x4BlkIdx}][k - 1] \qquad (8\text{-}314)$$

- The inverse transform coefficient scanning process as described in subclause 8.5.4 is invoked with chromaList as the input and the two-dimensional array c^r(chroma4x4BlkIdx) with elements c_{ij}^r(chroma4x4BlkIdx) as the output.

- The 4x4 array c(chroma4x4BlkIdx) with elements c_{ij}(chroma4x4BlkIdx) with i, j = 0..3 except for the combination i = 0, j = 0 is derived as follows. The derivation of c_{00}(chroma4x4BlkIdx) is described below.

$$c_{ij}(\text{chroma4x4BlkIdx}) = c_{ij}^{r}(\text{chroma4x4BlkIdx}) + c_{ij}^{s} \qquad (8\text{-}315)$$

- The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.8 is invoked with c(chroma4x4BlkIdx) as the input and r as the output.

- The 4x4 array u with elements u_{ij} is derived as follows.

$$u_{ij} = \text{Clip1}(r_{ij}) \text{ with } i, j = 0..3 \qquad (8\text{-}316)$$

- The picture construction process prior to deblocking filter process in subclause 8.5.9 is invoked with chroma4x4BlkIdx, u as the input and S' as the output.

The derivation of the DC transform coefficient level c_{00}(chroma4x4BlkIdx) is specified as follows. The DC transform coefficients of the 4 prediction 4x4 chroma blocks of the current component of the macroblock, $c_{00}{}^{p}$(chroma4x4BlkIdx), are assembled into a 2x2 matrix, and a 2x2 transform is applied to the DC transform coefficients of these blocks according to Equation 8-304 resulting in DC transform coefficients $dc_{ij}{}^{p}$.

These DC transform coefficients are then quantised with the quantisation parameter QS_C, as given by:

$$dc_{ij}{}^{s} = (\text{Sign}(dc_{ij}{}^{p}) * (\text{Abs}(dc_{ij}{}^{p}) * \text{LevelScale2}(QS_C \% 6, 0, 0) + (1 << (15 + QS_C / 6)))) >> (16 + QS_C / 6) \quad \text{with } i, j = 0, 1 \qquad (8\text{-}317)$$

The parsed chroma DC prediction residual transform coefficients, ChromaDCLevel[iCbCr][k] with k = 0..3 are added to these quantised DC transform coefficients of the prediction block, as given by:

$$dc_{ij}{}^{r} = dc_{ij}{}^{s} + \text{ChromaDCLevel}[\text{iCbCr}][j * 2 + i] \quad \text{with } i, j = 0, 1 \qquad (8\text{-}318)$$

The 2x2 array f with elements f_{ij} and i, j = 0..1 is derived using Equation 8-307.

The 2x2 array f with elements f_{ij} and i, j = 0..1 is copied as follows.

$$c_{00}(j * 2 + i) = f_{ij} \quad \text{with } i, j = 0, 1 \qquad (8\text{-}319)$$

8.7 Deblocking filter process

A conditional filtering shall be 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 and 8.6) 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 or macroblock pair above (if any) and the macroblock or macroblock pair 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.

The deblocking filter process is invoked for the luma and chroma components separately. For each macroblock, vertical edges are filtered first, from left to right, and then horizontal edges are filtered from top to bottom. The luma deblocking filter process is performed on four 16-sample edges and the deblocking filter process for each chroma components is performed on two 8-sample edges, for the horizontal direction as shown on the left side of Figure 8-9 and for the vertical direction as shown on the right side of Figure 8-9. Sample values above and to the left of the current macroblock that may have already been modified by the deblocking filter process operation on previous macroblocks shall be used as input to the deblocking filter process on the current macroblock and may be further modified during the filtering of the current macroblock. Sample values modified during filtering of vertical edges are used as input for the filtering of the horizontal edges for the same macroblock.

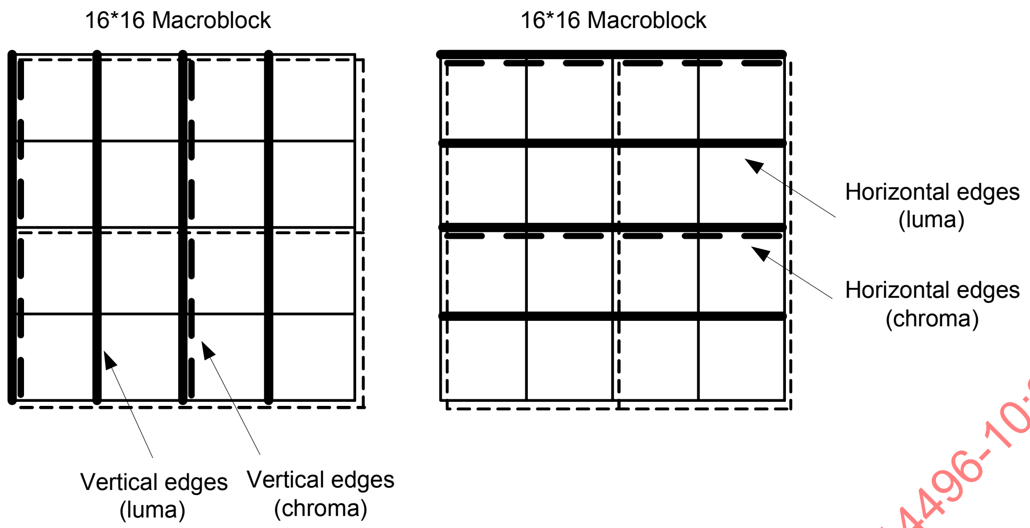


Figure 8-9 – Boundaries in a macroblock to be filtered (luma boundaries shown with solid lines and chroma boundaries shown with dashed lines)

For each macroblock in ascending order of mbAddr, the following applies.

1. The variables fieldModeMbFlag, filterInternalEdgesFlag, filterLeftMbEdgeFlag and filterTopMbEdgeFlag are derived as follows.
   - The variable fieldModeMbFlag is derived as follows.
     - If any of the following conditions is true, fieldModeMbFlag is set equal to 1.
       - field_pic_flag is equal to 1
       - MbaffFrameFlag is equal 1 and the macroblock mbAddr is a field macroblock
     - Otherwise, fieldModeMbFlag is set equal to 0.
   - The variable filterInternalEdgesFlag is derived as follows.
     - If disable_deblocking_filter_idc for the slice that contains the macroblock mbAddr is equal to 1, the variable filterInternalEdgesFlag is set equal to 0;
     - Otherwise (disable_deblocking_filter_idc for the slice that contains the macroblock mbAddr is not equal to 1), the variable filterInternalEdgesFlag is set equal to 1.
   - The variable filterLeftMbEdgeFlag is derived as follows.
     - If any of the following conditions is true, the variable filterLeftMbEdgeFlag is set equal to 0.
       - the left vertical macroblock edge of the macroblock mbAddr represents a picture boundary
       - disable_deblocking_filter_idc for the slice that contains the macroblock mbAddr is equal to 1
       - disable_deblocking_filter_idc for the slice that contains the macroblock mbAddr is equal to 2 and the left vertical macroblock edge of the macroblock mbAddr represents a slice boundary
     - Otherwise, the variable filterLeftMbEdgeFlag is set equal to 1.
   - The variable filterTopMbEdgeFlag is derived as follows.
     - If any of the following conditions is true, the variable filterTopMbEdgeFlag is set equal to 0.
       - the top horizontal macroblock edge of the macroblock mbAddr represents a picture boundary
       - disable_deblocking_filter_idc for the slice that contains the macroblock mbAddr is equal to 1

    - disable_deblocking_filter_idc for the slice that contains the macroblock mbAddr is equal to 2 and the top horizontal macroblock edge of the macroblock mbAddr represents a slice boundary

  - Otherwise, the variable filterTopMbEdgeFlag is set equal to 1.

2. Given the variables fieldModeMbFlag, filterInternalEdgesFlag, filterLeftMbEdgeFlag and filterTopMbEdgeFlag the deblocking filtering is controlled as follows.

   - When filterLeftMbEdgeFlag is equal to 1, the filtering of the left vertical luma edge is specified as follows.

     - The process specified in subclause 8.7.1 is invoked with mbAddr, chromaEdgeFlag = 0, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, and (xE_k, yE_k) = (0, k) with k = 0..15 as input and S'_L as output.

   - When filterInternalEdgesFlag is equal to 1, the filtering of the internal vertical luma edges is specified as follows.

     - The process specified in subclause 8.7.1 is invoked with mbAddr, chromaEdgeFlag = 0, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, and (xE_k, yE_k) = (4, k) with k = 0..15 as input and S'_L as output.

     - The process specified in subclause 8.7.1 is invoked with mbAddr, chromaEdgeFlag = 0, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, and (xE_k, yE_k) = (8, k) with k = 0..15 as input and S'_L as output.

     - The process specified in subclause 8.7.1 is invoked with mbAddr, chromaEdgeFlag = 0, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, and (xE_k, yE_k) = (12, k) with k = 0..15 as input and S'_L as output.

   - When filterTopMbEdgeFlag is equal to 1, the filtering of the top horizontal luma edge is specified as follows.

     - If MbaffFrameFlag is equal to 1, (mbAddr % 2) is equal to 0, mbAddr is greater than or equal to 2 * PicWidthInMbs, the macroblock mbAddr is a frame macroblock, and the macroblock (mbAddr - 2 * PicWidthInMbs + 1) is a field macroblock, the following applies.

       - The process specified in subclause 8.7.1 is invoked with mbAddr, chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = 1, and (xE_k, yE_k) = (k, 0) with k = 0..15 as input and S'_L as output.

       - The process specified in subclause 8.7.1 is invoked with mbAddr, chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = 1, and (xE_k, yE_k) = (k, 1) with k = 0..15 as input and S'_L as output.

     - Otherwise, the process specified in subclause 8.7.1 is invoked with mbAddr, chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, and (xE_k, yE_k) = (k, 0) with k = 0..15 as input and S'_L as output.

   - When filterInternalEdgesFlag is equal to 1, the filtering of the internal horizontal luma edges is specified as follows.

     - The process specified in subclause 8.7.1 is invoked with mbAddr, chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, and (xE_k, yE_k) = (k, 4) with k = 0..15 as input and S'_L as output.

     - The process specified in subclause 8.7.1 is invoked with mbAddr, chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, and (xE_k, yE_k) = (k, 8) with k = 0..15 as input and S'_L as output.

     - The process specified in subclause 8.7.1 is invoked with mbAddr, chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, and (xE_k, yE_k) = (k, 12) with k = 0..15 as input and S'_L as output.

   - For both chroma components iCbCr = 0 and 1, the following applies.

     - When filterLeftMbEdgeFlag is equal to 1, the filtering of the left vertical chroma edge is specified as follows.

       - The process specified in subclause 8.7.1 is invoked with mbAddr, chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 1, fieldModeFilteringFlag = 1, and (xE_k, yE_k) = (0, k) with k = 0..7 as input and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as output.

     - When filterInternalEdgesFlag is equal to 1, the filtering of the internal vertical chroma edge is specified as follows.

       - The process specified in subclause 8.7.1 is invoked with mbAddr, chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, and (xE_k, yE_k) = (4, k) with k = 0..7 as input and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as output.

- When filterTopMbEdgeFlag is equal to 1, the filtering of the top horizontal chroma edge is specified as follows.
  - If MbaffFrameFlag is equal to 1, (mbAddr % 2) is equal to 0, mbAddr is greater than or equal to 2 * PicWidthInMbs, the macroblock mbAddr is a frame macroblock, and the macroblock (mbAddr − 2 * PicWidthInMbs + 1) is a field macroblock, the following applies.
    - The process specified in subclause 8.7.1 is invoked with mbAddr, chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeFilteringFlag = 1, and (xE_k, yE_k) = (k, 0) with k = 0..7 as input and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as output.
    - The process specified in subclause 8.7.1 is invoked with mbAddr, chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeFilteringFlag = 1, and (xE_k, yE_k) = (k, 1) with k = 0..7 as input and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as output.
  - Otherwise, the process specified in subclause 8.7.1 is invoked with mbAddr, chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, and (xE_k, yE_k) = (k, 0) with k = 0..7 as input and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as output.
- When filterInternalEdgesFlag is equal to 1, the filtering of the internal horizontal chroma edge is specified as follows.
  - The process specified in subclause 8.7.1 is invoked with mbAddr, chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, and (xE_k, yE_k) = (k, 4) with k = 0..7 as input and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as output.

    NOTE - When field mode filtering (fieldModeFilteringFlag is equal to 1) is applied across the top horizontal edges of a frame macroblock, this vertical filtering across the top or bottom macroblock boundary may involve some samples that extend across an internal block edge that is also filtered internally in frame mode.

    NOTE – In all cases, 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. When field mode filtering (fieldModeFilteringFlag is equal to 1) is applied to the top edges of a frame macroblock, 2 horizontal luma, 2 horizontal chroma edges for Cb, and 2 horizontal chroma edges for Cr between the frame macroblock and the above macroblock pair are filtered using field mode filtering, for a total of up to 5 horizontal luma edges, 3 horizontal chroma edges for Cb, and 3 horizontal chroma edges for Cr filtered that are considered to be controlled by the frame macroblock. In all other cases, at most 4 horizontal luma, 2 horizontal chroma edges for Cb, and 2 horizontal chroma edges for Cr are filtered that are considered to be controlled by a particular macroblock.

Finally, 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

Input to this process are mbAddr, chromaEdgeFlag, the chroma component index iCbCr (when chromaEdgeFlag is equal to 1), verticalEdgeFlag, fieldModeFilteringFlag, and a set of sixteen luma (when chromaEdgeFlag is equal to 0) or eight chroma (when chromaEdgeFlag is equal to 1) sample locations (xE_k, yE_k), with k = 0 .. nE - 1, expressed relative to the upper left corner of the macroblock mbAddr. 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 16;
- Otherwise (chromaEdgeFlag is equal to 1), nE is 8.

Let s' be a variable specifying a luma or chroma sample array, be 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 variable dy is derived as follows.

- If fieldModeFilteringFlag is equal to 1 and MbaffFrameFlag is equal to 1, dy is set equal to 2.
- Otherwise (fieldModeFilteringFlag is equal to 0 or MbaffFrameFlag is equal to 0), dy is set equal to 1.

The position of the upper-left luma sample of the macroblock mbAddr is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddr as input and the output being assigned to (xP, yP).

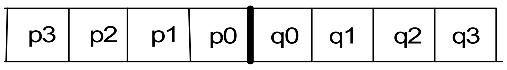


Figure 8-10 – Convention for describing samples across a 4x4 block horizontal or vertical boundary

For each sample location (xE_k, yE_k), k = 0 .. nE - 1, the following applies.

- 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 8-10 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,

    $q_i = s'[\ xP + xE_k + i,\ yP + yE_k\ ]$ (8-320)

    $p_i = s'[\ xP + xE_k - i - 1,\ yP + yE_k\ ]$ (8-321)

  - Otherwise (verticalEdgeFlag is equal to 0),

    $q_i = s'[\ xP + xE_k,\ yP + dy * (\ yE_k + i\ ) - (yE_k \% 2\ )\ ]$ (8-322)

    $p_i = s'[\ xP + xE_k,\ yP + dy * (\ yE_k - i - 1\ ) - (yE_k \% 2\ )\ ]$ (8-323)

- The process specified in subclause 8.7.2 is invoked with the sample values p_i and q_i (i = 0..3), chromaEdgeFlag, verticalEdgeFlag, and fieldModeFilteringFlag as input, and the output is assigned to the filtered results sample values p'_i and q'_i with i = 0..2.
- 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,

    $s'[\ xP + xE_k + i,\ yP + yE_k\ ] = q'_i$ (8-324)

    $s'[\ xP + xE_k - i - 1,\ yP + yE_k\ ] = p'_i$ (8-325)

  - Otherwise (verticalEdgeFlag is equal to 0),

    $s'[\ xP + xE_k,\ yP + dy * (\ yE_k + i\ ) - (\ yE_k \% 2\ )\ ] = q'_i$ (8-326)

    $s'[\ xP + xE_k,\ yP + dy * (\ yE_k - i - 1\ ) - (\ yE_k \% 2\ )\ ] = p'_i$ (8-327)

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, verticalEdgeFlag, and fieldModeFilteringFlag.

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 following applies.
  - If fieldModeFilteringFlag is equal to 0, the bS used for filtering a set of samples of a horizontal or vertical chroma edge shall be 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.
  - Otherwise (fieldModeFilteringFlag is equal to 1), the bS used for filtering a set of samples of a horizontal or vertical chroma edge shall be 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 same field, where (x, y) is the location of the chroma sample q_0 inside the chroma array for that field.

The process specified in subclause 8.7.2.2 is invoked with p_0, q_0, p_1, q_1, chromaEdgeFlag, and bS as input, and the output is 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..3), 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:

for i = 0..2, $p'_i = p_i$ (8-328)

for i = 0..2, $q'_i = q_i$ (8-329)

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.

Let the variable mixedModeEdgeFlag be derived as follows.

- If MbaffFrameFlag is equal to 1 and the samples p_0 and q_0 are in different macroblock pairs, one of which is a field macroblock pair and the other is a frame macroblock pair, mixedModeEdgeFlag is set equal to 1
- Otherwise, mixedModeEdgeFlag is set equal to 0.

The variable bS is derived as follows.

- If the block edge is also a macroblock edge and any of the following conditions are true, a value of bS equal to 4 shall be the output:
  - the samples p_0 and q_0 are both in frame macroblocks and either or both of the samples p_0 or q_0 is in a macroblock coded using an Intra macroblock prediction mode
  - the samples p_0 and q_0 are both in frame macroblocks and either or both of the samples p_0 or q_0 is in a macroblock that is in a slice with slice_type equal to SP or SI
  - MbaffFrameFlag is equal to 1 or field_pic_flag is equal to 1, and verticalEdgeFlag is equal to 1, and either or both of the samples p_0 or q_0 is in a macroblock coded using an Intra macroblock prediction mode.
  - MbaffFrameFlag is equal to 1 or field_pic_flag is equal to 1, and verticalEdgeFlag is equal to 1, and either or both of the samples p_0 or q_0 is in a macroblock that is in a slice with slice_type equal to SP or SI
- Otherwise, if any of the following conditions are true, a value of bS equal to 3 shall be the output:
  - mixedModeEdgeFlag is equal to 0 and either or both of the samples p_0 or q_0 is in a macroblock coded using an Intra macroblock prediction mode
  - mixedModeEdgeFlag is equal to 0 and either or both of the samples p_0 or q_0 is in a macroblock that is in a slice with slice_type equal to SP or SI
  - mixedModeEdgeFlag is equal to 1, verticalEdgeFlag is equal to 0, and either or both of the samples p_0 or q_0 is in a macroblock coded using an Intra macroblock prediction mode
  - mixedModeEdgeFlag is equal to 1, verticalEdgeFlag is equal to 0, and either or both of the samples p_0 or q_0 is in a macroblock that is in a slice with slice_type equal to SP or SI
- Otherwise, if the following condition is true, a value of bS equal to 2 shall be the output:

  - the 4x4 luma block containing sample p_0 or the 4x4 luma block containing 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 shall be the output:
  - mixedModeEdgeFlag is equal to 1
  - mixedModeEdgeFlag is equal to 0 and for the prediction of the macroblock/sub-macroblock partition containing the sample p_0 different reference pictures or a different number of motion vectors 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 a prediction is formed using an index into list 0 or an index into list 1, and also without regard to whether or not the index position within a reference picture list is different or not.
  - mixedModeEdgeFlag is equal to 0 and 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 component of the motion vectors used is greater than or equal to 4 in units of quarter luma frame samples.
  - mixedModeEdgeFlag is equal to 0 and two motion vectors and two different reference pictures are used to predict the macroblock/sub-macroblock partition containing the sample p_0 and two motion vectors for the same two reference pictures are used to predict the macroblock/sub-macroblock partition containing the sample q_0 and the absolute difference between the horizontal or vertical component of the two motion vectors used in the prediction of the two macroblock/sub-macroblock partitions for the same reference picture is greater than or equal to 4 in units of quarter luma frame samples.
  - mixedModeEdgeFlag is equal to 0 and two motion vectors for the same reference picture are used to predict the macroblock/sub-macroblock partition containing the sample p_0 and two motion vectors for the same reference picture as used to predict the macroblock/sub-macroblock partition containing the sample p_0 are used to predict the macroblock/sub-macroblock partition containing the sample q_0 and both of the following conditions are true:
    - The absolute difference between the horizontal or vertical component of list 0 motion vectors used in the prediction of the two macroblock/sub-macroblock partitions is greater than or equal to 4 in quarter luma frame samples or the absolute difference between the horizontal or vertical component of the list 1 motion vectors used in the prediction of the two macroblock/sub-macroblock partitions is greater than or equal to 4 in units of quarter luma frame samples.
    - The absolute difference between the horizontal or vertical component of list 0 motion vector used in the prediction of the macroblock/sub-macroblock partition containing the sample p_0 and the list 1 motion vector used in the prediction of the macroblock/sub-macroblock partition containing the sample q_0 is greater than or equal to 4 in units of quarter luma frame samples or the absolute difference between the horizontal or vertical component of the list 1 motion vector used in the prediction of the macroblock/sub-macroblock partition containing the sample p_0 and list 0 motion vector used in the prediction of the macroblock/sub-macroblock partition containing the sample q_0 is greater than or equal to 4 in units of quarter luma frame samples.

      NOTE – A vertical difference of 4 in units of quarter luma frame samples is a difference of 2 in units of quarter luma field samples

- Otherwise, a value of bS equal to 0 shall be 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, chromaEdgeFlag, and bS, for the set of input samples, as specified in 8.7.2.

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_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 to the value of QP_C that corresponds to a value of 0 for QP_Y as specified in subclause 8.5.5.
  - Otherwise (the macroblock containing the sample z_0 is not an I_PCM macroblock), qP_z is set 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 8.5.5.

Let qPav be a variable specifying an average quantisation parameter. It is derived as follows.

$$qPav = (qP_p + qP_q + 1) >> 1 \qquad (8\text{-}330)$$

NOTE - In SP and SI slices, qP_{av} is derived in the same way as in other slice types. QS_Y from Equation 7-17 is not used in the deblocking filter.

Let indexA be a variable that is used to access the α table (Table 8-14) as well as the t_{C0} table (Table 8-15), 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 8-14). The variables indexA and indexB are derived as follows, where the values of FilterOffsetA and FilterOffsetB are the values of those variables specified in subclause 7.4.3 for the slice that contains the macroblock containing sample q_0..

$$\text{indexA} = \text{Clip3}(0, 51, qP_{av} + \text{FilterOffsetA}) \qquad (8\text{-}331)$$

$$\text{indexB} = \text{Clip3}(0, 51, qP_{av} + \text{FilterOffsetB}) \qquad (8\text{-}332)$$

The threshold variables α and β are specified in Table 8-14 depending on the values of indexA and indexB.

The variable filterSamplesFlag is derived by

$$\text{filterSamplesFlag} = (bS \mathrel{!=} 0 \ \&\& \ \text{Abs}(p_0 - q_0) < \alpha \ \&\& \ \text{Abs}(p_1 - p_0) < \beta \ \&\& \ \text{Abs}(q_1 - q_0) < \beta) \qquad (8\text{-}333)$$

Table 8-14 – Derivation of indexA and indexB from offset dependent threshold variables α and β

	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 8-14 (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 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.

The filtered result samples p'_0 and q'_0 are derived by

$$\Delta = \text{Clip3}(-t_C, t_C, ((((q_0 - p_0) << 2) + (p_1 - q_1) + 4) >> 3)) \qquad (8\text{-}334)$$

$p'_0 = \mathrm{Clip1}(\ p_0 + \Delta\ )$ (8-335)

$q'_0 = \mathrm{Clip1}(\ q_0 - \Delta\ )$ (8-336)

where the threshold t_C is determined as follows.

- If chromaEdgeFlag is equal to 0,

  $t_C = t_{C0} + (\ (\ a_p < \beta\ )\ ?\ 1 : 0\ ) + (\ (\ a_q < \beta\ )\ ?\ 1 : 0\ )$ (8-337)

- Otherwise (chromaEdgeFlag is equal to 1),

  $t_C = t_{C0} + 1$ (8-338)

The threshold t_{C0} is specified in Table 8-15 depending on the values of indexA and bS.

Let a_p and a_q be two threshold variables specified by

$a_p = \mathrm{Abs}(\ p_2 - p_0\ )$ (8-339)

$a_q = \mathrm{Abs}(\ q_2 - q_0\ )$ (8-340)

The filtered result sample p'_1 is derived as follows

- If chromaEdgeFlag is equal to 0 and a_p is less than β,

  $p'_1 = p_1 + \mathrm{Clip3}(\ -t_{C0}, t_{C0}, (\ p_2 + (\ (\ p_0 + q_0 + 1\ ) >> 1\ ) - (\ p_1 << 1\ )\ ) >> 1\ )$ (8-341)

- Otherwise (chromaEdgeFlag is equal to 1 or a_p is greater than or equal to β),

  $p'_1 = p_1$ (8-342)

The filtered result sample q'_1 is derived as follows

- If chromaEdgeFlag is equal to 0 and a_q is less than β,

  $q'_1 = q_1 + \mathrm{Clip3}(\ -t_{C0}, t_{C0}, (\ q_2 + (\ (\ p_0 + q_0 + 1\ ) >> 1\ ) - (\ q_1 << 1\ )\ ) >> 1\ )$ (8-343)

- Otherwise (chromaEdgeFlag is equal to 1 or a_q is greater than or equal to β),

  $q'_1 = q_1$ (8-344)

The filtered result samples p'_2 and q'_2 are always set equal to the input samples p_2 and q_2:

$p'_2 = p_2$ (8-345)

$q'_2 = q_2$ (8-346)

Table 8-15 – Value of filter clipping 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 8-15 (concluded) – Value of filter clipping 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

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, the variable 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 8-339 and 8-340, 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,

$$a_p < \beta \ \ \&\& \ \ \mathrm{Abs}(\ p_0 - q_0\ ) < (\ (\ \alpha >> 2\ ) + 2\ ) \qquad (8\text{-}347)$$

  then the variables p'_0, p'_1, and p'_2 are derived by

$$p'_0 = (\ p_2 + 2{*}p_1 + 2{*}p_0 + 2{*}q_0 + q_1 + 4\ ) >> 3 \qquad (8\text{-}348)$$

$$p'_1 = (\ p_2 + p_1 + p_0 + q_0 + 2\ ) >> 2 \qquad (8\text{-}349)$$

$$p'_2 = (\ 2{*}p_3 + 3{*}p_2 + p_1 + p_0 + q_0 + 4\ ) >> 3 \qquad (8\text{-}350)$$

- Otherwise (chromaEdgeFlag is equal to 1 or the condition in Equation 8-347 does not hold), the variables p'_0, p'_1, and p'_2 are derived by

$$p'_0 = (\ 2{*}p_1 + p_0 + q_1 + 2\ ) >> 2 \qquad (8\text{-}351)$$

$$p'_1 = p_1 \qquad (8\text{-}352)$$

$$p'_2 = p_2 \qquad (8\text{-}353)$$

The filtered result samples q'_i ($i = 0..2$) are derived as follows.

- If chromaEdgeFlag is equal to 0 and the following condition holds,

$$a_q < \beta \ \ \&\& \ \ \mathrm{Abs}(\ p_0 - q_0\ ) < (\ (\ \alpha >> 2\ ) + 2\ ) \qquad (8\text{-}354)$$

  then the variables q'_0, q'_1, and q'_2 are derived by

$$q'_0 = (\ p_1 + 2{*}p_0 + 2{*}q_0 + 2{*}q_1 + q_2 + 4\ ) >> 3 \qquad (8\text{-}355)$$

$$q'_1 = (\ p_0 + q_0 + q_1 + q_2 + 2\ ) >> 2 \qquad (8\text{-}356)$$

$$q'_2 = (\ 2{*}q_3 + 3{*}q_2 + q_1 + q_0 + p_0 + 4\ ) >> 3 \qquad (8\text{-}357)$$

- Otherwise (chromaEdgeFlag is equal to 1 or the condition in Equation 8-354 does not hold), the variables q'_0, q'_1, and q'_2 are derived by

$$q'_0 = (\ 2{*}q_1 + q_0 + p_1 + 2\ ) >> 2 \qquad (8\text{-}358)$$

$$q'_1 = q_1 \qquad (8\text{-}359)$$

$$q'_2 = q_2 \qquad (8\text{-}360)$$

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), or ae(v) (see subclause 9.3).

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, this process is invoked only when entropy_coding_mode_flag is equal to 0.

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 shall be equivalent to the following:

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

The variable codeNum is then assigned as follows:

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

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 the syntax element shall be determined first. The range of this syntax element may be between 0 and x, with x being greater than or equal to 1 and is used in the derivation of the value of a syntax element as follows
  - If x is greater than 1, codeNum and the value of the syntax element shall be 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 )
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
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
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 when parsing syntax elements with descriptor equal to ce(v) in subclause 7.3.5.3.1 and when entropy_coding_mode_flag is equal to 0.

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 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.

The process is specified in the following ordered steps:

1. All transform coefficient levels, with indices 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 (see subclause 9.2.1) as follows.

   - If the number of non-zero transform coefficient levels TotalCoeff(coeff_token) is equal to 0, the list coeffLevel containing 0 values is returned and no further step is 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 (see 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 (see subclause 9.2.3).

     c. The level and run information are combined into the list coeffLevel (see subclause 9.2.4).

9.2.1 Parsing process for total number of transform coefficient levels and 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 of the current block of transform.

Outputs of this process are TotalCoeff(coeff_token) and TrailingOnes(coeff_token).

The syntax element coeff_token is decoded using one of the five VLCs specified in five right-most columns of Table 9-5. Each VLC specifies both TotalCoeff(coeff_token) and TrailingOnes(coeff_token) for a given codeword coeff_token. VLC selection is dependent upon a variable nC that is derived as follows.

- If the CAVLC parsing process is invoked for ChromaDCLevel, nC is set equal to –1,

- Otherwise, the following applies.
  - When the CAVLC parsing process is invoked for Intra16x16DCLevel, luma4x4BlkIdx is set equal to 0.
  - The variables blkA and blkB are derived as follows.
    - If the CAVLC parsing process is invoked for Intra16x16DCLevel, Intra16x16ACLevel, or LumaLevel, the process specified in subclause 6.4.7.3 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 to blkA, 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 6.4.7.4 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 to blkA, and the 4x4 chroma block specified by mbAddrB\iCbCr\chroma4x4BlkIdxB is assigned to blkB.
  - Let nA and nB be the number of non-zero transform coefficient levels (given by TotalCoeff(coeff_token)) in the block of transform coefficient levels blkA located to the left of the current block and the block of transform coefficient levels blkB located above the current block, respectively.
  - With N replaced by A and B, in mbAddrN, blkN, and nN the following applies.
    - If any of the following conditions is true, nN is set equal to 0.
      - mbAddrN is not available
      - The current macroblock is coded using an Intra prediction mode, constrained_intra_pred_flag is equal to 1 and mbAddrN is coded using Inter prediction and slice data partitioning is in use (nal_unit_type is in the range of 2 to 4, inclusive).
      - The macroblock mbAddrN has mb_type equal to P_Skip or B_Skip
      - 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, or is a chroma block, nA and nB is the number of decoded non-zero AC transform coefficient levels.

      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.
  - Given the values of nA and nB, the variable nC is derived as follows.
    - If both mbAddrA and mbAddrB are available, the variable nC is set equal to (nA + nB + 1) >> 1.
    - Otherwise (mbAddrA is not available or mbAddrB is not available), the variable nC is set equal to nA + nB.

The value of TotalCoeff(coeff_token) resulting from decoding coeff_token shall be in the range of 0 to maxNumCoeff, inclusive.

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	-
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	-
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 level containing transform coefficient levels.

Initially an index i is set equal to 0. Then the following procedure is iteratively applied TrailingOnes(coeff_token) times to decode the trailing one transform coefficient levels (if any):

- A 1-bit syntax element trailing_ones_sign_flag is decoded and evaluated as follows.
  - If trailing_ones_sign_flag is equal to 0, the value +1 is assigned to level[i].
  - Otherwise (trailing_ones_sign_flag is equal to 1), the value -1 is assigned to level[i].
- The index i is incremented by 1.

Following the decoding of the trailing one transform coefficient levels, a 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.

The following procedure is then applied iteratively (TotalCoeff(coeff_token) – TrailingOnes(coeff_token)) times to decode the remaining levels (if any):

- The syntax element level_prefix is decoded using the VLC specified in Table 9-6.
- The variable levelSuffixSize is set equal to the variable suffixLength with the exception of the following two cases.
- When level_prefix is equal to 14 and suffixLength is equal to 0, levelSuffixSize is set equal to 4.

- When level_prefix is equal to 15, levelSuffixSize is set equal to 12.
- 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 shall be inferred to be equal to 0.
- A variable levelCode is set equal to (level_prefix << suffixLength) + level_suffix.
- When level_prefix is equal to 15 and suffixLength is equal to 0, levelCode is incremented by 15.
- When the index i is equal to TrailingOnes(coeff_token) and TrailingOnes(coeff_token) is smaller than 3, levelCode is incremented by 2.
- The variable level[i] is derived as follows.
  - If levelCode is an even number, the value (levelCode + 2) >> 1 is assigned to level[i].
  - Otherwise, the value (-levelCode – 1) >> 1 is assigned to level[i].
- When suffixLength is equal to 0, suffixLength is set equal to 1.
- When the absolute value of level[i] is greater than (3 << (suffixLength – 1)) and suffixLength is less than 6, suffixLength is incremented by 1.
- The index i is incremented by 1.

Table 9-6 – Codeword table for level_prefix

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 run.

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 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 Table 9-7 and Table 9-8 are used.

The following procedure is then applied iteratively (TotalCoeff(coeff_token) – 1) times:

- The variable run[i] is derived as follows.
    - If zerosLeft is greater than zero, a value run_before is decoded based on Table 9-10 and zerosLeft. run[i] is set equal to run_before.
    - Otherwise (zerosLeft is equal to 0), run[i] is set equal to 0.
- The value of run[i] is subtracted from zerosLeft and the result assigned to zerosLeft. The result of the subtraction shall be greater than or equal to 0.
- The index i is incremented by 1.

Finally the value of zerosLeft is assigned to run[i].

Table 9-7 – total_zeros tables for 4x4 blocks with TotalCoeff(coeff_token) 1 to 7

total_zeros	**TotalCoeff(coeff_token)**						
	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 TotalCoeff(coeff_token) 8 to 15

total_zeros	TotalCoeff(coeff_token)							
	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 blocks

total_zeros	TotalCoeff(coeff_token)		
	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 level, a list of runs called run, 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 procedure is iteratively applied TotalCoeff(coeff_token) times:

- coeffNum is incremented by run[i] + 1.
- coeffLevel[coeffNum] is set equal to level[i].
- The index i is decremented by 1.

9.3 CABAC parsing process for slice data

This process is invoked when parsing syntax elements with descriptor ae(v) in subclauses 7.3.4 and 7.3.5 when entropy_coding_mode_flag is equal to 1.

Inputs to this process are a request for a value of a syntax element and values of prior parsed syntax elements.

Output of this process is the value of the syntax element.

When starting the parsing of the slice data of a slice in subclause 7.3.4, the initialisation process of the CABAC parsing process is invoked as specified in subclause 9.3.1.

The parsing of syntax elements proceeds as follows:

For each requested value of a syntax element a binarization is derived as described in subclause 9.3.2.

The binarization for the syntax element and the sequence of parsed bins determines the decoding process flow as described in subclause 9.3.3.

For each bin of the binarization of the syntax element, which is indexed by the variable binIdx, a context index ctxIdx is derived as specified in subclause 9.3.3.1.

For each ctxIdx the arithmetic decoding process is invoked as specified in subclause 9.3.3.2.

The resulting sequence (b_0 .. b_{binIdx}) of parsed bins is compared to the set of bin strings given by the binarization process after decoding of each bin. When the sequence matches a bin string in the given set, the corresponding value shall be assigned to the syntax element.

In case the request for a value of a syntax element is processed for the syntax element mb_type and the decoded value of mb_type is I_PCM, the decoding engine shall be initialised after the decoding of the pcm_alignment_zero_bit and all pcm_byte data as specified in subclause 9.3.1.2.

The whole CABAC parsing process is illustrated in the flowchart of Figure 9-1 with the abbreviation SE for syntax element.

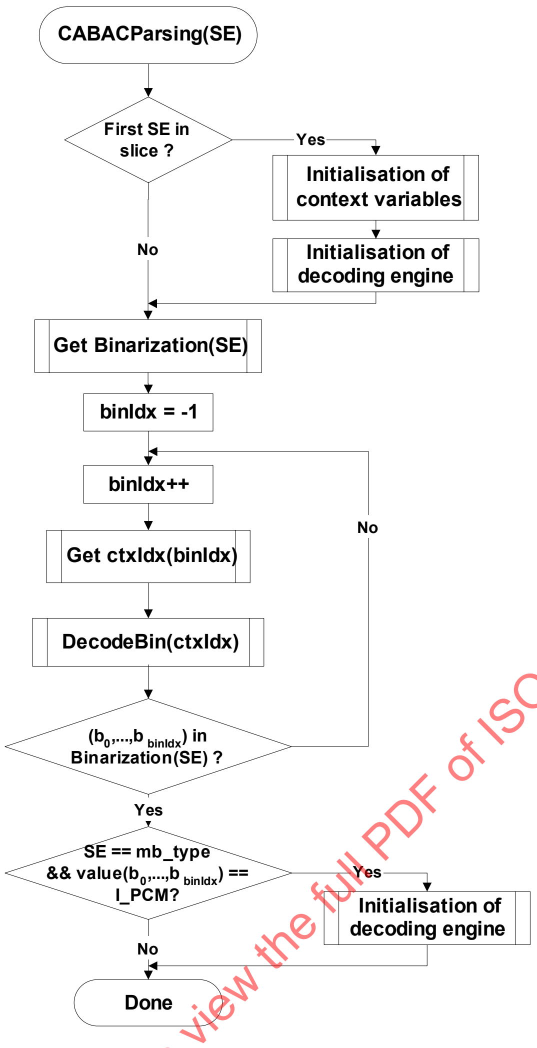

CABACParsing(SE)
First SE in slice ?
Yes
Initialisation of context variables
Initialisation of decoding engine
No
Get Binarization(SE)
binIdx = -1
binIdx++
No
Get ctxIdx(binIdx)
DecodeBin(ctxIdx)
$(b_0,\ldots,b_{\text{binIdx}})$ in Binarization(SE) ?
Yes
SE == mb_type && value$(b_0,\ldots,b_{\text{binIdx}})$ == I_PCM?
Yes
Initialisation of decoding engine
No
Done

Figure 9-1 – Illustration of CABAC parsing process for a syntax element SE (informative)

9.3.1 Initialisation process

Outputs of this process are initialised CABAC internal variables.

The processes in subclauses 9.3.1.1 and 9.3.1.2 are invoked when starting the parsing of the slice data of a slice in subclause 7.3.4.

The process in subclause 9.3.1.2 is also invoked after decoding the pcm_alignment_zero_bit and all pcm_byte data for a macroblock of type I_PCM.

9.3.1.1 Initialisation process for context variables

Outputs of this process are the initialised CABAC context variables indexed by ctxIdx.

Table 9-12 to Table 9-23 contain the values of the variables n and m used in the initialisation of context variables that are assigned to all syntax elements in subclauses 7.3.4 and 7.3.5 except for the end-of-slice flag.

For each context variable, the two variables pStateIdx and valMPS are initialised.

NOTE - The variable pStateIdx corresponds to a probability state index and the variable valMPS corresponds to the value of the most probable symbol as further described in subclause 9.3.3.2.

The two values assigned to pStateIdx and valMPS for the initialisation are derived from SliceQP_Y, which is derived in Equation 7-16. Given the two table entries (m, n),

1. $\text{preCtxState} = \text{Clip3}(1, 126, ((m * \text{SliceQP}_Y) >> 4) + n)$

```
2.  if( preCtxState  <=  63 ) {
        pStateIdx = 63 - preCtxState
        valMPS = 0
    } else {
        pStateIdx = preCtxState - 64
        valMPS = 1
    }
```

In Table 9-11, the ctxIdx for which initialisation is needed for each of the slice types are listed. Also listed is the table number that includes the values of m and n needed for the initialisation. For P, SP and B slice type, the initialisation depends also on the value of the cabac_init_idc syntax element. Note that the syntax element names do not affect the initialisation process.

Table 9-11 – Association of ctxIdx and syntax elements for each slice type in the initialisation process

	Syntax element	Table	Slice type			
			SI	I	P, SP	B
slice_data( )	mb_skip_flag	Table 9-13 Table 9-14			11-13	24-26
	mb_field_decoding_flag	Table 9-18	70-72	70-72	70-72	70-72
macroblock_layer( )	mb_type	Table 9-12, Table 9-13, Table 9-14.	0-10	3-10	14-20	27-35
	coded_block_pattern (luma)	Table 9-18	73-76	73-76	73-76	73-76
	coded_block_pattern (chroma)	Table 9-18	77-84	77-84	77-84	77-84
	mb_qp_delta	Table 9-17	60-63	60-63	60-63	60-63
mb_pred( )	prev_intra4x4_pred_mode_flag	Table 9-17	68	68	68	68
	rem_intra4x4_pred_mode	Table 9-17	69	69	69	69
	intra_chroma_pred_mode	Table 9-17	64-67	64-67	64-67	64-67
mb_pred( ) and sub_mb_pred( )	ref_idx_l0	Table 9-16			54-59	54-59
	ref_idx_l1	Table 9-16				54-59
	mvd_l0[ ][ ][0]	Table 9-15			40-46	40-46
	mvd_l1[ ][ ][0]	Table 9-15				40-46
	mvd_l0[ ][ ][1]	Table 9-15			47-53	47-53
	mvd_l1[ ][ ][1]	Table 9-15				47-53
sub_mb_pred( )	sub_mb_type	Table 9-13 Table 9-14			21-23	36-39
residual_block_cabac( )	coded_block_flag	Table 9-18	85-104	85-104	85-104	85-104
	significant_coeff_flag[ ]	Table 9-19, Table 9-22.	105-165, 277-337	105-165, 277-337	105-165, 277-337	105-165, 277-337
	last_significant_coeff_flag[ ]	Table 9-20, Table 9-23.	166-226, 338-398	166-226, 338-398	166-226, 338-398	166-226, 338-398
	coeff_abs_level_minus1[ ]	Table 9-21	227-275	227-275	227-275	227-275

NOTE – ctxIdx equal to 276 is associated with the end_of_slice_flag and the bin of mb_type, which specifies the I_PCM macroblock type. The decoding process specified in subclause 9.3.3.2.4 applies to ctxIdx equal to 276. This decoding process, however, may also be implemented by using the decoding process specified in subclause 9.3.3.2.1. In this case, the initial values associated with ctxIdx equal to 276 are specified to be pStateIdx = 63 and valMPS = 0, where pStateIdx = 63 represents a non-adapting probability state.

Table 9-12 – Values of variables m and n for ctxIdx from 0 to 10

Initialisation variables	**ctxIdx**										
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
m	20	2	3	20	2	3	-28	-23	-6	-1	7
n	-15	54	74	-15	54	74	127	104	53	54	51

Table 9-13 – Values of variables m and n for ctxIdx from 11 to 23

Value of cabac_init_idc	**Initialisation variables**	**ctxIdx**												
		11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
0	**m**	23	23	21	1	0	-37	5	-13	-11	1	12	-4	17
	n	33	2	0	9	49	118	57	78	65	62	49	73	50
1	**m**	22	34	16	-2	4	-29	2	-6	-13	5	9	-3	10
	n	25	0	0	9	41	118	65	71	79	52	50	70	54
2	**m**	29	25	14	-10	-3	-27	26	-4	-24	5	6	-17	14
	n	16	0	0	51	62	99	16	85	102	57	57	73	57

Table 9-14 – Values of variables m and n for ctxIdx from 24 to 39

Value of cabac_init_idc	**Initialisation variables**	**ctxIdx**															
		24	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**
0	**m**	18	9	29	26	16	9	-46	-20	1	-13	-11	1	-6	-17	-6	9
	n	64	43	0	67	90	104	127	104	67	78	65	62	86	95	61	45
1	**m**	26	19	40	57	41	26	-45	-15	-4	-6	-13	5	6	-13	0	8
	n	34	22	0	2	36	69	127	101	76	71	79	52	69	90	52	43
2	**m**	20	20	29	54	37	12	-32	-22	-2	-4	-24	5	-6	-14	-6	4
	n	40	10	0	0	42	97	127	117	74	85	102	57	93	88	44	55

Table 9-15 – Values of variables m and n for ctxIdx from 40 to 53

Value of cabac_init_idc	Initialisation variables	ctxIdx													
		40	41	42	43	44	45	46	47	48	49	50	51	52	53
0	m	-3	-6	-11	6	7	-5	2	0	-3	-10	5	4	-3	0
	n	69	81	96	55	67	86	88	58	76	94	54	69	81	88
1	m	-2	-5	-10	2	2	-3	-3	1	-3	-6	0	-3	-7	-5
	n	69	82	96	59	75	87	100	56	74	85	59	81	86	95
2	m	-11	-15	-21	19	20	4	6	1	-5	-13	5	6	-3	-1
	n	89	103	116	57	58	84	96	63	85	106	63	75	90	101

Table 9-16 – Values of variables m and n for ctxIdx from 54 to 59

Value of cabac_init_idc	Initialisation variables	ctxIdx					
		54	55	56	57	58	59
0	m	-7	-5	-4	-5	-7	1
	n	67	74	74	80	72	58
1	m	-1	-1	1	-2	-5	0
	n	66	77	70	86	72	61
2	m	3	-4	-2	-12	-7	1
	n	55	79	75	97	50	60

Table 9-17 – Values of variables m and n for ctxIdx from 60 to 69

Initialisation variables	ctxIdx									
	60	61	62	63	64	65	66	67	68	69
m	0	0	0	0	-9	4	0	-7	13	3
n	41	63	63	63	83	86	97	72	41	62

Table 9-18 – Values of variables m and n for ctxIdx from 70 to 104

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
70	0	11	0	45	13	15	7	34	**88**	-11	115	-13	108	-4	92	5	78
71	1	55	-4	78	7	51	-9	88	**89**	-12	63	-3	46	0	39	-6	55
72	0	69	-3	96	2	80	-20	127	**90**	-2	68	-1	65	0	65	4	61
73	-17	127	-27	126	-39	127	-36	127	**91**	-15	84	-1	57	-15	84	-14	83
74	-13	102	-28	98	-18	91	-17	91	**92**	-13	104	-9	93	-35	127	-37	127
75	0	82	-25	101	-17	96	-14	95	**93**	-3	70	-3	74	-2	73	-5	79
76	-7	74	-23	67	-26	81	-25	84	**94**	-8	93	-9	92	-12	104	-11	104
77	-21	107	-28	82	-35	98	-25	86	**95**	-10	90	-8	87	-9	91	-11	91
78	-27	127	-20	94	-24	102	-12	89	**96**	-30	127	-23	126	-31	127	-30	127
79	-31	127	-16	83	-23	97	-17	91	**97**	-1	74	-5	54	3	55	0	65
80	-24	127	-22	110	-27	119	-31	127	**98**	-6	97	6	60	7	56	-2	79
81	-18	95	-21	91	-24	99	-14	76	**99**	-7	91	6	59	7	55	0	72
82	-27	127	-18	102	-21	110	-18	103	**100**	-20	127	6	69	8	61	-4	92
83	-21	114	-13	93	-18	102	-13	90	**101**	-4	56	-1	48	-3	53	-6	56
84	-30	127	-29	127	-36	127	-37	127	**102**	-5	82	0	68	0	68	3	68
85	-17	123	-7	92	0	80	11	80	**103**	-7	76	-4	69	-7	74	-8	71
86	-12	115	-5	89	-5	89	5	76	**104**	-22	125	-8	88	-9	88	-13	98
87	-16	122	-7	96	-7	94	2	84									

Table 9-19 – Values of variables m and n for ctxIdx from 105 to 165

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
105	-7	93	-2	85	-13	103	-4	86	**136**	-13	101	5	53	0	58	-5	75
106	-11	87	-6	78	-13	91	-12	88	**137**	-13	91	-2	61	-1	60	-8	80
107	-3	77	-1	75	-9	89	-5	82	**138**	-12	94	0	56	-3	61	-21	83
108	-5	71	-7	77	-14	92	-3	72	**139**	-10	88	0	56	-8	67	-21	64
109	-4	63	2	54	-8	76	-4	67	**140**	-16	84	-13	63	-25	84	-13	31
110	-4	68	5	50	-12	87	-8	72	**141**	-10	86	-5	60	-14	74	-25	64
111	-12	84	-3	68	-23	110	-16	89	**142**	-7	83	-1	62	-5	65	-29	94
112	-7	62	1	50	-24	105	-9	69	**143**	-13	87	4	57	5	52	9	75
113	-7	65	6	42	-10	78	-1	59	**144**	-19	94	-6	69	2	57	17	63
114	8	61	-4	81	-20	112	5	66	**145**	1	70	4	57	0	61	-8	74
115	5	56	1	63	-17	99	4	57	**146**	0	72	14	39	-9	69	-5	35
116	-2	66	-4	70	-78	127	-4	71	**147**	-5	74	4	51	-11	70	-2	27
117	1	64	0	67	-70	127	-2	71	**148**	18	59	13	68	18	55	13	91
118	0	61	2	57	-50	127	2	58	**149**	-8	102	3	64	-4	71	3	65
119	-2	78	-2	76	-46	127	-1	74	**150**	-15	100	1	61	0	58	-7	69
120	1	50	11	35	-4	66	-4	44	**151**	0	95	9	63	7	61	8	77
121	7	52	4	64	-5	78	-1	69	**152**	-4	75	7	50	9	41	-10	66
122	10	35	1	61	-4	71	0	62	**153**	2	72	16	39	18	25	3	62
123	0	44	11	35	-8	72	-7	51	**154**	-11	75	5	44	9	32	-3	68
124	11	38	18	25	2	59	-4	47	**155**	-3	71	4	52	5	43	-20	81
125	1	45	12	24	-1	55	-6	42	**156**	15	46	11	48	9	47	0	30
126	0	46	13	29	-7	70	-3	41	**157**	-13	69	-5	60	0	44	1	7
127	5	44	13	36	-6	75	-6	53	**158**	0	62	-1	59	0	51	-3	23
128	31	17	-10	93	-8	89	8	76	**159**	0	65	0	59	2	46	-21	74
129	1	51	-7	73	-34	119	-9	78	**160**	21	37	22	33	19	38	16	66
130	7	50	-2	73	-3	75	-11	83	**161**	-15	72	5	44	-4	66	-23	124
131	28	19	13	46	32	20	9	52	**162**	9	57	14	43	15	38	17	37
132	16	33	9	49	30	22	0	67	**163**	16	54	-1	78	12	42	44	-18
133	14	62	-7	100	-44	127	-5	90	**164**	0	62	0	60	9	34	50	-34
134	-13	108	9	53	0	54	1	67	**165**	12	72	9	69	0	89	-22	127
135	-15	100	2	53	-5	61	-15	72									

Table 9-20 – Values of variables m and n for ctxIdx from 166 to 226

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
166	24	0	11	28	4	45	4	39	**197**	26	-17	28	3	36	-28	28	-3
167	15	9	2	40	10	28	0	42	**198**	30	-25	28	4	38	-28	24	10
168	8	25	3	44	10	31	7	34	**199**	28	-20	32	0	38	-27	27	0
169	13	18	0	49	33	-11	11	29	**200**	33	-23	34	-1	34	-18	34	-14
170	15	9	0	46	52	-43	8	31	**201**	37	-27	30	6	35	-16	52	-44
171	13	19	2	44	18	15	6	37	**202**	33	-23	30	6	34	-14	39	-24
172	10	37	2	51	28	0	7	42	**203**	40	-28	32	9	32	-8	19	17
173	12	18	0	47	35	-22	3	40	**204**	38	-17	31	19	37	-6	31	25
174	6	29	4	39	38	-25	8	33	**205**	33	-11	26	27	35	0	36	29
175	20	33	2	62	34	0	13	43	**206**	40	-15	26	30	30	10	24	33
176	15	30	6	46	39	-18	13	36	**207**	41	-6	37	20	28	18	34	15
177	4	45	0	54	32	-12	4	47	**208**	38	1	28	34	26	25	30	20
178	1	58	3	54	102	-94	3	55	**209**	41	17	17	70	29	41	22	73
179	0	62	2	58	0	0	2	58	**210**	30	-6	1	67	0	75	20	34
180	7	61	4	63	56	-15	6	60	**211**	27	3	5	59	2	72	19	31
181	12	38	6	51	33	-4	8	44	**212**	26	22	9	67	8	77	27	44
182	11	45	6	57	29	10	11	44	**213**	37	-16	16	30	14	35	19	16
183	15	39	7	53	37	-5	14	42	**214**	35	-4	18	32	18	31	15	36
184	11	42	6	52	51	-29	7	48	**215**	38	-8	18	35	17	35	15	36
185	13	44	6	55	39	-9	4	56	**216**	38	-3	22	29	21	30	21	28
186	16	45	11	45	52	-34	4	52	**217**	37	3	24	31	17	45	25	21
187	12	41	14	36	69	-58	13	37	**218**	38	5	23	38	20	42	30	20
188	10	49	8	53	67	-63	9	49	**219**	42	0	18	43	18	45	31	12
189	30	34	-1	82	44	-5	19	58	**220**	35	16	20	41	27	26	27	16
190	18	42	7	55	32	7	10	48	**221**	39	22	11	63	16	54	24	42
191	10	55	-3	78	55	-29	12	45	**222**	14	48	9	59	7	66	0	93
192	17	51	15	46	32	1	0	69	**223**	27	37	9	64	16	56	14	56
193	17	46	22	31	0	0	20	33	**224**	21	60	-1	94	11	73	15	57
194	0	89	-1	84	27	36	8	63	**225**	12	68	-2	89	10	67	26	38
195	26	-19	25	7	33	-25	35	-18	**226**	2	97	-9	108	-10	116	-24	127
196	22	-17	30	-7	34	-30	33	-25									

Table 9-21 – Values of variables m and n for ctxIdx from 227 to 275

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
227	-3	71	-6	76	-23	112	-24	115	**252**	-12	73	-6	55	-16	72	-14	75
228	-6	42	-2	44	-15	71	-22	82	**253**	-8	76	0	58	-7	69	-10	79
229	-5	50	0	45	-7	61	-9	62	**254**	-7	80	0	64	-4	69	-9	83
230	-3	54	0	52	0	53	0	53	**255**	-9	88	-3	74	-5	74	-12	92
231	-2	62	-3	64	-5	66	0	59	**256**	-17	110	-10	90	-9	86	-18	108
232	0	58	-2	59	-11	77	-14	85	**257**	-11	97	0	70	2	66	-4	79
233	1	63	-4	70	-9	80	-13	89	**258**	-20	84	-4	29	-9	34	-22	69
234	-2	72	-4	75	-9	84	-13	94	**259**	-11	79	5	31	1	32	-16	75
235	-1	74	-8	82	-10	87	-11	92	**260**	-6	73	7	42	11	31	-2	58
236	-9	91	-17	102	-34	127	-29	127	**261**	-4	74	1	59	5	52	1	58
237	-5	67	-9	77	-21	101	-21	100	**262**	-13	86	-2	58	-2	55	-13	78
238	-5	27	3	24	-3	39	-14	57	**263**	-13	96	-3	72	-2	67	-9	83
239	-3	39	0	42	-5	53	-12	67	**264**	-11	97	-3	81	0	73	-4	81
240	-2	44	0	48	-7	61	-11	71	**265**	-19	117	-11	97	-8	89	-13	99
241	0	46	0	55	-11	75	-10	77	**266**	-8	78	0	58	3	52	-13	81
242	-16	64	-6	59	-15	77	-21	85	**267**	-5	33	8	5	7	4	-6	38
243	-8	68	-7	71	-17	91	-16	88	**268**	-4	48	10	14	10	8	-13	62
244	-10	78	-12	83	-25	107	-23	104	**269**	-2	53	14	18	17	8	-6	58
245	-6	77	-11	87	-25	111	-15	98	**270**	-3	62	13	27	16	19	-2	59
246	-10	86	-30	119	-28	122	-37	127	**271**	-13	71	2	40	3	37	-16	73
247	-12	92	1	58	-11	76	-10	82	**272**	-10	79	0	58	-1	61	-10	76
248	-15	55	-3	29	-10	44	-8	48	**273**	-12	86	-3	70	-5	73	-13	86
249	-10	60	-1	36	-10	52	-8	61	**274**	-13	90	-6	79	-1	70	-9	83
250	-6	62	1	38	-10	57	-8	66	**275**	-14	97	-8	85	-4	78	-10	87
251	-4	65	2	43	-9	58	-7	70									

Table 9-22 – Values of variables m and n for ctxIdx from 277 to 337

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
277	-6	93	-13	106	-21	126	-22	127	**308**	-16	96	-1	51	-16	77	-10	67
278	-6	84	-16	106	-23	124	-25	127	**309**	-7	88	7	49	-2	64	1	68
279	-8	79	-10	87	-20	110	-25	120	**310**	-8	85	8	52	2	61	0	77
280	0	66	-21	114	-26	126	-27	127	**311**	-7	85	9	41	-6	67	2	64
281	-1	71	-18	110	-25	124	-19	114	**312**	-9	85	6	47	-3	64	0	68
282	0	62	-14	98	-17	105	-23	117	**313**	-13	88	2	55	2	57	-5	78
283	-2	60	-22	110	-27	121	-25	118	**314**	4	66	13	41	-3	65	7	55
284	-2	59	-21	106	-27	117	-26	117	**315**	-3	77	10	44	-3	66	5	59
285	-5	75	-18	103	-17	102	-24	113	**316**	-3	76	6	50	0	62	2	65
286	-3	62	-21	107	-26	117	-28	118	**317**	-6	76	5	53	9	51	14	54
287	-4	58	-23	108	-27	116	-31	120	**318**	10	58	-13	49	-1	66	15	44
288	-9	66	-26	112	-33	122	-37	124	**319**	-1	76	4	63	-2	71	5	60
289	-1	79	-10	96	-10	95	-10	94	**320**	-1	83	6	64	-2	75	2	70
290	0	71	-12	95	-14	100	-15	102	**321**	-7	99	-2	69	-1	70	-2	76
291	3	68	-5	91	-8	95	-10	99	**322**	-14	95	-2	59	-9	72	-18	86
292	10	44	-9	93	-17	111	-13	106	**323**	2	95	6	70	14	60	12	70
293	-7	62	-22	94	-28	114	-50	127	**324**	0	76	10	44	16	37	5	64
294	15	36	-5	86	-6	89	-5	92	**325**	-5	74	9	31	0	47	-12	70
295	14	40	9	67	-2	80	17	57	**326**	0	70	12	43	18	35	11	55
296	16	27	-4	80	-4	82	-5	86	**327**	-11	75	3	53	11	37	5	56
297	12	29	-10	85	-9	85	-13	94	**328**	1	68	14	34	12	41	0	69
298	1	44	-1	70	-8	81	-12	91	**329**	0	65	10	38	10	41	2	65
299	20	36	7	60	-1	72	-2	77	**330**	-14	73	-3	52	2	48	-6	74
300	18	32	9	58	5	64	0	71	**331**	3	62	13	40	12	41	5	54
301	5	42	5	61	1	67	-1	73	**332**	4	62	17	32	13	41	7	54
302	1	48	12	50	9	56	4	64	**333**	-1	68	7	44	0	59	-6	76
303	10	62	15	50	0	69	-7	81	**334**	-13	75	7	38	3	50	-11	82
304	17	46	18	49	1	69	5	64	**335**	11	55	13	50	19	40	-2	77
305	9	64	17	54	7	69	15	57	**336**	5	64	10	57	3	66	-2	77
306	-12	104	10	41	-7	69	1	67	**337**	12	70	26	43	18	50	25	42
307	-11	97	7	46	-6	67	0	68									

Table 9-23 – Values of variables m and n for ctxIdx from 338 to 398

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
338	15	6	14	11	19	-6	17	-13	**369**	32	-26	31	-4	40	-37	37	-17
339	6	19	11	14	18	-6	16	-9	**370**	37	-30	27	6	38	-30	32	1
340	7	16	9	11	14	0	17	-12	**371**	44	-32	34	8	46	-33	34	15
341	12	14	18	11	26	-12	27	-21	**372**	34	-18	30	10	42	-30	29	15
342	18	13	21	9	31	-16	37	-30	**373**	34	-15	24	22	40	-24	24	25
343	13	11	23	-2	33	-25	41	-40	**374**	40	-15	33	19	49	-29	34	22
344	13	15	32	-15	33	-22	42	-41	**375**	33	-7	22	32	38	-12	31	16
345	15	16	32	-15	37	-28	48	-47	**376**	35	-5	26	31	40	-10	35	18
346	12	23	34	-21	39	-30	39	-32	**377**	33	0	21	41	38	-3	31	28
347	13	23	39	-23	42	-30	46	-40	**378**	38	2	26	44	46	-5	33	41
348	15	20	42	-33	47	-42	52	-51	**379**	33	13	23	47	31	20	36	28
349	14	26	41	-31	45	-36	46	-41	**380**	23	35	16	65	29	30	27	47
350	14	44	46	-28	49	-34	52	-39	**381**	13	58	14	71	25	44	21	62
351	17	40	38	-12	41	-17	43	-19	**382**	29	-3	8	60	12	48	18	31
352	17	47	21	29	32	9	32	11	**383**	26	0	6	63	11	49	19	26
353	24	17	45	-24	69	-71	61	-55	**384**	22	30	17	65	26	45	36	24
354	21	21	53	-45	63	-63	56	-46	**385**	31	-7	21	24	22	22	24	23
355	25	22	48	-26	66	-64	62	-50	**386**	35	-15	23	20	23	22	27	16
356	31	27	65	-43	77	-74	81	-67	**387**	34	-3	26	23	27	21	24	30
357	22	29	43	-19	54	-39	45	-20	**388**	34	3	27	32	33	20	31	29
358	19	35	39	-10	52	-35	35	-2	**389**	36	-1	28	23	26	28	22	41
359	14	50	30	9	41	-10	28	15	**390**	34	5	28	24	30	24	22	42
360	10	57	18	26	36	0	34	1	**391**	32	11	23	40	27	34	16	60
361	7	63	20	27	40	-1	39	1	**392**	35	5	24	32	18	42	15	52
362	-2	77	0	57	30	14	30	17	**393**	34	12	28	29	25	39	14	60
363	-4	82	-14	82	28	26	20	38	**394**	39	11	23	42	18	50	3	78
364	-3	94	-5	75	23	37	18	45	**395**	30	29	19	57	12	70	-16	123
365	9	69	-19	97	12	55	15	54	**396**	34	26	22	53	21	54	21	53
366	-12	109	-35	125	11	65	0	79	**397**	29	39	22	61	14	71	22	56
367	36	-35	27	0	37	-33	36	-16	**398**	19	66	11	86	11	83	25	61
368	36	-34	28	0	39	-36	37	-14									

9.3.1.2 Initialisation process for the arithmetic decoding engine

This process is invoked before decoding the first macroblock of a slice or after the decoding of the pcm_alignment_zero_bit and all pcm_byte data for a macroblock of type I_PCM.

Outputs of this process are the initialised decoding engine registers codIRange and codIOffset both in 16 bit register precision.

The status of the arithmetic decoding engine is represented by the variables codIRange and codIOffset. In the initialisation procedure of the arithmetic decoding process, codIRange is set equal to 0x01FE and codIOffset is set equal to the value returned from read_bits(9) interpreted as a 9 bit binary representation of an unsigned integer with most significant bit written first.

NOTE – The description of the arithmetic decoding engine in this Recommendation | International Standard utilizes 16 bit register precision. However, the minimum register precision for the variables codIRange and codIOffset is 9 bits.

9.3.2 Binarization process

Input to this process is a request for a syntax element.

Output of this process is the binarization of the syntax element, maxBinIdxCtx, ctxIdxOffset, and bypassFlag.

Table 9-24 specifies the type of binarization process, maxBinIdxCtx, and ctxIdxOffset associated with each syntax element.

The specification of the unary (U) binarization process, the truncated unary (TU) binarization process, the concatenated unary / k-th order Exp-Golomb (UEGk) binarization process, and the fixed-length (FL) binarization process are given in subclauses 9.3.2.1 to 9.3.2.4, respectively. Other binarizations are specified in subclauses 9.3.2.5 to 9.3.2.7.

Except for I slices, the binarizations for the syntax element mb_type as specified in subclause 9.3.2.5 consist of bin strings given by a concatenation of prefix and suffix bit strings. The UEGk binarization as specified in 9.3.2.3, which is used for the binarization of the syntax elements mvd_lX (X = 0, 1) and coeff_abs_level_minus1, and the binarization of the coded_block_pattern also consist of a concatenation of prefix and suffix bit strings. For these binarization processes, the prefix and the suffix bit string are separately indexed using the binIdx variable as specified further in subclause 9.3.3. The two sets of prefix bit strings and suffix bit strings are referred to as the binarization prefix part and the binarization suffix part, respectively.

Associated with each binarization or binarization part of a syntax element is a specific value of the context index offset (ctxIdxOffset) variable and a specific value of the maxBinIdxCtx variable as given in Table 9-24. When two values for each of these variables are specified for one syntax element in Table 9-24, the value in the upper row is related to the prefix part while the value in the lower row is related to the suffix part of the binarization of the corresponding syntax element.

The use of the DecodeBypass process and the variable bypassFlag is derived as follows.

- If no value is assigned to ctxIdxOffset for the corresponding binarization or binarization part in Table 9-24 labelled as “na”, all bins of the bit strings of the corresponding binarization or of the binarization prefix/suffix part shall be decoded by invoking the DecodeBypass process as specified in subclause 9.3.3.2.3. In such a case, bypassFlag is set equal to 1, where bypassFlag is used to indicate that for parsing the value of the bin from the bitstream the DecodeBypass process shall be applied.
- Otherwise, for each possible value of binIdx up to the specified value of MaxBinIdxCtx given in Table 9-24, a specific value of the variable ctxIdx is further specified in subclause 9.3.3. bypassFlag is set equal to 0.

The possible values of the context index ctxIdx are in the range of 0 to 398, inclusive. The value assigned to ctxIdxOffset specifies the lower value of the range of ctxIdx assigned to the corresponding binarization or binarization part of a syntax element.

ctxIdx = ctxIdxOffset = 276 is assigned to the syntax element end_of_slice_flag and the bin of mb_type, which specifies the I_PCM macroblock type as further specified in subclause 9.3.3.1. For parsing the value of the corresponding bin from the bitstream, the arithmetic decoding process for decisions before termination (DecodeTerminate) as specified in subclause 9.3.3.2.4 shall be applied.

NOTE – The bins of mb_type in I slices and the bins of the suffix for mb_type in SI slices that correspond to the same value of binIdx share the same ctxIdx. The last bin of the prefix of mb_type and the first bin of the suffix of mb_type in P, SP, and B slices may share the same ctxIdx.

Table 9-24 – Syntax elements and associated types of binarization, maxBinIdxCtx, and ctxIdxOffset

Syntax element	Type of binarization	maxBinIdxCtx	ctxIdxOffset
mb_type (SI slices only)	prefix and suffix as specified in subclause 9.3.2.5	prefix: 0 suffix: 6	prefix: 0 suffix: 3
mb_type (I slices only)	as specified in subclause 9.3.2.5	6	3
mb_skip_flag (P, SP slices only)	FL, cMax=1	0	11
mb_type (P, SP slices only)	prefix and suffix as specified in subclause 9.3.2.5	prefix: 2 suffix: 5	prefix: 14 suffix: 17
sub_mb_type (P, SP slices only)	as specified in subclause 9.3.2.5	2	21
mb_skip_flag (B slices only)	FL, cMax=1	0	24
mb_type (B slices only)	prefix and suffix as specified in subclause 9.3.2.5	prefix: 3 suffix: 5	prefix: 27 suffix: 32
sub_mb_type (B slices only)	as specified in subclause 9.3.2.5	3	36
mvd_l0[ ][ ][0], mvd_l1[ ][ ][0]	prefix and suffix as given by UEG3 with signedValFlag=1, uCoff=9	prefix: 4 suffix: na	prefix: 40 suffix: na (uses DecodeBypass)
mvd_l0[ ][ ][1], mvd_l1[ ][ ][1]		prefix: 4 suffix: na	prefix: 47 suffix: na (uses DecodeBypass)
ref_idx_l0, ref_idx_l1	U	2	54
mb_qp_delta	as specified in subclause 9.3.2.7	2	60
intra_chroma_pred_mode	TU, cMax=3	1	64
prev_intra4x4_pred_mode_flag	FL, cMax=1	0	68
rem_intra4x4_pred_mode	FL, cMax=7	0	69
mb_field_decoding_flag	FL, cMax=1	0	70
coded_block_pattern	prefix and suffix as specified in subclause 9.3.2.6	prefix: 3 suffix: 1	prefix: 73 suffix: 77
coded_block_flag	FL, cMax=1	0	85
significant_coeff_flag (frame coded blocks only)	FL, cMax=1	0	105
last_significant_coeff_flag (frame coded blocks only)	FL, cMax=1	0	166
coeff_abs_level_minus1	prefix and suffix as given by UEG0 with signedValFlag=0, uCoff=14	prefix: 1 suffix: na	prefix: 227 suffix: na, (uses DecodeBypass)
coeff_sign_flag	FL, cMax=1	0	na, (uses DecodeBypass)
end_of_slice_flag	FL, cMax=1	0	276
significant_coeff_flag (field coded blocks only)	FL, cMax=1	0	277
last_significant_coeff_flag (field coded blocks only)	FL, cMax=1	0	338

9.3.2.1 Unary (U) binarization process

Input to this process is a request for a U binarization for a syntax element.

Output of this process is the U binarization of the syntax element.

The bin string of a syntax element having (unsigned integer) value synElVal is a bit string of length synElVal + 1 indexed by BinIdx. The bins for binIdx less than synElVal are equal to 1. The bin with binIdx equal to synElVal is equal to 0.

Table 9-25 illustrates the bin strings of the unary binarization for a syntax element.

Table 9-25 – Bin string of the unary binarization (informative)

Value of syntax element	Bin string					
0	0					
1	1	0				
2	1	1	0			
3	1	1	1	0		
4	1	1	1	1	0	
5	1	1	1	1	1	0
…						
binIdx	0	1	2	3	4	5

9.3.2.2 Truncated unary (TU) binarization process

Input to this process is a request for a TU binarization for a syntax element and cMax.

Output of this process is the TU binarization of the syntax element.

For syntax element (unsigned integer) values less than cMax, the U binarization process as specified in subclause 9.3.2.1 is invoked. For the syntax element value equal to cMax the bin string is a bit string of length cMax with all bins being equal to 1.

NOTE – TU binarization is always invoked with a cMax value equal to the largest possible value of the syntax element being decoded.

9.3.2.3 Concatenated unary/ k-th order Exp-Golomb (UEGk) binarization process

Input to this process is a request for a UEGk binarization for a syntax element, signedValFlag and uCoff.

Output of this process is the UEGk binarization of the syntax element.

A UEGk bin string is a concatenation of a prefix bit string and a suffix bit string. The prefix of the binarization is specified by invoking the TU binarization process for the prefix part Min(uCoff, Abs(synElVal)) of a syntax element value synElVal as specified in subclause 9.3.2.2 with cMax = uCoff, where uCoff > 0.

The UEGk bin string is derived as follows.

- If one of the following is true, the bin string of a syntax element having value synElVal consists only of a prefix bit string,
  - signedValFlag is equal to 0 and the prefix bit string is not equal to the bit string of length uCoff with all bits equal to 1.
  - signedValFlag is equal to 1 and the prefix bit string is equal to the bit string that consists of a single bit with value equal to 0.
- Otherwise, the bin string of the UEGk suffix part of a syntax element value synElVal is specified by a process equivalent to the following pseudo-code:

```
if( Abs( synElVal ) >= uCoff ) {
    sufS = Abs( synElVal ) − uCoff
    stopLoop = 0
    do {
        if( sufS >= ( 1 << k ) ) {
            put( 1 )
            sufS = sufS − ( 1<<k )
            k++
        } else {
            put( 0 )
            while( k− − )
                put( ( sufS >> k ) & 0x01 )
            stopLoop = 1
        }
    } while( !stopLoop )
}
if( signedValFlag && synElVal ! = 0)
    if( synElVal > 0 )
        put( 0 )
    else
        put( 1 )
```

NOTE – The specification for the k-th order Exp-Golomb (EGk) code uses 1's and 0's in reverse meaning for the unary part of the Exp-Golomb code of 0-th order as specified in subclause 9.1.

9.3.2.4 Fixed-length (FL) binarization process

Input to this process is a request for a FL binarization for a syntax element and cMax.

Output of this process is the FL binarization of the syntax element.

FL binarization is constructed by using an fixedLength-bit unsigned integer bin string of the syntax element value, where fixedLength = Ceil(Log2(cMax + 1)). The indexing of bins for the FL binarization is such that the binIdx = 0 relates to the least significant bit with increasing values of binIdx towards the most significant bit.

9.3.2.5 Binarization process for macroblock type and sub-macroblock type

Input to this process is a request for a binarization for syntax elements mb_type or sub_mb_type.

Output of this process is the binarization of the syntax element.

The binarization scheme for decoding of macroblock type in I slices is specified in Table 9-26.

For macroblock types in SI slices, the binarization consists of bin strings specified as a concatenation of a prefix and a suffix bit string as follows.

The prefix bit string consists of a single bit, which is specified by b_0 = ((mb_type = = SI) ? 0 : 1). For the syntax element value for which b_0 is equal to 0, the bin string only consists of the prefix bit string. For the syntax element value for which b_0 is equal to 1, the binarization is given by concatenating the prefix b_0 and the suffix bit string as specified in Table 9-26 for macroblock type in I slices indexed by subtracting 1 from the value of mb_type in SI slices.

Table 9-26 – Binarization for macroblock types in I slices

Value (name) of mb_type	Bin string						
0 (I_4x4)	0						
1 (I_16x16_0_0_0)	1	0	0	0	0	0	
2 (I_16x16_1_0_0)	1	0	0	0	0	1	
3 (I_16x16_2_0_0)	1	0	0	0	1	0	
4 (I_16x16_3_0_0)	1	0	0	0	1	1	
5 (I_16x16_0_1_0)	1	0	0	1	0	0	0
6 (I_16x16_1_1_0)	1	0	0	1	0	0	1
7 (I_16x16_2_1_0)	1	0	0	1	0	1	0
8 (I_16x16_3_1_0)	1	0	0	1	0	1	1
9 (I_16x16_0_2_0)	1	0	0	1	1	0	0
10 (I_16x16_1_2_0)	1	0	0	1	1	0	1
11 (I_16x16_2_2_0)	1	0	0	1	1	1	0
12 (I_16x16_3_2_0)	1	0	0	1	1	1	1
13 (I_16x16_0_0_1)	1	0	1	0	0	0	
14 (I_16x16_1_0_1)	1	0	1	0	0	1	
15 (I_16x16_2_0_1)	1	0	1	0	1	0	
16 (I_16x16_3_0_1)	1	0	1	0	1	1	
17 (I_16x16_0_1_1)	1	0	1	1	0	0	0
18 (I_16x16_1_1_1)	1	0	1	1	0	0	1
19 (I_16x16_2_1_1)	1	0	1	1	0	1	0
20 (I_16x16_3_1_1)	1	0	1	1	0	1	1
21 (I_16x16_0_2_1)	1	0	1	1	1	0	0
22 (I_16x16_1_2_1)	1	0	1	1	1	0	1
23 (I_16x16_2_2_1)	1	0	1	1	1	1	0
24 (I_16x16_3_2_1)	1	0	1	1	1	1	1
25 (I_PCM)	1	1					
binIdx	0	1	2	3	4	5	6

The binarization schemes for P macroblock types in P and SP slices and for B macroblocks in B slices are specified in Table 9-27.

The bin string for I macroblock types in P and SP slices corresponding to mb_type values 5 to 30 consists of a concatenation of a prefix, which consists of a single bit with value equal to 1 as specified in Table 9-27 and a suffix as specified in Table 9-26, indexed by subtracting 5 from the value of mb_type.

mb_type equal to 4 (P_8x8ref0) is not allowed..

For I macroblock types in B slices (mb_type values 23 to 48) the binarization consists of bin strings specified as a concatenation of a prefix bit string as specified in Table 9-27 and suffix bit strings as specified in Table 9-26, indexed by subtracting 23 from the value of mb_type.

Table 9-27 – Binarization for macroblock types in P, SP, and B slices

Slice type	Value (name) of mb_type	Bin string						
P, SP slice	0 (P_L0_16x16)	0	0	0				
	1 (P_L0_L0_16x8)	0	1	1				
	2 (P_L0_L0_8x16)	0	1	0				
	3 (P_8x8)	0	0	1				
	4 (P_8x8ref0)	na						
	5 to 30 (Intra, prefix only)	1						
B slice	0 (B_Direct_16x16)	0						
	1 (B_L0_16x16)	1	0	0				
	2 (B_L1_16x16)	1	0	1				
	3 (B_Bi_16x16)	1	1	0	0	0	0	
	4 (B_L0_L0_16x8)	1	1	0	0	0	1	
	5 (B_L0_L0_8x16)	1	1	0	0	1	0	
	6 (B_L1_L1_16x8)	1	1	0	0	1	1	
	7 (B_L1_L1_8x16)	1	1	0	1	0	0	
	8 (B_L0_L1_16x8)	1	1	0	1	0	1	
	9 (B_L0_L1_8x16)	1	1	0	1	1	0	
	10 (B_L1_L0_16x8)	1	1	0	1	1	1	
	11 (B_L1_L0_8x16)	1	1	1	1	1	0	
	12 (B_L0_Bi_16x8)	1	1	1	0	0	0	0
	13 (B_L0_Bi_8x16)	1	1	1	0	0	0	1
	14 (B_L1_Bi_16x8)	1	1	1	0	0	1	0
	15 (B_L1_Bi_8x16)	1	1	1	0	0	1	1
	16 (B_Bi_L0_16x8)	1	1	1	0	1	0	0
	17 (B_Bi_L0_8x16)	1	1	1	0	1	0	1
	18 (B_Bi_L1_16x8)	1	1	1	0	1	1	0
	19 (B_Bi_L1_8x16)	1	1	1	0	1	1	1
	20 (B_Bi_Bi_16x8)	1	1	1	1	0	0	0
	21 (B_Bi_Bi_8x16)	1	1	1	1	0	0	1
	22 (B_8x8)	1	1	1	1	1	1	
	23 to 48 (Intra, prefix only)	1	1	1	1	0	1	
binIdx		0	1	2	3	4	5	6

For P, SP, and B slices the specification of the binarization for sub_mb_type is given in Table 9-28.

Table 9-28 – Binarization for sub-macroblock types in P, SP, and B slices

Slice type	Value (name) of sub_mb_type	Bin string					
P, SP slice	0 (P_L0_8x8)	1					
	1 (P_L0_8x4)	0	0				
	2 (P_L0_4x8)	0	1	1			
	3 (P_L0_4x4)	0	1	0			
B slice	0 (B_Direct_8x8)	0					
	1 (B_L0_8x8)	1	0	0			
	2 (B_L1_8x8)	1	0	1			
	3 (B_Bi_8x8)	1	1	0	0	0	
	4 (B_L0_8x4)	1	1	0	0	1	
	5 (B_L0_4x8)	1	1	0	1	0	
	6 (B_L1_8x4)	1	1	0	1	1	
	7 (B_L1_4x8)	1	1	1	0	0	0
	8 (B_Bi_8x4)	1	1	1	0	0	1
	9 (B_Bi_4x8)	1	1	1	0	1	0
	10 (B_L0_4x4)	1	1	1	0	1	1
	11 (B_L1_4x4)	1	1	1	1	0	
	12 (B_Bi_4x4)	1	1	1	1	1	
binIdx		0	1	2	3	4	5

9.3.2.6 Binarization process for coded block pattern

Input to this process is a request for a binarization for the syntax element coded_block_pattern.

Output of this process is the binarization of the syntax element.

The binarization of coded_block_pattern consists of a concatenation of a prefix part and a suffix part. The prefix part of the binarization is given by the FL binarization of CodedBlockPatternLuma with cMax = 15. The suffix part consists of the TU binarization of CodedBlockPatternChroma with cMax = 2. The relationship between the value of the syntax element coded_block_pattern and the values of CodedBlockPatternLuma and CodedBlockPatternChroma is given as specified in subclause 7.4.5.

9.3.2.7 Binarization process for mb_qp_delta

Input to this process is a request for a binarization for the syntax element mb_qp_delta.

Output of this process is the binarization of the syntax element.

The bin string of mb_qp_delta is derived by the U binarization of the mapped value of the syntax element mb_qp_delta, where the assignment rule between the signed value of mb_qp_delta and its mapped value is given as specified in Table 9-3.

9.3.3 Decoding process flow

Input to this process is a binarization of the requested syntax element, maxBinIdxCtx, bypassFlag and ctxIdxOffset as specified in subclause 9.3.2.

Output of this process is the value of the syntax element.

This process specifies how each bit of a bit string is parsed for each syntax element.

After parsing each bit, the resulting bit string is compared to all bin strings of the binarization of the syntax element and the following applies.

- If the bit string is equal to one of the bin strings, the corresponding value of the syntax element is the output.
- Otherwise (the bit string is not equal to one of the bin strings), the next bit is parsed.

While parsing each bin, the variable binIdx is incremented by 1 starting with binIdx being set equal to 0 for the first bin.

When the binarization of the corresponding syntax element consists of a prefix and a suffix binarization part,, the variable binIdx is set equal to 0 for the first bin of each part of the bin string (prefix part or suffix part). In this case, after parsing the prefix bit string, the parsing process of the suffix bit string related to the binarizations specified in subclauses 9.3.2.3 and 9.3.2.5 is invoked depending on the resulting prefix bit string as specified in subclauses 9.3.2.3 and 9.3.2.5. Note that for the binarization of the syntax element coded_block_pattern, the suffix bit string is present regardless of the prefix bit string of length 4 as specified in subclause 9.3.2.6.

Depending on the variable bypassFlag, the following applies.

- If bypassFlag is equal to 1, the bypass decoding process as specified in subclause 9.3.3.2.3 shall be applied for parsing the value of the bins from the bitstream.
- Otherwise (bypassFlag is equal to 0), the parsing of each bin is specified by the following two ordered steps:
  1. Given binIdx, maxBinIdxCtx and ctxIdxOffset, ctxIdx is derived as specified in subclause 9.3.3.1.
  2. Given ctxIdx, the value of the bin from the bitstream as specified in subclause 9.3.3.2 is decoded.

9.3.3.1 Derivation process for ctxIdx

Inputs to this process are binIdx, maxBinIdxCtx and ctxIdxOffset.

Output of this process is ctxIdx.

Table 9-29 shows the assignment of ctxIdx increments (ctxIdxInc) to binIdx for all ctxIdxOffset values except those related to the syntax elements coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1.

The ctxIdx to be used with a specific binIdx is specified by first determining the ctxIdxOffset associated with the given bin string or part thereof. The ctxIdx is determined as follows.

- If the ctxIdxOffset is listed in Table 9-29, the ctxIdx for a binIdx is the sum of ctxIdxOffset and ctxIdxInc, which is found in Table 9-29. When more than one value is listed in Table 9-29 for a binIdx, the assignment process for ctxIdxInc for that binIdx is further specified in the subclauses given in parenthesis of the corresponding table entry.
- Otherwise (ctxIdxOffset is not listed in Table 9-29), the ctxIdx is specified to be the sum of the following terms: ctxIdxOffset and ctxIdxBlockCatOffset(ctxBlockCat) as specified in Table 9-30 and ctxIdxInc(ctxBlockCat). Subclause 9.3.3.1.3 specifies which ctxBlockCat is used. Subclause 9.3.3.1.1.9 specifies the assignment of ctxIdxInc(ctxBlockCat) for coded_block_flag, and subclause 9.3.3.1.3 specifies the assignment of ctxIdxInc(ctxBlockCat) for significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1.

All bins with binIdx greater than maxBinIdxCtx are parsed using ctxIdx assigned to maxBinIdxCtx.

All entries in Table 9-29 labelled with "na" correspond to values of binIdx that do not occur for the corresponding ctxIdxOffset.

ctxIdx = 276 is assigned to the binIdx of mb_type indicating the I_PCM mode. For parsing the value of the corresponding bins from the bitstream, the arithmetic decoding process for decisions before termination as specified in subclause 9.3.3.2.4 shall be applied.

Table 9-29 – Assignment of ctxIdxInc to binIdx for all ctxIdxOffset values except those related to the syntax elements coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1

ctxIdxOffset	binIdx						
	0	1	2	3	4	5	>= 6
0	0,1,2 (subclause 9.3.3.1.1.3)	na	na	na	na	na	na
3	0,1,2 (subclause 9.3.3.1.1.3)	ctxIdx=276	3	4	5,6 (subclause 9.3.3.1.2)	6,7 (subclause 9.3.3.1.2)	7
11	0,1,2 (subclause 9.3.3.1.1.1)	na	na	na	na	na	na
14	0	1	2,3 (subclause 9.3.3.1.2)	na	na	na	na
17	0	ctxIdx=276	1	2	2,3 (subclause 9.3.3.1.2)	3	3
21	0	1	2	na	na	na	na
24	0,1,2 (subclause 9.3.3.1.1.1)	na	na	na	na	na	na
27	0,1,2 (subclause 9.3.3.1.1.3)	3	4,5 (subclause 9.3.3.1.2)	5	5	5	5
32	0	ctxIdx=276	1	2	2,3 (subclause 9.3.3.1.2)	3	3
36	0	1	2,3 (subclause 9.3.3.1.2)	3	3	3	na
40	0,1,2 (subclause 9.3.3.1.1.7)	3	4	5	6	6	6
47	0,1,2 (subclause 9.3.3.1.1.7)	3	4	5	6	6	6
54	0,1,2,3 (subclause 9.3.3.1.1.6)	4	5	5	5	5	5
60	0,1 (subclause 9.3.3.1.1.5)	2	3	3	3	3	3
64	0,1,2 (subclause 9.3.3.1.1.8)	3	3	na	na	na	na
68	0	na	na	na	na	na	na
69	0	0	0	na	na	na	na
70	0,1,2 (subclause 9.3.3.1.1.2)	na	na	na	na	na	na
73	0,1,2,3 (subclause 9.3.3.1.1.4)	0,1,2,3 (subclause 9.3.3.1.1.4)	0,1,2,3 (subclause 9.3.3.1.1.4)	0,1,2,3 (subclause 9.3.3.1.1.4)	na	na	na
77	0,1,2,3 (subclause 9.3.3.1.1.4)	4,5,6,7 (subclause 9.3.3.1.1.4)	na	na	na	na	na
276	0	na	na	na	na	na	na

Table 9-30 shows the values of ctxIdxBlockCatOffset depending on ctxBlockCat for the syntax elements coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1. The specification of ctxBlockCat is given in Table 9-32.

Table 9-30 – Assignment of ctxIdxBlockCatOffset to ctxBlockCat for syntax elements coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1

Syntax element	ctxBlockCat (as specified in Table 9-32)				
	0	1	2	3	4
coded_block_flag	0	4	8	12	16
significant_coeff_flag	0	15	29	44	47
last_significant_coeff_flag	0	15	29	44	47
coeff_abs_level_minus1	0	10	20	30	39

9.3.3.1.1 Assignment process of ctxIdxInc using neighbouring syntax elements

Subclause 9.3.3.1.1.1 specifies the derivation process of ctxIdxInc for the syntax element mb_skip_flag.

Subclause 9.3.3.1.1.2 specifies the derivation process of ctxIdxInc for the syntax element mb_field_decoding_flag.

Subclause 9.3.3.1.1.3 specifies the derivation process of ctxIdxInc for the syntax element mb_type.

Subclause 9.3.3.1.1.4 specifies the derivation process of ctxIdxInc for the syntax element coded_block_pattern.

Subclause 9.3.3.1.1.5 specifies the derivation process of ctxIdxInc for the syntax element mb_qp_delta.

Subclause 9.3.3.1.1.6 specifies the derivation process of ctxIdxInc for the syntax elements ref_idx_l0 and ref_idx_l1.

Subclause 9.3.3.1.1.7 specifies the derivation process of ctxIdxInc for the syntax elements mvd_l0 and mvd_l1.

Subclause 9.3.3.1.1.8 specifies the derivation process of ctxIdxInc for the syntax element intra_chroma_pred_mode.

Subclause 9.3.3.1.1.9 specifies the derivation process of ctxIdxInc for the syntax element coded_block_flag.

9.3.3.1.1.1 Derivation process of ctxIdxInc for the syntax element mb_skip_flag

Output of this process is ctxIdxInc.

When MbaffFrameFlag is equal to 1 and mb_field_decoding_flag has not been decoded (yet) for the current macroblock pair with top macroblock address 2 * (CurrMbAddr / 2), the inference rule for the syntax element mb_field_decoding_flag as specified in subclause 7.4.4 shall be applied.

The derivation process for neighbouring macroblocks specified in subclause 6.4.7.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

Let the variable condTermFlagN (with N being either A or B) be derived as follows.

- If mbAddrN is not available or mb_skip_flag for the macroblock mbAddrN is equal to 1, condTermFlagN is set equal to 0.
- Otherwise (mbAddrN is available and mb_skip_flag for the macroblock mbAddrN is equal to 0), condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived by

$$\text{ctxIdxInc} = \text{condTermFlagA} + \text{condTermFlagB} \qquad (9\text{-}1)$$

9.3.3.1.1.2 Derivation process of ctxIdxInc for the syntax element mb_field_decoding_flag

Output of this process is ctxIdxInc.

The derivation process for neighbouring macroblock addresses and their availability in MBAFF frames as specified in subclause 6.4.6 is invoked and the output is assigned to mbAddrA and mbAddrB.

When both macroblocks mbAddrN and mbAddrN + 1 have mb_type equal to P_Skip or B_Skip, the inference rule for the syntax element mb_field_decoding_flag as specified in subclause 7.4.4 shall be applied for the macroblock mbAddrN.

Let the variable condTermFlagN (with N being either A or B) be derived as follows.

- If any of the following conditions is true, condTermFlagN is set equal to 0,
  - mbAddrN is not available
  - the macroblock mbAddrN is a frame macroblock.
- Otherwise, condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived by

ctxIdxInc = condTermFlagA + condTermFlagB (9-2)

9.3.3.1.1.3 Derivation process of ctxIdxInc for the syntax element mb_type

Input to this process is ctxIdxOffset.

Output of this process is ctxIdxInc.

The derivation process for neighbouring macroblocks specified in subclause 6.4.7.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

Let the variable condTermFlagN (with N being either A or B) be derived as follows.

- If any of the following conditions is true, condTermFlagN is set equal to 0
  - mbAddrN is not available
  - ctxIdxOffset is equal to 0 and mb_type for the macroblock mbAddrN is equal to SI
  - ctxIdxOffset is equal to 3 and mb_type for the macroblock mbAddrN is equal to I_4x4
  - ctxIdxOffset is equal to 27 and the macroblock mbAddrN is skipped
  - ctxIdxOffset is equal to 27 and mb_type for the macroblock mbAddrN is equal to B_Direct_16x16
- Otherwise, condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived as

ctxIdxInc = condTermFlagA + condTermFlagB (9-3)

9.3.3.1.1.4 Derivation process of ctxIdxInc for the syntax element coded_block_pattern

Inputs to this process are ctxIdxOffset and binIdx.

Output of this process is ctxIdxInc.

Depending on the value of the variable ctxIdxOffset, the following applies.

- If ctxIdxOffset is equal to 73, the following applies
  - The derivation process for neighbouring 8x8 luma blocks specified in subclause 6.4.7.2 is invoked with luma8x8BlkIdx = binIdx as input and the output is assigned to mbAddrA, mbAddrB, luma8x8BlkIdxA, and luma8x8BlkIdxB.
  - Let the variable condTermFlagN (with N being either A or B) be derived as follows.
    - If any of the following conditions is true, condTermFlagN is set equal to 0
      - mbAddrN is not available
      - mb_type for the macroblock mbAddrN is equal to I_PCM
      - the macroblock mbAddrN is not skipped and ((CodedBlockPatternLuma >> luma8x8BlkIdxN) & 1) is not equal to 0 for the macroblock mbAddrN
    - Otherwise, condTermFlagN is set equal to 1.
  - The variable ctxIdxInc is derived as

    ctxIdxInc = condTermFlagA + 2 * condTermFlagB (9-4)

- Otherwise (ctxIdxOffset is equal to 77), the following applies.
  - The derivation process for neighbouring macroblocks specified in subclause 6.4.7.1 is invoked and the output is assigned to mbAddrA and mbAddrB.
  - Let the variable condTermFlagN (with N being either A or B) be derived as follows.
    - If mbAddrN is available and mb_type for the macroblock mbAddrN is equal to I_PCM, condTermFlagN is set equal to 1
    - Otherwise, if any of the following conditions is true, condTermFlagN is set equal to 0
      - mbAddrN is not available or the macroblock mbAddrN is skipped
      - binIdx is equal to 0 and CodedBlockPatternChroma for the macroblock mbAddrN is equal to 0
      - binIdx is equal to 1 and CodedBlockPatternChroma for the macroblock mbAddrN is not equal to 2
    - Otherwise, condTermFlagN is set equal to 1.
  - The variable ctxIdxInc is derived as

$$\text{ctxIdxInc} = \text{condTermFlagA} + 2 * \text{condTermFlagB} + ((\text{binIdx} == 1) ? 4 : 0) \quad (9\text{-}5)$$

NOTE – When a macroblock uses an Intra_16x16 prediction mode, the values of CodedBlockPatternLuma and CodedBlockPatternChroma for the macroblock are derived from mb_type as specified in Table 7-8.

9.3.3.1.1.5 Derivation process of ctxIdxInc for the syntax element mb_qp_delta

Output of this process is ctxIdxInc.

Let prevMbAddr be the macroblock address of the macroblock that precedes the current macroblock in decoding order. When the current macroblock is the first macroblock of a slice, prevMbAddr is marked as not available.

Let the variable ctxIdxInc be derived as follows.

- If any of the following conditions is true, ctxIdxInc is set equal to 0
  - prevMbAddr is not available or the macroblock prevMbAddr is skipped
  - mb_type of the macroblock prevMbAddr is equal to I_PCM
  - The macroblock prevMbAddr is not coded in Intra_16x16 prediction mode and both CodedBlockPatternLuma and CodedBlockPatternChroma for the macroblock prevMbAddr are equal to 0
  - mb_qp_delta for the macroblock prevMbAddr is equal to 0
- Otherwise, ctxIdxInc is set equal to 1.

9.3.3.1.1.6 Derivation process of ctxIdxInc for the syntax elements ref_idx_l0 and ref_idx_l1

Inputs to this process are mbPartIdx and the reference picture list suffix lX, where X = 0 or 1.

Output of this process is ctxIdxInc.

Let currSubMbType be set equal to sub_mb_type[mbPartIdx].

The derivation process for neighbouring partitions specified in subclause 6.4.7.5 is invoked with mbPartIdx, currSubMbType, and subMbPartIdx = 0 as input and the output is assigned to mbAddrA\mbPartIdxA and mbAddrB\mbPartIdxB.

With ref_idx_lX[mbPartIdxN] (with N being either A or B) specifying the syntax element for the macroblock mbAddrN, let the variable refIdxZeroFlagN be derived as follows.

- If MbaffFrameFlag is equal to 1, the current macroblock is a frame macroblock, and the macroblock mbAddrN is a field macroblock

$$\text{refIdxZeroFlagN} = ((\text{ref\_idx\_lX}[\text{mbPartIdxN}] > 1) ? 0 : 1) \quad (9\text{-}6)$$

- Otherwise,

$$\text{refIdxZeroFlagN} = ((\text{ref\_idx\_lX}[\text{mbPartIdxN}] > 0) \; ? \; 0 : 1) \qquad (9\text{-}7)$$

Let the variable predModeEqualFlag be specified as follows.

- If the macroblock mbAddrN has mb_type equal to P_8x8 or B_8x8, the following applies.
  - If SubMbPredMode(sub_mb_type[mbPartIdxN]) is not equal to Pred_LX and not equal to BiPred, predModeEqualFlag is set equal to 0, where sub_mb_type specifies the syntax element for the macroblock mbAddrN.
  - Otherwise, predModeEqualFlag is set equal to 1.
- Otherwise, the following applies.
  - If MbPartPredMode(mb_type, mbPartIdxN) is not equal to Pred_LX and not equal to BiPred, predModeEqualFlag is set equal to 0, where mb_type specifies the syntax element for the macroblock mbAddrN.
  - Otherwise, predModeEqualFlag is set equal to 1.

Let the variable condTermFlagN (with N being either A or B) be derived as follows.

- If any of the following conditions is true, condTermFlagN is set equal to 0
  - mbAddrN is not available
  - the macroblock mbAddrN has mb_type equal to P_Skip or B_Skip
  - The macroblock mbAddrN is coded in Intra prediction mode
  - predModeEqualFlag is equal to 0
  - refIdxZeroFlagN is equal to 1
- Otherwise, condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived as

$$\text{ctxIdxInc} = \text{condTermFlagA} + 2 * \text{condTermFlagB} \qquad (9\text{-}8)$$

9.3.3.1.1.7 Derivation process of ctxIdxInc for the syntax elements mvd_l0 and mvd_l1

Inputs to this process are mbPartIdx, subMbPartIdx, the reference picture list suffix lX, and ctxIdxOffset

Output of this process is ctxIdxInc.

Let currSubMbType be set equal to sub_mb_type[mbPartIdx].

The derivation process for neighbouring partitions specified in subclause 6.4.7.5 is invoked with mbPartIdx, currSubMbType, and subMbPartIdx as input and the output is assigned to mbAddrA\mbPartIdxA\subMbPartIdxA and mbAddrB\mbPartIdxB\subMbPartIdxB.

Let the variable compIdx be derived as follows.

- If ctxIdxOffset is equal to 40, compIdx is set equal to 0.
- Otherwise (ctxIdxOffset is equal to 47), compIdx is set equal to 1.

Let the variable predModeEqualFlag be specified as follows.

- If the macroblock mbAddrN has mb_type equal to P_8x8 or B_8x8, the following applies.
  - If SubMbPredMode(sub_mb_type[mbPartIdxN]) is not equal to Pred_LX and not equal to BiPred, predModeEqualFlag is set equal to 0, where sub_mb_type specifies the syntax element for the macroblock mbAddrN.
  - Otherwise, predModeEqualFlag is set equal to 1.

- Otherwise, the following applies.
  - If MbPartPredMode(mb_type, mbPartIdxN) is not equal to Pred_LX and not equal to BiPred, predModeEqualFlag is set equal to 0, where mb_type specifies the syntax element for the macroblock mbAddrN.
  - Otherwise, predModeEqualFlag is set equal to 1.

Let the variable absMvdCompN (with N being either A or B) be derived as follows.

- If any of the following conditions is true, absMvdCompN is set equal to 0
  - mbAddrN is not available
  - the macroblock mbAddrN has mb_type equal to P_Skip or B_Skip
  - The macroblock mbAddrN is coded in Intra prediction mode
  - predModeEqualFlag is equal to 0
- Otherwise, the following applies
  - If compIdx is equal to 1, MbaffFrameFlag is equal to 1, the current macroblock is a frame macroblock, and the macroblock mbAddrN is a field macroblock

    absMvdCompN = Abs(mvd_lX[mbPartIdxN][subMbPartIdxN][compIdx]) * 2 (9-9)

  - Otherwise, if compIdx is equal to 1, MbaffFrameFlag is equal to 1, the current macroblock is a field macroblock, and the macroblock mbAddrN is a frame macroblock

    absMvdCompN = Abs(mvd_lX[mbPartIdxN][subMbPartIdxN][compIdx]) / 2 (9-10)

  - Otherwise,

    absMvdCompN = Abs(mvd_lX[mbPartIdxN][subMbPartIdxN][compIdx]) (9-11)

The variable ctxIdxInc is derived as follows

- If (absMvdCompA + absMvdCompB) is less than 3, ctxIdxInc is set equal to 0.
- Otherwise, if (absMvdCompA + absMvdCompB) is greater than 32, ctxIdxInc is set equal to 2.
- Otherwise ((absMvdCompA + absMvdCompB) is in the range of 3 to 32, inclusive), ctxIdxInc is set equal to 1.

9.3.3.1.1.8 Derivation process of ctxIdxInc for the syntax element intra_chroma_pred_mode

Output of this process is ctxIdxInc.

The derivation process for neighbouring macroblocks specified in subclause 6.4.7.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

Let the variable condTermFlagN (with N being replaced by either A or B) be derived as follows.

- If any of the following conditions is true, condTermFlagN is set equal to 0
  - mbAddrN is not available
  - The macroblock mbAddrN is coded in Inter prediction mode
  - mb_type for the macroblock mbAddrN is equal to I_PCM
  - intra_chroma_pred_mode for the macroblock mbAddrN is equal to 0
- Otherwise, condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived by

ctxIdxInc = condTermFlagA + condTermFlagB (9-12)

9.3.3.1.1.9 Derivation process of ctxIdxInc for the syntax element coded_block_flag

Input to this process is ctxBlockCat and additional input is specified as follows.

- If ctxBlockCat is equal to 0, no additional input
- Otherwise, if ctxBlockCat is equal to 1 or 2, luma4x4BlkIdx
- Otherwise, if ctxBlockCat is equal to 3, the chroma component index iCbCr
- Otherwise (ctxBlockCat is equal to 4), chroma4x4BlkIdx and the chroma component index compIdx

Output of this process is ctxIdxInc(ctxBlockCat).

Let the variable transBlockN (with N being either A or B) be derived as follows.

- If ctxBlockCat is equal to 0, the following applies.
  - The derivation process for neighbouring macroblocks specified in subclause 6.4.7.1 is invoked and the output is assigned to mbAddrN (with N being either A or B).
  - The variable transBlockN is derived as follows.
    - If mbAddrN is available and the macroblock mbAddrN is coded in Intra_16x16 prediction mode, the luma DC block of macroblock mbAddrN is assigned to transBlockN
    - Otherwise, transBlockN is marked as not available.
- Otherwise, if ctxBlockCat is equal to 1 or 2, the following applies.
  - The derivation process for neighbouring 4x4 luma blocks specified in subclause 6.4.7.3 is invoked with luma4x4BlkIdx as input and the output is assigned to mbAddrN, luma4x4BlkIdxN (with N being either A or B).
  - The variable transBlockN is derived as follows.
    - If mbAddrN is available, the macroblock mbAddrN is not skipped, mb_type for the macroblock mbAddrN is not equal to I_PCM, and ((CodedBlockPatternLuma >> (luma4x4BlkIdxN >>2)) & 1) is not equal to 0 for the macroblock mbAddrN, the 4x4 luma block with luma4x4BlkIdxN of macroblock mbAddrN is assigned to transBlockN.
    - Otherwise, transBlockN is marked as not available.
- Otherwise, if ctxBlockCat is equal to 3, the following applies.
  - The derivation process for neighbouring macroblocks specified in subclause 6.4.7.1 is invoked and the output is assigned to mbAddrN (with N being either A or B).
  - The variable transBlockN is derived as follows.
    - If mbAddrN is available, the macroblock mbAddrN is not skipped, mb_type for the macroblock mbAddrN is not equal to I_PCM, and CodedBlockPatternChroma is not equal to 0 for the macroblock mbAddrN, the chroma DC block of chroma component iCbCr of macroblock mbAddrN is assigned to transBlockN.
    - Otherwise, transBlockN is marked as not available.
- Otherwise (ctxBlockCat is equal to 4), the following applies.
  - The derivation process for neighbouring 4x4 chroma blocks specified in subclause 6.4.7.4 is invoked with chroma4x4BlkIdx as input and the output is assigned to mbAddrN, chroma4x4BlkIdxN (with N being either A or B).
  - The variable transBlockN is derived as follows.
    - If mbAddrN is available, the macroblock mbAddrN is not skipped, mb_type for the macroblock mbAddrN is not equal to I_PCM, and CodedBlockPatternChroma is equal to 2 for the macroblock mbAddrN, the 4x4 chroma block with chroma4x4BlkIdxN of the chroma component iCbCr of macroblock mbAddrN is assigned to transBlockN.
    - Otherwise, transBlockN is marked as not available.

Let the variable condTermFlagN (with N being either A or B) be derived as follows.

- If any of the following conditions is true, condTermFlagN is set equal to 0
  - mbAddrN is not available and the current macroblock is coded in Inter prediction mode
  - mbAddrN is available and transBlockN is not available and mb_type for the macroblock mbAddrN is not equal to I_PCM
  - The current macroblock is coded in Intra prediction mode, constrained_intra_pred_flag is equal to 1, the macroblock mbAddrN is available and coded in Inter prediction mode, and slice data partitioning is in use (nal_unit_type is in the range of 2 through 4, inclusive).
- Otherwise, if any of the following conditions is true, condTermFlagN is set equal to 1
  - mbAddrN is not available and the current macroblock is coded in Intra prediction mode
  - mb_type for the macroblock mbAddrN is equal to I_PCM
- Otherwise, condTermFlagN is set equal to the value of the coded_block_flag of the transform block transBlockN that was decoded for the macroblock mbAddrN.

The variable ctxIdxInc(ctxBlockCat) is derived by

$$\text{ctxIdxInc(ctxBlockCat)} = \text{condTermFlagA} + 2 * \text{condTermFlagB} \tag{9-13}$$

9.3.3.1.2 Assignment process of ctxIdxInc using prior decoded bin values

Inputs to this process are ctxIdxOffset and binIdx.

Output of this process is ctxIdxInc.

Table 9-31 contains the specification of ctxIdxInc for the given values of ctxIdxOffset and binIdx.

For each value of ctxIdxOffset and binIdx, ctxIdxInc is derived by using some of the values of prior decoded bin values (b_0, b_1, b_2,…, b_k), where the value of the index k is less than the value of binIdx.

Table 9-31 – Specification of ctxIdxInc for specific values of ctxIdxOffset and binIdx

Value (name) of ctxIdxOffset	binIdx	ctxIdxInc
3	4	(b_3 != 0) ? 5: 6
	5	(b_3 != 0) ? 6: 7
14	2	(b_1 != 1) ? 2: 3
17	4	(b_3 != 0) ? 2: 3
27	2	(b_1 != 0) ? 4: 5
32	4	(b_3 != 0) ? 2: 3
36	2	(b_1 != 0) ? 2: 3

9.3.3.1.3 Assignment process of ctxIdxInc for syntax elements significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1

Inputs to this process are ctxIdxOffset and binIdx.

Output of this process is ctxIdxInc.

The assignment process of ctxIdxInc for syntax elements significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1 as well as for coded_block_flag depends on categories of different blocks denoted by the variable ctxBlockCat. The specification of these block categories is given in Table 9-32.

Table 9-32 – Specification of ctxBlockCat for the different blocks

Block description	maxNumCoeff	ctxBlockCat
block of luma DC transform coefficient levels (for macroblock coded in Intra_16x16 prediction mode)	16	0
block of luma AC transform coefficient levels (for macroblock coded in Intra_16x16 prediction mode)	15	1
block of luma transform coefficient levels (for macroblock not coded in Intra_16x16 prediction mode)	16	2
block of chroma DC transform coefficient levels	4	3
block of chroma AC transform coefficient levels	15	4

For the syntax elements significant_coeff_flag and last_significant_coeff_flag the scanning position scanningPos within the regarded block is assigned to ctxIdxInc, where scanningPos ranges from 0 to maxNumCoeff − 2, inclusive:

$$\text{ctxIdxInc} = \text{scanningPos} \qquad (9\text{-}14)$$

The scanning position for frame coded blocks relates to the zig-zag scan; the scanning position for field coded blocks relates to the field scan.

Let numDecodAbsLevelEq1 denotes the accumulated number of decoded transform coefficient levels with absolute value equal to 1, and let numDecodAbsLevelGt1 denotes the accumulated number of decoded transform coefficient levels with absolute value greater than 1. Both numbers are related to the same transform coefficient block, where the current decoding process takes place. Then, for decoding of coeff_abs_level_minus1, ctxIdxInc for coeff_abs_level_minus1 is specified depending on binIdx as follows.

- If binIdx is equal to 0, ctxIdxInc is derived by

$$\text{ctxIdxInc} = ((\text{numDecodAbsLevelGt1} \mathrel{!=} 0) ~?~ 0 : \text{Min}(4, 1 + \text{numDecodAbsLevelEq1})) \qquad (9\text{-}15)$$

- Otherwise (binIdx is greater than 0), ctxIdxInc is derived by

$$\text{ctxIdxInc} = 5 + \text{Min}(4, \text{numDecodAbsLevelGt1}) \qquad (9\text{-}16)$$

9.3.3.2 Arithmetic decoding process

Inputs to this process are the bypassFlag, ctxIdx as derived in subclause 9.3.3.1, and the state variables codIRange and codIOffset of the arithmetic decoding engine.

Output of this process is the value of the bin.

Figure 9-2 illustrates the whole arithmetic decoding process for a single bin. For decoding the value of a bin, the context index ctxIdx is passed to the arithmetic decoding process DecodeBin(ctxIdx), which is specified as follows.

- If bypassFlag is equal to 1, DecodeBypass( ) as specified in subclause 9.3.3.2.3 is invoked.
- Otherwise, if bypassFlag is equal to 0 and ctxIdx is equal to 276, DecodeTerminate( ) as specified in subclause 9.3.3.2.4 is invoked.
- Otherwise (bypassFlag is equal to 0 and ctxIdx is not equal to 276), DecodeDecision( ) as specified in subclause 9.3.3.2.1 shall be applied.

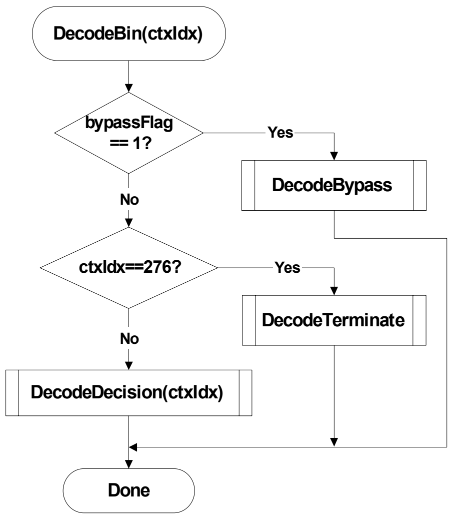


Figure 9-2 – Overview of the arithmetic decoding process for a single bin (informative)

NOTE - Arithmetic coding is based on the principle of recursive interval subdivision. Given a probability estimation p(0) and p(1) = 1 – p(0) of a binary decision (0, 1), an initially given code sub-interval with the range codIRange will be subdivided into two sub-intervals having range p(0) * codIRange and codIRange – p(0) * codIRange, respectively. Depending on the decision, which has been observed, the corresponding sub-interval will be chosen as the new code interval, and a binary code string pointing into that interval will represent the sequence of observed binary decisions. It is useful to distinguish between the most probable symbol (MPS) and the least probable symbol (LPS), so that binary decisions have to be identified as either MPS or LPS, rather than 0 or 1. Given this terminology, each context is specified by the probability p_{LPS} of the LPS and the value of MPS (valMPS), which is either 0 or 1.

The arithmetic core engine in this Recommendation | International Standard has three distinct properties:

- The probability estimation is performed by means of a finite-state machine with a table-based transition process between 64 different representative probability states $\{ p_{LPS}(pStateIdx) \mid 0 <= pStateIdx < 64 \}$ for the LPS probability p_{LPS}. The numbering of the states is arranged in such a way that the probability state with index pStateIdx = 0 corresponds to an LPS probability value of 0.5, with decreasing LPS probability towards higher state indices.
- The range codIRange representing the state of the coding engine is quantised to a small set $\{Q_1,\ldots,Q_4\}$ of pre-set quantisation values prior to the calculation of the new interval range. Storing a table containing all 64x4 pre-computed product values of $Q_i * p_{LPS}(pStateIdx)$ allows a multiplication-free approximation of the product codIRange * $p_{LPS}(pStateIdx)$.
- For syntax elements or parts thereof for which an approximately uniform probability distribution is assumed to be given a separate simplified encoding and decoding bypass process is used.

9.3.3.2.1 Arithmetic decoding process for a binary decision

Inputs to this process are ctxIdx, codIRange, and codIOffset.

Outputs of this process are the decoded value binVal, and the updated variables codIRange and codIOffset.

Figure 9-3 shows the flowchart for decoding a single decision (DecodeDecision).

1. The value of the variable codIRangeLPS is derived as follows.
   - Given the current value of codIRange, the variable qCodIRangeIdx is derived by

     qCodIRangeIdx =(codIRange >> 6) & 0x03 (9-17)

   - Given qCodIRangeIdx and pStateIdx associated with ctxIdx, the value of the variable rangeTabLPS as specified in Table 9-33 is assigned to codIRangeLPS:

     codIRangeLPS = rangeTabLPS[pStateIdx][qCodIRangeIdx] (9-18)

2. The variable codIRange is set equal to codIRange – codIRangeLPS and the following applies.
   - If codIOffset is greater than or equal to codIRange, the variable binVal is set equal to 1 - valMPS, codIOffset is decremented by codIRange, and codIRange is set equal to codIRangeLPS.
   - Otherwise, the variable binVal is set equal to valMPS.

Given the value of binVal, the state transition is performed as specified in subclause 9.3.3.2.1.1. Depending on the current value of codIRange, renormalization is performed as specified in subclause 9.3.3.2.2.

9.3.3.2.1.1 State transition process

Inputs to this process are the current pStateIdx, the decoded value binVal and valMPS values of the context variable associated with ctxIdx.

Outputs of this process are the updated pStateIdx and valMPS of the context variable associated with ctxIdx.

Depending on the decoded value binVal, the update of the two variables pStateIdx and valMPS associated with ctxIdx is derived as follows:

```
if( binVal  = =  valMPS )
    pStateIdx = transIdxMPS( pStateIdx )
else {                                                          (9-19)
    if( pStateIdx  = =  0 )
        valMPS = 1 – valMPS
    pStateIdx = transIdxLPS( pStateIdx )
}
```

Table 9-34 specifies the transition rules transIdxMPS( ) and transIdxLPS( ) after decoding the value of valMPS and 1 – valMPS, respectively.

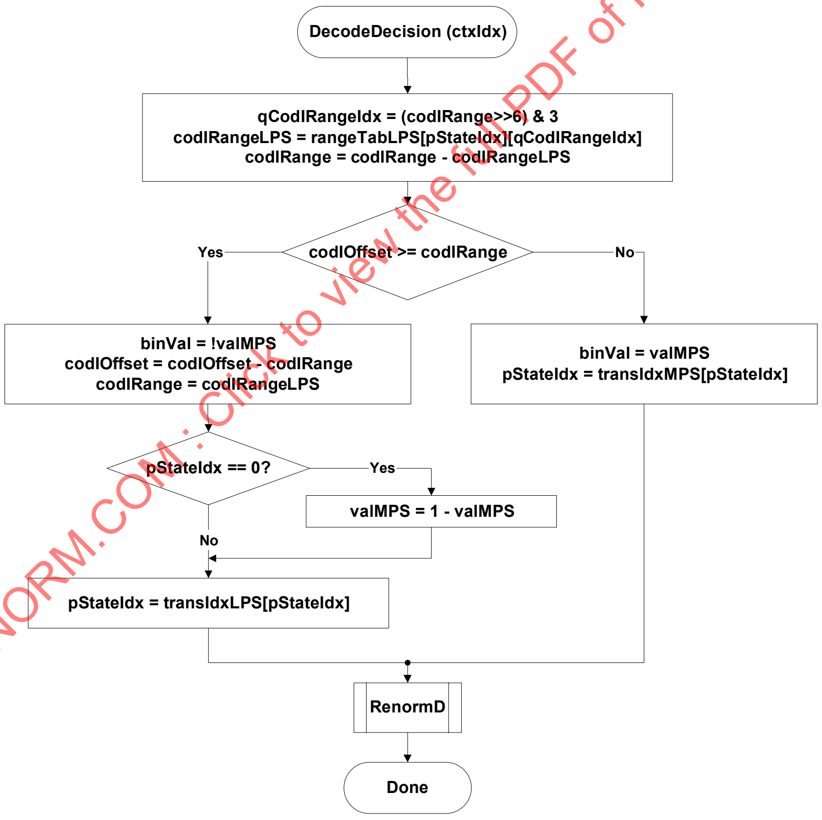

DecodeDecision (ctxIdx)
qCodIRangeIdx = (codIRange>>6) & 3
codIRangeLPS = rangeTabLPS[pStateIdx][qCodIRangeIdx]
codIRange = codIRange - codIRangeLPS
codIOffset >= codIRange
Yes
No
binVal = !valMPS
codIOffset = codIOffset - codIRange
codIRange = codIRangeLPS
binVal = valMPS
pStateIdx = transIdxMPS[pStateIdx]
pStateIdx == 0?
Yes
valMPS = 1 - valMPS
No
pStateIdx = transIdxLPS[pStateIdx]
RenormD
Done

Figure 9-3 – Flowchart for decoding a decision

Table 9-33 – Specification of rangeTabLPS depending on pStateIdx and qCodIRangeIdx

pStateIdx	qCodIRangeIdx				pStateIdx	qCodIRangeIdx			
	0	1	2	3		0	1	2	3
0	128	176	208	240	**32**	27	33	39	45
1	128	167	197	227	**33**	26	31	37	43
2	128	158	187	216	**34**	24	30	35	41
3	123	150	178	205	**35**	23	28	33	39
4	116	142	169	195	**36**	22	27	32	37
5	111	135	160	185	**37**	21	26	30	35
6	105	128	152	175	**38**	20	24	29	33
7	100	122	144	166	**39**	19	23	27	31
8	95	116	137	158	**40**	18	22	26	30
9	90	110	130	150	**41**	17	21	25	28
10	85	104	123	142	**42**	16	20	23	27
11	81	99	117	135	**43**	15	19	22	25
12	77	94	111	128	**44**	14	18	21	24
13	73	89	105	122	**45**	14	17	20	23
14	69	85	100	116	**46**	13	16	19	22
15	66	80	95	110	**47**	12	15	18	21
16	62	76	90	104	**48**	12	14	17	20
17	59	72	86	99	**49**	11	14	16	19
18	56	69	81	94	**50**	11	13	15	18
19	53	65	77	89	**51**	10	12	15	17
20	51	62	73	85	**52**	10	12	14	16
21	48	59	69	80	**53**	9	11	13	15
22	46	56	66	76	**54**	9	11	12	14
23	43	53	63	72	**55**	8	10	12	14
24	41	50	59	69	**56**	8	9	11	13
25	39	48	56	65	**57**	7	9	11	12
26	37	45	54	62	**58**	7	9	10	12
27	35	43	51	59	**59**	7	8	10	11
28	33	41	48	56	**60**	6	8	9	11
29	32	39	46	53	**61**	6	7	9	10
30	30	37	43	50	**62**	6	7	8	9
31	29	35	41	48	**63**	2	2	2	2

Table 9-34 – State transition table

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

9.3.3.2.2 Renormalization process in the arithmetic decoding engine

Inputs to this process are bits from slice data and the variables codIRange and codIOffset.

Outputs of this process are the updated variables codIRange and codIOffset.

A flowchart of the renormalization is shown in Figure 9-4. The current value of codIRange is first compared to 0x0100 and further steps are specified as follows.

- If codIRange is greater than or equal to 0x0100, no renormalization is needed and the RenormD process is finished;
- Otherwise (codIRange is less than 0x0100), the renormalization loop is entered. Within this loop, the value of codIRange is doubled, i.e., left-shifted by 1 and a single bit is shifted into codIOffset by using read_bits(1).

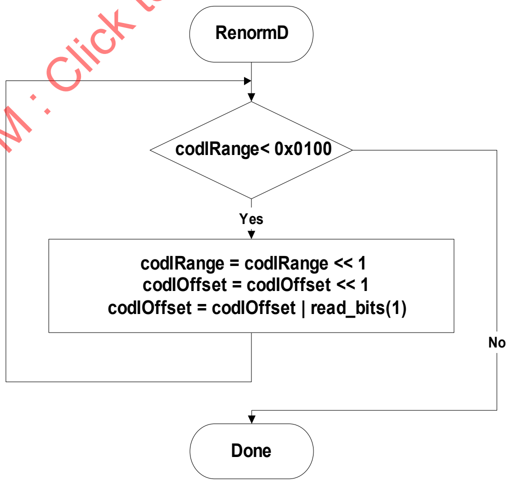

RenormD
codIRange< 0x0100
Yes
codIRange = codIRange << 1
codIOffset = codIOffset << 1
codIOffset = codIOffset | read_bits(1)
No
Done

Figure 9-4 – Flowchart of renormalization

9.3.3.2.3 Bypass decoding process for binary decisions

Inputs to this process are bits from slice data and the variables codIRange and codIOffset.

Outputs of this process are the updated variable codIOffset and the decoded value binVal.

The bypass decoding process is invoked when bypassFlag is equal to 1. Figure 9-5 shows a flowchart of the corresponding process.

First, the value of codIOffset is doubled, i.e., left-shifted by 1 and a single bit is shifted into codIOffset by using read_bits(1). Then, the value of codIOffset is compared to the value of codIRange and further steps are specified as follows.

- If codIOffset is greater than or equal to codIRange, the variable binVal is set equal to 1 and codIOffset is decremented by codIRange.
- Otherwise (codIOffset is less than codIRange), the variable binVal is set equal to 0.

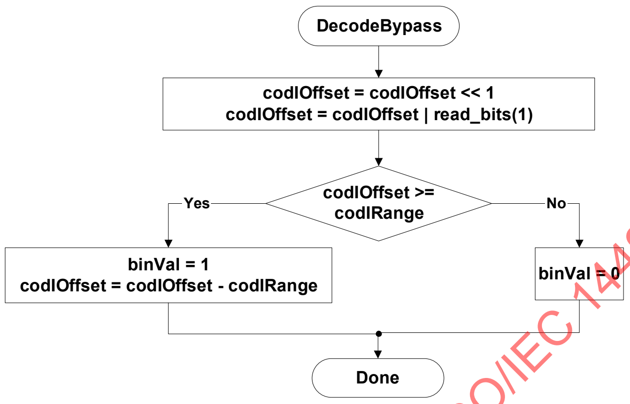

DecodeBypass
codIOffset = codIOffset << 1
codIOffset = codIOffset | read_bits(1)
codIOffset >= codIRange
Yes
No
binVal = 1
codIOffset = codIOffset - codIRange
binVal = 0
Done

Figure 9-5 – Flowchart of bypass decoding process

9.3.3.2.4 Decoding process for binary decisions before termination

Inputs to this process are bits from slice data and the variables codIRange and codIOffset.

Outputs of this process are the updated variables codIRange and codIOffset, and the decoded value binVal.

This special decoding routine applies to decoding of end_of_slice_flag and of the bin indicating the I_PCM mode corresponding to ctxIdx equal to 276. Figure 9-6 shows the flowchart of the corresponding decoding process, which is specified as follows.

First, the value of codIRange is decremented by 2. Then, the value of codIOffset is compared to the value of codIRange and further steps are specified as follows.

- If codIOffset is greater than or equal to codIRange, the variable binVal is set equal to 1, no renormalization is carried out, and CABAC decoding is terminated. The last bit inserted in register codIOffset is equal to 1. When decoding end_of_slice_flag, this last bit inserted in register codIOffset is interpreted as rbsp_stop_one_bit.
- Otherwise (codIOffset is less than codIRange), the variable binVal is set equal to 0 and renormalization is performed as specified in subclause 9.3.3.2.2.

  NOTE – This procedure may also be implemented using DecodeDecision(ctxIdx) with ctxIdx = 276. In the case where the decoded value is equal to 1, seven more bits would be read by DecodeDecision(ctxIdx) and a decoding process would have to adjust its bitstream pointer accordingly to properly decode following syntax elements.

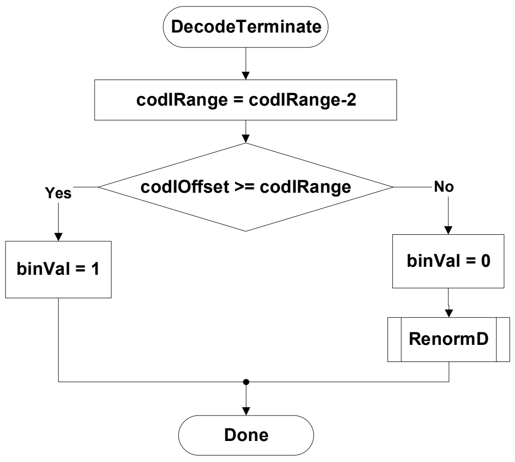


Figure 9-6 – Flowchart of decoding a decision before termination

9.3.4 Arithmetic encoding process (informative)

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

Inputs to this process are decisions that are to be encoded and written.

Outputs of this process are bits that are written to the RBSP.

This informative subclause describes an arithmetic encoding engine that matches the arithmetic decoding engine described in subclause 9.3.3.2. The encoding engine is essentially symmetric with the decoding engine, i.e., procedures are called in the same order. The following procedures are described in this section: InitEncoder, EncodeDecision, EncodeBypass, EncodeTerminate, which correspond to InitDecoder, DecodeDecision, DecodeBypass, and DecodeTerminate, respectively. The state of the arithmetic encoding engine is represented by a value of the variable codILow pointing to the lower end of a sub-interval and a value of the variable codIRange specifying the corresponding range of that sub-interval.

9.3.4.1 Initialisation process for the arithmetic encoding engine (informative)

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

This process is invoked before encoding the first macroblock of a slice, and after encoding the pcm_alignment_zero_bit and all pcm_byte data for a macroblock of type I_PCM.

Outputs of this process are the values codILow, codIRange, firstBitFlag, bitsOutstanding, and symCnt of the arithmetic encoding engine.

In the initialisation procedure of the encoder, codILow is set equal to 0, and codIRange is set equal to 0x01FE. Furthermore, a firstBitFlag is set equal to 1, and bitsOutstanding and symCnt counters are set equal to 0.

NOTE – The minimum register precision required for codILow is 10 bits and for CodIRange is 9 bits. The precision required for the counters bitsOutstanding and symCnt should be sufficiently large to prevent overflow of the related registers. When MaxBinCountInSlice denotes the maximum total number of binary decisions to encode in one slice, the minimum register precision required for the variables bitsOutstanding and symCnt is given by Ceil(Log2(MaxBinCountInSlice + 1)).

9.3.4.2 Encoding process for a binary decision (informative)

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

Inputs to this process are the context index ctxIdx, the value of binVal to be encoded, and the variables codIRange, codILow and symCnt.

Outputs of this process are the variables codIRange, codILow, and symCnt.

Figure 9-7 shows the flowchart for encoding a single decision. In a first step, the variable codIRangeLPS is derived as follows.

Given the current value of codIRange, codIRange is mapped to the index qCodIRangeIdx of a quantised value of codIRange by using Equation 9-17. The value of qCodIRangeIdx and the value of pStateIdx associated with ctxIdx are used to determine the value of the variable rangeTabLPS as specified in Table 9-33, which is assigned to codIRangeLPS. The value of codIRange – codIRangeLPS is assigned to codIRange.

In a second step, the value of binVal is compared to valMPS associated with ctxIdx. When binVal is different from valMPS, codIRange is added to codILow and codIRange is set equal to the value codIRangeLPS. Given the encoded decision, the state transition is performed as specified in subclause 9.3.3.2.1.1. Depending on the current value of codIRange, renormalization is performed as specified in subclause 9.3.4.3. Finally, the variable symCnt is incremented by 1.

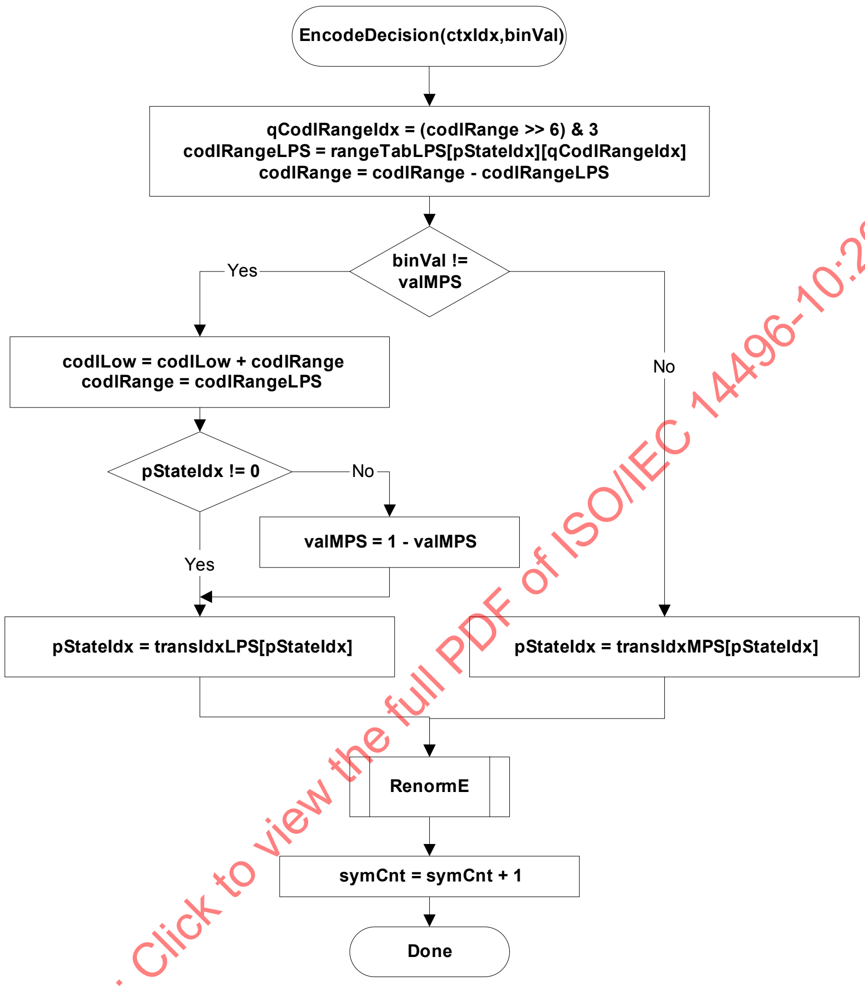

EncodeDecision(ctxIdx,binVal)
qCodIRangeIdx = (codIRange >> 6) & 3
codIRangeLPS = rangeTabLPS[pStateIdx][qCodIRangeIdx]
codIRange = codIRange - codIRangeLPS
binVal != valMPS
Yes
No
codILow = codILow + codIRange
codIRange = codIRangeLPS
pStateIdx != 0
No
valMPS = 1 - valMPS
Yes
pStateIdx = transIdxLPS[pStateIdx]
pStateIdx = transIdxMPS[pStateIdx]
RenormE
symCnt = symCnt + 1
Done

Figure 9-7 – Flowchart for encoding a decision

9.3.4.3 Renormalization process in the arithmetic encoding engine (informative)

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

Inputs to this process are the variables codIRange, codILow, firstBitFlag, and bitsOutstanding.

Outputs of this process are zero or more bits written to the RBSP and the updated variables codIRange, codILow, firstBitFlag, and bitsOutstanding.

Renormalization is illustrated in Figure 9-8.

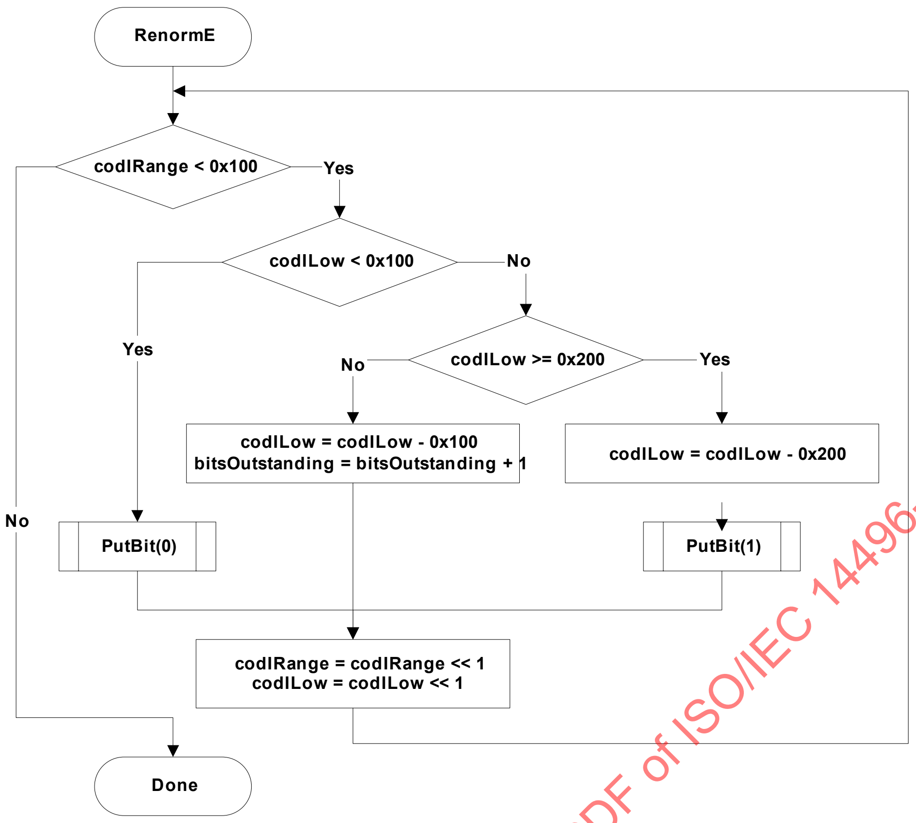


Figure 9-8 – Flowchart of renormalization in the encoder

The PutBit( ) procedure described in Figure 9-9 provides carry over control. It uses the function WriteBits(B, N) that writes N bits with value B to the bitstream and advances the bitstream pointer by N bit positions. This function assumes the existence of a bitstream pointer with an indication of the position of the next bit to be written to the bitstream by the encoding process.

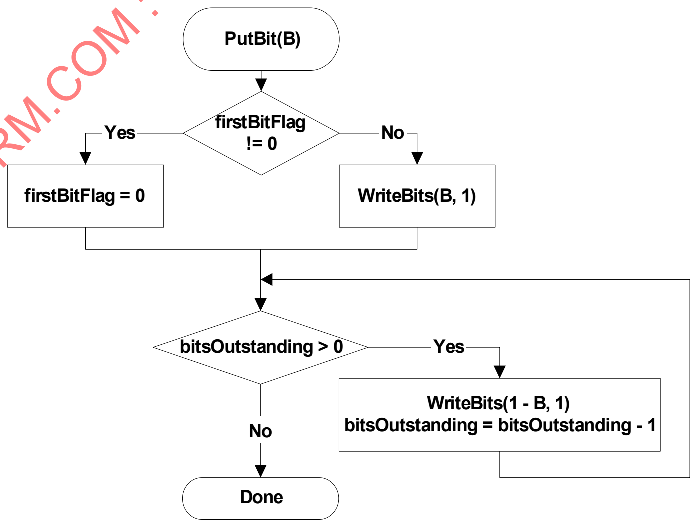


Figure 9-9 – Flowchart of PutBit(B)

9.3.4.4 Bypass encoding process for binary decisions (informative)

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

Inputs to this process are the variables binVal, codILow, codIRange, bitsOutstanding, and symCnt.

Output of this process is a bit written to the RBSP and the updated variables codILow, bitsOutstanding, and symCnt.

This encoding process applies to all binary decisions with bypassFlag equal to 1. Renormalization is included in the specification of this process as given in Figure 9-10.

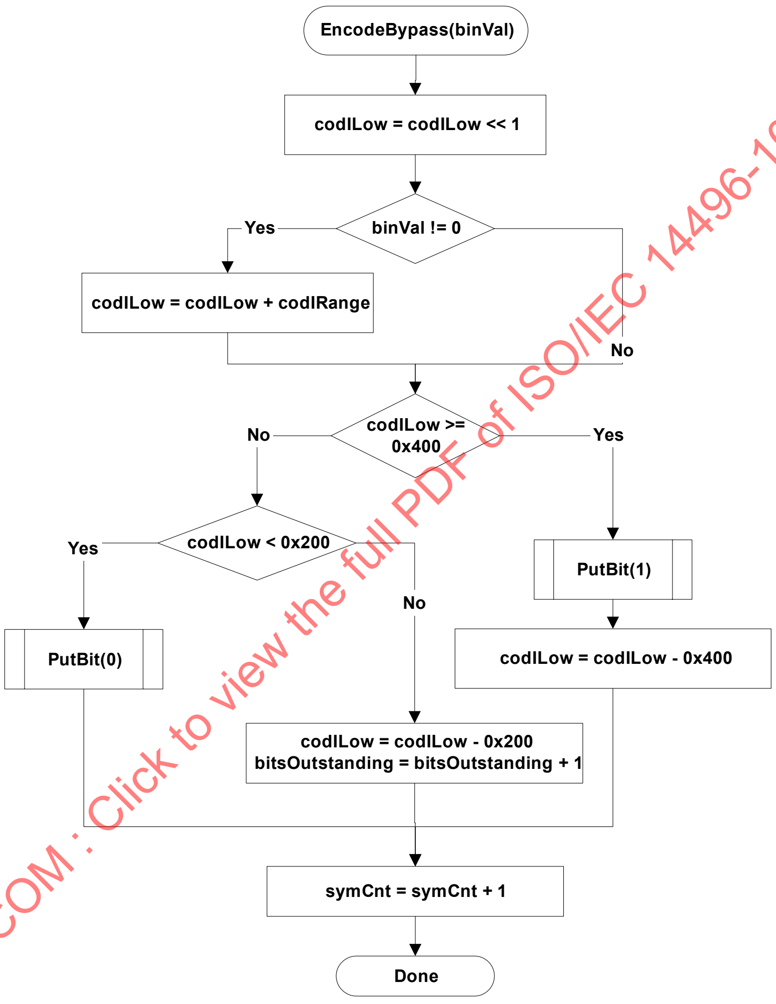


Figure 9-10 – Flowchart of encoding bypass

9.3.4.5 Encoding process for a binary decision before termination (informative)

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

Inputs to this process are the variables binVal, codIRange, codILow, bitsOutstanding, and symCnt.

Outputs of this process are zero or more bits written to the RBSP and the updated variables codILow, codIRange, bitsOutstanding, and symCnt.

This encoding routine shown in Figure 9-11 applies to encoding of the end_of_slice_flag and of the bin indicating the I_PCM mb_type both associated with ctxIdx equal to 276.

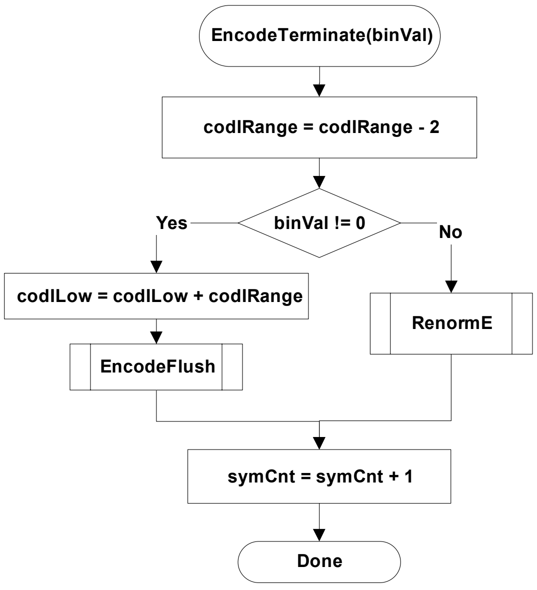


Figure 9-11 – Flowchart of encoding a decision before termination

When the value of binVal to encode is equal to 1, CABAC encoding is terminated and the flushing procedure shown in Figure 9-12 is applied. In this flushing procedure, the last bit written by WriteBits(B, N) is equal to 1. When encoding end_of_slice_flag, this last bit is interpreted as the rbsp_stop_one_bit.

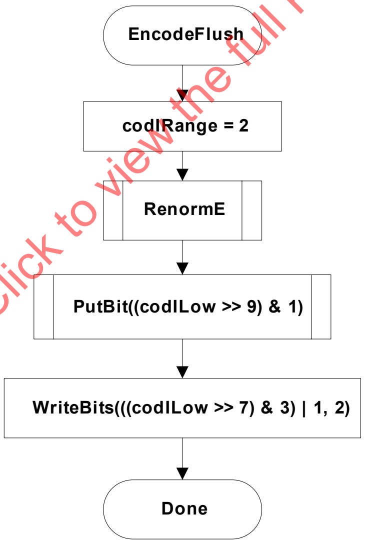

EncodeFlush
codIRange = 2
RenormE
PutBit((codILow >> 9) & 1)
WriteBits(((codILow >> 7) & 3) | 1, 2)
Done

Figure 9-12 – Flowchart of flushing at termination

9.3.4.6 Byte stuffing process (informative)

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

This process is invoked after encoding the last macroblock of the last slice of a picture and after encapsulation.

Inputs to this process are the number of bytes NumBytesInVclNALunits of all VCL NAL units of a picture, the number of macroblocks PicSizeInMbs in the picture, and the number of binary symbols BinCountsInNALunits resulting from encoding the contents of all VCL NAL units of the picture.

Outputs of this process are zero or more bytes appended to the NAL unit.

Let the variable k be set equal to Ceil((Ceil((3 * BinCountsInNALunits – 3 * 96 * PicSizeInMbs) / 32) – NumBytesInVclNALunits) / 3). Depending on the variable k the following applies.

- If k is less than or equal to 0, no cabac_zero_word is appended to the NAL unit.
- Otherwise (k is greater than 0), the 3-byte sequence 0x000003 is appended k times to the NAL unit after encapsulation, where the first two bytes 0x0000 represent a cabac_zero_word and the third byte 0x03 represents an emulation_prevention_three_byte.

Annex A
Profiles and levels

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

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 Recommendation | 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.

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

Each level specifies a set of limits on the values that may be taken by the syntax elements of this Recommendation | 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 Recommendation | 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 Recommendation | 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

A.2.1 Baseline profile

Bitstreams conforming to the 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.
– Sequence parameter sets shall have frame_mbs_only_flag equal to 1.
– 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.
– Picture parameter sets shall have num_slice_groups_minus1 in the range of 0 to 7, inclusive.
– The level constraints specified for the Baseline profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the Baseline profile is specified by profile_idc being equal to 66.

Decoders conforming to the Baseline profile at a specific level shall be capable of decoding all bitstreams in which profile_idc is equal to 66 or constraint_set0_flag is equal to 1 and in which level_idc represents a level less than or equal to the specified level.

A.2.2 Main profile

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

– Only I, P, and B 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.
– 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.
– The level constraints specified for the Main profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the Main profile is specified by profile_idc being equal to 77.

Decoders conforming to the Main profile at a specified level shall be capable of decoding all bitstreams in which profile_idc is equal to 77 or constraint_set1_flag is equal to 1 and in which level_idc represents a level less than or equal to the specified level.

A.2.3 Extended profile

Bitstreams conforming to the Extended profile shall obey the following constraints:

– Sequence parameter sets shall have direct_8x8_inference_flag equal to 1.
– Picture parameter sets shall have entropy_coding_mode_flag equal to 0.
– Picture parameter sets shall have num_slice_groups_minus1 in the range of 0 to 7, inclusive.
– The level constraints specified for the Extended profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the Extended profile is specified by profile_idc being equal to 88.

Decoders conforming to the Extended profile at a specified level shall be capable of decoding all bitstreams in which profile_idc is equal to 88 or constraint_set2_flag is equal to 1 and in which level_idc represents a level less than or equal to specified level.

Decoders conforming to the Extended profile at a specified level shall also be capable of decoding all bitstreams in which profile_idc is equal to 66 or constraint_set0_flag is equal to 1, in which level_idc represents a level less than or equal to the specified level.

A.3 Levels

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.

A.3.1 Profile-independent level limits

Let the variable fR be derived as follows.

- If picture n is a frame, fR is set equal to 1 ÷ 172.
- Otherwise (picture n is a field), fR is set equal to 1 ÷ (172 * 2).

Bitstreams conforming to any 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, and PicSizeInMbs is the number of macroblocks in picture n.

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 256 * ChromaFormatFactor * (PicSizeInMbs + 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 256 * ChromaFormatFactor * 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 <= MaxDpbSize, where MaxDpbSize is equal to Min(1024 * MaxDPB / (PicWidthInMbs * FrameHeightInMbs * 256 * ChromaFormatFactor), 16) and MaxDPB is given in Table A-1 in units of 1024 bytes. max_dec_frame_buffering is also called DPB size.

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] is given by Equation E-13 and CpbSize[SchedSelIdx] is given by Equation E-14 when vcl_hrd_parameters_present_flag is equal to 1. 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. CpbSize[SchedSelIdx] is also called CPB size.

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] is given by Equation E-13 and CpbSize[SchedSelIdx] is given by Equation E-14 when nal_hrd_parameters_present_flag equal to 1. 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) Vertical motion vector component range does not exceed MaxVmvR in units of luma frame samples, where MaxVmvR is specified in Table A-1

l) Horizontal motion vector range does not exceed the range of -2048 to 2047.75, inclusive, in units of luma samples

m) Number of motion vectors per two consecutive macroblocks in decoding order (also applying to the total from the last macroblock of a slice and the first macroblock of the next slice in decoding order, and in particular also applying to the total from the last macroblock of the last slice of a picture and the first macroblock of the first slice of the next picture in decoding order) 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.

n) Number of bits of macroblock_layer( ) data for any macroblock is not greater than 128 + 2048 * ChromaFormatFactor. Depending on entropy_coding_mode_flag, the bits of macroblock_layer( ) data are counted as follows.

- If entropy_coding_mode_flag is equal to 0, the number of bits of macroblock_layer( ) data is given by the number of bits in the macroblock_layer( ) syntax structure for a macroblock.
- Otherwise (entropy_coding_mode_flag is equal to 1), the number of bits of macroblock_layer( ) data for a macroblock is given by the number of times read_bits(1) is called in subclauses 9.3.3.2.2 and 9.3.3.2.3 when parsing the macroblock_layer( ) associated with the macroblock.

Table A-1 below specifies the limits for each level. Entries marked "-" in Table A-1 denote the absence of a corresponding limit.

Conformance to a particular level shall be specified by setting the syntax element level_idc equal to a value of ten times the level number 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 MaxDPB (1024 bytes)	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	148.5	64	175	[-64,+63.75]	2	-
1.1	3 000	396	337.5	192	500	[-128,+127.75]	2	-
1.2	6 000	396	891.0	384	1 000	[-128,+127.75]	2	-
1.3	11 880	396	891.0	768	2 000	[-128,+127.75]	2	-
2	11 880	396	891.0	2 000	2 000	[-128,+127.75]	2	-
2.1	19 800	792	1 782.0	4 000	4 000	[-256,+255.75]	2	-
2.2	20 250	1 620	3 037.5	4 000	4 000	[-256,+255.75]	2	-
3	40 500	1 620	3 037.5	10 000	10 000	[-256,+255.75]	2	32
3.1	108 000	3 600	6 750.0	14 000	14 000	[-512,+511.75]	4	16
3.2	216 000	5 120	7 680.0	20 000	20 000	[-512,+511.75]	4	16
4	245 760	8 192	12 288.0	20 000	25 000	[-512,+511.75]	4	16
4.1	245 760	8 192	12 288.0	50 000	62 500	[-512,+511.75]	2	16
4.2	491 520	8 192	12 288.0	50 000	62 500	[-512,+511.75]	2	16
5	589 824	22 080	41 400.0	135 000	135 000	[-512,+511.75]	2	16
5.1	983 040	36 864	69 120.0	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.3.3 shows the effect of these limits on frame rates for several example picture formats.

A.3.2 Profile-specific level limits

a) In bitstreams conforming to the Main profile, the removal time of access unit 0 shall satisfy the constraint that the number of slices in picture 0 is less than or equal to (PicSizeInMbs + MaxMBPS * (t_r(0) - $t_{r,n}$(0))) ÷ SliceRate, where SliceRate is the value specified in Table A-3 that applies to picture 0.

b) In bitstreams conforming to the Main profile, the difference between consecutive removal time of access units n and n - 1 (with n > 0) shall satisfy the constraint that the number of slices in picture n is less than or equal to MaxMBPS * (t_r(n) - t_r(n - 1)) ÷ SliceRate, where SliceRate is the value specified in Table A-3 that applies to picture n.

c) In bitstreams conforming to the Main profile, sequence parameter sets shall have direct_8x8_inference_flag equal to 1 for the levels specified in Table A-3.

NOTE – direct_8x8_inference_flag is not relevant to the Baseline profile as it does not allow B slice types (specified in subclause A.2.1), and direct_8x8_inference_flag is equal to 1 for all levels of the Extended profile (specified in subclause A.2.3).

d) In bitstreams conforming to the Main and Extended profiles, sequence parameter sets shall have frame_mbs_only_flag equal to 1 for the levels specified in Table A-3 for the Main profile and in Table A-4 for the Extended profile.

NOTE – frame_mbs_only_flag is equal to 1 for all levels of the Baseline profile (specified in subclause A.2.1).

e) In bitstreams conforming to the Main and Extended profiles, the value of sub_mb_type in B macroblocks shall not be equal to B_Bi_8x4, B_Bi_4x8, or B_Bi_4x4 for the levels in which MinLumaBiPredSize is shown as 8x8 in Table A-3 for the Main profile and in Table A-4 for the Extended profile.

f) In bitstreams conforming to the Baseline and Extended profiles, $(\text{xInt}_{\text{max}} - \text{xInt}_{\text{min}} + 6) * (\text{yInt}_{\text{max}} - \text{yInt}_{\text{min}} + 6) <= \text{MaxSubMbRectSize}$ in macroblocks coded with mb_type equal to P_8x8, P_8x8ref0 or B_8x8 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 list (list 0 or list 1) for each 8x8 sub-macroblock, where NumSubMbPart(sub_mb_type) > 1, where MaxSubMbRectSize is specified in Table A-2 for the Baseline profile and in Table A-4 for the Extended 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

A.3.2.1 Baseline profile limits

Table A-2 specifies limits for each level that are specific to bitstreams conforming to the Baseline profile. Entries marked "-" in Table A-2 denote the absence of a corresponding limit.

Table A-2 – Baseline profile level limits

Level number	MaxSubMbRectSize
1	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.3.2.2 Main profile limits

Table A-3 specifies limits for each level that are specific to bitstreams conforming to the Main profile. Entries marked "-" in Table A-3 denote the absence of a corresponding limit.